COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

MASTER'S DEGREE IN TELECOMMUNICATIONS ENGINEERING

MASTER'S THESIS

SENSITIVITY ANALYSIS OF THE CNN LEARNING PROCESS USING SYNTHETIC IMAGES

Author: Irene España Novillo

Supervisors:
Dr. Eugenio Francisco Sánchez Úbeda
Dr. Jaime Boal Martín-Larrauri

MADRID
February 2021

This disertation was typeset with LaTeX and compiled in TeXstudio using the TeXLive 2017 distribution. The font families used are Bitstream Charter, Utopia, Bookman and Computer Modern. Unless otherwise noted, all figures were created by the author using Microsoft PowerPoint®, GIMP®, R® and Python®.

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título "Sensitivity analysis of the CNN learning process using synthetic images" en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2020/2021 es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Fdo.: Irene España Novillo Fecha: 09 / 02 / 2021

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Firmado digitalmente por BOAL MARTIN LARRAURI JAIME - 05304600H
Fecha: 2021.02.10 08:55:47 +01'00'

Fdo.: Jaime Boal Martín-Larrauri Fecha: 10 / 02 / 2021

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Eugenio Fco. Sánchez Úbeda

Firmado digitalmente por Eugenio Fco. Sánchez Úbeda
Fecha: 2021.02.10 09:47:34 +01'00'

Fdo.: Eugenio Francisco Sánchez Úbeda Fecha: 10 / 02 / 2021

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

MASTER'S DEGREE IN TELECOMMUNICATIONS ENGINEERING

MASTER'S THESIS

SENSITIVITY ANALYSIS OF THE CNN LEARNING PROCESS USING SYNTHETIC IMAGES

Author: Irene España Novillo

Supervisors:
Dr. Eugenio Francisco Sánchez Úbeda
Dr. Jaime Boal Martín-Larrauri

MADRID
February 2021

This disertation was typeset with LaTeX and compiled in TeXstudio using the TeXLive 2017 distribution. The font families used are Bitstream Charter, Utopia, Bookman and Computer Modern. Unless otherwise noted, all figures were created by the author using Microsoft PowerPoint®, GIMP®, R® and Python®.

ANALISIS DE SENSIBILIDAD DEL PROCESO DE APRENDIZAJE DE LA CNN USANDO IMÁGENES SINTÉTICAS

Autora: España Novillo, Irene.
Directores: Sánchez Úbeda, Eugenio Francisco y Boal Martín-Larrauri, Jaime.
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas.

RESUMEN DEL PROYECTO

Incrementar la interpretabilidad de las CNN es uno de los principales retos en el campo de Deep Learning actualmente y en general, se ha abordado desde enfoques muy complejos. Este proyecto presenta una nueva metodología más sencilla para entender el proceso que sigue una CNN para aprender a clasificar imágenes con niveles tan altos de precisión. Varios modelos compuestos por una capa convolucional con un único filtro se han entrenado con conjuntos de datos generados sintéticamente de tal forma que se centran en resolver un problema específico. Se ha comprobado que obtienen buenos resultados, así como que facilitan la interpretación de las características y los filtros, incluso permitiendo aplicar técnicas de Machine Learning para realizar los análisis, debido a la baja cantidad de parámetros a estudiar.

Palabras clave: Deep Learning, CNN, IA interpretable, modelo mínimo, proceso de aprendizaje.

1. Introducción

Los elevados niveles de precisión alcanzados por las redes neuronales convolucionales y la eliminación de la intervención manual, han incrementado su uso en cualquier actividad que implique reconocimiento de imágenes. Sin embargo, estos modelos poseen una elevada complejidad que hacen que su funcionamiento sea difícil de entender. Hasta ahora, las aproximaciones al problema de la interpretabilidad de las CNN se llevan a cabo mediante el análisis de complejas arquitecturas entrenadas con conjuntos de imágenes reales. La metodología que se presenta en este proyecto hace uso del enfoque opuesto, estudiando el problema desde una perspectiva más sencilla.

2. Definición del proyecto

Se ha realizado la descripción e implementación de una metodología análisis del proceso de aprendizaje de las CNN, la cual aumenta la interpretabilidad de las mismas. Partiendo de un conjunto de imágenes sencillas generado sintéticamente, se entrenan una serie de modelos con arquitecturas de baja complejidad y se realiza el análisis de los resultados obtenidos de manera exhaustiva. Los problemas estudiados son el escalado y rotación de figuras geométricas con y sin ruido.

3. Descripción de la metodología

El diagrama de flujo seguido en el proceso se muestra en la Figura 1 y consta principalmente de cuatro fases:

- **Generación del conjunto de datos**. Los conjuntos tanto de entrenamiento como de validación contienen imágenes de figuras geométricas pertenecientes a tres clases distintas: elipses, rectángulos y triángulos. Dependiendo del problema, se aplica una transformación a las imágenes: escalado (variación del tamaño), rotación (giro respecto al eje perpendicular al plano) y adicción de ruido.
- **Búsqueda del modelo mínimo**. Con el dataset generado se entrenan una serie de arquitecturas modificando los valores de los hiperparámetros y se selecciona aquel modelo que consigue los mayores niveles de precisión con el mínimo número de parámetros (menor complejidad).
- **Interpretación de resultados**. Engloba el análisis de las características obtenidas por el modelo mínimo y cómo le permiten realizar la clasificación, además del análisis del significado físico del modelo; identificando qué partes de la imagen son importantes y cómo el filtro es capaz de extraerlas. Para los experimentos de los casos con ruido, además se ha realizado un análisis de las muestras clasificadas erróneamente por el modelo, pero este estudio es opcional.
- **Conclusiones**. El proceso de análisis da como resultado una serie de lecciones aprendidas sobre el funcionamiento del proceso de aprendizaje de los modelos estudiados.

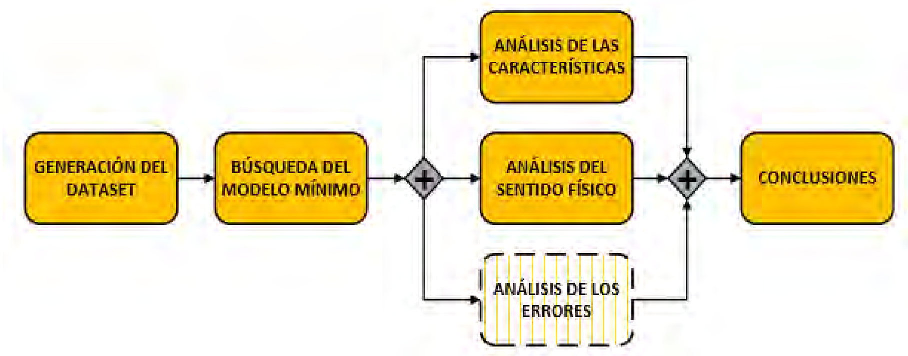

Figura 1. Diagrama de flujo de la metodología.

4. Resultados

- En cuanto al problema de escalado y rotación sin ruido, los modelos con una sola característica son capaces de realizar la clasificación correctamente, logrando precisiones de alrededor del 100% en el primer caso y alrededor del 60% en el segundo. Sin embargo, inevitablemente se introduce sesgo en estos conjuntos de datos, lo que hace necesario analizar los mismos problemas añadiendo ruido a las imágenes. El filtro se centra en extraer los bordes de la imagen.
- Para el problema de escalado y rotación con ruido, se obtienen modelos más complejos. Para los primeros, la arquitectura tiene cuatro características y dieciséis para los segundos, clasificando en ambos casos más del 90% de las imágenes. El filtro también extrae bordes en este caso y tiene un comportamiento opuesto dependiendo de la combinación de colores de fondo y figura (blanco o negro).

Un ejemplo del análisis del significado físico del modelo que permite ver los mapas de activación resultantes tras la aplicación del filtro y las localizaciones de las características extraídas en distintas imágenes se muestra en la Figura 2.

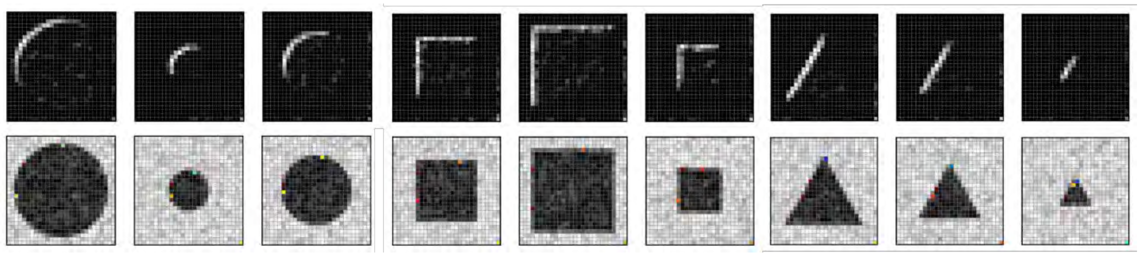

Figura 2. Ejemplo de mapas de activación y representación de los píxeles de máxima activación para imágenes del problema de escalado y ruido.

5. Conclusiones

Este proyecto ha demostrado que es posible entrenar modelos de CNN muy simples capaces de realizar la clasificación correctamente, contrariamente a los enfoques generalizados que se centran en entrenar arquitecturas muy complejas. En lugar de utilizar una metodología *top-bottom* en la que se construye un modelo complejo y luego se reduce el número de parámetros tratando de no perder precisión, en este caso las arquitecturas están diseñadas para ser simples desde el principio y resolver un solo problema, lo cual es posible gracias a los conjuntos de datos sintéticos. Se debe prestar especial atención a la semilla de inicialización y al número de *epochs* cuando se busca el modelo mínimo. Además, se debe agregar ruido a los conjuntos de datos que se generan sintéticamente para evitar sesgar el conjunto de datos.

En cuanto a la interpretación de los resultados, la baja cantidad de características extraídas por el modelo mínimo permite analizarlos mediante técnicas de ML. Los modelos de CNN están extrayendo detalles muy específicos de las imágenes como características, que no son intuitivas para los humanos pero que permiten crear un espacio en el que las muestras son claramente separables.

SENSITIVITY ANALYSIS OF THE CNN LEARNING PROCESS USING SYNTHETIC IMAGES

Author: España Novillo, Irene.
Supervisors: Sánchez Úbeda, Eugenio Francisco y Boal Martín-Larrauri, Jaime.
Collaborating Entity: ICAI – Universidad Pontificia Comillas.

ABSTRACT

Increasing CNN interpretability is one of the fundamental challenges in the Deep Learning field nowadays and generally, it has been addressed from very complex approaches. A new and simpler methodology to understand the process a CNN follows to learn to classify images with such high levels of accuracy is presented in this thesis. Several models comprised of one convolutional layer with a single filter have been trained with datasets generated synthetically so they focus on solving a specific problem. They have been proved to obtain good results as well as ease feature and filter interpretation, even enable to apply ML techniques to perform the analyses, due to the lower amount of parameters to study.

Keywords: Deep Learning, CNN, interpretable AI, minimum model, learning process.

1. Introduction

The high levels of precision achieved by CNN and the elimination of manual intervention have increased their use in any activity that involves image recognition. However, these models have a high degree of complexity, which makes their operation difficult to understand. Until now, the approaches to the problem of CNN interpretability have been carried out by analyzing complex architectures trained with real image datasets. The methodology presented in this thesis makes use of the opposite approach, studying the problem from a simpler perspective.

2. Project definition

The description and implementation of a methodology to analyse the CNN learning process which increases their interpretability, has been carried out. Starting from a dataset of synthetically generated simple images, a series of models with low complexity architectures are trained and the results obtained are analyzed exhaustively. The problems addressed are the scaling and rotation of geometric figures with and without noise.

3. Methodology description

The workflow followed in the process is shown in Figure 1 and it is comprised of four main phases:

- **Dataset generation**. Both training and validation datasets contain images of geometric figures belonging to three different classes: ellipses, rectangles and triangles. Depending on the problem, a transformation is applied to the images:

scaling (size variation), rotation (about the axis perpendicular to the plane) and noise addition.

- **Minimum model search**. Once the dataset has been built, a series of architectures are trained varying the values of the hyperparamenters following a grid-search method. The model that achieves the highest levels of accuracy with the minimum number of parameters and hence, the lower complexity, is selected.
- **Interpretation of results**. It includes the analysis of the features obtained by the minimum model and how they allow it to perform the classification, as well as the analysis of the physical meaning of the model; identifying which parts of the image are important and how the filter is able to extract them. For the case of experiments with noise, an analysis of the samples misclassified by the model has also been performed, but this study is optional.
- **Conclusions**. The interpretation of results outputs a series of lessons learned about the operation of the CNN learning process of the models studied.

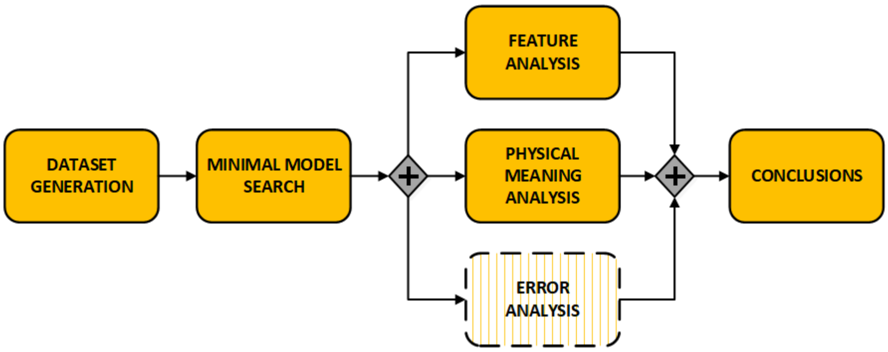

Figure 1. Workflow chart of the methodology.

4. Results

- Regarding scaling and rotation problem without noise, models with just one feature are able to perform the classification correctly, achieving accuracies of around 100% in the first case, and around 60% for the second one. However, bias is inevitably introduced into this datasets, what makes necessary to analyse the same problems adding noise to the images. The filter is focused on extracting borders of the image.
- Regarding scaling and rotation problem with noise, more complex models are obtained. For the former, the architecture has four features and sixteen for the latter, classifying in both cases more than the 90% of the images. The filter is also extracting borders in this case and it has an opposite behaviour depending on the combination of background and figure colors (white or black)

An example of the analysis of the physical meaning of the model that shows the resulting activation maps after applying the filter and the locations of the extracted features in different images is shown in Figure 2.

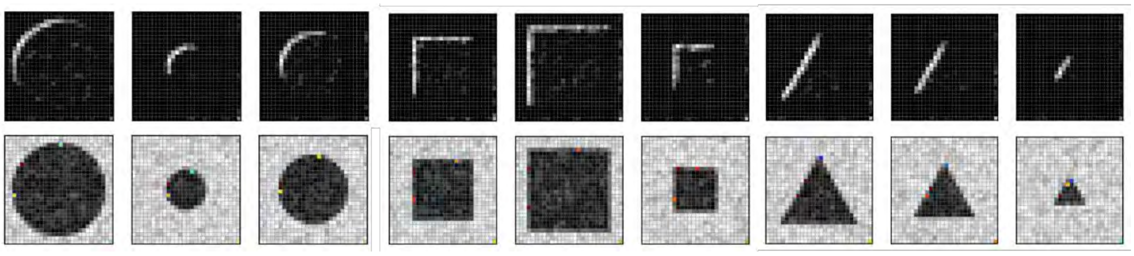

Figure 2. Example of activation maps and maximum activation pixel representation for images belonging to the scaling plus noise problem.

5. Conclusions

This paper has demonstrated that it is possible to train very simple CNN models able to perform the classification correctly, contrary to widespread approaches which focus on training very complex architectures. Instead of using a top-bottom methodology in which a complex model is built and after that the number of parameters is reduced trying not to lose accuracy, the architectures are designed to be simple from the beginning and to solve just one problem, which is possible to thanks to the synthetic datasets. Special attention should be paid to the initialization seed and the number of epochs when looking for the minimum model and noise has to be added to datasets that are synthetically generated so that bias is avoided.

Regarding interpretation of results, the low amount of features extracted by the minimum model enables to analyse them using ML techniques. CNN models are extracting very specific details of the images as features, which are not the intuitive for humans but enable to create a space in which samples are clearly separable.

A mi abuelo Plácido, me sigues enseñando a luchar cada día.

A mi padre, a mi madre y a mi hermano.
Por el apoyo, la comprensión y el cariño.

A mi amigo Julián,
desde que no estás me ando buscando
y no me encuentro ni en el espejo.

Talent is actually something very common.
The rare thing is perseverance.
Doris Lessing (1919–2013)

Agradecimientos

Indudablemente, la realización de este trabajo no habría sido posible sin el valioso apoyo de mis dos directores, Eugenio y Jaime, que han empujado este proyecto desde el día –1. Ha sido un placer veros sacar vuestra vena más investigadora y poder apreciar la pasión y el empeño que ponéis en vuestro trabajo. Gracias, espero que hayáis disfrutado tanto como yo estos últimos meses. Asimismo, no puedo olvidarme del resto de profesores del máster, que me han enseñado a ser la mejor versión de mi misma, tanto profesional como personalmente. Gracias de todo corazón por la confianza que siempre habéis depositado en mí.

Durante estos dos años, un pilar fundamental ha sido mi familia. Gracias a mis padres y mi hermano, por animarme a continuar mis estudios y permitirme hacerlo en ICAI. No puedo estar más orgullosa de tener a mi lado a personas que son igualmente tenaces y generosas. Al resto de mi familia: abuelas, por darme siempre los mejores consejos; tíos, tías y primos, por mantenernos siempre unidos y haber confiado en mi capacidad para salir adelante.

No puedo olvidarme todos mis compañeros de clase, hemos compartido nervios, frustración y risas, pero sobre todo, hemos crecido juntos. En especial quiero darle las gracias a Marta, mi pareja en esta aventura que comenzó en una clase de laboratorio de electrónica en septiembre de 2018 y que a día de hoy sigue caminando a mi lado. Gracias a Carlos, por escuchar mis rayadas y saber siempre sacarme una sonrisa, además de ser mi "zoundfriend" haciendo que el último verano pasase volando.

Gracias también a mis compañeras de piso: Elena y Sisa, sobre todo por vuestra ayuda en este último mes. A Esther, por brindarme tu ayuda en lo que a temas de Biología se refiere así como incitarme a comer sushi más a menudo de lo que me atrevo a admitir. Gracias a Teresa, por los apuntes de contrabando y tu interés por mi trabajo; y a Dani, por los consejos literarios. Gracias a Juan, por los viernes de Catán que siempre me han ayudado a desconectar. A mis compañeros del departamento de GIRS, por ser todos ejemplos a seguir con su trabajo y dedicación, valores que he trasladado a este proyecto. Cuando pienso en Madrid, pienso en todos vosotros.

Un inmenso gracias a mis amigas de toda la vida, por confiar en mí más que yo misma. No olvidéis que os quiero. Finalmente, gracias a quien siempre está pero nunca se le ve, por estar a mi lado durante los buenos momentos y los que no lo han sido tanto.

Al resto de personas que no he mencionado pero que merecen estar aquí: ¡GRACIAS!

Acknowledgment

With no doubt, carrying out this thesis would not have been possible without the valuable encouragement of my two directors, Eugenio and Jaime, who have supported this project since day –1. It has been a pleasure to see you bring out your most investigative streak and to be able to appreciate the passion and commitment that you put into your work. Thank you, I hope you have enjoyed as much as I have these last months. Likewise, I cannot forget the rest of the teachers on the master's degree, who have taught me to be the best version of myself, both professionally and personally. Thank you from the bottom of my heart for the trust you have always put in me.

During these two years, a fundamental pillar has been my family. Thanks to my parents and my brother, for encouraging me to continue my studies and allowing me to do it at ICAI. I could not be more proud to have by my side people who are equally tenacious and generous. To the rest of my family: grandmothers, for always giving me the best advice; uncles, aunts and cousins, for always sticking together and trusting in my ability to get ahead.

I cannot forget all my classmates, we have shared nerves, frustration and laughter, but above all, we have grown together. I especially want to thank Marta, my partner in this adventure that began in an electronics laboratory class in September 2018 and who continues walking by my side to this day. Thanks to Carlos, for listening to my problems and always knowing how to make me smile, as well as being my "zoundfriend" making the last summer fly by.

Thanks also to my roommates: Elena and Sisa, especially for your help in this last month. To Esther, for giving me your help in matters of biology as well as encouraging me to eat sushi more often than I dare to admit. Thanks to Teresa, for the smuggling notes and your interest in my work; and Dani for literary advice. Thanks to Juan, for Fridays with Catan that have always helped me to disconnect. To my colleagues from the GIRS department, for being all examples to follow with their work and dedication, values that I have put in the development of this project. When I think of Madrid, I think of all of you.

A big thank you to my lifelong friends, for trusting me more than myself. Do not forget that I love you. Finally, thanks to the one who is always there but is never seen, for being by my side during the good times and those that have not been so good.

To the rest of the people I have not mentioned but who deserve to be here: THANK YOU!

Contents

List of Figures

List of Tables

1 Introduction

Do not wait until the conditions
are perfect to begin.
Beginning makes the conditions perfect.
Alan Cohen (1950–)

The first chapter presents the rationale behind this project as well as its main objective and the tools used in its development. In addition, it provides the reader with an overview of the dissertation organization in order to ease the reading.

The exponential growth in data generation and collection during the last decade, characterised by containing a wide variety of both structured and unstructured information, along with the increasing processing technology speed, has resulted in numerous applications in the field of Big Data. In particular, variety takes on special importance since data is intaken from different sources such as IoT sensors or web navigation, which gives rise to a diversity of formats and types. Finding out and understanding completely which is the best way to analyse each kind of data can be crucial to complete any project successfully.

Images are the representative set of unstructured data par excellence, considering that its analysis can be also applied to audio and video. The massive use of social networks as well as the spread of cameras in multiple fields like industry, medicine or agriculture (given its low price in relation to the amount of information that can obtain) are only two of the many factors that have made images such a powerful resource. Most of applications involve a pipeline of extraction, preprocessing and classification of images in order to obtain a result. This dissertation focuses on the classification task, as it is one of the most challenging nowadays.

Image classification is the process of taking an image as input for an algorithm and outputting a class or probability that the input belongs to a particular class. Computer vision is a subfield of Artificial Intelligence that aims to make a processor able to identify images from the real world and act accordingly, in the same way a human would do. Traditional computer vision involves extracting features from pictures manually, selecting the more relevant ones for the problem being solved and feeding them to the classification algorithm. This process is characteristic of the Machine Learning field as it requires human intervention performing the manual feature extraction, which makes the process harder.

With the emergence of Deep Learning this problem is solved since it is the algorithm itself the one in charge of both selecting and extracting the proper features, as well as classifying the images, performing in this way an end-to-end learning without the need of human intervention. The family of methods used in Deep Learning are based on Artificial Neural Networks (ANN), computing systems inspired by the biological neural networks that compose animal brains. One of the most used kinds of ANN in Deep Learning is the Convolutional Neural Network (CNN), which represents a huge breakthrough in computer vision.

CNNs are especially suited for image and speech recognition, since the spectrum of sound frequencies can be represented visually as spectrograms. They are also used for video applications, since it is, in fact, a sequence of images (frames) captured and displayed at a given frequency. Due to the high levels of accuracy achieved, CNNs are widely used in diverse applications such as self-driving cars, facial recognition or document analysis.

Nevertheless, as happens with all Machine Learning models, there is a trade-off between interpretability and precision of the results obtained. Thus, while simple models are very comprehensive but not too accurate; very complex and accurate models are seen as "black boxes", due to its low interpretability. Particularly, CNNs are deeply difficult to interpret which represents a drawback for a method with such a good performance. In order to better understand the process the convolutional neural network is following to learn, in this dissertation a deep analysis of the learning process of a CNN is carried out, based on synthetic images.

1.1. Motivation

For the last few years there has been a rapid rise in the use of convolutional neural networks, consolidating their role as the main algorithm for image detection, recognition, segmentation or classification and also, outperforming the results obtained by traditional computer vision techniques and, in some cases, even improving the human vision abilities. Moreover, they are likely to become an important component in most automation processes within the next decade since cameras are already sensors commonly used in the Industry 4.0 due to its great relationship information-cost.

The low interpretability that characterises CNNs not only prevents from choosing the optimal architecture for each application but it also makes harder to improve the existing ones. In addition, previous work has addressed the problem of CNN interpretability focusing mainly on the training of complex architectures with large datasets and, after that, trying to visualize the feature maps to understand how the network is learning. Nevertheless, simpler approaches consisting on training networks with controlled synthetic datasets, building them layer by layer and, at the same time, interpreting both the features and results obtained after each layer, seem to be a valid solution.

1.2. Objective

This thesis presents an alternate solution to analyse the way in which a CNN learns, based on the use of an image dataset of geometric shapes generated synthetically, to which different modifications have been applied (e.g., rotation or scaling). This controlled environment eases the interpretation of network parameters as it enables to visualize both the feature maps and the filters clearer, without additional noise derived from real images.

The aim is, therefore, to understand the learning process convolutional networks are following to achieve such accurate results by analysing the impact of different geometric transformations over a synthetic dataset.

In this way, for each controlled image training set, the optimal architecture is found, containing the minimum number of layers to perform the classification with a high level of accuracy. Understanding how the CNN works in this group of controlled environments and finding out which are the best architectures for basic problems enables to establish the baseline for developing most complex architectures in addition to improving already existing ones, applied to different use cases.

1.3. Resources

Research is mainly carried out in Python 3 [1] as it is a programming language widely used for Machine Learning applications due to its compatibility with numerous libraries and its easy usage. Moreover, Python enables both building prototypes quickly and convenient application deployment. For Deep Learning purposes the Keras API is used over the TensorFlow 2 back end [2], working with the `tf.keras` module. R programming language is also used to perform certain analyses over the results given by the models.

The Google Colab platform is used to implement the different experiments since it provides GPUs at no cost, which significantly accelerate the CNN training process. For algorithms implemented in R, the chosen IDE is RStudio. On the other hand, in order to analyse deeply the results given by the experiments, apart from using the most common Python and R visualization libraries, other tools such as Power BI, for creating dashboards with the most representative graphs, and GIMP, for studying the network behaviour over the input images at pixel level are used.

1.4. Dissertation outline

The dissertation consists of 5 chapters, being this one the first of all containing the introduction to the problem concerning the project. The next chapter contains a detailed description of the internal structure of CNNs in addition to a list of the most known architectures. The third one provides an extensive review that covers most of the approaches to convolutional networks interpretation and visualization that can be found in the literature. The fourth chapter contains all the information related to the new methodology that is presented in the thesis as well as the results obtained from series of experiments that have been carried out. The last chapter includes a summary with the most important points that have been developed in the document in addition to several future improvements over the approach that has been presented. Finally, the thesis contains an appendix in which the different contributions of this project to the Sustainable Development Goals (SDG) are stated.

2
Convolutional Neural Networks

Nothing in life is to be feared.
It is only to be understood.
Marie Curie (1867–1934)

This chapter provides an overview of Convolutional Neural Networks starting from exploring its origins to defining its operation and analysing the internal structure. After that, several of the most popular network architectures are presented as well as the concept of *transfer learning*, setting the basis for the understanding of the remaining chapters.

Artificial Intelligence (AI) aims to develop computer systems capable of imitating human behaviour or tasks (and in many cases, outperforming them), such as speech recognition, decision-making or visual perception. Enabling machines to learn can make them “smart” enough to perform certain functions considered as repetitive, dangerous or even impossible (due to its high computational complexity) for human beings. Within the field of AI, the discipline known as Machine Learning (ML) comprises all computer algorithms that improve automatically through experience without being explicitly programmed [3].

There are several categories into which ML algorithms are divided, standing out the *Brain-Inspired* ones, which tries to replicate the brain functioning, such as Spiking Neural Networks (SNN) and the traditional Artificial Neural Networks (ANN, or just NN). The former includes the concept of *time* into the learning process, using discrete events that take place at specific points in time (spikes), whereas the latter focuses on continuous values [4]. However, both of them consist of a series of neurons connected between them and disposed in several layers.

Finally, within the NN algorithms the category concerning this dissertation is found: Deep Learning (DL). DL puts a special focus on large neural networks which are trained with datasets containing huge amounts of examples. DL learns to represent the world as a nested hierarchy of concepts, defining each concept in relation to simpler ones, and more abstract representations computed in terms of less abstract ones [5]. A diagram that helps to understand the taxonomy of AI is shown in Figure 2.1.

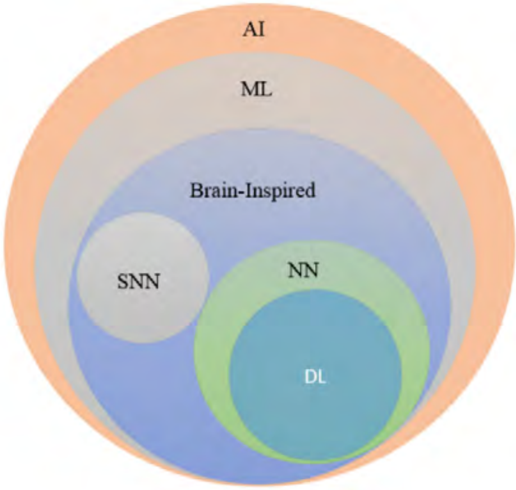

Figure 2.1. Taxonomy of AI [6].

It is important to highlight the main difference between ML and DL techniques, apart from the fact that the latter uses larger and more complex architectures. In traditional ML, the procedure begins with the extraction of new variables from a raw dataset by hand. The data scientist tries to extract the most explicative and representative variables, using a traditional technique (such as SIFT for images [7]). Once the variables have been found, an algorithm of ML is applied in order to solve the specific task, for example, a classification problem. DL algorithms, on the other hand, extract variables automatically from the dataset, without the need for human intervention. It is the algorithm itself the one in charge of both learning which features are important for the specific tasks and solving the problem. Figure 2.2 shows that this difference in procedures can be better appreciated applied to a classification task.

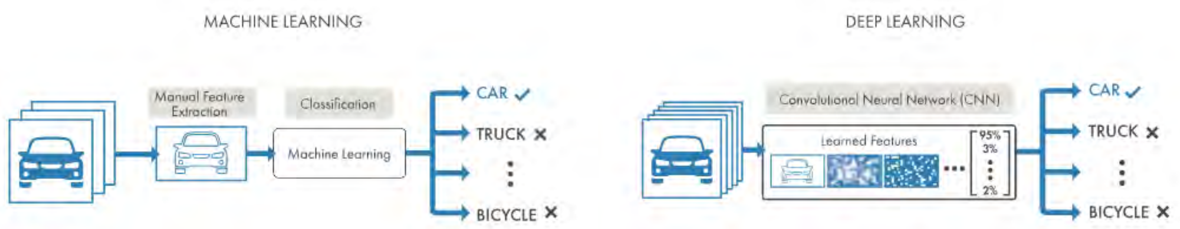

Figure 2.2. Comparison between a Machine Learning approach (left) and a Deep Learning approach (right) for a vehicle image classification task [8].

One of the most widespread algorithms nowadays within the field of DL are Convolutional Neural Networks (CNN), which is specially well-suited for image and speech recognition tasks. CNN hierarchical structure was first proposed in 1988 by Fukushima, who called it *Neocognitron* and applied it to a visual pattern-recognition system trained to recognize handwritten numerals [9]. Nevertheless, this advance could not have been possible without the discovery made by Hubel and Wiesel in 1959 [10]. Through a series of experiments carried out to study the cat's visual cortex, they found out that some cells were only stimulated by borders presented in a specific position and orientation. In order to do so, they connected a series of electrodes to different levels of the cat's visual pathway from retina to striate cortex. This method enabled them to observe the responses at several distinct levels, concluding also that patterns most effective activating cells at one level, may no longer be the most effective at the next. Figure 2.3 shows four example cases of the experiment.

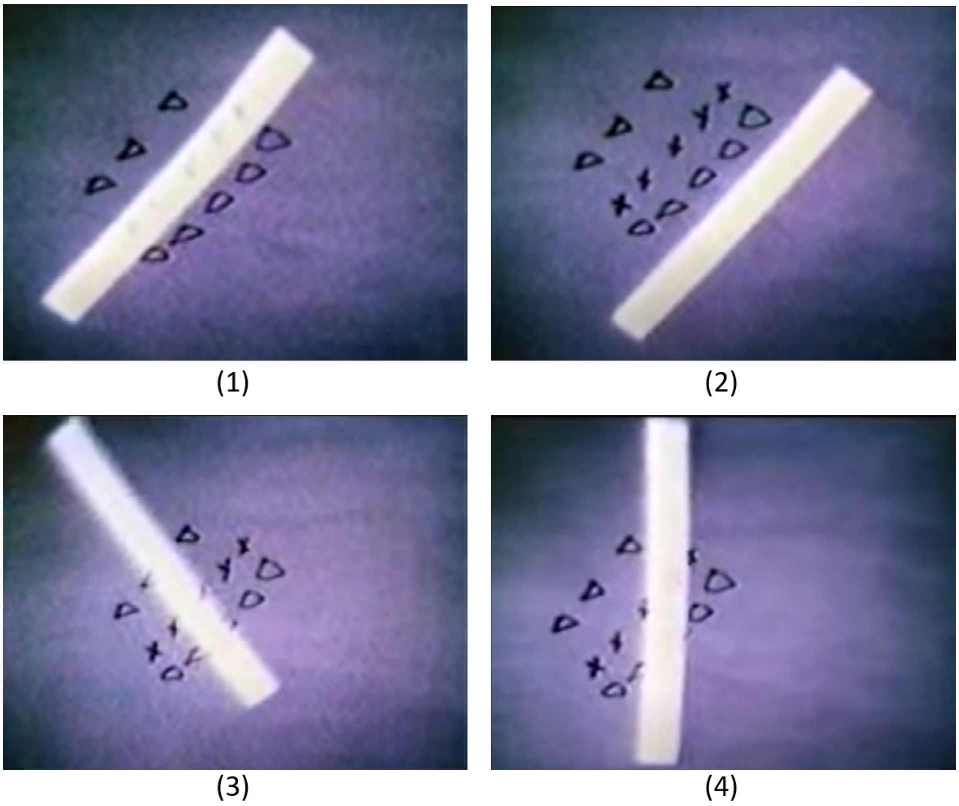

Figure 2.3. Cat's visual cortex experiment by Hubel and Wiesel. Black marks indicate the position and orientation of the light stimuli in response to which the cells are activated. (1) The cells are activated as the light coincides in position and orientation with the pattern. (2), (3) and (4) The cells are not activated since there is no match between the light and the marks in both position and orientation. Modified from [11].

Two significant outcomes were obtained from this research: firstly, there are neurons specialised in extracting certain geometric features or patterns in particular points of images and secondly, these neurons are organized in different layers, meaning that each layer is, at the same time, focused on a given pattern and therefore contains the neurons in charge of obtaining that kind of feature. This is the basis for the functioning of CNN: a series of *pattern-specialised neurons* (called filters) organised in a hierarchical structure.

2.1. Definition and structure

Artificial neural networks can be defined as computational learning systems composed by a series of nodes (neurons) organised hierarchically that translate a data input of one form into a desired output, usually in another form [12]. The NN general structure consists of an input layer, several hidden layers and a final output layer, as it is shown in Figure 2.4.

A neuron, at the same time, has several inputs which are multiplied each one by it weight and passed through an activation function obtaining the output. This process is similar to the one followed by biological neurons, in which synaptic impulses are carried toward the cell body and, after performing a series of chemical reactions, the output impulses are carried away from the cell body through the axon. Figure 2.5 shows the similarity between a biological and an artificial neuron.

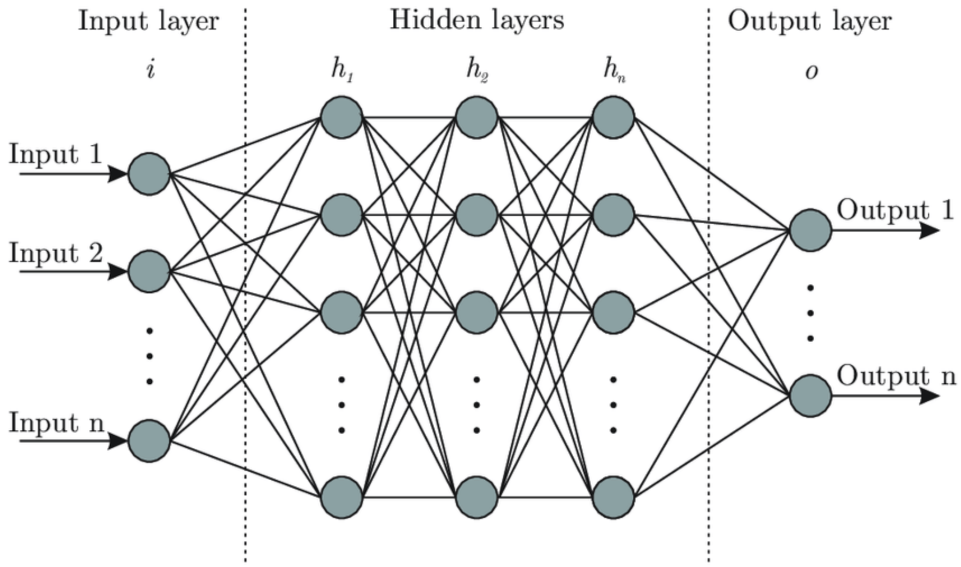

Figure 2.4. Structure of an Artificial Neural Network (ANN). Neurons are plotted with a gray circle [13].

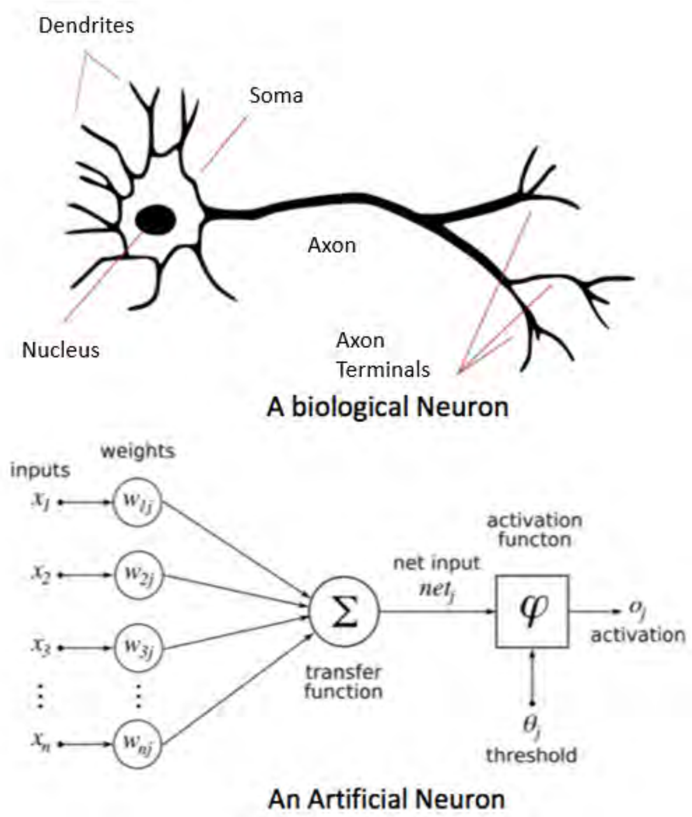

Figure 2.5. Comparison between a biological neuron and an artificial neuron. Modified from [14].

CNN, just like NN, are composed of an input layer, an output layer and a series of hidden layers. The first difference is found in the number of hidden layers which is far superior in the case of CNN. In addition, the layers in a CNN are divided into feature detection and classification layers and the neurons are disposed in a multidimensional way, contrary to traditional neural networks. These differences can be better observed in Figure 2.6, which shows the internal structure of a CNN.

Regarding the learning process of a CNN, firstly, all raw pixel values of an image are taken as inputs. Then, a series of multidimensional matrices are applied to operate those inputs (filters) and extract the image features. Once the features are available, the classification is performed based on the fact that images from different classes originate different features.

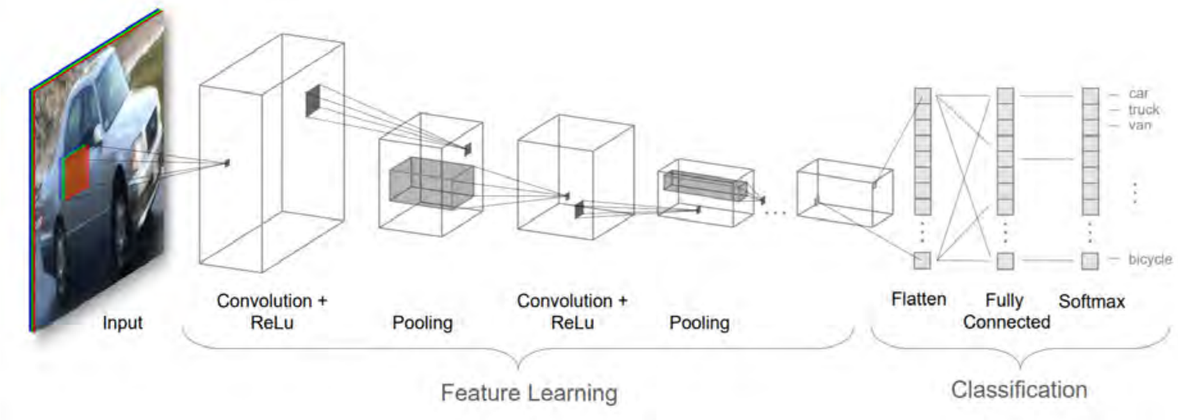

Figure 2.6. Structure of an Convolutional Neural Network (CNN). Feature learning neurons are disposed in different 3D blocks whereas classification neurons are disposed just like in ANN. Bear in mind that this figure is showing the output volume of each layer, not the neurons disposition of each layer itself [15].

2.1.1. General concepts

Once the reader has an overall idea about the structure of a CNN and its differences with a traditional ANN, it is important to describe more specific concepts of this kind of network that will help to understand its operation.

- **Tensors**: a tensor is a container which can house data in N dimensions, along with its linear operations [16] or in other words, a multi-dimensional array with a uniform type [17]. Figure 2.7 shows this generalization of scalars, vectors and matrices. In CNN, neurons in feature learning layers are arranged in a 3-dimensional manner (a tensor with 3 dimensions), using width, height and depth as shown in Figure 2.8. Each layer accepts an input 3D volume and transforms it to an output 3D volume through a differentiable function [18]. This specific organization of the CNN layers has a physical sense, as images are themselves a tensor: three matrices of pixels put together with the red, green and blue components as can be seen in the input part of Figure 2.6.

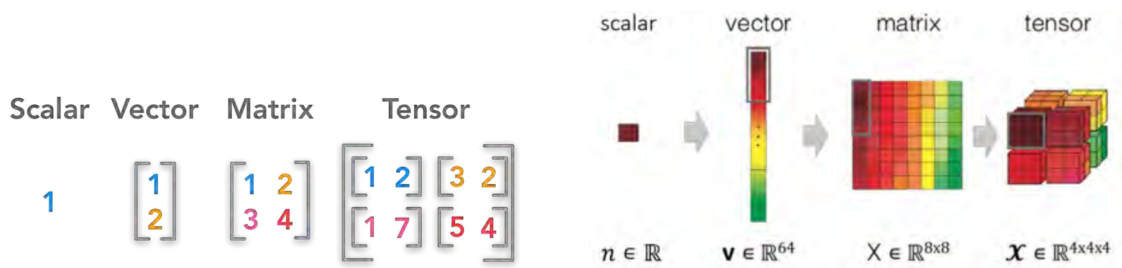

Figure 2.7. Difference between a scalar, a vector, a matrix and a tensor. Modified from [19] and [20].

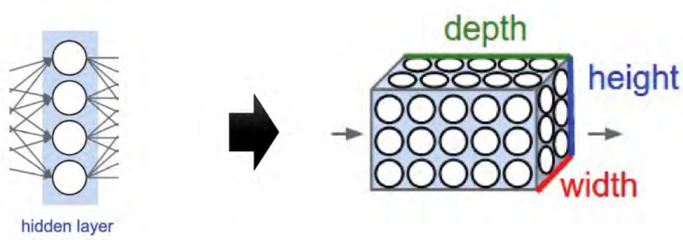

Figure 2.8. Arrangement transformation of neurons from an ANN hidden layer (left) to a CNN feature learning layer (right). Modified from [21].

- **Connectivity and Weight sharing**: due to the size and depth of the CNN layers, that is to say, the huge number of neurons, this network is characterised by learning a large amount of parameters. In order to avoid the *curse of dimensionality*, the network must be pruned in some way, being impractical to connect neurons in each volume to all neurons in the next one as it happens in Dense Neural Networks (DNN). A CNN bypasses the curse of dimensionality by [22]:

  – Local connectivity: each neuron is connected only to a subset of neurons in the following layer, unlike a DNN where all the neurons are fully connected (Figure 2.9).

Figure 2.9. Comparision between global connectivity and local connectivity. Modified from [23].

  – Weight sharing: the same weights are used for two layers in the model meaning that the same parameters will be used to represent two different transformations in the system [24] (Figure 2.10).

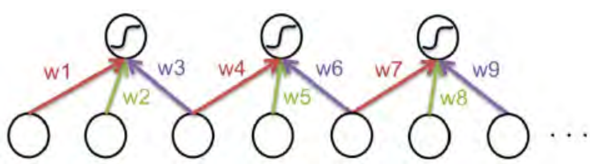

Figure 2.10. Weight sharing. Edges that have the same color have the same weight [23].

  This two methods help reduce the number of parameters in the whole architecture making the computation more effcient.

- **Connectivity in terms of tensors**: a CNN connects each neuron to only a local region of the input volume. The connections are local in space (along width and height), but always full along the entire depth of the input volume [25]. An example is shown in Figure 2.11 where the connectivity between the input layer (which only function is to read the image so no parameters are learnt in this layer) and the first convolution layer can be observed.

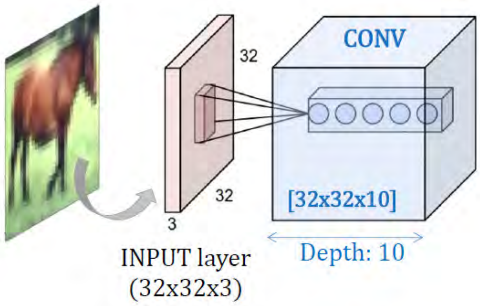

Figure 2.11. Example of connectivity in terms of tensors between the input layer and the first convolutional layer. Modified from [21].

- **Batches**: the batch size refers to the number of training examples used in one iteration to update the internal parameters of the model. Therefore, batches are packs of images (subsets of the training data) that in each iteration are utilized to change the network weights. The network does not employ the whole training dataset for weight update due to its large size, which would involve a high computational and memory cost. With the images in the batch, the gradient descent method is calculated in order to find the optimal coefficients which result in the minimum error. There are three main alternatives for estimating the gradient [23]:
  - <u>Batch Gradient Descent or Batch mode</u>: batch size is set equal to the total number of examples in the training set. Figure 2.12 shows how this option manages to draw a perfect trajectory to the minimum. Nevertheless, the computation time in this case is too high, which slows the calculation.
  - <u>Stochastic Gradient Descent of Stochastic mode</u>: batch size is set to one so the parameters are updated after each image. Since all the gradient and the neural network parameters are updated after each sample, the trajectory to the minimum is completely random as can be seen in Figure 2.12.
  - <u>Mini-Batch Gradient Descent of Mini-Batch mode</u>: the batch size is greater than one but less than the total number of samples in the training dataset. With this option a balance is struck between the computational time spent in the calculations and the randomness of the trajectory towards the minimum.

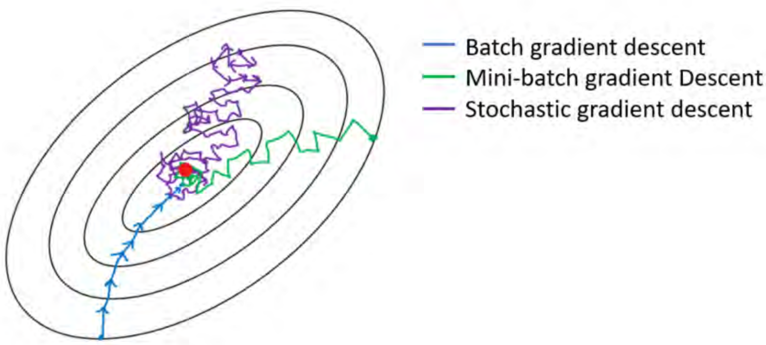

Figure 2.12. Kinds of gradient descent trajectory towards the minimum depending on the batch size [26].

Due to the trade-off between how quickly a CNN learns and the stability of the learning process, the batch size most commonly used is the Mini-Batch, typically it is set to a power of 2 since GPU hardware achieves better run times (number of physical processors is often a power of 2, so using a number of virtual processors different from a power of 2 leads to poor performance as they cannot be distributed equally to parallelize the work). Most common values are 32 and 64.

With smaller batch sizes a larger variance is reached, which is translated into a regularizing effect: weights are updated in a more reasonable way and the neural network will acquire a better generalization ability (convergence towards the minimum). In addition, smaller batches sizes eases parallelization enabling to fit one batch of training data in GPU memory [27].

- **Epochs**: the number of epochs is an hyperparameter that defines the total number of times that the complete training dataset will be passed through the neural network. An epoch is comprised of one or more batches, depending on the batch mode used. Normally, the number of epochs is large, around hundreds or thousands, and typically during training only the maximum number of epochs is set so the network continues until the error has been sufficiently minimized.

  To summarise, the batch size is a number of samples (images) processed before the model is updated whereas the number of epochs is the number of complete passes through the training dataset [28]. A pseudocode that combines concepts of batches and epochs is shown in down below and together with an example can be useful to fully understand these two important hyperparameters.

```
for i in range(num_epochs):
        np.random.shuffle(data)
        for batck in batches(data, batch_size=32):
                grad = compute_gradient(batck, params)
                params = params - learning_rate * grad
```

  In [28] a good and concrete example is explained:

  Assume you have a dataset with 200 samples (rows of data) and you choose a batch size of 5 and 1,000 epochs. This means that the dataset will be divided into 40 batches, each with five samples. The model weights will be updated after each batch of five samples. This also means that one epoch will involve 40 batches or 40 updates to the model. With 1,000 epochs, the model will be exposed to or pass through the whole dataset 1,000 times. That is a total of 40,000 batches during the entire training process.

  Note that with the Batch mode the iteration and epoch values are equivalent.

- **Dropout**: is a technique where randomly selected neurons (both in hidden and visible layers) are ignored during the training of the network. This means that neither the contribution of this neurons is considered during the forward pass nor their weights are updated on the backward pass. This means that their contribution to the activation of downstream neurons is temporally removed on the forward pass and any weight updates are not applied to the neuron on the backward pass [29]. A probability *p* is selected so at each training stage, an individual node would be either dropped of the net with probability *1-p* or kept with probability *p* [30]. Figure 2.13 shows the difference between a neural network before and after applying dropout.

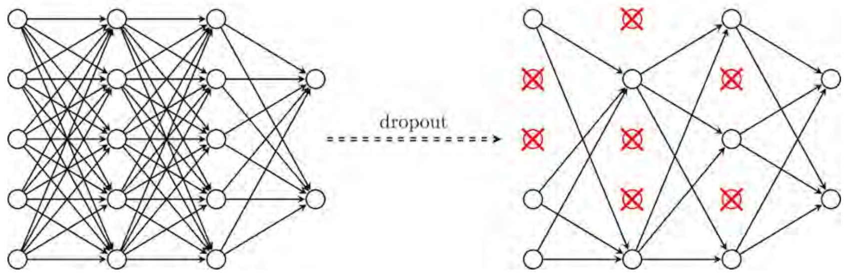

Figure 2.13. Standard Neural Network (left) and Neural Network after aplying dropout (right) [31].

Dropout is a useful technique to prevent overfitting since it avoids co-dependency amongst neurons, which limits the individual learning power of each neuron. A regularization effect is achieved since Dropout forces the neural network to learn more robust features that are useful for the classification task.

2.1.2. Types of layers

CNN architecture is comprised by several types of layers, each one of them has a specific set of characteristics and is in charge of performing a explicit task.

- **Input layer**: this layer contains the raw pixel values of the original image. There are different possible depth values for this layer depending on the number of channels the input image has or, what it is the same, the image colour space. For example, if the image is in grayscale, this layer will have a depth of one; if it is in RGB, the depth will be equal to three since it has three colour channels (Red, Green and Blue) whereas if the image is in CMYK it has four channels (Cyan, Magenta, Yellow and Key or Black) so the input layer will have a volume with depth of four. The height and width of this layer will be also equal to this specific dimensions of the input image. Therefore, it is important that all images in the training dataset have the same dimensions, as they will determine the first layer of the architecture. Figure 2.14 shows an example of both a 3x3 pixels RGB image matrix and a grayscale image matrix, note that the input layer dimensions will be different for each case.

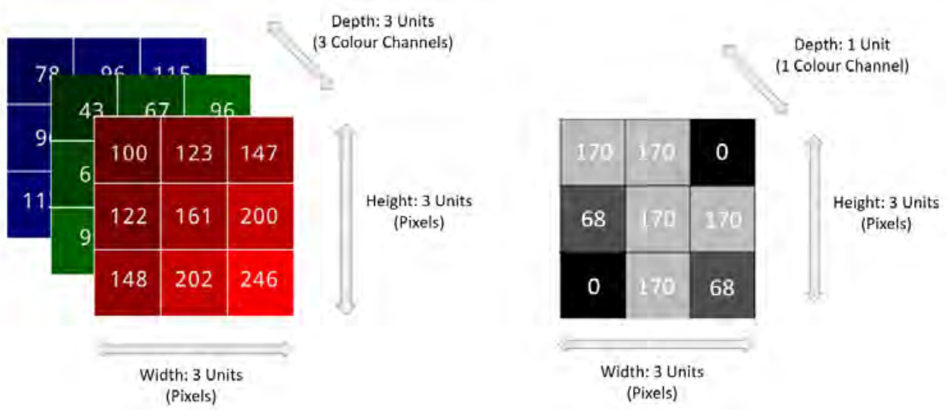

Figure 2.14. Example of 3x3x3 RGB Image and 3x3x1 Grayscale image. Modified from [32] and [33].

In a real case, image resolutions are larger, reaching dimensions of 7680x4320 pixels in 8K. The CNN aims to reduce images into a form that is computationally easy to manipulate and process, without losing important details or features that are critical for getting a good classification [34].

- **Convolutional layer**: it represents the main building block of the CNN since it performs most of the computationally heavy work. Its aim is to extract the features from the input image. In order to do so, this layer performs the convolutional operation, understood as the merging of two sets of information or the mathematical combination of two functions that produces a third function. In this case, the two pieces of information convolved are the input image and a filter or kernel.

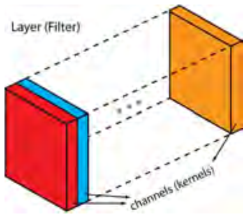

Figure 2.15. Difference between filter and kernel in a convolutional layer of a CNN [35].

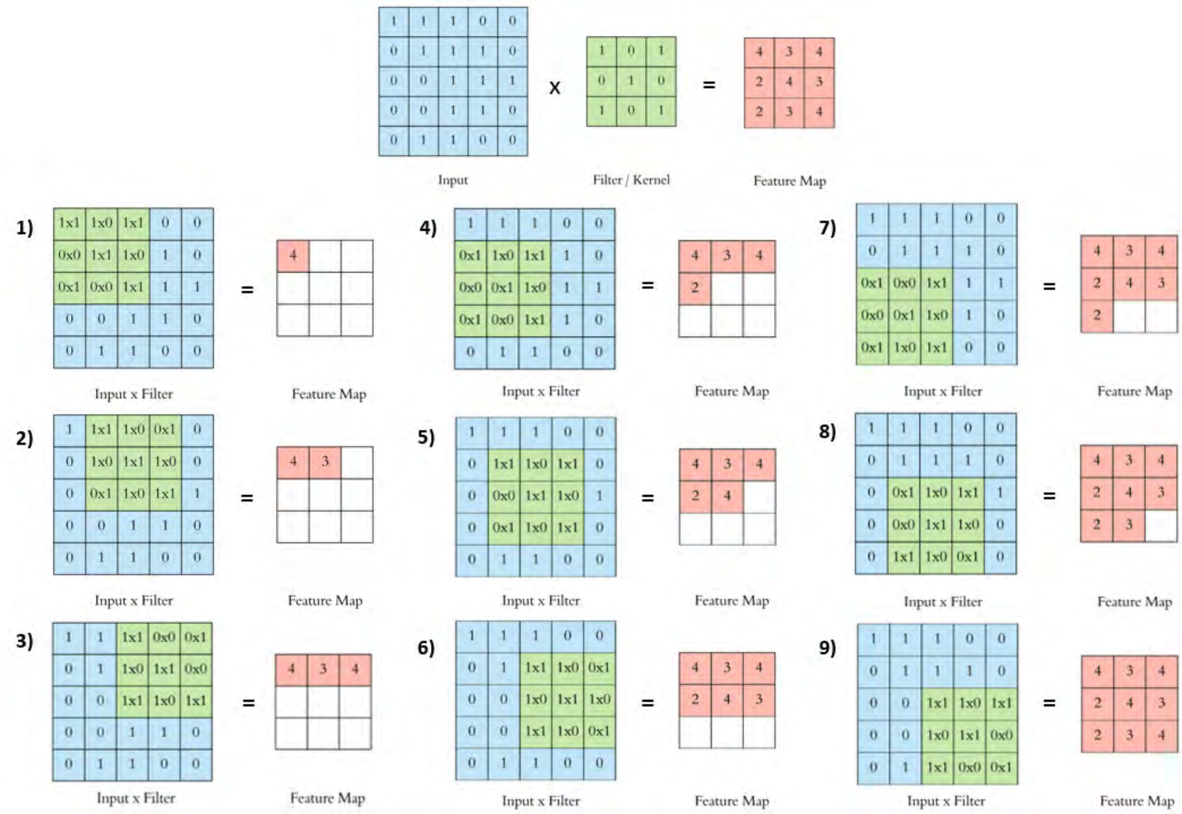

Figure 2.16. Steps followed in the convolution operation of a 5x5x1 image with a 3x3x1 kernel to obtain a 3x3x1 feature map. Modified from [36].

In image processing, filters are numerical matrices used to transform the input images. It is important to highlight the difference between the terms “kernel” and “filter”. A kernel refers to a 2D array of weights whereas a “filter” is a 3D structure of multiple kernels stacked together as Figure 2.15 shows [35].

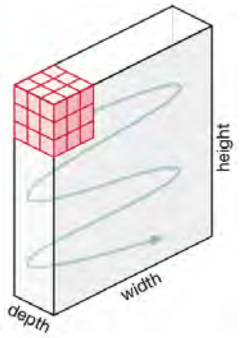

Figure 2.17. Movement of the kernel. The filter moves to the right till it parses the complete width. Moving on, it hops down to the beginning (left) of the image and repeats the process until the entire image is traversed [34].

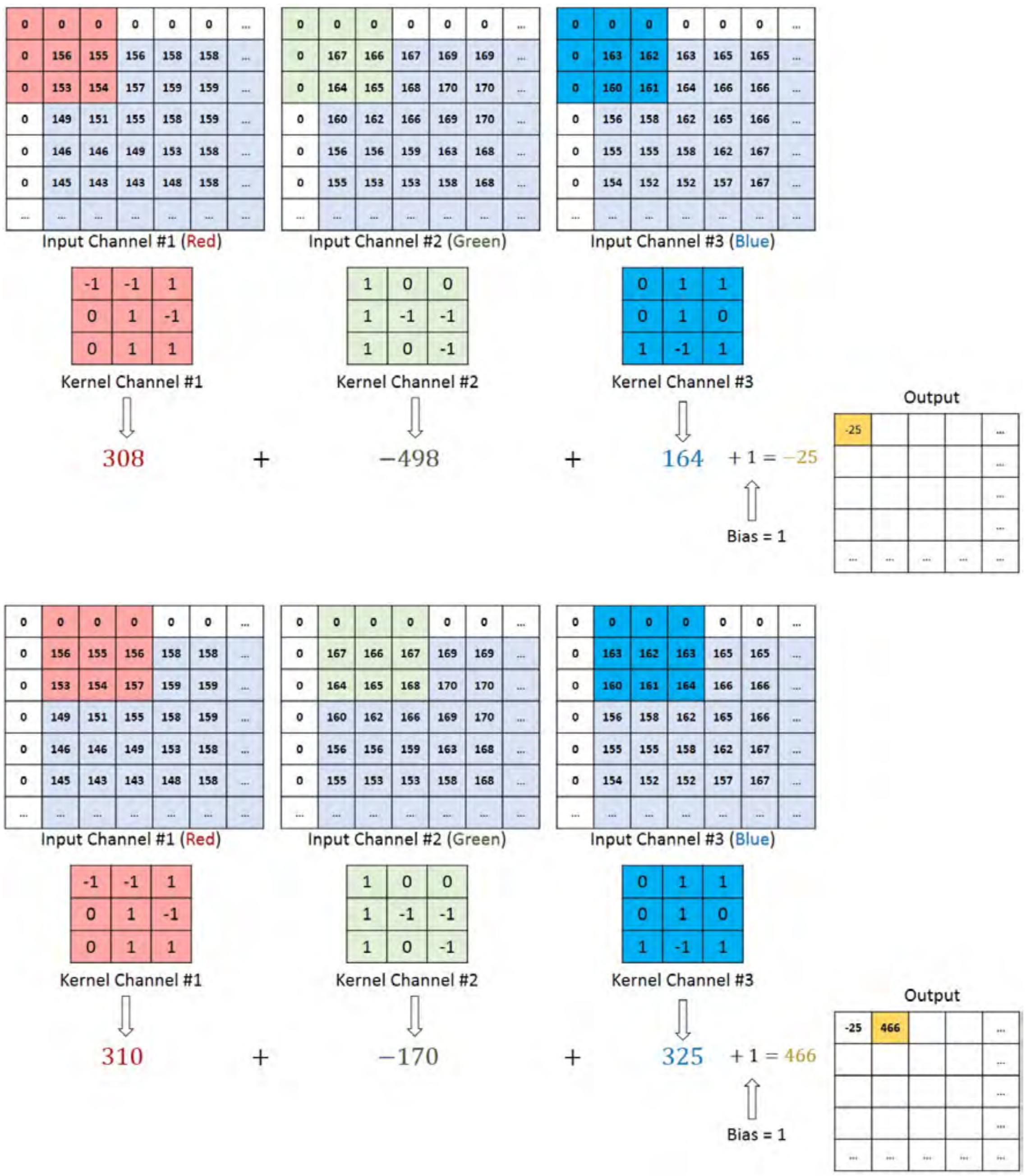

Figure 2.18. First two steps followed in the convolution operation of a MxNx3 image with a 3x3x3 kernel to obtain a 3x3x1 feature map. Modified from [34].

In CNN, filters are also the set of parameters learned by each convolutional layer, used to extract the image features. They are comprised by a small area (width and height) but extend through the full depth of the input volume. Figure 2.16 shows the convolution operation of a filter composed by single kernel over an input volume of one unit of depth (e.g., a grayscale image): the filter slides over the input and a series of matrix multiplications between the kernel and local regions of the input are performed, producing a feature map as output as it can be seen in Figure 2.17.

A similar process is followed when the input volume depth is higher than one and several kernels are applied, as shown in Figure 2.18 . In this case, the input volume has depth equal to three and so does the filter. Each image channel is multiplied by its corresponding kernel which passes through the whole width (as happens in 2.16) and all the results are summed with a bias, giving one depth channel convolved feature output [34].

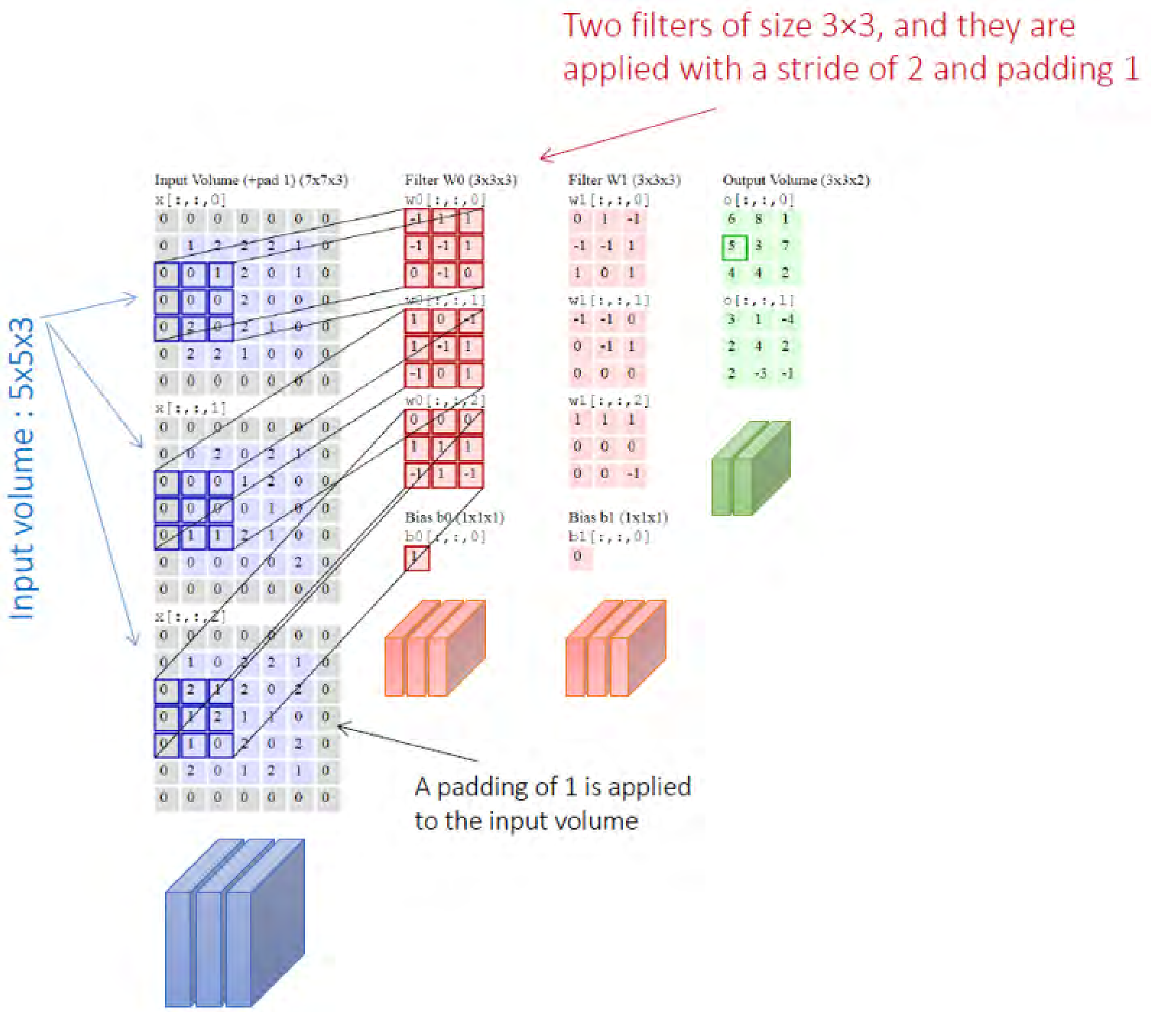

Figure 2.19. Convolution with two filters composed by three channels each giving an output volume with two channels depth. [21].

Note that the same filter channel (kernel) is applied over all the area of an input channel, but different kernels are applied to each input channel. Talking in terms of neurons, it is the same as saying that all neurons in a convolutional layer apply the same kernel, each one of them over a particular local area of the input to which that neuron is connected. In other words, every entry in the 3D output volume can be interpreted as an output of a neuron that looks at only a small region in the input and shares parameters with all neurons to the left and right spatially (since these numbers all result form applying the same filter) [21]. In this way, the depth of output volume will be equal to the number of kernels the filter has applied to the input. The width and depth depend on the dimensions of the filter and another two parameters explained further in this section: stride and padding. One more graphical example of these concepts is shown in Figure 2.19.

The deeper the convolutional layer is in the overall network architecture, the more detailed the features it extracts are. The first convolutional layers extract high-level (simpler) features such as edges or colour whereas the top convolutional layers extract low-level (more complex) features such as particular parts of an object. This fact shows the hierarchical nature of the network, which orders features from general to more class-specific. In [37] a series of experiments are carried out to visualize the features of a fully trained model. As seen in Figure 2.20 the features extracted in Layer 2 correspond to corners and other edge/colours conjunctions while Layer 4 is focused on obtaining object-part features such as dog faces or bird's legs.

This fact reminds of the function of human brain when recognizing objects and the similarity can be best appreciated in Figure 2.21, highlighting what was explained in previous sections: the attempt or inspiration of AI to imitate human behaviour. The work of the human visual system starts in the eyes, which receive the light rays reflected by objects. The rays are captured and processed into light impulses by the retina and sent to a part of the central nervous system. The impulses finally reach the primary visual cortex passing through the optic nerve. During the path followed from the retina to the visual cortex, the input impulses are preprocessed but it is not until they reach the visual cortex that the brain makes sense of what the eye sees [38]. Complex properties as colour, shape, texture and stereoscopic depth are processed by different layers of the visual cortex to finally recognise the object, just like in a CNN each convolutional layer is specialised in extracting an specific kind of feature, going from the simpler to the most complex ones.

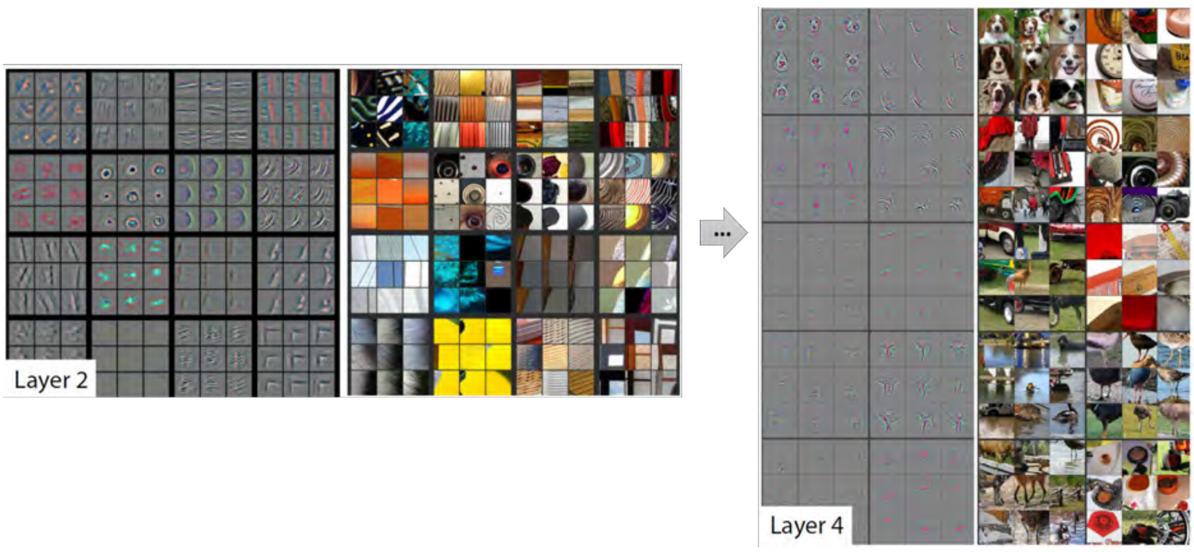

Figure 2.20. Visualization of features in Layers 2 and 4 from a fully trained model. The top 9 activations (or filter representations) in a random subset of feature maps across the validation data, projected down to pixel space, together with the original images are shown. Modified from [37].

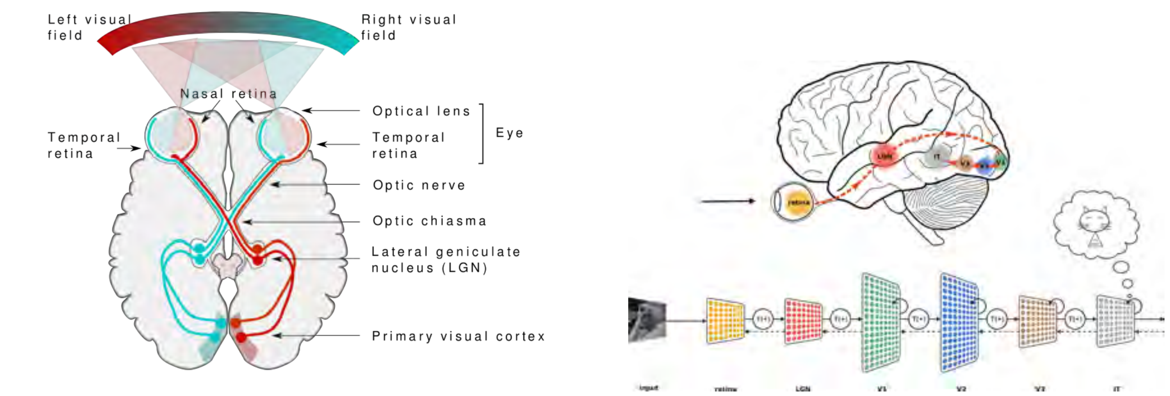

Figure 2.21. The Visual pathway followed inside the human brain (right) [39] and its correspondence with the internal structure of a CNN (left) [40].

There are two important parameters to specify in a convolutional layer that have been mentioned previously and are related with the movement of the filter along each layer of the input: stride and padding.

– Stride: it specifies the size of the step the convolution filter moves each time it slides over the input. When the stride is equal to one, then the filter moves one pixel at a time whereas when it is equal to two, the filter jumps two pixels at a time as it slides over the input. With the stride parameter two factors are controlled: speed and size of the output volume. If the stride is higher than one it means that the movement of the filter will be quicker since it is not applied several times over all the pixels (there is less overlap). This is specially useful when the input images have high resolution, meaning a lot of pixels (bigger dimensions). In addition, bigger strides make the resulting feature map smaller as they skip over potential locations. Figure 2.22 shows two examples of stride over inputs with different dimensions.

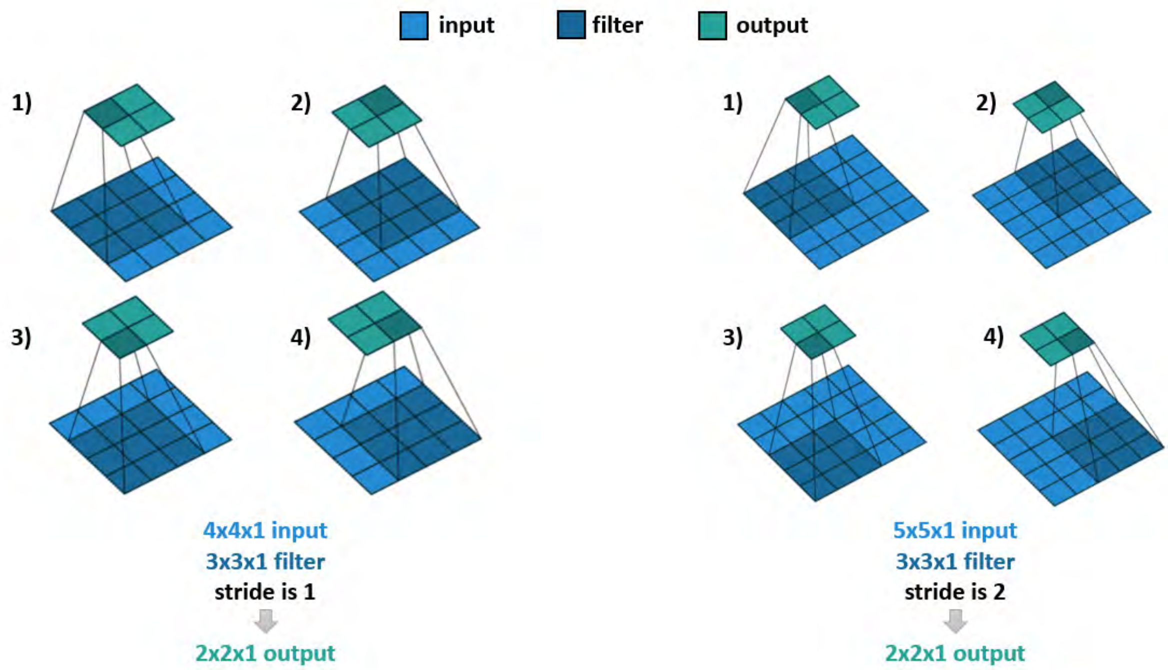

Figure 2.22. Convolution operation over different inputs using distinct strides. Modified from [41].

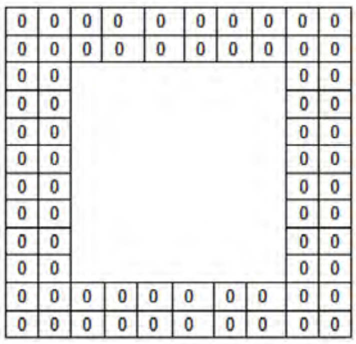

Figure 2.23. Example of input with padding 2, padding 1 is very usual. Modified from [42].

– Padding: as pointed out previously, with stride the dimensions (width and height) of the input are reduced after the convolution. Sometimes it is necessary for the convolution output to have the same width and height as the input and that is achieved with padding. Padding is a hyperparameter that indicates the number of "frames´´ of zeros that need to be added to the input. An example of padding equal to 2 is shown in Figure 2.23. Without padding, the convolution shrinks the output losing information on corners so padding allows an output with the same dimensions than the input, but also with bigger dimensions than the input as shown in Figure 2.24, although this last option is less used.

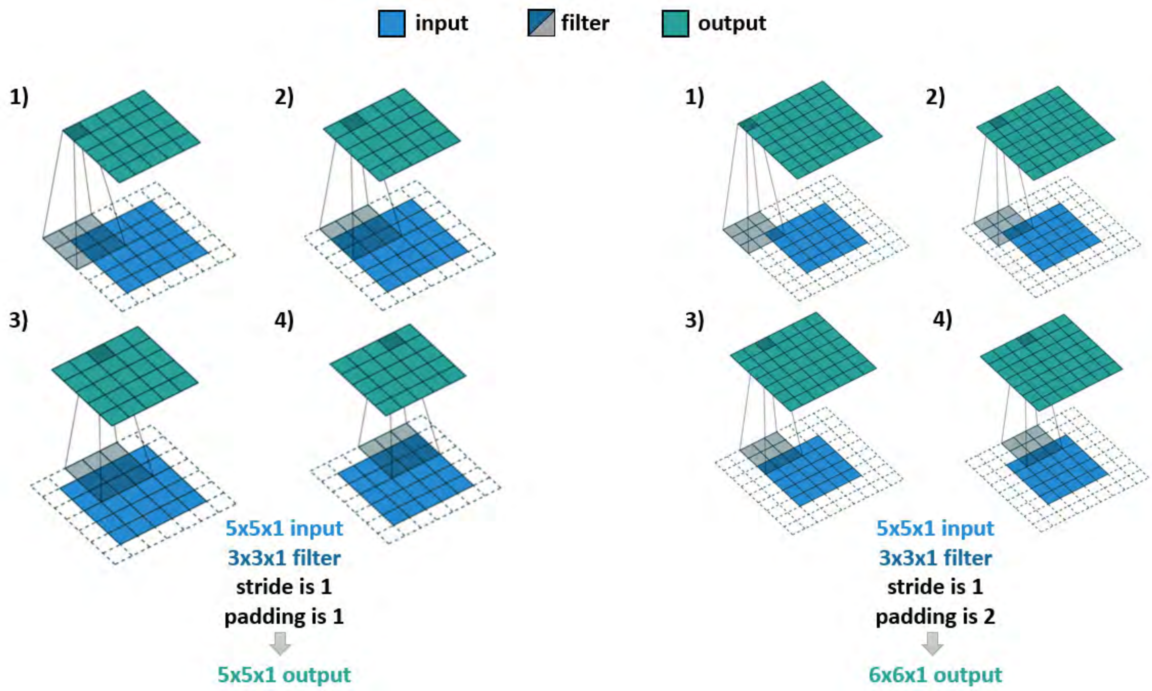

Figure 2.24. Examples of input with padding 1 and padding 2. In the first case input and output have the same dimensions whereas in the second case the output is bigger than the input. Modified from [41].

Another important hyperparameter to choose in a convolutional layer when building the architecture of a CNN is the number of different filters to pass through the input and the filter size. In order to choose the correct filter size a priori, it is important to consider the importance of the local information around a point to make a decision: bigger filters does not focus very much on the central information but on the surroundings. All these parameters are related to the complexity of the network and will be analysed in more detailed during the experiments carried out in these dissertation.

- **ReLU layer**: Convolutional Neural Networks, like the rest of Artificial Neural networks have to be able to deal with non-linearity, so that the network is able to learn both linear an non-linear relationships present in the input data. In other words, the network can learn more complex structures. Activation functions like sigmoid, hyperbolic tangent or, in the case of CNN, ReLU are used to achieve this goal. After each convolutional layer, a ReLU layer is applied, managing to introduce a non-linearity to a system where only linear operations (multiplications and sums) have been performed. ReLU stands for Rectified Linear Unit and it is defined as $f(x) = max(0, x)$, where x represents the input of the neuron. Basically, ReLU function maps negative values to zero while keeping positive values invariable.

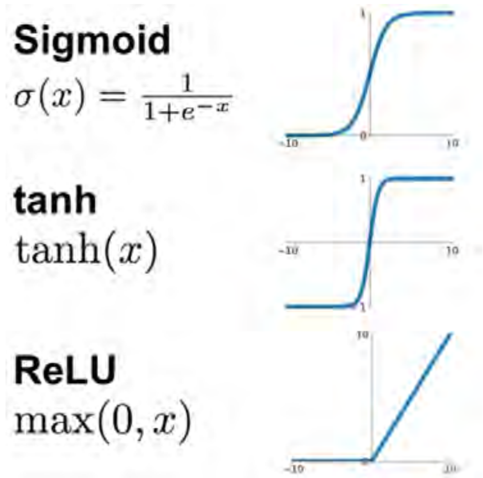

Figure 2.25. Graphical representation of three of the most common activation functions: sigmoid, hyperbolic tangent and rectified linear unit. ReLU function is the one used in CNN. Modified from [43].

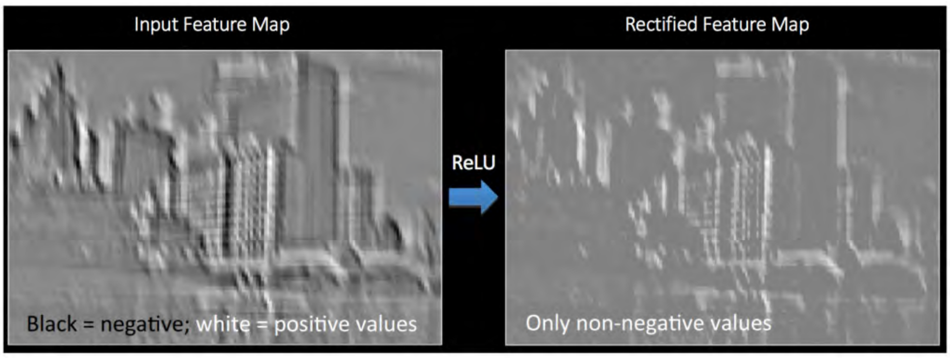

Figure 2.26. Example of a feature map before and after the ReLU layer [44].

Its graphical representation can be found in Figure 2.25 together with the sigmoid and hyperbolic tangent activation functions. The latter two functions cannot be used in networks with many layers due to the **vanishing gradient problem**, preventing deep networks from completing its training [45]. The vanishig gradient problem appears when using gradient-based learning methods and backpropagation because in some cases, the update received by each one of the weights in the network, which is proportional to the partial derivative of the error function, will be vanishingly small, preventing the weights from changing their value and, in the worst case, turning the network untrainable [46]. In addition, sigmoid and hyperbolic tangent functions have another general problem which is saturation. This means that they are only really sensitive to changes around their mid-point of their input (0.5 for sigmoid and 0.0 for tanh) and large values snap to 1.0 whereas small values snap to 0 or –1 for sigmoid and tanh respetively [45]. The ReLU activation functions overcomes the vanishing gradient problem, allowing for faster and more effective training. Figure 2.26 shows a feature map before and after passing through the ReLU layer so it can be observed how the negative pixels are transformed whereas the rest remain unchanged.

From the previous explanation, one can draw the conclusion that the final feature maps are not obtained after the sums, but after applying the ReLU function to them. The output feature maps of the ReLU layer will maintain the same dimensions as the input. Any type of convolution involves a ReLU operation because without it, the network will not achieve its true potential [36].

- **Pooling layer**: after each Convolutional plus ReLU layer it is common to add a Pooling layer. Its function is to continuously and progressively reduce the dimensionality. In this way, the number of parameters is reduced achieving several objectives. In the first place, it shortens the training time and decreases the computation power required to process the data. Moreover, the pooling layer enables the network to extract dominant features and to control overfitting, since only the most important parameters are kept [34].

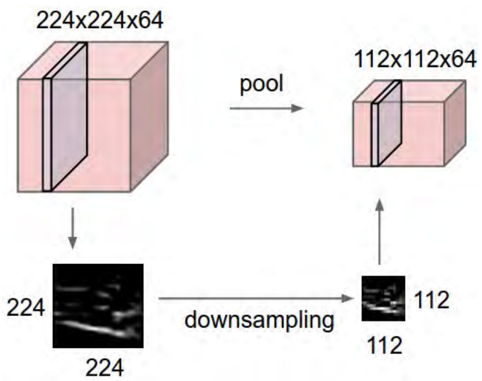

Figure 2.27. The input volume of size [224x224x64] is pooled with filter size 2, stride 2 into output volume of size [112x112x64]. Note that the volume depth is preserved [21].

Convolutional layers can also reduce the dimensions of feature maps but in this case, the pooling layer always reduces width and height, keeping the depth intact and downsampling each feature map independently. In Figure 2.27 it can be observed how the pooling layer operates, downsampling each volume spatially, independently in each depth slice of the input volume [21]. This is one of the most common pooling operations: a filter of size 2x2 applied with a stride of 2 units.

Figure 2.28 shows the process for performing the pooling operation over a 5x5 feature map with a pooling of size 3x3 and a stride of one, which results in an output image of 3x3 dimensions.

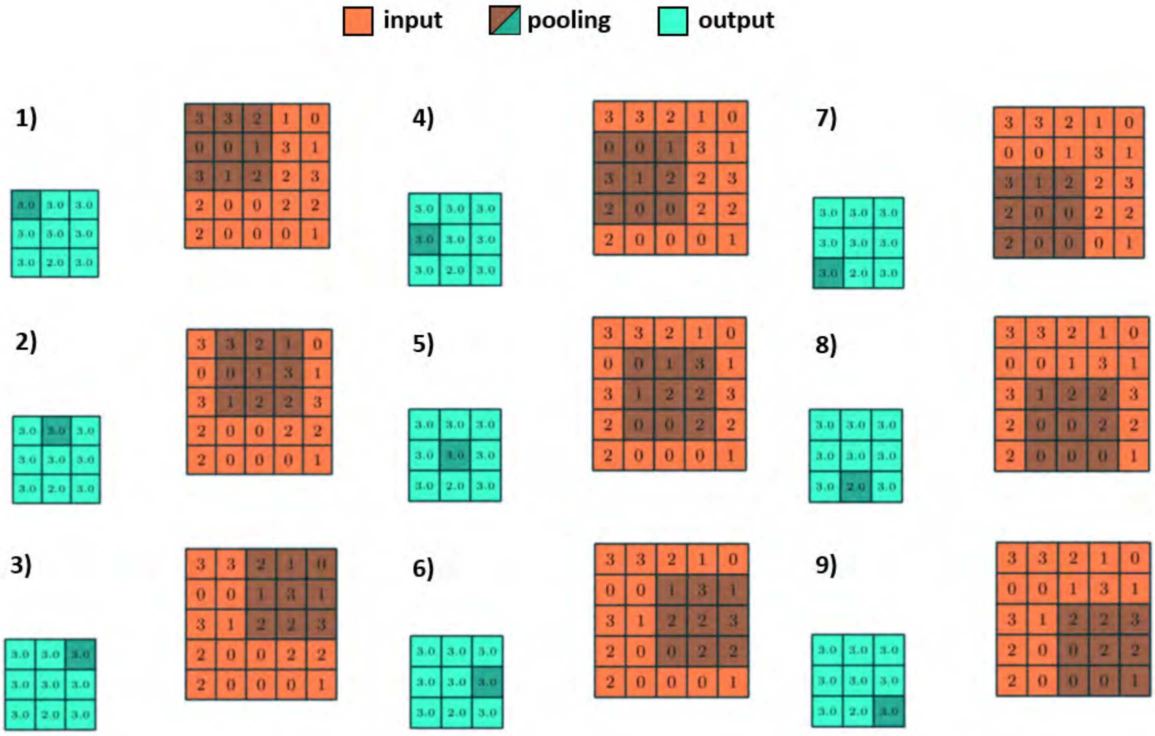

Figure 2.28. 3x3 pooling over 5x5 convolved feature. Modified from [34].

By down-sampling feature maps, pooling layers manage to summarize the presence of features in several parts of the images, making the network more robust to changes in the location of the features in the input, what it is known as *local translation invariance*. There are several ways to perform the pooling operation:

- Max-Pooling: it consists in selecting the maximum value from each region covered by the kernel. It also performs as a noise suppressor since it discards activations due to noise by taking only the most relevant values as well as reducing the dimensionality [34]. An example of its operation is shown in Figure 2.29.

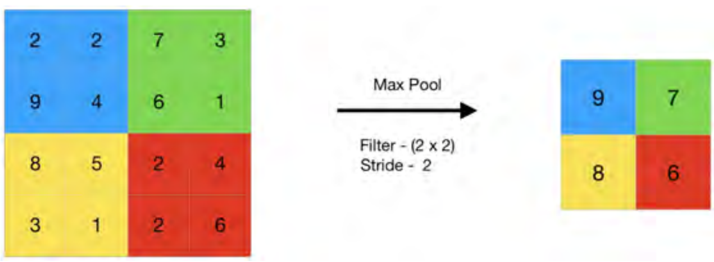

Figure 2.29. Max-Pooling operation over a 4x4 feature map with a filter of size 2x2 and a stride of 2 units [47].

- Average-Pooling: this technique calculates the average of the values from the region covered by the filter. It is not as effective as Max-Pooling to supress noise, since it simply performs dimensionality reduction [34]. Figure 2.30 shows an example of this operation.

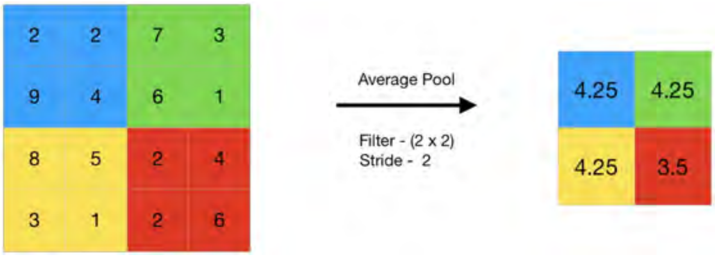

Figure 2.30. Average-Pooling operation over a 4x4 feature map with a filter of size 2x2 and a stride of 2 units [47].

– Global Pooling: the third type of pooling down samples the entire feature map into a single value, instead of using a filter of a determined size. This process is equivalent to set the filter size equal to the dimensions of the feature map. The operation used to pool the features is usually one of the two explained previously: Max or Average pooling as can be seen in Figure 2.31. Global Pooling summarizes radically the presence of a feature in an image.

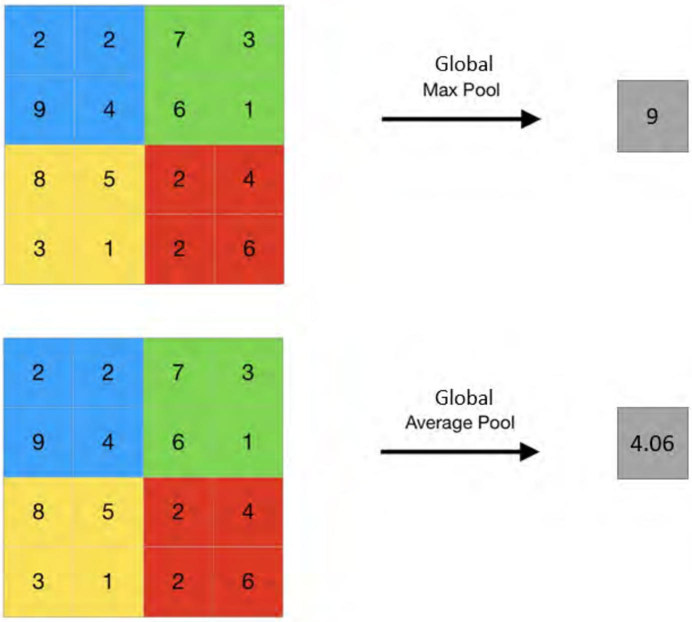

Figure 2.31. Max Global-Pooling and Average Global-Pooling operations over a 4x4 feature map. Modified from [47].

The Feature Extraction part of the CNN is composed by several blocks of Convolution, ReLU and Pooling layers, placed each one after the other and managing passing from wide, high and shallow volumes to narrow, short and deep volumes as can be seen in Figure 2.6. This means that the network has obtained a small amount of features (the most relevant ones) but from multitude of different kinds (using different types of filters).

- **Flattening layer**: after finishing the feature extraction, composed basically by the layers explained before (Convolution, ReLU and Pooling), the feature maps obtained are flattened as shown in Figure 2.32 meaning that each matrix is transformed into a 1D array. As it can be seen in Figure 2.33 each one of the feature maps obtained as output of the previous layers are flattened giving a long vector that contains all of the features extracted from the input. Each feature is passed to a neuron and, when the CNN receives a new image, the neurons containing the features extracted from that image will activate. In this way, the flattening layer is the first layer in the classification part of the CNN.

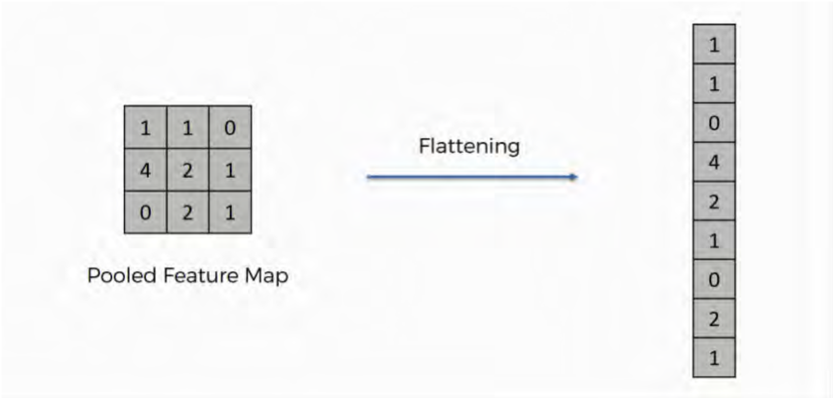

Figure 2.32. Flattening of a feature map into a vector [48].

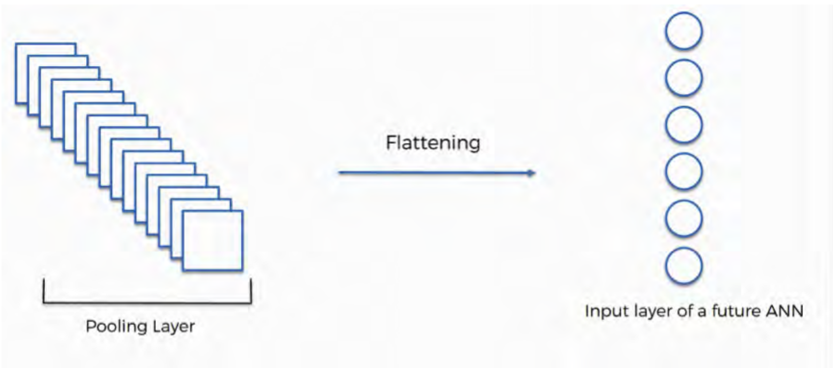

Figure 2.33. Relocating all the neurons of the feature maps in a CNN containing the extracted features into a flatten layer of neurons as input to the classifier [48].

- **Fully-connected layer**: finally, one or several fully-connected layers are added to the architecture forming a Multi-Level Perceptron (note the importance of a previous flattening layer, to obtain a 1D input to feed the ANN). This way, the output features contained in the flattening layer are fed to a feed-forward neural network with backpropagation applied to every iteration of training [34]. The output layer is composed of a number of neurons equal to the number of classes presented in the classification problem.

  Over a series of epochs, the model is able to distinguish between dominating and low-level features in images and classify them using the Softmax Classification technique [34]. When a series of features associated to a certain class are activated, it means that the input image belongs to that class and the neuron of the output layer representing the class will be activated.

  In Figure 2.34 the fully-connected layer is linked to all the neurons in the previous layer (the flattening layer) as well as with all the neurons in the output layer. It is important to highlight that the output layer should change depending on the type of problem: classification or regression. In this case, the classification problem is using a softmax function which outputs a probability for the input image of belonging to each one of the classes (the neuron activated with the highest probability will correspond to the class associated). On the other hand, in this layer a dropout operation is commonly performed as explained in section 2.1.1, in order to reduce the number of parameters and prevent overfitting.

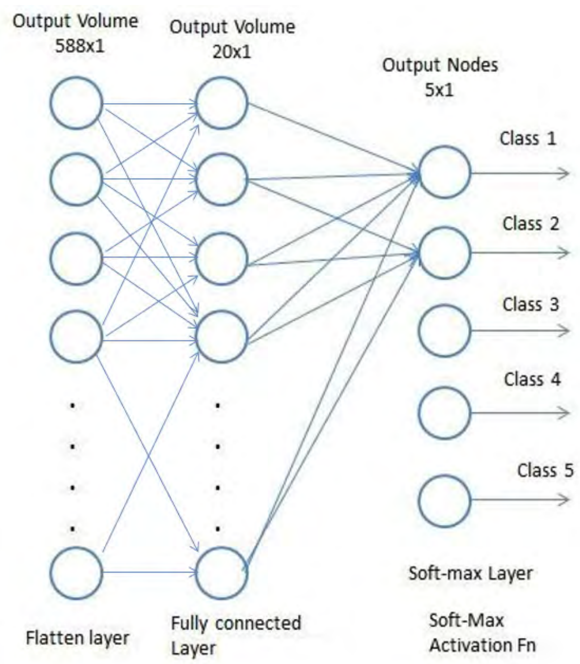

Figure 2.34. Fully-connected layer with softmax classifier as output layers of the CNN architecture to perform a classification problem. Modified from [34].

Figure 2.35. Node-link 2D diagram of a CNN trained to recognize handwritten digits. On the left there is a drawing path whereby the user can interact with the network changing its input. Layers from the architecture can be also hidden or shown through the layer visibility panel. The third filter from the first convolutional layer is also shown [49].

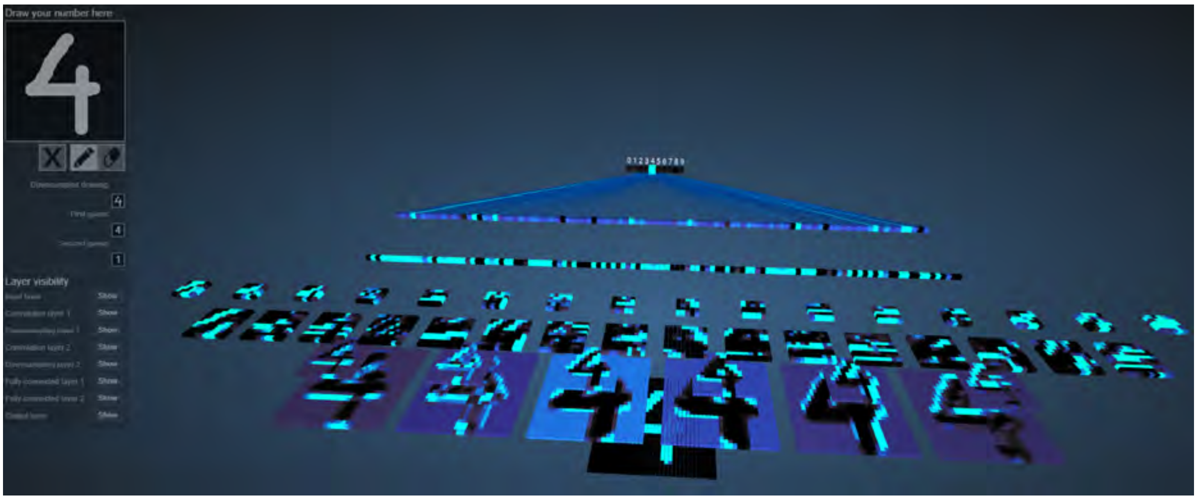

Figure 2.36. Node-link 3D diagram of a CNN trained to recognize handwritten digits. On the left there is a drawing path whereby the user can interact with the network changing its input. Layers from the architecture can be also hidden or shown through the layer visibility panel. Links from the last fully connected layer are also shown [50].

In order to have a overall picture of the behaviour of the CNN as well as fully understand its learning process, [51] presents two versions of an online interactive visualization in which the internal network structure can be better appreciated: one of them with a two-dimensional layout (Figure 2.35) and the other one with a three-dimensional layout (Figure 2.36). The visualization has been built over a trained CNN with the augmented version of the well-known MNIST dataset [52] so it excels at categorizing centred upright numbers.

The user is able to write on a drawing pad and observe how the network changes its response to different inputs. The visualization shows the output from each layer: activation maps for both convolutional and max-pooling layers as well as flattened arrays for fully-connected layers. Hoovering through pixels of each output, it can be observed to which neurons of the previous layer is connected the neuron responsible for that output pixel, as shown in Figure 2.35 and 2.36 for a neuron in a convolutional layer and for a neuron in a fully-connected layer, respectively. In addition, clicking over a pixel shows how the calculations to obtain that value have been performed. This is specially useful in the convolutional layers since the filters are plotted and the user can appreciate how each filter extracts a different feature (Figure 2.35). Color is also important in the representation of features since bright colors simbolize stronger neuron activations.

The input layer of the network corresponds to the image drew by the user in black and white. The first convolutional layer consists of six filters, each one of them focused on extracting a certain feature, such as filter 3 and 4 (the one shown in Figure 2.35 and the next one on the left) that extract diagonal edges. After that, a max-pooling operation is performed, resulting in a reduction in width and height dimensions. The second convolutional layer consists of 16 filters with very low interpretability, in addition to the max-pooling layer. Finally, all the features extracted are flattened (the flattening layer is not shown in the visualization) and passed to the classifier which consists of 2 fully connected layers. It is interesting to observe that depending on the number drawn on the pad, some neurons or others are activated, as the features representing the number.

2.2. Popular CNN architectures

Since its first appearence in 1988, many CNN architectures have been proposed over the years and some of them have become popular due to its high performance obtained on different well-known datasets, and have been considered benchmarks for performing object recognition tasks. In general, CNN architectures consists of several stacks of the basic layers explained in section 2.1.2: convolutional and max-pooling layers for the feature extraction part, followed by fully-connected and SoftMax layers for the classifier. Nevertheless, there has been an evolution in architectures through the years, always pursuing more accurate and faster results, which has resulted in some differences in topologies of modern deep learning networks.

- **LeNet-5**: it was the first proposed by LeCun in 1988 to perform a handwriting document recognition task [53], but limited computation capability and memory capacity made the algorithm difficult to implement until about 2010 [6]. This network played a pioneering role in introducing concept of average-pooling layer. It consists of two convolutional layers, two pooling layers and three fully connected layers, using the last one of them a Softmax activation function to classify among 10 categories. This architecture has about 60000 parameters [54] (Figure 2.37).

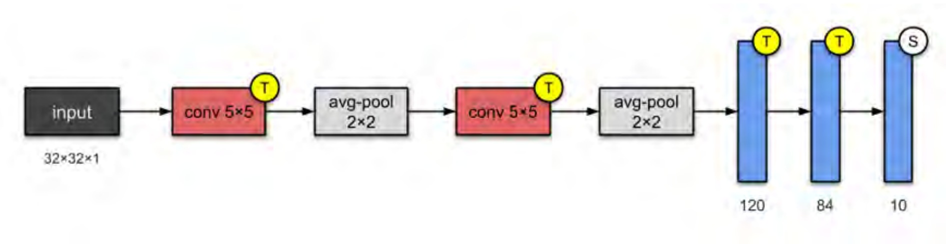

Figure 2.37. LeNet-5 architecture. Red rectangles represent Convolutional operations, gray rectangles represent Pooling operations and blue rectangles represent Dense layers, T stands for Tanh activaction function and S for Softmax function [54].

- **AlexNet**: it was proposed by Alex Krizhevesky in 2012 [55], winning that year's ImageNet Large Scale Visual Recognition Challenge (ILSVRC), which is explained in Section 2.2.1. AlexNet beat the accuracy reached by all the previous traditional machine learning approaches, achieving a significant breakthrough in the field of computer vision [6]. It was the first network to implement ReLU as activation functions as well as max-pooling layers. It consists of five convolutional layers with different filter sizes, three max-pooling layers and three fully connected layers reaching a total of 60 million parameters (Figure 2.38).

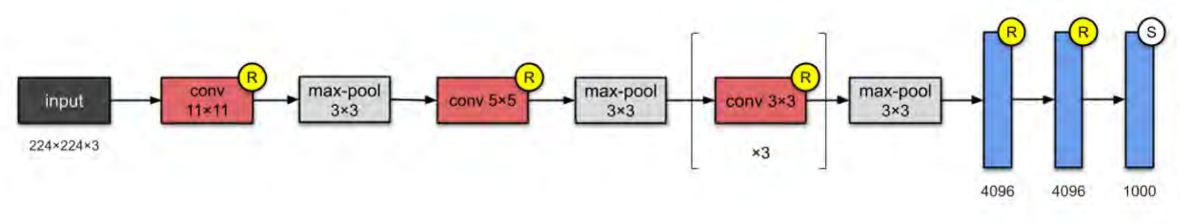

Figure 2.38. AlexNet architecture. Red rectangles represent Convolutional operations, gray rectangles represent Pooling operations and blue rectangles represent Dense layers, R stands for ReLU activaction function and S for Softmax function. Square brackets indicate a set of repeated layers [54].

- **ZFNet**: the winner from the 2013 ILSVRC (Section 2.2.1) was this network, made by Matthew Zeiler and Rob Fergue and named after their authors [37]. It is an improvement of AlexNet, using 7x7 kernels instead of 11x11 kernels in the first convolutional layer. This reduces the number of parameters significantly and improves the overall recognition accuracy, making a more efficient use of parameters as well as decreasing the complexity [6]. Figure 2.39 shows ZFNet architecture.

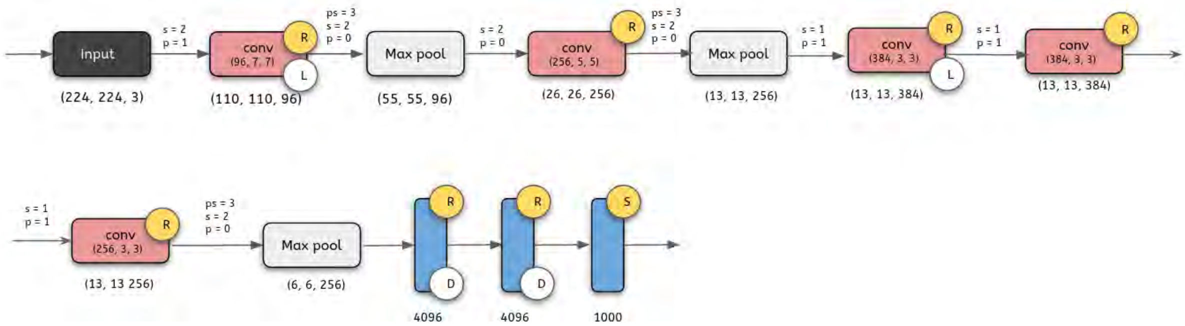

Figure 2.39. ZFNet architecture. Red rectangles represent Convolutional operations, gray rectangles represent Pooling operations and blue rectangles represent Dense layers, R stands for ReLU activaction function and S for Softmax function [56].

- **Network In Network (NIN)**: this model created in 2014 by a group of researchers from the National University of Singapore [57] introduced two new concepts: MLP convolutional layers and Global Average Pooling (GAP). The first concept is to perform convolutions with a 1x1 filter and using a mini neural network with 1 hidden layer. The value of a pixel, instead of being a linear combination of the weights in a filter and the current sliding window, is the output of a multi-layer perceptron, which helps to add more non-linearity to the models [54], [6].

  The second concept is to use GAP as a substitute of fully-connected layers: taking average of each final feature map extracted from the input image and feeding the resulting vector into the softmax layer [54]. This reduces the number of network parameters significantly, generating a final low dimensional feature vector without reducing the dimension of the feature maps in intermediate steps [6].

- **VGGNET**: the Visual Geometry Group (VGG) designed a set of CNN models with different depths: VGG-11, VGG-16 and VGG-19, which had 11, 16 and 19 layers respectively [58]. The basic building blocks of the VGG architecture consist of several convolutional layers with ReLU activation functions followed by a max-pooling layer and several fully-connected layers with softmax activation. All versions of the VGG architectures end with three fully-connected layers but the number of convolutional layers varies: VGG-11 contains eight convolutional layers, VGG-16 has thirteen convolutional layers and VGG-19 has sixteen convolutional layers [6]. VGG nets use smaller filter sizes than AlexNet (2x2 and 3x3). The architecture of VGG-16 is shown in Figure 2.40, consisting of 138 million parameters [54].

  The main contribution from this work was the experimental verification of depth as a crucial hyperparameter to achieve higher classification accuracy in CNNs. In this way, adding layers to the CNN is the most straightforward way of improving performance of deep neural networks [54]. The VGG reached the second place in the 2014 ILSVRC (Section 2.2.1).

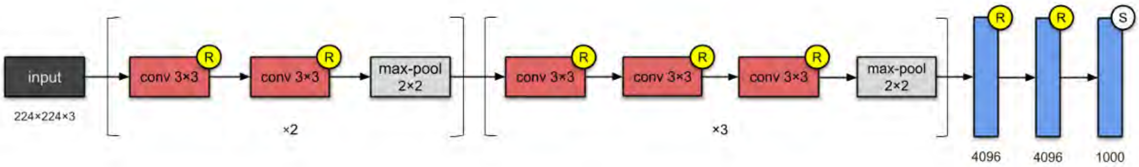

Figure 2.40. VGG-16 architecture. Red rectangles represent Convolutional operations, gray rectangles represent Pooling operations and blue rectangles represent Dense layers, R stands for ReLU activaction function and S for Softmax function. Square brackets indicate a set of repeated layers [54].

- **GoogLeNet/Inception-V1**: this network, proposed by Christian Szegedy of Google, was the winner of the ILSVRC 2014 [59] (Section 2.2.1). The objective was to reduce the computation complexity compared to the traditional CNN and managed to do it with this 22-layer architecture, which has a total amount of 5 million parameters, a much lower amount than its predecessor AlexNet or VGG. The model incorporates "Inception modules´´, an approach uses the concept of *Network In Network* explained previously in this section [6], [54].

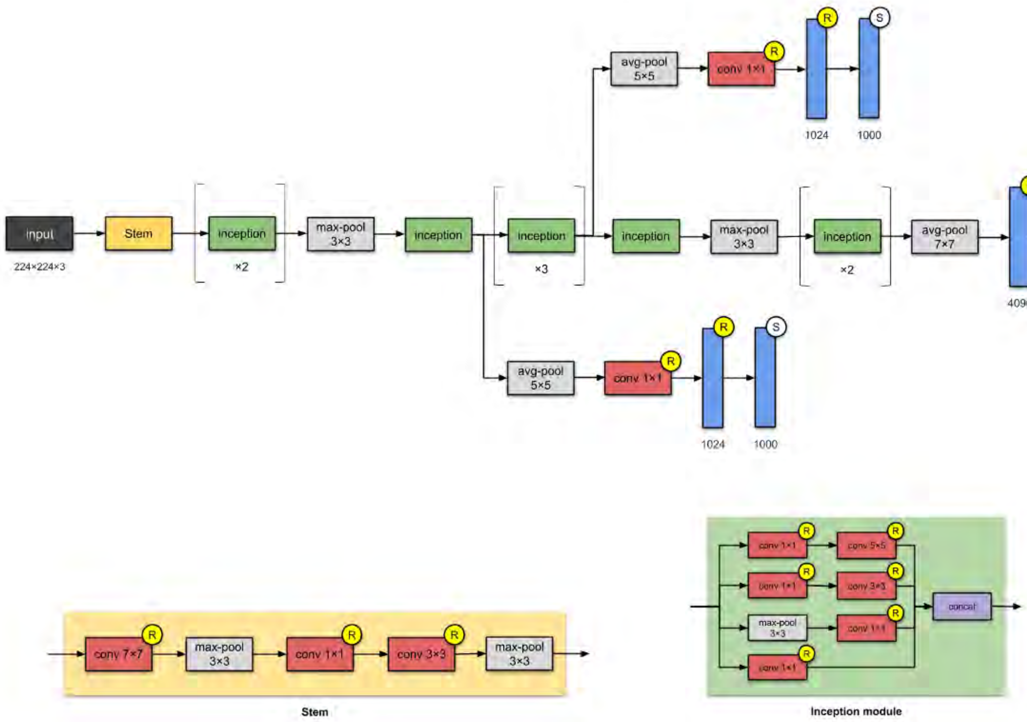

Figure 2.41. GoogLeNet or Inception-V1 architecture. Red rectangles represent Convolutional operations, gray rectangles represent Pooling operations and blue rectangles represent Dense layers, R stands for ReLU activaction function and S for Softmax function. Square brackets indicate a set of repeated layers or modules [54].

An inception module is based on several basic ideas. First of all, having parallel towers of convolutions with different filters, followed by concatenation, captures different features at 1×1, 3×3 and 5×5, thereby "clustering´´ them. Secondly, 1×1 convolutions are used for dimensionality reduction and due to its activation function, its addition also adds non-linearity (based on the Network In Network architecture) [54].

In addition, as it can be seen in Figure 2.41 two auxiliary classifiers are introduced in order to encourage discrimination in lower stages of the training, increase gradient signal that gets propagated back and provide additional regularisation. It is important to highlight that the branches connected to the auxiliary classifiers are discarded when performing inference [54].

Based on the architecture of Inception-V1, several successor networks were developed. In 2015 Christian Szegedy presents **Inception-V3** [60], an improvement of Inception-V1 with 24 million parameters and one of the first models to use batch normalisation. After that, an improvement of Inception-V3 is developed in 2016: **Inception-V4** [61], which has 43 million parameters and adds more inception modules among other changes.

- **ResNet-50**: the Residual Network architecture, better known as ResNet and developed by Kaiming He, was the winner of the ILSVRC 2015 (Section 2.2.1) [62]. There are several ResNet variants with different number of layers: 34, 50, 101, 152 and 1202, but the most popular one is ResNet-50, which contains forty-nine convolution layers and one fully-connected layer at the end of the network, resulting in a total of 26 million parameters [6], [54].

  The predecessors of ResNet had an increasing number of layers but they also suffered from the vanishing gradient problem. The aim was to design a ultra-deep network in which accuracy did not get saturated neither degraded rapidly [6], [54]. Figure 2.42 shows the architecture of ResNet-50. The Identity block is the basic building block from ResNet, being a traditional feed forward network with residual connection. The idea behind this is the one shown in Figure 2.43 where to make the prediction equal to the actual value, the network has to be able to compute the residual value [63]. This residual connections, also known as *skip connections via addition*, backpropagate through the identity function, simply multiplying the gradient by one. In this way, the gradient is preserved in the early layers, avoiding the vainishing gradient problem [64].

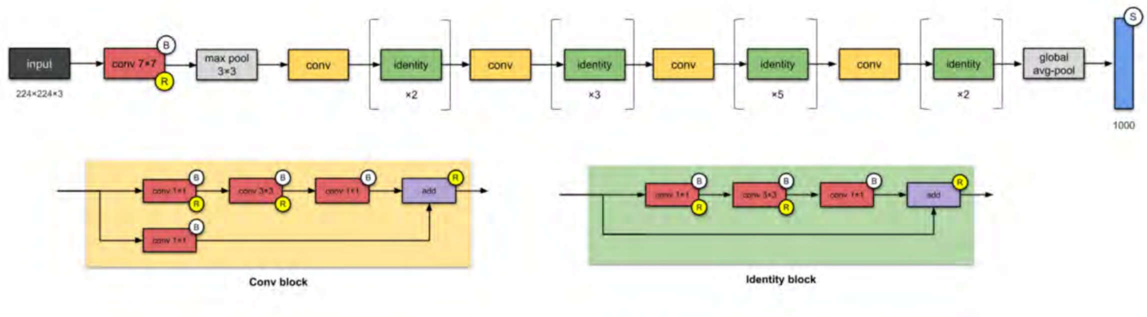

Figure 2.42. ResNet-50 architecture. Red rectangles represent Convolutional operations, gray rectangles represent Pooling operations and blue rectangles represent Dense layers, R stands for ReLU activaction function and S for Softmax function. Square brackets indicate a set of repeated layers or modules [54].

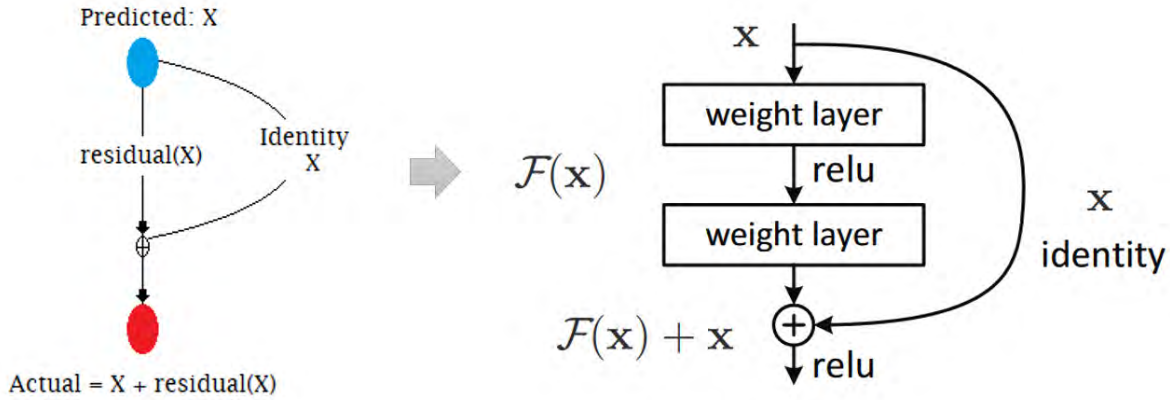

Figure 2.43. Residual learning. Flowchart representing the concept (left) and a building block from the architecture (right). Modified from [63] and [62].

There are also several advanced architectures that combine Inception and Residual blocks. One of the most popular networks of this kind is the **Inception-ResNet-V2** [61], developed in 2016 and has 56 million parameters [54]. In 2017, it is developed the ResNeXt-50 [65] with 25 million parameters, which adds parallel branches within each identity block.

- **DenseNet**: Gao Huang among other researchers developed in 2017 DenseNet [66] consisting of densely connected CNN layers where each layer is connected with all the preceding layers, hence taking all the previous feature maps as input. This approach manages to make the network more efficient since the features extracted in one layer can be reused in all successor layers, which reduces network parameters dramatically [6]. Figure 2.44 shows an example diagram of a DenseNet. An important hyperparameter in this kind of models is the *Growth rate*, represented as k, which is the number of feature maps produced by each function H_l. The paper presenting DenseNet [66] showed that a relatively small growth rate is sufficient to obtain state-of-the-art results.

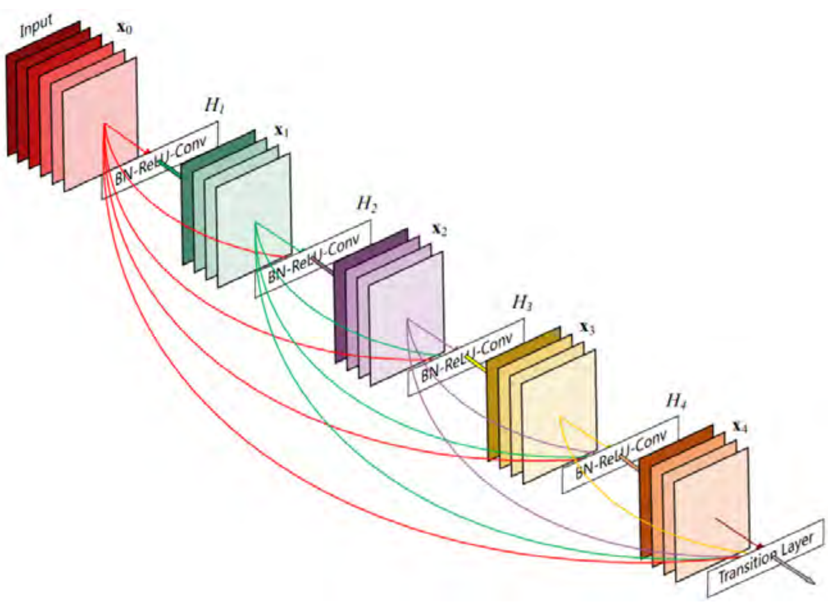

Figure 2.44. A 5-layer dense block with a growth rate of k = 4. Each layer takes all preceding feature-maps as input [66].

2.2.1. ImageNet Large Scale Visual Recognition Challenge (ILSVRC)

ImageNet is an image dataset created as a result of an ongoing research project to provide scientists around the world an easy accessible image database for use in visual object recognition software research [67]. ImageNet currently has more than 14 million images belonging to more than 20000 categories [68]. ImageNet images follows the organisation established for WordNet concepts.

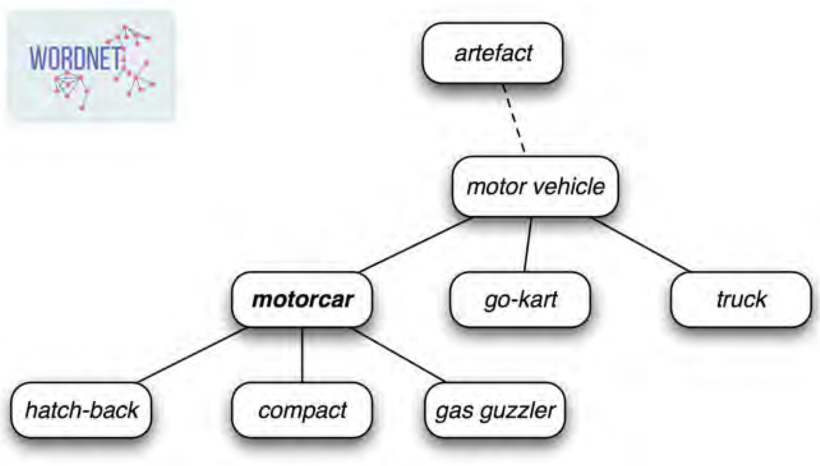

Figure 2.45. Fragment of WordNet Concept Hierarchy: nodes correspond to synsets; edges indicate the hypernym/hyponym relation, i.e. the relation between superordinate and subordinate concepts. Modified from [69]

Figure 2.46. A snapshot of two root-to-leaf branches of ImageNet: the top row is from the mammal subtree; the bottom row is form the vehicle subtree. For each synset, 6 randomly sampled images are presented. Modified from [70].

Wordnet [71] is a lexical datasbase of semantic relations between words in more than 200 languages, first created in English by a group of researchers in Princeton University [72]. It is widely used for Natural Language Processing applications such as machine translation or natural-language generation. As it can be appreciated from the parallelism existing between

Figures 2.45 and 2.46 ImageNet dataset is organised according to the WordNet hierarchy. In WordNet, each concept is described by multiple words or word phrases and is called “synonym set´´ or “synset´´. ImageNet aims to provide on average 1000 images to illustrate each synset, which are organised hierarchically in high level categories and subcategories [67].

Every year between 2010 and 2017 a competition known as ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [73] was held with the aim of measuring the progress in computer vision benchmarking the state of the art as well as promoting the development of better computer vision techniques [74]. The ILSVRC uses a reduced list of only 1000 image classes, such as pencil, coffee mug and many animal breeds. As a result, most popular architectures presented previously (Section 2.2) have been discovered in this competition. Although more recently the challenge tasks are changing to increase difficulty such as labelling videos, generally, the challenge tasks every year can be classify in three blocks [74]:

- **Image classification**: the algorithm has to predict the classes of objects present in an image.
- **Single-object localization**: the algorithm has to perform first an image classification and, after that, draw a bounding box over an image around one example of each object present.
- **Object detection**: the algorithm has to perform first an image classification and, after that, draw a bounding box around each object present.

Figure 2.47 shows the Top-5 error rate, that is, the fraction of test images for which the correct label is not among the five labels considered most probable by the model, obtained from 2010 to 2015 in ILSVRC competition. There are several important facts to highlight from this image. First of all, the accuracy of the models has been improving significantly over the years or, what is the same, the error rate has been decreasing.

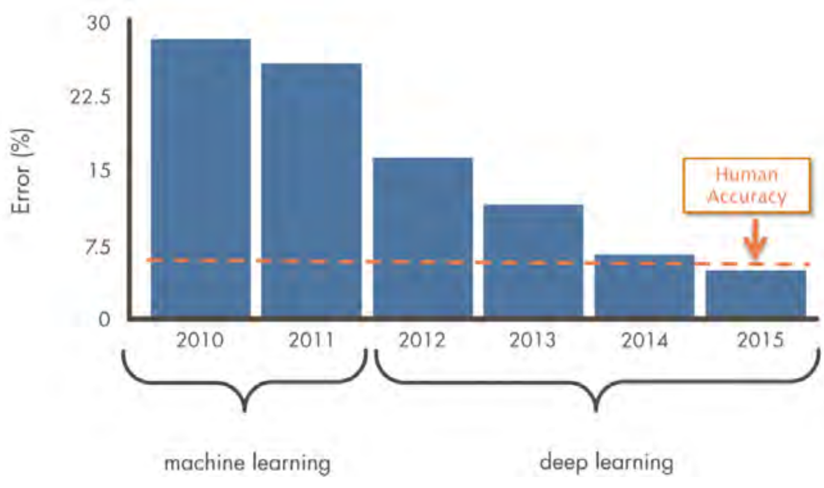

Figure 2.47. ILSVRC Top-5 Error rate on ImageNet [15].

Secondly, the first two years that the competition was held, the models developed used Machine Learning techniques whereas from 2012 until 2015 (and subsequent years) the competitors have developed Deep Learning architectures. Note that AlexNet, a pioneer on this field, was developed in 2012 winning that year’s ILSVRC (Section 2.2). A significant decrease in the error rate from 2011 to 2012 can be observed, evidencing that Deep Learning models are much more powerful for image recognition than Machine Learning models.

Finally, in 2015 the models developed in the ILSVRC competition, achieved to outperform humans at classifying images for fist time in history. In other words, since then, there are some images that humans are not able to classify but machines are.

2.2.2. Pretrained Networks and Transfer Learning

Training a CNN with the same amount of parameters as the models explained previously (Section 2.2) can be very expensive in terms of both computational power and time, even with the processing capacity available today. This, combined with the fact that those several popular CNN models are known as "standards" for multitude of different tasks of image classification (since they have been trained on ImageNet to recognise hundreds of different objects), gives purpose to pretrained networks.

A pretrained network, as its name implies, is a model that has been trained previously so it has already learnt to extract powerful and informative features and it can be used without the need of spending an excessive amount of time or computing power. There are three approaches to make use of a pretrained networks depending on the problem that has to be solved [75]:

- **Classification**: apply pretrained networks directly to classification problems.
- **Feature Extraction**: use a pretrained network as a feature extractor by using the layer activations as features. Then, these activations can be used as features to train another machine learning model, such as a support vector machine (SVM).
- **Transfer Learning**: Take layers from a network trained on a large data set and fine-tune on a new data set. This technique is widely used nowadays and it would be explained in more detail down below.

In addition to choosing an approach of the previous one, the data scientist has to choose also the pretrained network that better suites the problem being addressed. Pretrained networks have different characteristics to take into account, but the most important ones are accuracy, speed and size. Choosing a model is generally a trade-off between these characteristics [75]. The plot in Figure 2.48 represents the three mentioned characteristics for group of well-known pretrained models.

Transfer Learning, unlike the other two approaches, enables to better adapt the network to the problem addressed. That is because the data scientist can unfreeze some of the top layers from the architecture and retrain them with the new dataset. Normally, the layers being unfrozen are the ones belonging to the classification part of the network since the feature extraction part is very powerful in pretrained networks and is standard for any problem, but the classification part usually differs from the problem being addressed. Sometimes, one or two of the top layers from the feature extraction part of the network are unfrozen as they are the ones in charged of obtaining the most class-specific features. To clarify this process the Figure 2.49 shows an explanatory diagram.

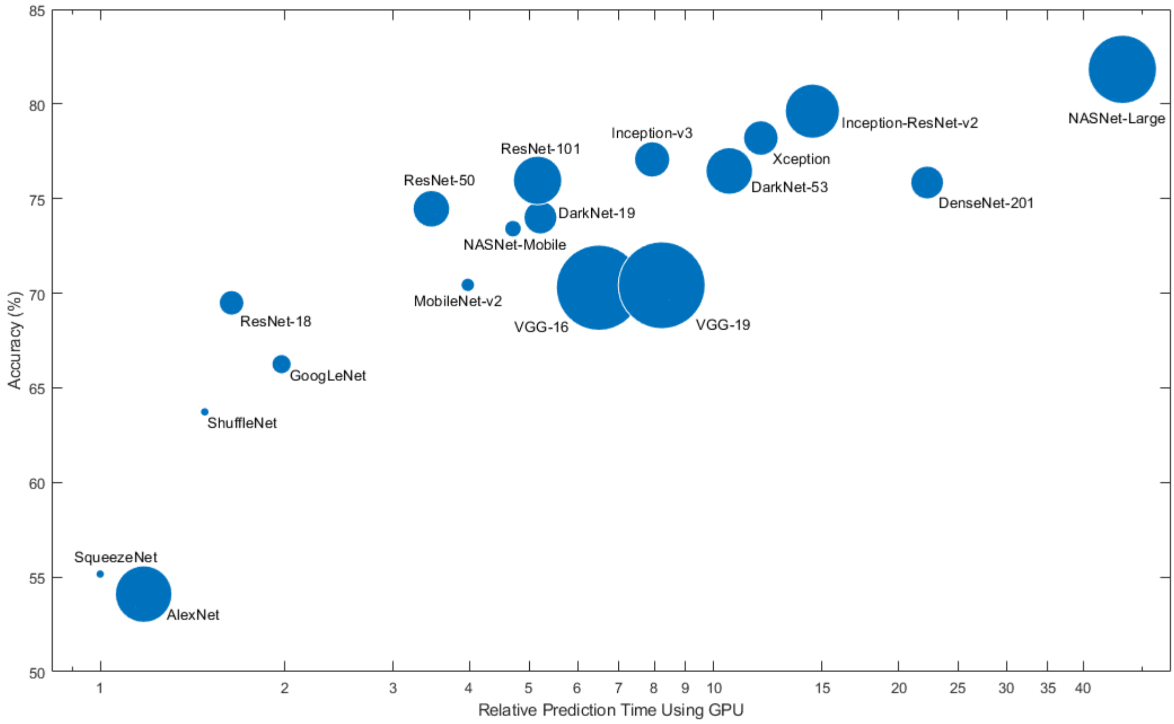

Figure 2.48. An indication of the relative predicition speed/time (x-axis), accuracy (y-axis) and size (point's size) of the most popular pretrained networks. The exact prediction and training iteration times depend on the hardware and mini-batch size that you use [75]

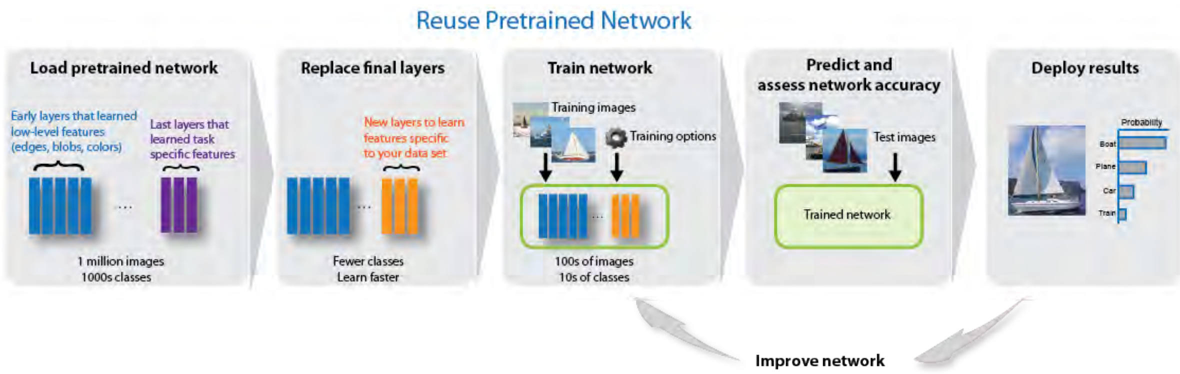

Figure 2.49. Transfer learning workflow [75].

For example, if the problem consists in classifying between dogs and cats, the VGG-16 network can be used applying transfer learning, for what the top three layers are unfrozen (the ones belonging to the classification part, Figure 2.40) and retrained with a dataset that only contains images of dogs and cats.

3

Literature Review

I start where the last man left off.

Thomas A. Edison (1847–1931)

Increasing the interpretability of a CNN has raised attention among researchers over the last years. A compilation of the existing related work on this topic is described in this chapter in order to examine the different techniques applied so far and how they differ from the approach presented in this thesis.

As explained in section 1.2 from chapter 1 with little detail, the methodology presented in this thesis to analyse the learning process of the CNN can be divided in two stages. First of all, the optimal or minimal architecture must be found, understood as the one that is able to obtain high levels of accuracy in both training and validation datasets with the minimum number of parameters. Once this model is obtained, a series of techniques have to be applied in order to understand how the network is identifying the decisive features to perform the classification.

Since the process followed makes use of two very different types of techniques, the present chapter is divided in two sections, focused each one of them on studying the state of art of one of the groups. The first section includes all the approaches to search the minimal CNN architecture that is able to recognise the majority of images from the dataset whereas the second section is focused on approaches to enhance the one mentioned model interpretability.

3.1. Minimum model search

When applying Machine Learning to solve any problem, the minimum model is pursued, understood as the one that achieves the highest precision without increasing too much the complexity. As it was previously explained in Chapter 1 this fact is due to the existing trade-off between interpretability and precision of the model whereby highly accurate models are really hard to understand whereas models easily interpretable are not too accurate.

As the main purpose of this thesis is to understand how the network is learning and CNN are intrinsically complex, the best option is to find the minimum model, as the simpler the network architecture is, the easier it will be to analyse. For this reason, several techniques to find CNN minimum models are presented in this section. This study also underlines the importance of the initialization of the weights, which can significantly increase training performance if these initial weights are close to the final values.

Talking about minimum model leads inevitably to the topic of *model compression* which goal is to achieve a simplified model from the original one without a significant loss in accuracy or performance. A simplified model is one that is reduced in size (meaning that the compressed model has fewer parameters and thus, uses less RAM while running) and/or latency (meaning that there is a decrease in the time it takes for the model to make a prediction based on an input to the trained model, reducing at the same time its energy consumption) [76]. As it can be appreciated, both searching for the minimum model and model compression have the same objective as well as the methods for achieving it. In practice several of the following approaches can be combined.

The first and most common technique is pruning, which involves removing connections between neurons setting individual weights to zero or entire neurons, setting all weight values related to that node to zero. As stated in [76], the motivation behind pruning is that neural networks tend to be over-parametrized, with multiple features encoding nearly the same information. Removing weights would lower the amount of parameters in the network while keeping the same architecture, whereas removing neurons will change the structure of the model, making it smaller. Figure 3.1 shows an example of both kinds of pruning.

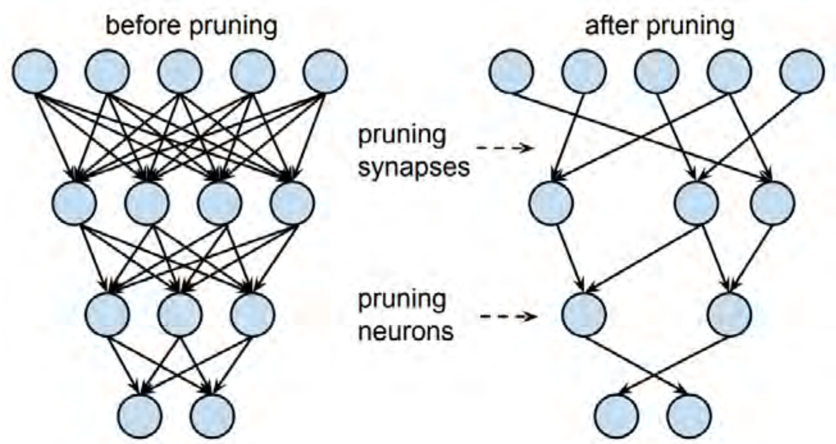

Figure 3.1. Neural network before and after pruning weights/synapses and nodes/neurons [77].

It is easier to prune weights without decreasing the overall accuracy of the network but it requires a higher amount of hardware computation capacity to perform a certain amount of sparsity to be efficient. On the other hand, neuron pruning will decrease hardware computation since the network architecture is smaller but removing entire nodes can more easily decrease the accuracy of the network [78]. In addition, both weight-based and neuron-based pruning are included in the group of unstructured pruning, which involves removing individual weights or neurons [76].

One of the most common methods is known as *magnitude-based pruning* presented in [77] and based on establishing a threshold chosen as a "quality parameter" multiplied by the standard deviation of a layer's weights. In this way, all weights with value below the threshold, will be set to zero (removed from the network). After pruning, the new sparse network has to be retrained so the remaining connections can compensate for the connections that have been removed. This is one of the simplest ways of performing pruning.

Activations on training data can be also used to prune the network. According to [78], when doing inference (running a dataset through a network), some neurons always output near-zero values. Those are the neurons that can be removed from the architecture without a high impact on the model. Intuitively, if a neuron rarely activates with a high value, then it is rarely used to perform the model's task.

In addition, redundancy of parameters can also be used to remove neurons since if two neurons in the same layer have very similar weights of activations it can mean they have the same functionality so one of them is redundant [78]. The final network after pruning ideally would have neurons with unique parameters, each one of them performing a different operation (like extracting a different feature).

Another pruning technique that can be classified as neuron pruning (hence, unstructured pruning) is presented by [79] based on *The Lottery Ticket Hypothesis*. This hypothesis states that any randomly-initialized dense neural network contains a subnetwork that is initialized such that, when trained in isolation, it can match the test accuracy of the original network after training for at most the same number of iterations. This subnetworks are known as *winning tickets* since their connections have initial weights that make training particularly effective. In other words, they have won the "initialization lottery" [79].

This methodology enables to, once the winning ticket network has been identified, to prune the rest of connections, reducing the number of parameters without losing accuracy, since it is this subnetwork the one contributing to solving the problem, whereas the other weights and neurons are residual.

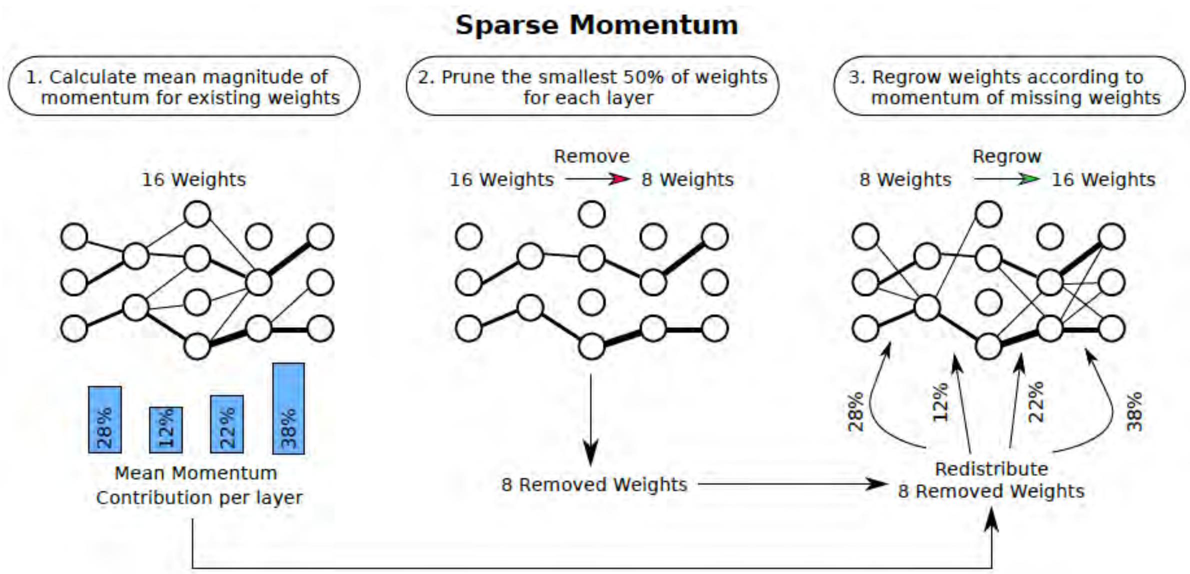

Figure 3.2. Sparse Momentum flow that is applied at the end of each epoch [80].

Since unstructured pruning increase the sparsity of the network, known as the proportion of zero to non-zero weights, the concept of *sparse learning* is becoming increasingly important. This methodology (that is not classified strictly as a pruning technique) is understood as accelerated training that makes use of sparse weights or nodes settled at widely spaced intervals, being opposite to dense learning, in which all the neurons are connected to each other. [80] proposes a sparse learning algorithm called *sparse momentum*, which can rival dense neural network performance while reducing training times. This technique uses exponentially smoothed gradients (momentum) to identify layers and weights which reduce the error efficiently, leaving this connections in the architecture and pruning those that does not produce such a high decrease in the error rate. Figure 3.2 shows a flowchart explaining the process.

In order to apply this methodology is necessary to fully train the network first so the link between neurons with the highest weights can be selected through the calculation of the momentum and the links with the smallest ways can be removed. In this way, only the strongest activations (the ones that contribute more to the network task) are selected. Finally, the removed weights are redistributed proportionate to the momentum magnitude, starting from those neurons with the largest momentum.

The intuitive idea behind the momentum is to estimate how efficient the average weight in each layer is at reducing the overall error. This magnitude is first used to prune the network and, after that, grow and redistribute weights. This method reduces the number of network parameters, yielding speedups during training without losing overall accuracy [80].

In addition to unstructured pruning techniques, another group of methodologies known as *Structured pruning* is commonly used and well-suited for CNN. Structured pruning does not give as a result matrices with sparse connectivity patterns (a lot of the elements set to zero) because it involves removing entire blocks of weights such as channels of filters from a trained network [76]. As it can be observed in Figure 3.3 there are three types of structured pruning: filter pruning; which removes whole filters, channel pruning; which removes whole channels and filter shape pruning, which removes the weights in the same location of all filters in one specific layer [81].

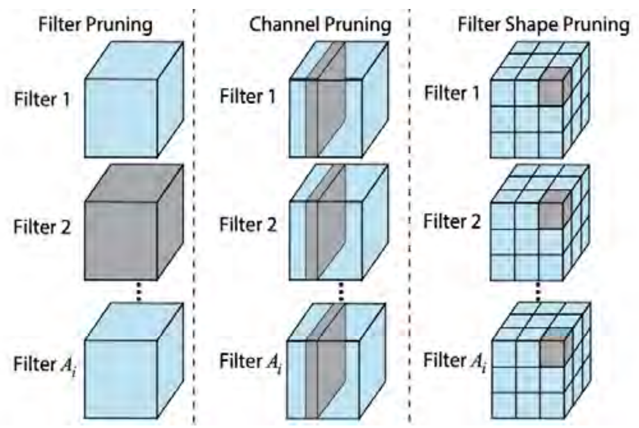

Figure 3.3. Three types of structured pruning [81].

Magnitude-based pruning can also be applied in order to perform structured pruning but there are other approaches such as the one presented in [82]. The algorithm introduced trains “pruning agents” that remove unnecessary CNN filters from the original model. These agents aim to maximize a reward function which is defined as the drop in performance between the original and the pruned model, forcing the agents to keep performance above a specified level. In this way, an agent is trained for each convolutional layer and it learns, by combining different filters, which modifications to the original model will increase the network efficiency white-out losing accuracy [76]. It is interesting to notice that this methodology makes use of distinctive concepts of Reinforcement Learning such as *agent* or *reward function*.

Although pruning is widely used, it has several drawbacks such as the generalization power of given pruning methods across different architectures. The pruning process also involves a lot of fine-tuning which increases the time to generate a final model [76].

Another model compression technique is quantization, which involves decreasing the size of the weights learnt by the network. In other words, mapping values from a large set to values in a smaller set, being the output a smaller range of possible values than the input without losing too much information in the process [76]. Applied to the neural networks, quantization aims at reducing the precision of network components making the model more lightweight but without a high difference in accuracy.

The quantization of neural networks can be seen as a way to compress the network, as if it was an image or audio file. Some approaches take advantage of the fact that weights are normally distributed within a certain range and reduced 32-bit floating point parameters to 8-bit fixed points [76]. In practice, quantization implementations are generally tied to the features of the hardware being used, which has to be able to handle numbers of the specific size.

Another approach to compress models is the Low-rank approximation, whose goal is to approximate the redundant filters of a layer using a linear combination of fewer filters. For example, in Figure 3.4, it can be observed that in image (a) shows filters of a convolutional layer, many of wich look nearly the same, whereas image (b) has performed an approximation of all filters. In addition, the filters in (b) have a lower rank, defined as the number of linearly-independent columns of the filter [76]. [83] makes use of this technique, which has several points in common with the Singular Value Decomposition method.

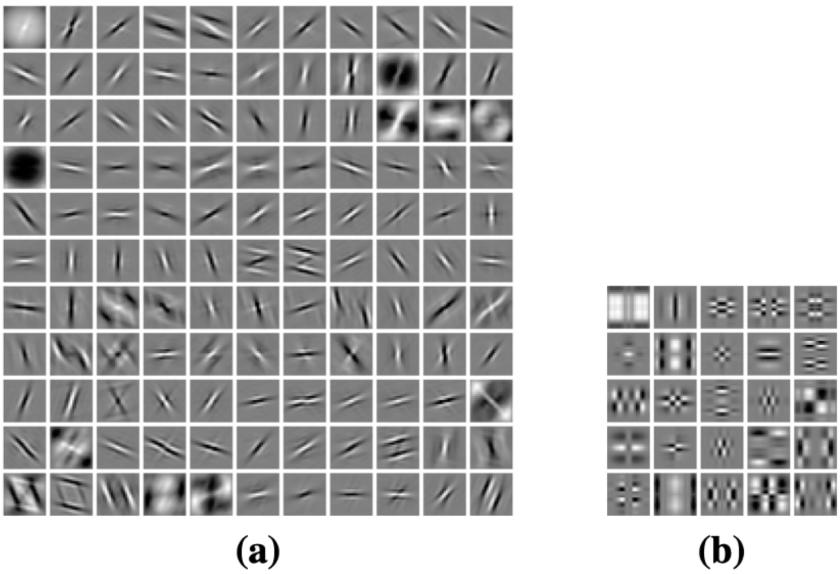

Figure 3.4. Convolutional filter bank (a) learned for the extraction of linear structures in retinal scan images, along with its separable low-rank approximation (b) [84].

Although these techniques present remarkable improvements over the original CNN architectures, the resulting models still have a high degree of complexity, which makes the results difficult to understand. This is due to the fact that the general process followed is to train a very complex network and after that, try to reduce the size of it without losing accuracy. That is the reason why in this thesis a grid search method is proposed to find the hyperparameters that make up the minimum CNN. With grid search method, the size of the models is more controlled and the complexity can be increased in a progressive way.

Grid-searching is the process that takes a set of possible network hyperparameters as input and builds a model with each parameter combination, storing at the same time both training and validation accuracy. In this way, after all models have been generated, the architecture with the best trade-off value between accuracy and complexity is selected as the minimum model.

3.2. Model interpretability

Most of the references agree on visualization as one of the most powerful tools to properly interpret the results given by the CNN. There are different kinds of visualizations that can be obtained from CNNs such as the structure of their internal architecture, the extracted features (activation maps) in each layer, which enable, in turn, to infer the filters learnt by convolutional layers; or the parts of the images that are decisive when classifying them in a specific category (heat maps).

In order to present the range of visualization techniques in the most comprehensive way for the reader, this section is divided in two blocks. The first one contains references from works focused on displaying the CNN intermediate layers so that filters can be studied, whereas the second block covers visualization methodologies used to understand the classification output given by the CNN.

3.2.1. Intermediate layers visualization

This section encompass a series of approaches that aim to clarify how the internal structure of the CNN is working to perform the classification task. In order to better organise the wide range of existing techniques, a non-canonical classification is proposed depending on whether the visualization changes or not to different input images: *static visualizations* and *dynamic visualizations*.

On the one hand, static visualizations is the category that gathers all the plots extracted from the CNN intermediate layers that stay constant no matter which input image is being fed to the network. It mainly include the techniques used to represent or that takes as basis the filters learned by the CNN in the convolutional layers since once the training is finished, the network has already learnt all the filters that are considered important to perform the classification task, meaning they will not change when a new image is presented to the network.

On the other hand, the category of dynamic visualizations is comprised of plots that change depending on the image that is being inputted to the CNN. The visualizations par excellence included in this category are the feature maps or activation maps which are outputted by each layer belonging to the feature extraction part of the network. The activation maps show, in fact, which parts of the image “activate” certain neurons after having applied the filter, showing the features that are being detected. Since each image is different, the activation maps will be different too for each input sample.

3.2.1.1. Static visualizations

Filters are the static visualization used in CNN par excellence. As it has been previously explained in chapter 2, they are composed of different kernels (channels). A kernel is simply a matrix of numbers that after performing an operation over an image (which is, in fact, another matrix) gives a new image as a result with a specific effect. The operation performed between the two matrices is known as convolution and it is widely used, not only in CNNs but in telecommunications sector for signal processing.

In image processing, depending on the desired effect over the final image a certain kernel must be applied. For a CNN the process is similar: the network learns which filters needs to apply to the input image to obtain the desired features. For example, if horizontal borders are essential to differentiate images belonging to one class from another, a Sobel filter will outline the borders which will pass as features to the following layer.

Plotting CNN filters is very simple: iterating through all the convolutional layers, the weights for each layer are extracted and plotted with all of their channels/kernels. [51] presents a very complete visualization which, among other features, displays the kernels learnt for each filter in the convolutional layers as it has been previously explained. Figure 3.5 shows some filter plots from the VGG-16 architecture, where dark squares represent small or inhibitory weights whereas the light squares indicate large or excitatory weights [85]. Using this intuition, it can be observed that the filters on the first row detect a gradient from light in the top left to dark in the bottom right corners or, what is the same, a diagonal border from the bottom left to the top right corner.

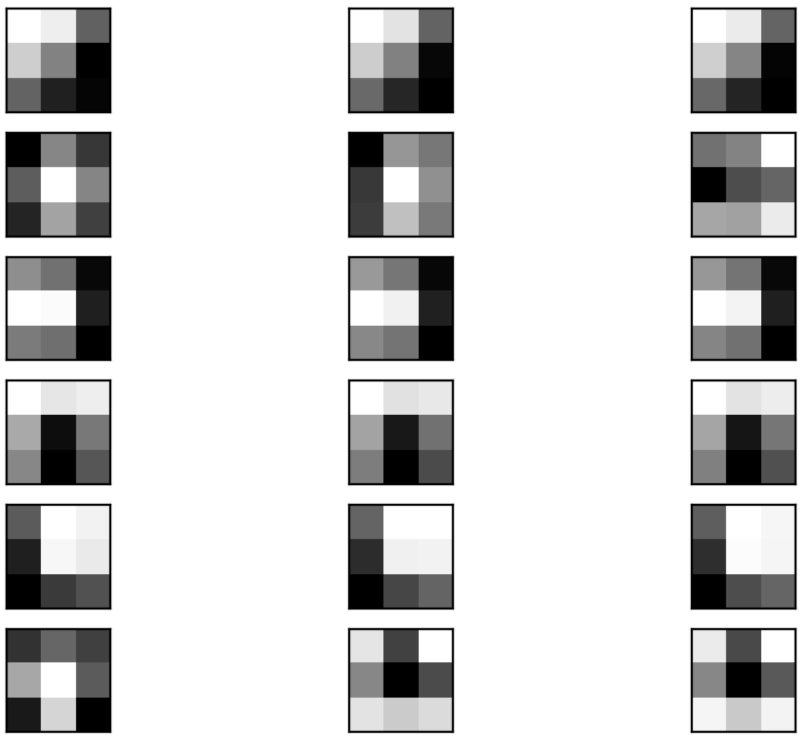

Figure 3.5. Plot of the first six filters from VGG-16 architecture with one subplot per channel [85].

To know more about the wide range of existing filters, some of the typical ones are presented in [86] like the sharpen kernel shown in Figure 3.6, which emphasizes differences in adjacent pixel values making the image look more vivid or the emboss kernel shown in Figure 3.7 which emphasizes the differences of pixels in a given direction. The emboss kernel is similar to the Sobel kernel and sometimes they are referred to mean the same, it is also equivalent to the filters in the first row of Figure 3.5.

Figure 3.6. Sharpen kernel matrix plus effect of appying the kenel on an image [86].

Figure 3.7. Emboss kernel matrix plus effect of applying the kernel on an image. In this case the direction of the emphasized differences of pixels is along a line from the top left to the bottom right [86].

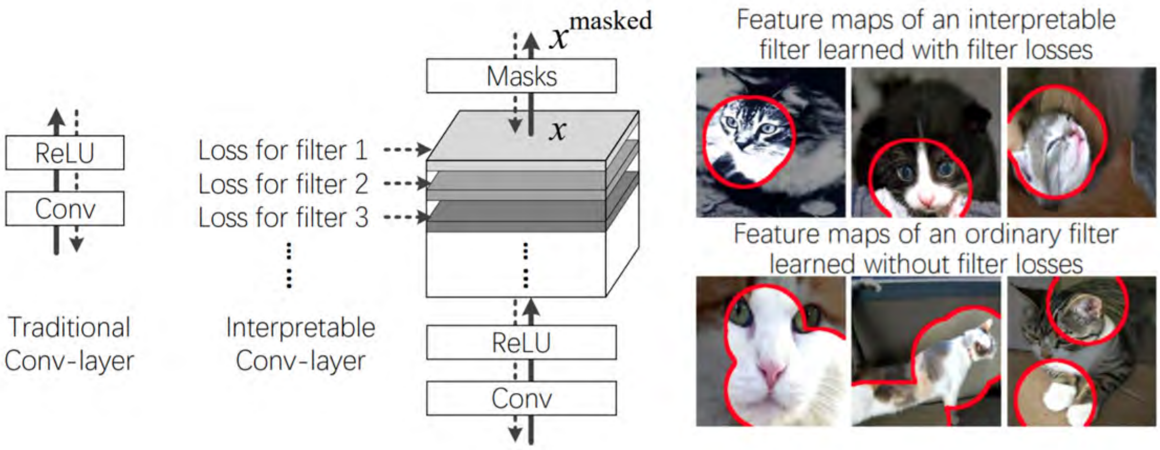

Figure 3.8. Structures of an ordinary conv-layer and an interpretable conv-layer. Solid and dashed lines indicate the forward and backward propagations, respectively (left). Comparison of an interpretable filter's feature maps with a filter's feature maps in a traditional CNN (right). Modified from [87]

In addition, some researchers have tried to make filters easier to understand as [87] shows. In this study, a generic method is proposed to learn *interpretable convolutional filters*, making use of a new CNN architecture. The aim of this model is to obtain filters specialized in recognising a single object/feature through different input images, as opposed to what happens with ordinary

CNN filters that usually identify a mixture of features. The algorithm proposed to purify the semantic meaning of each filter in a CNN formulates filter loss and uses this definition to push a filter to represent a specific object part of the category and keep silent on images from other categories. In addition, a mask layer is assign to the feature map to filter out noisy activations. Figure 3.8 shows the comparison between traditional and interpretable convolutional layer architectures where the concepts previously explained can be observed as well as the final results of both kinds of filters applied over the input images.

Another kind of static representations are the ones plotting the internal structure of the CNN. This visualizations are static since the architecture is defined in the moment in which the model is built and remains unchanged through the entire process. Before starting the network analysis it is important to know how the model has been built, and it is even more important to have in mind its structure if several models have been built as it happens in the grid-search method. Knowing both the internal structure and the total number of parameters trained makes possible to get an overall view of the model complexity that, together with the values of training and validation accuracy, enable to decide which architecture is the optimal one.

Related to the visualizations representing the internal architecture of the network, node-link diagrams as the one shown in Figure 3.9 (left) are the most common used technique, specially for fully-connected neural networks and, in the case of CNN, the classification part of the network that contains the fully-connected layers. According to [51], the node-link visualization enables to quickly appreciate that all neurons have been treated as identical processor units choosing an arrangement of neurons defined by a simple graph. This simplicity, just using circles for the neurons and lines for the connections between them, is what makes the representation so powerful for a better understanding of how the model works.

Nevertheless, as modern implementations of neural networks make use of more complex architectures (that is the case of the CNN), the graph-like visualizations become insufficient to cover all the details presented in these models. This can be better observed in Figure 3.9 (right) which illustrates how convolutional layers can be interpreted as graph-like connections and, as [51] states, it would be difficult to mentally assimilate the structure of a CNN with a classic node-link visualization. That is the reason why the most common way to represent a CNN nowadays is to use a solid block replacing each layer of nodes (neurons) and a single arrow pointing from one layer to the next or edges connecting the outskirts of the layers to represent dense connections, as it is shown in Figure 3.10.

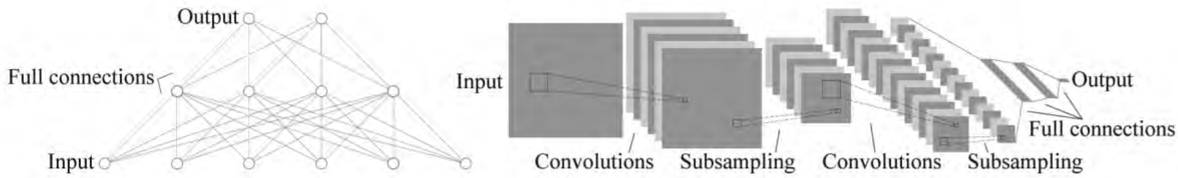

Figure 3.9. Typical node-link diagrams of fully-connected (left) and convolutional (right) neural networks [51].

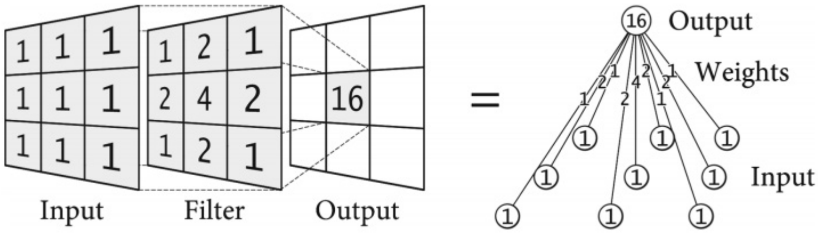

Figure 3.10. A 3 × 3 convolution filter is equivalent to a node with nine weighted connections, where the filter values correspond to the weights [51].

In this thesis, both filter visualization and internal architecture visualizations are used to study the CNN. On the one hand, filter representation helps to figure out which kind of features the network is extracting. This can be managed by visualizing the filter values so it can be identified which parts of the image the filter is focused on, mapping the filter to the input image. On the other hand, the representation of the network structure enables to figure out how is workflow of the learning process: which actions are performed at each step, how many parameters are generated and the number of values one layer takes as input and outputs to the next one. In this way, the actions performed at each step can be controlled.

3.2.1.2. Dynamic visualizations

One of the most common dynamic visualizations is the activation map, a simple technique to represent what neurons output for each layer. The location of the highest outputs correspond to the parts of the image that activate the network. They are also known as feature maps, since a high activation means that a feature was found in the image. They are widely used, and their dimensions are especially referred to when representing the whole architecture of a CNN, since they are the output from one layer and the input for the next one.

Figure 3.11. Node-link 2D diagram of a CNN trained to recognize handwritten digits. On the left there is a drawing path whereby the user can interact with the network changing its input. Layers from the architecture can be also hidden or shown through the layer visibility panel. The activation level of each node is encoded in hue and brightness [51].

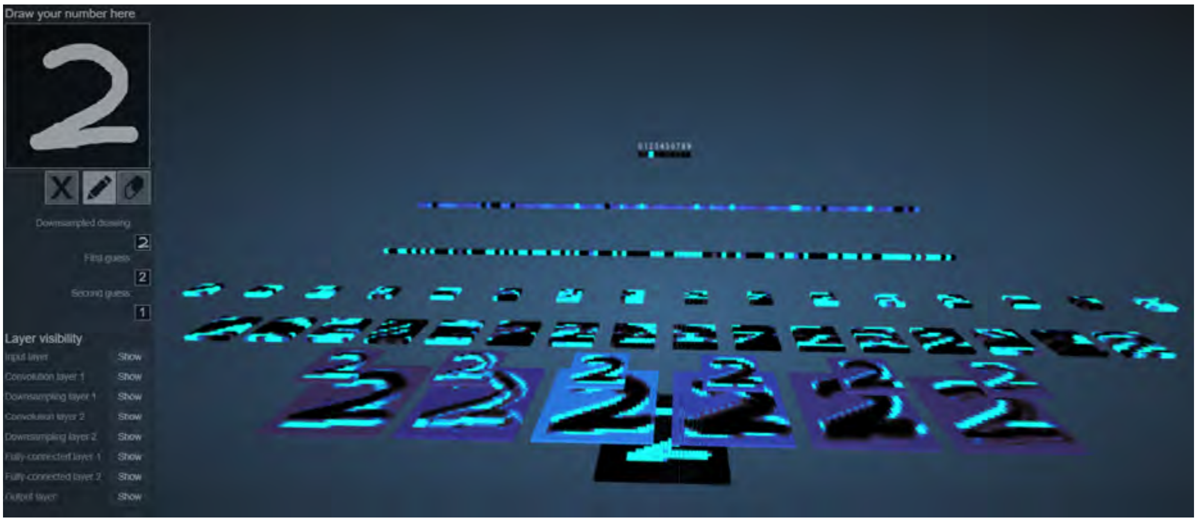

Figure 3.12. Node-link 3D diagram of a CNN trained to recognize handwritten digits. On the left there is a drawing path whereby the user can interact with the network changing its input. Layers from the architecture can be also hidden or shown through the layer visibility panel. The activation level of each node is encoded in hue and brightness [51].

Although block representation seems to be a good improvement, as it has been previously explained in section 3.2.1.1, it is still not enough to fully understand and interpret the way a CNN learns. That is why [51] presents an interactive node-link visualization of neural networks, which, apart from showing connections between neurons it also shows the activation patterns of the network, or what is the same, the activation maps. That is one of the reasons why this tool is considered as a dynamic visualization. This article has already been mentioned in the Chapter 2 of the thesis, since it is a powerful tool to understand the functioning of a CNN.

Figures 3.11 and 3.12 show the tool in its 2D and 3D version respectively. It can be observed that the tool presents a fixed CNN architecture with, in addition to the input and output layers, two convolutional layers (the first one is composed of six filters whereas the second one contains sixteen), two pooling layers and two fully-connected layers. The visualization shows the activation maps for each convolutional layer and pooling layer. For fully connected layers the activation maps are also shown but they are flattened into a vector as this is the way in which the neurons are arranged.

As it has been previously explained in this chapter, activation maps or feature maps represent the value each neuron is outputting as a result of the convolution operation with different nuances. These representations are a useful way of understanding what features the CNN is learning or how it is making decisions. They are widely used in the literature as in [37], [87] and [88] among others. For example, in Figure 3.11 it can be appreciated which features are being extracted by some of the filters in the first convolutional layer. Starting from the right, the first filter seems to extract horizontal borders, the second one vertical borders whereas the third and the fourth are focused on diagonal borders, from upper-left to bottom-right and from upper-right to bottom-left respectively.

One of the most powerful tool functionalities is the one that enables the user to modify the input image and see instantly how the activation maps change as well as the output class (how the network responds to different input images). For example, in the case shown in Figure 3.11 it is interesting to try finding out which parts of the image are essential for the network to recognise a number two. In this way the user can experiment "erasing" some parts of the image as it appears in Figure 3.13, where the bottom line of the number 2 has been deleted and the network now recognises the number 7 instead of 2. This gives the idea that the bottom part is essential to recognise the number 2.

Figure 3.13. Node-link 2D diagram of a CNN trained to recognize handwritten digits. When the bottom part of the number 2 is erased the network classifies it as a number 7 [51].

Hovering over the activation maps from the different layers, connections between neurons are shown, and clicking over a pixel the user can observe the calculations each neuron applies as well as the exact value that outputs. In convolutional layers, it also appears the filter that specific layer is applying so, it can be said that this tool contains a mixture of both static (filters) and dynamic visualizations (feature maps). It is classified as *dynamic* since the core of the visualization lies on the activation maps (as they are, in fact, the ones representing the internal structure of the CNN), and the filters representation is treated just as additional information. In Figure 3.14 a series of images from different tool functionalities showing extra information about the network can be observed.

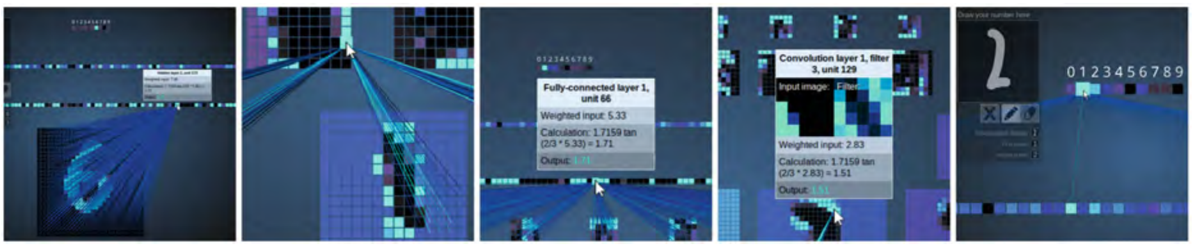

Figure 3.14. Screenshots from the visualization. From left to right, the images show: edges revealed by hovering on a fully-connected node, edges revealed by hovering on a convolutional node, details revealed by clicking on a fully-connected node, details revealed by clicking on a convolutional node, and finally the top layer activations of a network given ambiguous input [51].

The problem with this visualization technique regarding CNN interpretability is that the tool is focused on a very specific model, since it only presents a possible static architecture and it does not allow to experiment changing that architecture so that the user can observe how each hyperparameter influences the network performance. Along this line, [37] introduces a novel and more standard visualization technique called Multi-layered Deconvolutional Network or *Deconvnet*.

The aim of the deconvnet is to project the features extracted from the images by a regular CNN back to the input pixel space. To do so, the architecture of the deconvnet is made up of the same components as a CNN but in reverse: unpooling (an approximate inverse of max-pooling operation), rectification (same ReLU operation) and filtering (transposed versions of the same filters learnt by the original CNN applied to the rectified maps). Figure 3.15 shows a the typical architecture of a CNN and the same architecture but for a Deconvnet, where the latter will reconstruct an approximate version of the features projected over the original input image [37].

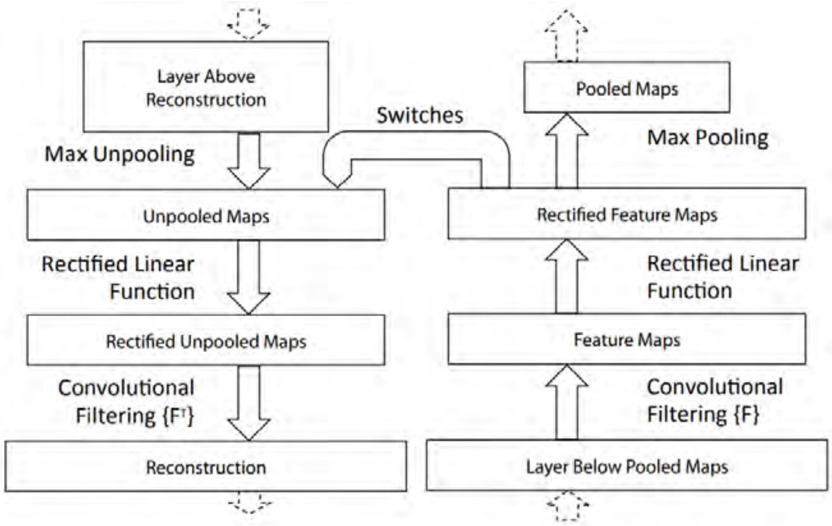

Figure 3.15. A deconvnet layer (left) attached to a convnet layer (right) [37].

In this way, the feature maps outputted by each convolutional layer can be projected down to pixel space enabling to interpret which features are extracted in each layer. Figure 3.16 shows the paper visualization of features extracted by layers from the first to fith one, as well as Figure 2.20 shown in Chapter 2 contains a visualization of features only in the second and fourth layer. With this visualization, [37] proves that the feature extraction goes from less to more specific features as layers are deeper in the architecture, showing the hierarchical nature of the CNN showing that:

- Layer 2: extracts corners, edges and colour.
- Layer 3: focuses on textures, mesh patterns and text.
- Layer 4: obtains more particular and class-specific features like dog faces or bird's legs with great variance depending on the input image.
- Layer 5: extracts entire objects with significant pose variation.

Deconvnets visualizations have, in fact, a more generic nature than the tool presented by [51], since they can be build from any trained CNN, whereas the interactive tool presents a static architecture.

Based on the deconvnet, [37] presents a couple of additional experiments. The first one studies the feature evolution during training, finding out that lower layers from the model only

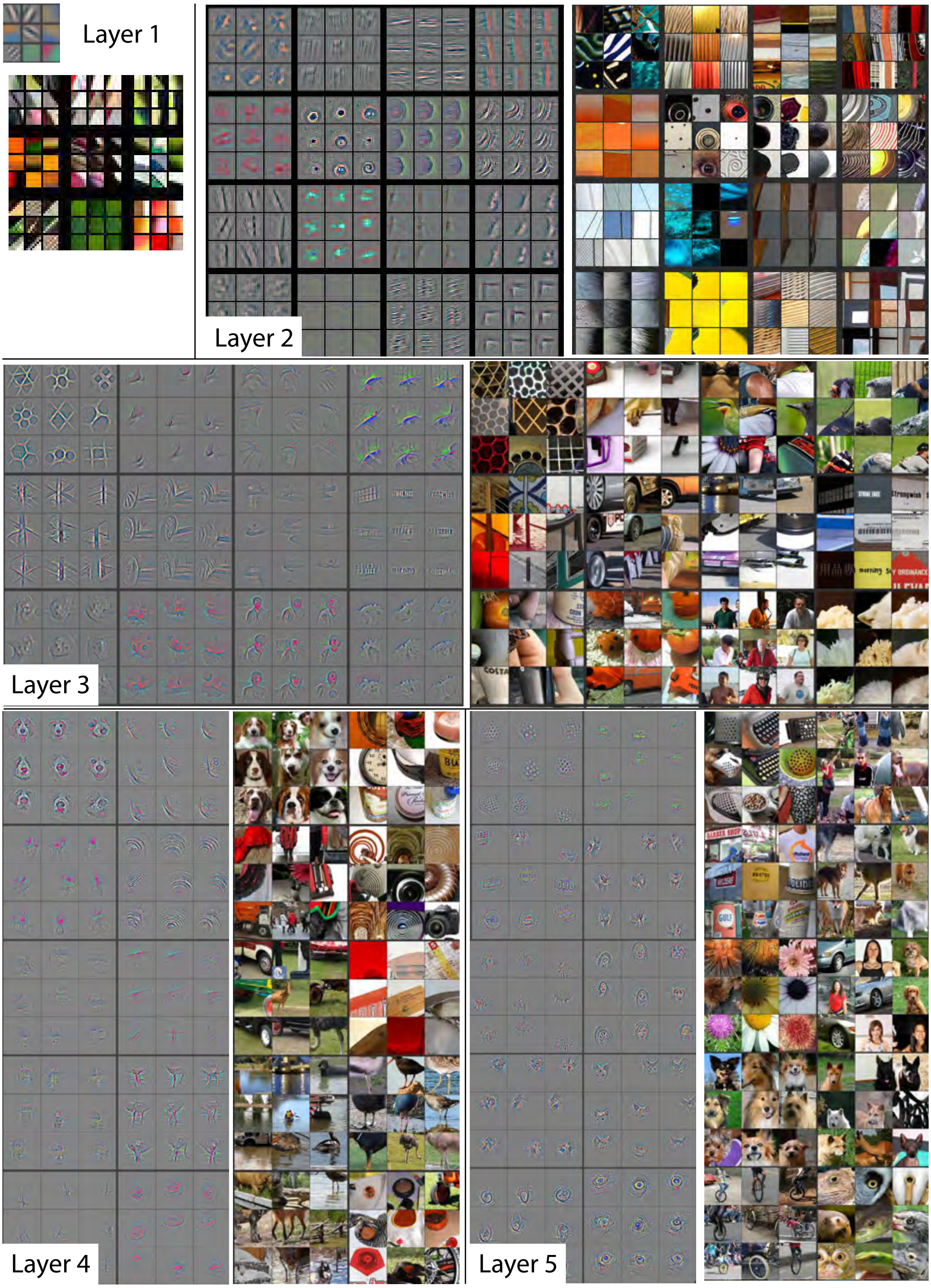

Figure 3.16. Visualization of features in a fully trained model. For layers 2-5 the top 9 activations in a random subset of feature maps across the validation data are shown, projected down to pixel space using deconvolutional network approach. For each feature map the corresponding image patches are also shown [37].

take a few epochs to converge whereas upper layers need a high amount of epochs to develop. This fact can be taken also as an indicative that first layers obtain more basic filters that are "easy" to learn, as the CNN quickly obtains them, while the more complex filters are the ones in the upper layers and the network needs more time to learnt them.

The second experiment analyses the feature invariance to different transformations such as scaling, rotation and translation. The conclusions obtained are that small transformations have a high impact on the first layer but not on the last layer. In addition, the network output is stable to translation and scaling, as well as rotation of objects with rotational symmetry, being variant to the rotation of the rest of the objects.

The networks studied by [37] have a more complex architecture and they are trained with images from the ImageNet database, what makes difficult to control and analyse the CNN learning process.

Another experiment carried out using feature maps is the correspondence analysis, also from [37]. A mechanism is developed for establishing correspondence between specific object parts in different images. First of all, certain parts from the input images are occluded using a gray square. After that, occluded images feature maps in layers 5 and 7 are subtracted from original images feature maps in the same layers, giving as a result an array with an specific feature (eyes or mouth) or a random one. This operation is repeated for different images with the same part occluded and the Hamming distance is calculated between different pairs of vectors. Finally, the mean of these distances can be used as an indicator of the consistency between the same object part through different images. Figure 3.17 shows several images used for this experiment. In [37] the object parts studied were right eye, left eye, nose and random parts from dogs with frontal pose. The table attach to the image shows that the lower scores are obtained for the eyes and nose, indicating that the model implicitly establishes some form of correspondence of parts among different images.

Occlusion Location	Mean Feature Sign Change Layer 5	Mean Feature Sign Change Layer 7
Right Eye	0.067 ± 0.007	0.069 ± 0.015
Left Eye	0.069 ± 0.007	0.068 ± 0.013
Nose	0.079 ± 0.017	0.069 ± 0.011
Random	0.107 ± 0.017	0.073 ± 0.014

Figure 3.17. Images used for correspondence experiments: original image in column 1, occlusion of the right eye, left eye, and nose in columns 2, 3 and 4 respectively and examples of random occlusions in other columns (left). Measure of correspondence for different object parts in 5 different dog images (right).

[88] makes use of a technique that is also used in the approach presented in this thesis: simplifying input images. In this way, images are modified so that they keep as little visual information as possible while still having the key objects to classifies them as a specific category. The difference between this methodology and the one used in the thesis is that [88] takes original complex images and tries to simplify them, whereas in the present dissertation, the complexity of the images is totally controlled as they are generated synthetically so the fact of increasing/decreasing complexity is straightforward.

Moreover, [88] makes use of the methodology explained before in Figure 3.17 which focus on occluding certain parts of the images, feeding them into the nework and record the change in activation compared to using the original images. A noticeable difference between both images will be an indicator that the given patch is occluding an important part of the image. This approach is used to identify the receptive fields, defined as the regions in the input space in which a particular feature appears. In this way, important regions from the image can be located for each convolutional layer, and following the activation of those locations through the network enables [88] to identify exactly which regions of the image lead to high unit activations.

Another very interesting approach to visualize the learning process of a CNN is presented by [89]. In this case, instead of being focused on convolutional layers, the representation displays activations of neurons in the first fully connected layer using a video format. Figure 3.18 shows a few frames obtained from the visualization video, in which it can be seen the neuron activity after every epoch. In particular, the visualization shows training data, testing data and both training and testing accuracy per epoch.

Training and testing data is plot almost in the same way, using UMAP to reduce neuron activation data to two dimensions so it can be displayed along Cartesian axes. The only difference is that training data is plot with dots whereas the testing data shows the images on the plot points, with a red border if the sample is misclassified. In both cases, each sample is displayed with a certain colour, corresponding to the one that has been assigned to the class it belongs to. Finally, a line chart shows the training and testing accuracy evolution through each epoch.

In Figure 3.18 it can be observed how as the number of epochs increases, the more separated the classes are in the space. This effect is due to the fact that the network is learning class-specific features that enables it to classify the images. The more separated the classes are, the easier is for the CNN to classify the input images making less mistakes and obtaining higher accuracies. Moreover, it is important to highlight the fact that neural networks are algorithms are known as search algorithms. This means that their aim is to look for an optimal solution in the input space and two networks with different architectures or even with the same architecture but distinct seeds can achieve both optimal solutions that are totally different. In other words, this visualization will change with each model unless two models have the same architecture and initialization seed, but that does not mean that different models cannot achieved equally good solutions.

This visualization tool is designed specially for people with no or limited knowledge about artificial neural networks. That is one of the reasons why the video format is such an useful technique, as the complete training process can be detailed observed, step by step, instead of just showing the final results (as it happens with most of the techniques previously explained), once the CNN has already been trained.

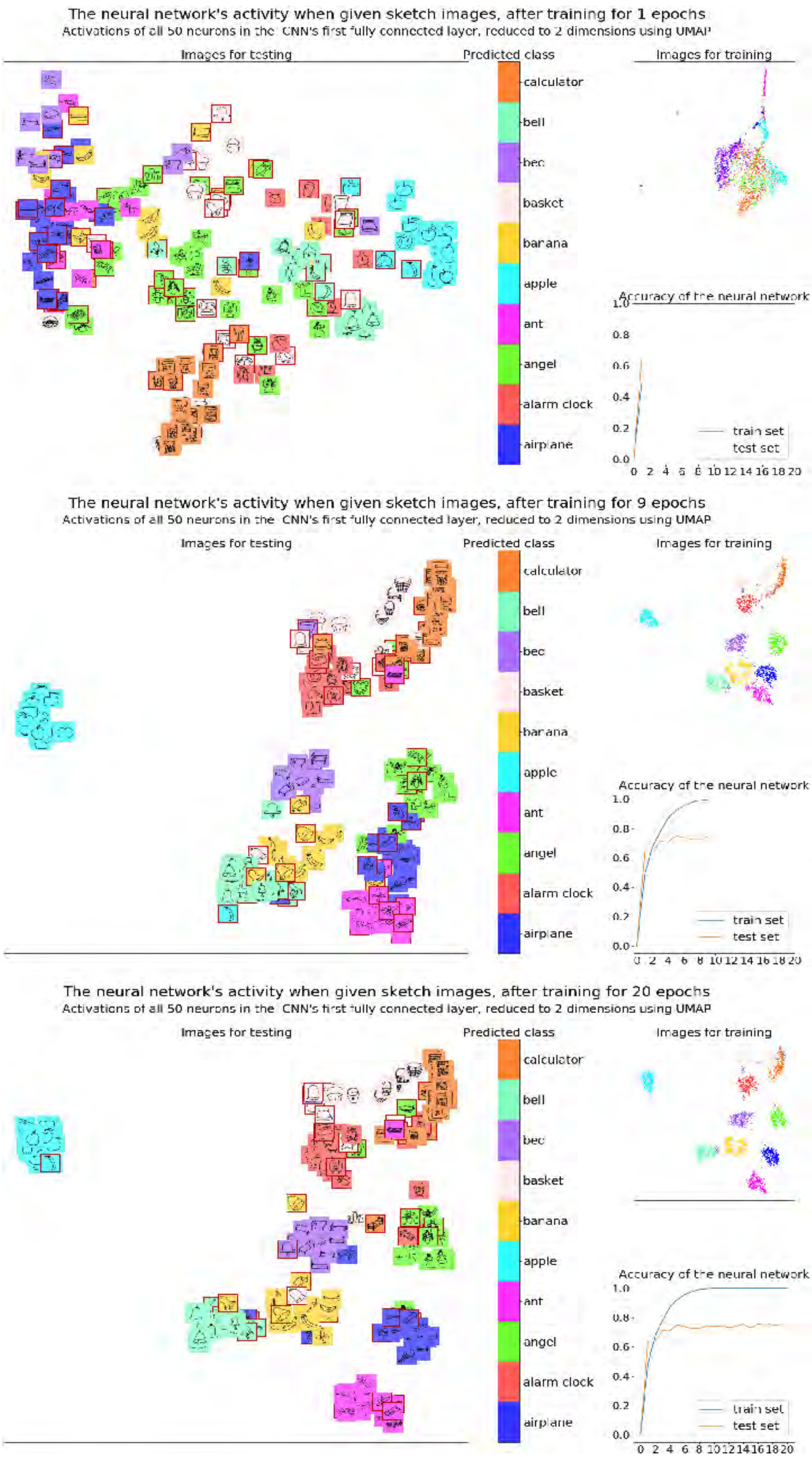

Figure 3.18. Visualization frames of epochs 1, 9 and 20 with epoch 9 as the supposed start of overfitting [89].

3.2.2. Output visualization

The techniques included in this section aim to visualize which parts of an image the CNN is focusing on in order to classify it. Hence, output visualization can be very useful, enabling to understand how the network is making decisions. In addition, it can be helpful to understand if the neural network is looking at appropriate parts of the image or if, on the contrary, the CNN is cheating, which can usually happen if the dataset has some kind of bias in the images.

For example, as stated in [90] based on the work of [91], although the aim of a CNN trained to identify chest illnesses is to be looking for actual visual signs of a disease, this cannot always be true. In the case of chest X-ray images if the patient is laying down, a metallic "L" token is placed directly on the patient's shoulder. This situation only happens if the patient is too weak to stand, so in the rest of the cases, the patient will stand and it would not be necessary to place the "L" token. A CNN can classify one of this X-ray images as "high probability of illness" not on the actual appearance of the disease but instead on the metallic token, if there are enough samples of images marked as "illness" and where the patient is lying. The CNN will have learnt the correlation between "metallic L on shoulder" and "patient too sick to stand", so, in the case of a patient standing the network could output "low probability of illness", even though the patient may be, in fact, ill. Therefore, the CNN will be looking for metal tokens, deviating from its initial purpose. This example ratifies the importance of output visualization when working with CNN.

One of the most commonly used visualizations are the heat maps. A heat map shows the magnitude of a phenomenon as colour in two dimensions, being the variation in colour by hue or intensity. There are two kinds of heat maps: the cluster heat map and the spatial heat map. The first one makes use of a matrix layout of fixed cell size whose rows and columns are discrete phenomena and categories whereas in a spatial heat map there is no notion of cells as the phenomenon is considered to vary continuously [92]. Figure 3.19 shows both visualizations so this difference can be better appreciated.

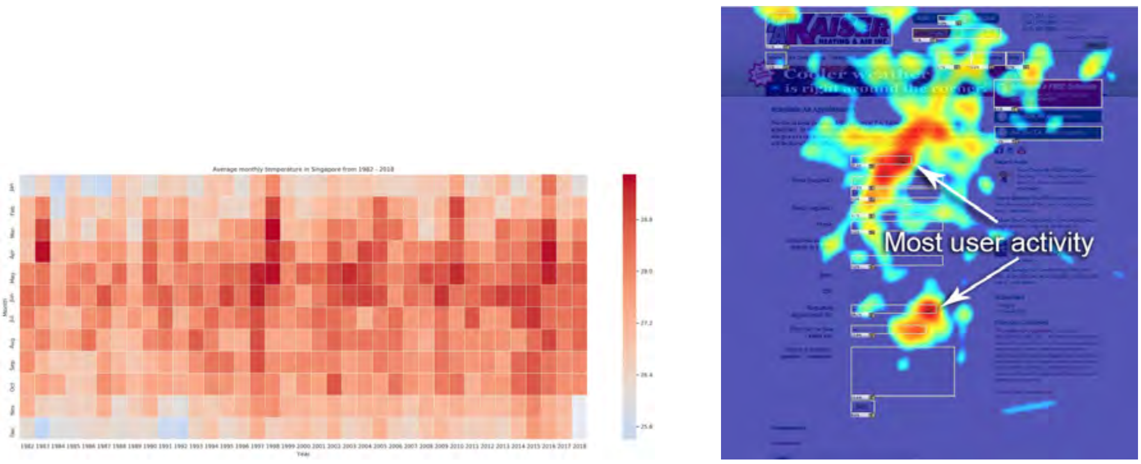

Figure 3.19. Cluster heat map (left) plotting the average monthly temperature in Singapore from 1982 to 2018 [93] and Spatial heat map (right) to represent a visual record of how users interact with every element on a website [94].

Two examples of heat maps applied to CNN are showed in Figure 3.20 where there are two equal images but, in the first one, the dog's mouth and cheeks are highlighted in red and yellow for the class *dog* whereas in the second one, the cat's face and body is highlighted for the class *cat*. If the CNN has been trained to classify between the categories *dog* and *no dog*, the output in this case should be similar to the first image. If the CNN has been trained to classify between the categories *cat* and *no cat*, its output should be similar to the second image. Finally, if the CNN is trained to distinguish between dogs and cats, the training dataset should include images of only dogs (labelled as the class *dog*) and of only cats (labelled as the class *cat*).

Figure 3.20. Heat maps over the output for two different CNN models: output of a CNN that has been trained to identify dogs (left) and output of a CNN that has been trained to identify cats (right) [95].

Once the concepts of activation map and heat map have been explained, it is important to highlight the difference between them. Although both can be classified as what this dissertation has named *dynamic* visualization techniques, activation maps plot the results of applying each one of the filters to the input image in a disaggregated way, whereas heat maps overlap all the output activated neurons together over the input image, in an aggregated way.

There are several papers on CNN heat map visualization. [96] proposes Class Activation Mapping (CAM) a technique that makes use of an specific kind of CNN composed of several convolutional layers over the last of which a Global Average Pooling (GAP) is applied (explained in subsection 2.1.2) and finally a fully-connected layer connected that outputs classification results. Figure 3.21 shows the architecture of the network.

In this case, the score that the network outputs for the class *Australian terrier* is calculated as shown in equation 3.1, being y_r, y_b and y_n the output for the neurons shown in the architecture or, what it is the same, the values that have been obtained after performing the GAP pooling over the feature maps A_1, A_2 and A_n).

$$y_{AustralianTerrier} = w_1 y_r + w_2 y_b + ... + w_n y_n \tag{3.1}$$

And in order to obtain the CAM heat map visualization the next calculation needs to be performed as in equation 3.2:

$$CAM^{out} = w_1 A_1 + w_2 A_2 + ... + w_n A_n \tag{3.2}$$

The difference between the two calculations is that instead of multiplying the weights w_1, w_2 ... w_n by individual results produced by GAP operation on feature maps A_k, these weights are multiplied by the feature maps directly [90]. In this way, instead of obtaining a single score corresponding to the class being activated (*Australian terrier*), a matrix of numbers is obtained, which corresponds to the actual heat map.

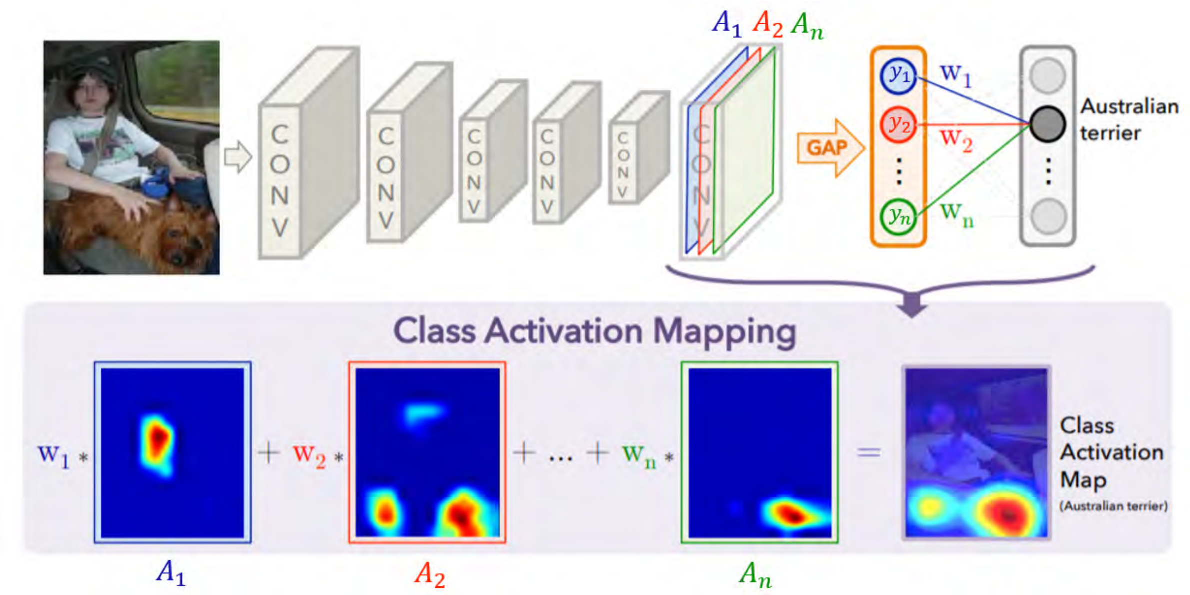

Figure 3.21. Class Activation Mapping architecture: the predicted class score is mapped back to the previous convolutional layer to generate the class activation maps (CAMs). The CAM highlights the class-specific discriminative regions. Modified from [96].

As a generalization of CAM algorithm, Grad-CAM is developed afterwards. Grad-CAM does not require a particular CNN architecture, as opposed to CAM which requires an architecture that applies global average pooling (GAP) to the final convolutional feature maps followed by a single fully connected layer that produces the predictions [97] as it has been explained previously in this document.

The main difference between CAM and Grad-CAM is the way in which the feature maps A_1, A_2 and A_n are weighted to make the final heat map. While in CAM these feature maps are weighted using the existing weights in the last fully-connected layer of the network, in Grad-CAM the feature maps are weighted using values calculated based on gradients (hence the name "Grad" that stands for Gradient). Therefore, Grad-CAM does not need an specific architecture since gradients can be calculated through any kind of neural network layer [97].

[37] uses heat maps to perform an occlusion sensitivity analysis over a trained network. The aim of this experiment is to find out if the network is truly identifying the location of the object in the image, or if it is just using the surrounding context. In order to do this, a gray square is placed over and moved to occlude sequential parts of the image while monitoring the output class given by the network. Figure 3.22 shows several visualizations of this experiment. For example, for the image belonging to the class *Pomeranian*. When the gray square covers the dog's face, it can be observed a sharp drop in both the strongest feature map activation (b) and in the probability of the correct class (e), meaning that the CNN is actually identifying objects within the scene. The same effect is observed for the classes *Car Wheel* and *Afghan Hound*.

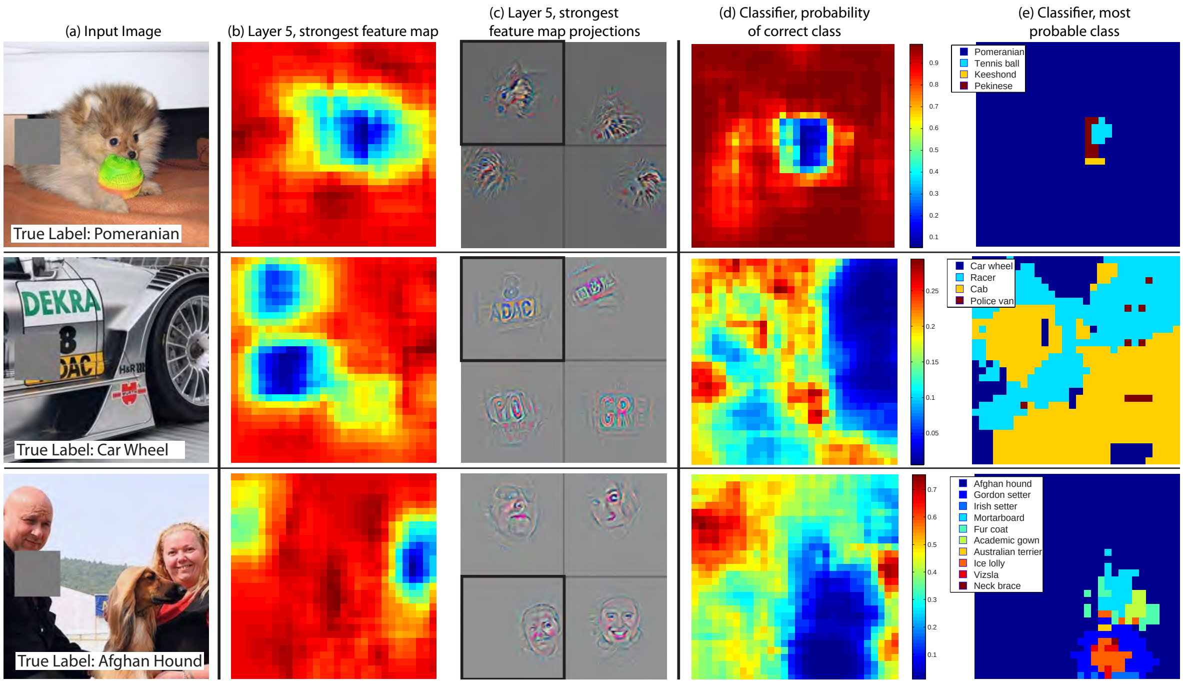

Figure 3.22. Three test examples where different portions of the scene are systematically covered up with a gray square (1st column) and the top (layer 5) feature maps ((b) and (c)) and classifier output ((d) and (e)) changes. (b): for each position of the gray scale the total activation in one layer 5 feature map is recorded (the one with the strongest response in the unoccluded image). (c): a visualization of this feature map projected down into the input image (black square), along with visualizations of this map from other images. The first row example shows the strongest feature to be the dog's face. When this is covered-up the activity in the feature map decreases (blue area in (b)). (d): a map of correct class probability, as a function of the position of the gray square. E.g. when the dog's face is obscured, the probability for *Pomeranian* drops significantly. (e): the most probable label as a function of occluder position [37].

Another technique commonly used to understand what a CNN is looking at during classification is guided backpropagation, also known as guided saliency. A saliency map is any visualization of an image (not limited to neural networks) in which the most important pixels are highlighted [98]. They are calculated after a network has finished training finding class score derivatives with respect to a certain image, taking the absolute value of each element of the the derivative matrix and plotting the matrix as if it is an image. This technique was fist introduced by [99] and is useful to detect the approximate location of an object in an image. Figure 3.23 shows three examples of images belonging to three different classes (snake, dog and spider) and its saliency maps both in colour and grayscale [98].

A more complex methodology to find out which parts of the image the model is paying more attention to is presented in [100]. The paper presents a new framework called Explainable Attribute-based Multi-task (EAT) whose overall operation is based on generating attribute-based textual explanations for the network and ground them to the attributes on the image to show visual explanations. With this framework, the model attention of the network os also increased, so that the model focus its effort in learning the discriminative area of the image, ignoring the background and making training more efficient. The framework pipeline is shown in Figure 3.24 in which the reader can understand better the logic behind the method, without going into further detail.

Finally, a detailed and complete compilation of currently visualization techniques for interpreting CNN is included in [101] which, apart from the references explained in this chapter, goes over other methods regarding diagnosis of CNN representations or evaluation metrics for network interpretability.

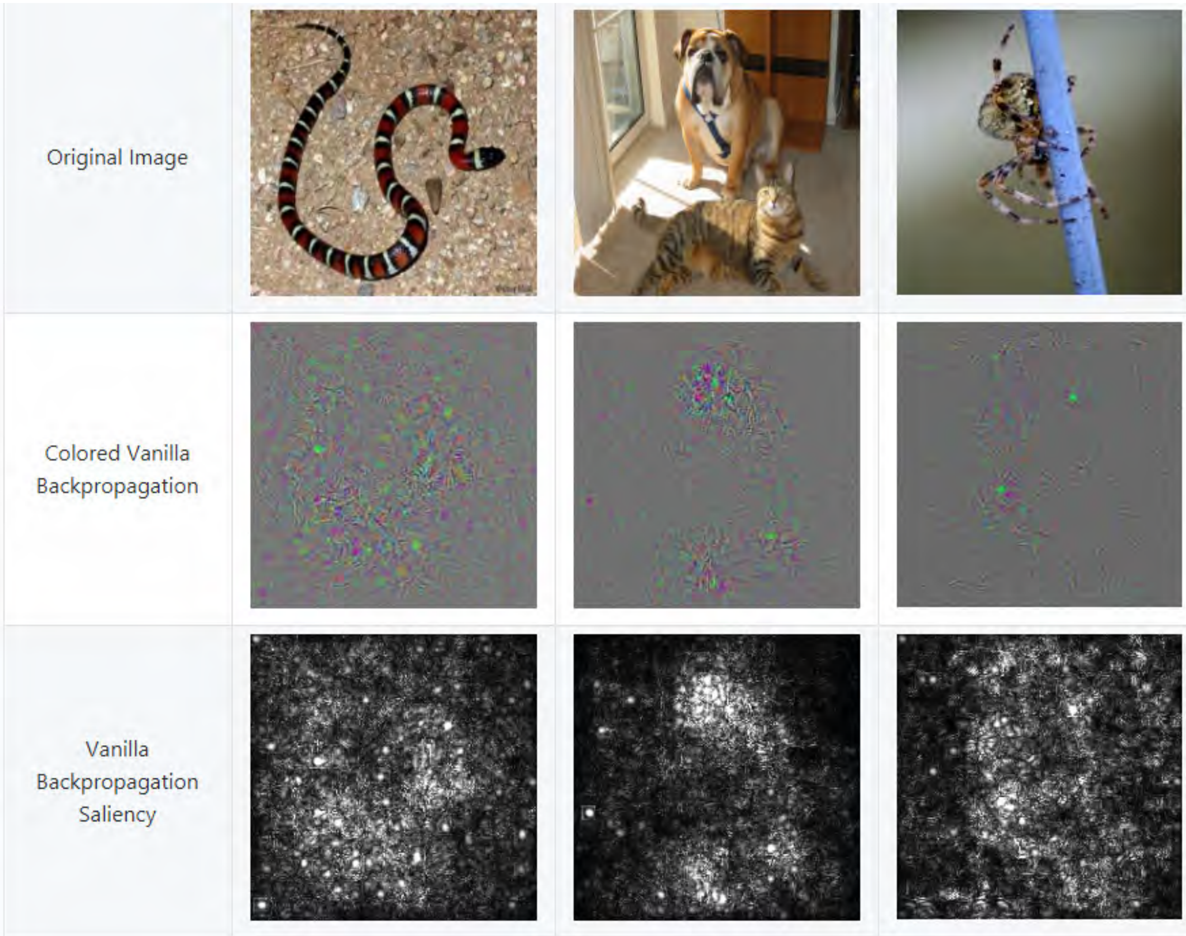

Figure 3.23. *Colored Vanilla Backpropagation* means a saliency map created with RGB colon channels. *Vanilla Backpropagation Saliency* is the result of converting the *Coloured Vanilla Backpropagation* image into a grayscale image [98]. Modified from [102].

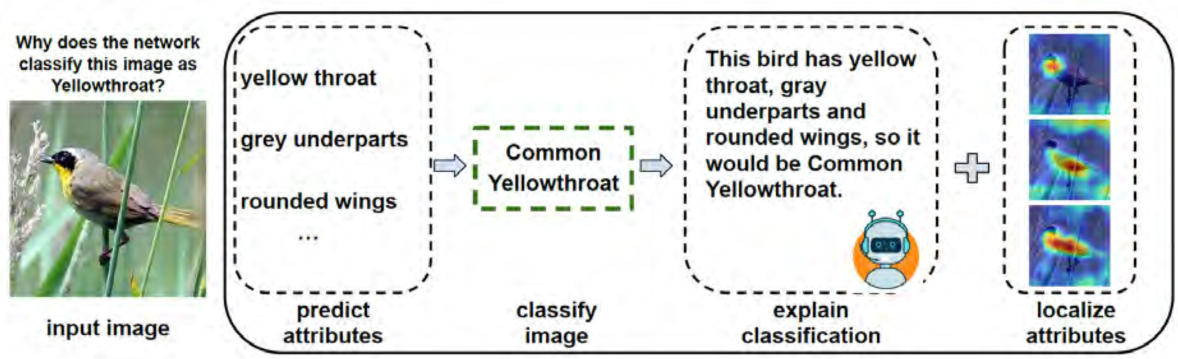

Figure 3.24. Taking an image as input, the first task is to predict the attributes of the foreground object, then classify image category according to the attributes. By calculating the attribute contribution to the classification result, it can be concluded which attributes influence the network decision [100].

Taking into a account the brief literature review included in this point, it can be observed that the problem of the CNN interpretability has been addressed from a very complicated perspective, trying to understand the learning process of very complex architectures comprised by several layers and trained with datasets containing real images. The key of this thesis is to analyse the problem with a much more simpler approach, using a controlled environment in which the minimum architecture is found and the dataset used for training and validation steps is synthetic. The idea of generate a dataset synthetically enables to study the training process with great detail, analysing and explaining step by step since the problem itself is more basic than the ones presented in the existing literature.

Due to its purpose, this thesis can be included in the recently born field that is known as Explainable Artificial Intelligence or XAI, which aims to explain how algorithms reach their conclusions helping to train the models in correctly avoiding any kind of bias, among other advantages as it is studied in [103].

4

Experiments and interpretation of results

See now the power of truth;
the same experiment which at first glance
seemed to show one thing,
when more carefully examined,
assures us of the contrary.
Galileo Galilei (1564–1642)

This chapter contains a detailed description of all the work that has been carried out during the research. It begins with an introduction exposing the overall methodology followed to perform the different experiments together with some other general common aspects. After that, each experiment is explained exhaustively including its three main sequential steps: dataset generation, minimum model search and interpretation of the results together with the model, trying to understand the network's logic behind its learning process. Notice that the motivation behind an experiment is usually due to issues arisen in the previous one so the complete research has a common thread.

In order to test the alternate solution to analyse the CNN learning process proposed by this thesis, a series of experiments are conducted. Depending on the problem they are trying to address; and hence, their purpose: as well as the datasets used to perform them, experiments are grouped in four main sets. In this way, the following of the trial process is easier for the reader.

As it has been stated before, the same experiment can be applied over different datasets. This happens specially with the first two groups of tests. In the first one (section 4.1), just some models are trained to scan the hyperparameter space and select which datasets are more suitable to try first. Moreover, for the same reason, the first group of experiments does not comprise all the three main steps that are going to be explained down below, as it is understood as a preparatory group of tests. In the second group (section 4.2), more exhaustive tests are performed but the models solve the problem in a similar way, so the interpretation of both model and results is going to be alike. Groups three (section 4.3) and four (section 4.4), contain just one experiment over a single dataset, but they are prepared to include more experiments with different datasets of the same kind, as it will explained in section 5.2 from Chapter 5.

The three main phases comprising an experiment are described as follows (notice that some groups of experiments only cover until the second phase):

- **Dataset generation**. The first stage is to obtain the data that is going to be used to both train and validate the deep learning models. As one of the main pillars of this thesis, datasets are generated synthetically so its complexity can be controlled and therefore, the learning process. Datasets are generated via Matlab and they contain 6000 images approximately, half of them for training and the other half for validation. In addition, each of the dataset contains 1000 images of each one of the three classes, resulting in balanced datasets. The images generated contain geometric shapes of ellipses, rectangles and triangles, which are the three classes. Images are in grayscale with the shape in black on a white background, except for the datasets used in experiments belonging to groups four and three, that also contains images with the figures in white on a black background. Over these datasets different geometric transformations are applied, more precisely, scaling and rotation. In addition, the datasets that are first generated and analysed does not contain any noise, whereas the last ones contain gaussian noise added to the original image.

- **Minimum model search**: the minimum model, as it was previously defined in section 3.1 is the one that achieves the highest precision without increasing excessively its complexity. The approach used in this thesis to tune the hyperparameter and find the minimum model is approach known as grid search, focused on methodically building and evaluating a model for each combination of network parameters specified in a grid. In order to enable a properly comparison of the different models generated, their information such as filename, hyperparameter values, number of parameters and features extracted, as well as levels of training and validation accuracy are saved in files organized in a table structure. For each experiment, several tables are extracted, and they are shown in the following sections.

  In addition, it is important to take into account that two hyperparameters are common to all the network architectures designed: padding and stride for both convolutional and pooling layers (the meaning of all these concepts has been already explained in Chapter 2). For convolutional layers, padding is set to value “same” and stride to 1, which, according to Keras documentation [104], means that the convolutional layer’s outputs will have the same spatial dimensions as its inputs. So, zero-padding will be added to the input image until this is achieved. For pooling layers, the padding is set to “valid” value, which means that no padding is added whe performing pooling operation, and the stride is set by default to the pool size. When using “valid” padding option, the resulting output has a shape (number of rows or columns) that can be calculated with the formula shown in 4.1 [105]:

$$output\_shape = (input\_shape - pool\_size + 1)/stride) \tag{4.1}$$

- **Results and model interpretation**: Once the minimum model has been found, the analysis of how the model has learnt is carried out. To obtain as much information about the learning process as possible is important to study at the same time several factors: the classification results given by the model in terms of accuracy, both the filters and features learnt by the internal layers, feature maps to see where the model is looking at to perform the task, etc. Several different techniques are used to perform these analyses but in summary, some well-known ML techniques among other methodologies are used to try to gain more comprehension of the features learnt by the model and, on the other hand, common representations such as feature maps and variations of them are used to give physical meaning to the results obtained.

As an additional step that is only performed in groups of experiments 3 and 4:

- **Mistakes interpretation**. For models generated in these last two experiments, incorrectly classified images are analysed as well, since there are not a lot of cases where the model makes an error. This study enables to gain insight into which parts of the image are crucial to classify an image correctly and what values the model expects the features to have.

Once the reader has review some general common points related to the experiments, then the complete experiments together with the discussion of the results obtained are detailed.

4.1. Group 1: Preliminary tests

The set of experiments included in this group were the first ones to be carried out and the objective is twofold. On the one hand, their aim is to start exploring the input hyperparameters space, setting initial values to the grid and analysing how the models behave with those hyperparameters, i.e. establishing an starting point for the minimum model search from which it is possible to figure out if the hyperparameter values have to be increased or decreased. On the other hand, these experiments are helpful to determine which datasets are the best ones to start examining since initially several were generated but it was not entirely clear which ones were the more suitable to begin the tests. In this way, a decision based on the data can be made or, what it is the same, a data-driven decision.

Due to the exploratory purpose of these experiments, only until the second phase of the experiments is performed, and that is minimum model search. The results are not analysed in detailed since the models obtained were not the optimal ones. Almost the same tests are performed over all the datasets.

4.1.1. Datasets tested

Three datasets were tested initially in order to see which ones are more suitable to perform the first experiments: dataset 1.1, 1.3 and 2.1. Their main features are described down below:

- **Dataset 1.1**. It comprises geometric shapes centred with a variable size, without rotation nor deformation. More specifically, there are ellipses, rectangles and equilateral triangles. They are centred in the same point and are limited by the radio that circumscribes the figure, so that they have a reasonable size. The images have a size of 224x224 pixels and the dataset contains 3000 images for training and 3000 for validation, belonging 1000 to each class. For triangles, the centre is located in the coordinates $(0, 0)$, calculating the vertices as it shown in Figure 4.2. Figure 4.1 shows some examples of the images that can be found in this dataset. Note that all the images are in grayscale with the figure in black on a white background.

  It is important to highlight that dataset 1.2 was also generated, with similar images to the ones in dataset 1.1 but with a size of 28x28 pixels and including the double amount of images in validation. The radio in which the figures are circumscribed is limited between 0.1 and 0.9 units. In addition, the triangles where better centred in the image since before they were shifted up, placing the barycentre in the coordinate $(\frac{3}{4}, \frac{3}{4})$ instead of $(\frac{2}{3}, \frac{1}{3})$ as it can be observed in Figure 4.3 . Nevertheless, this dataset contains figures so small that neither a human nor the network is able to recognise as Figure 4.4 shows. That is the reason why this dataset was not used in the following experiments and the motivation to generate the next one: Dataset 1.3.

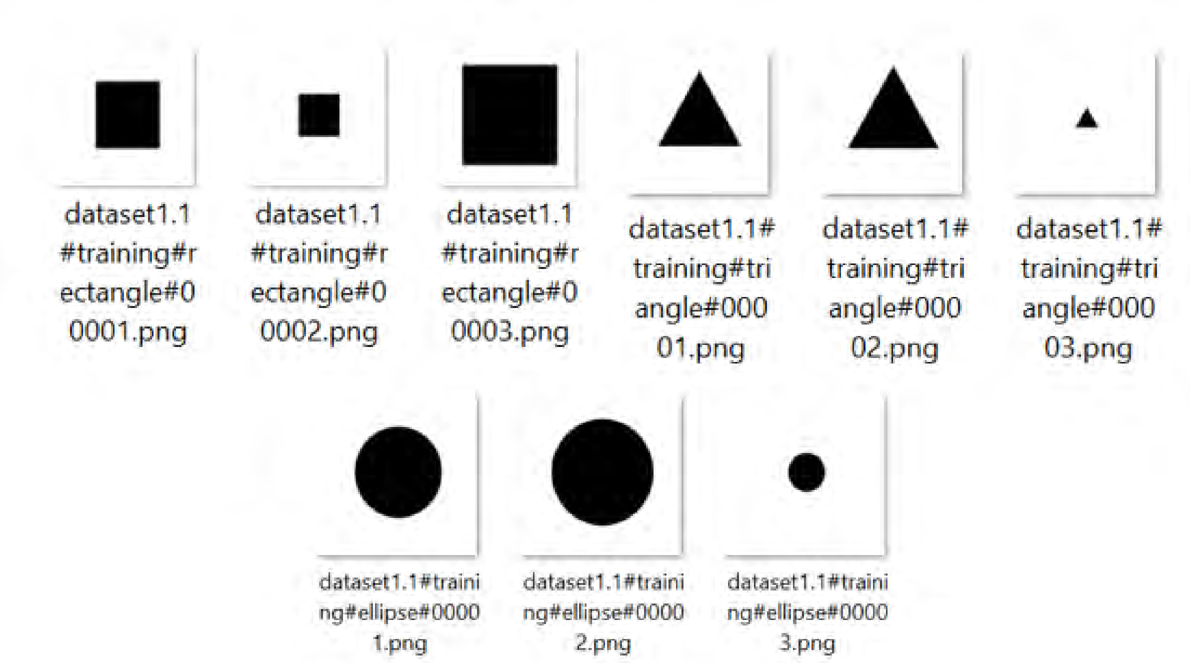

Figure 4.1. Examples of images from dataset 1.1 belonging to the three different classes.

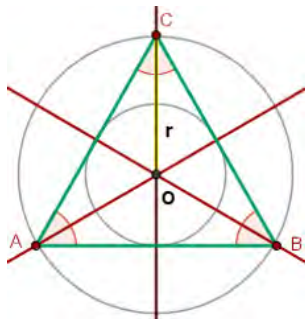

Figure 4.2. System to calculate triangle vertices giving the coordinates of its barycentre.

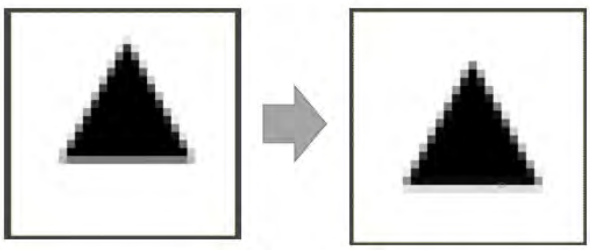

Figure 4.3. Example of an image with a triangle from dataset 1.1 (left) and the same triangle in dataset 1.2 (right).

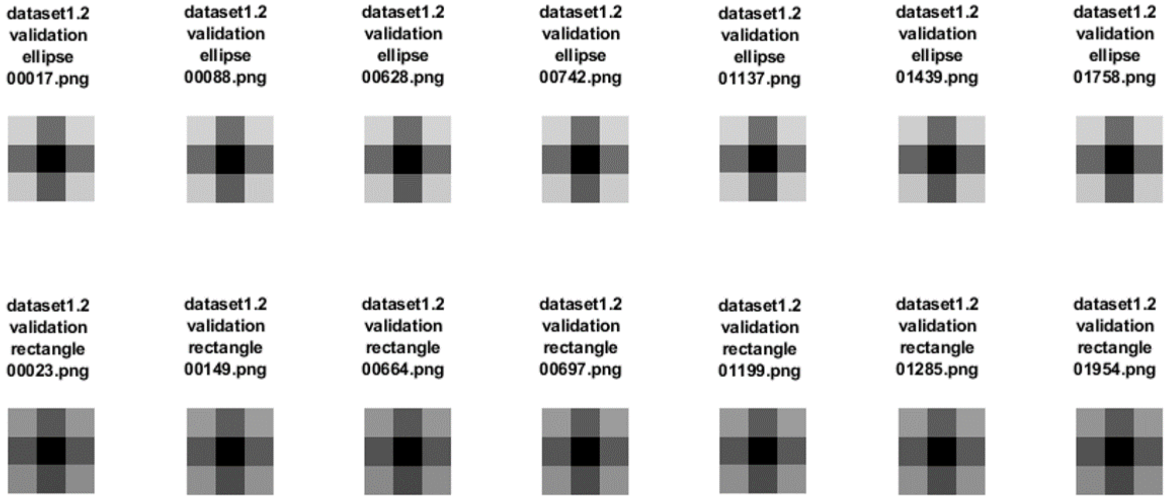

Figure 4.4. Several examples of images with the minimum size from dataset 1.2. The first row contains ellipses and the second row contains rectangles.

- **Dataset 1.3**. The images are similar to the ones in dataset 1.2 but with the difference that the minimum radio size is increased up to 0.3 so the figures are not so little that cannot be recognised. Figure 4.5 shows that now it is possible to recognise the figures. With this size, polygons can be differentiated perfectly.

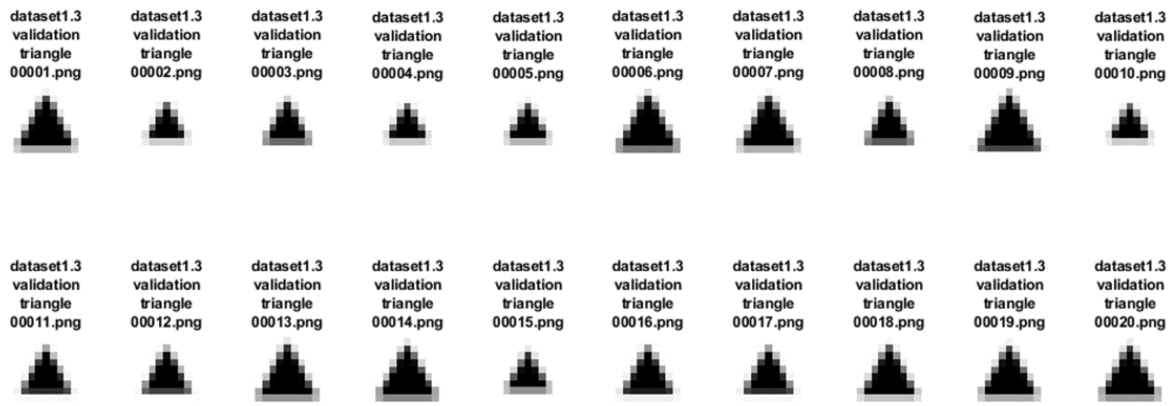

Figure 4.5. Several examples of triangles with different radio sizes between 0.3 and 0.5.

To sum up, this dataset contains 28x28 images with centred black figures on a white background and different sizes, varying the radio in which they are circumscribed from 0.3 to 0.9 units, so the scalability issue can be addressed. It contains 3000 images for training, 1000 belonging to each class and 6000 images for validation, 2000 of each class. Figure 4.6 shows some examples of images from the dataset. Notice that the quality is much lower in this case that it was in Figure 4.1 since the images in the first dataset had a better resolution: 224x224 against 28x28.

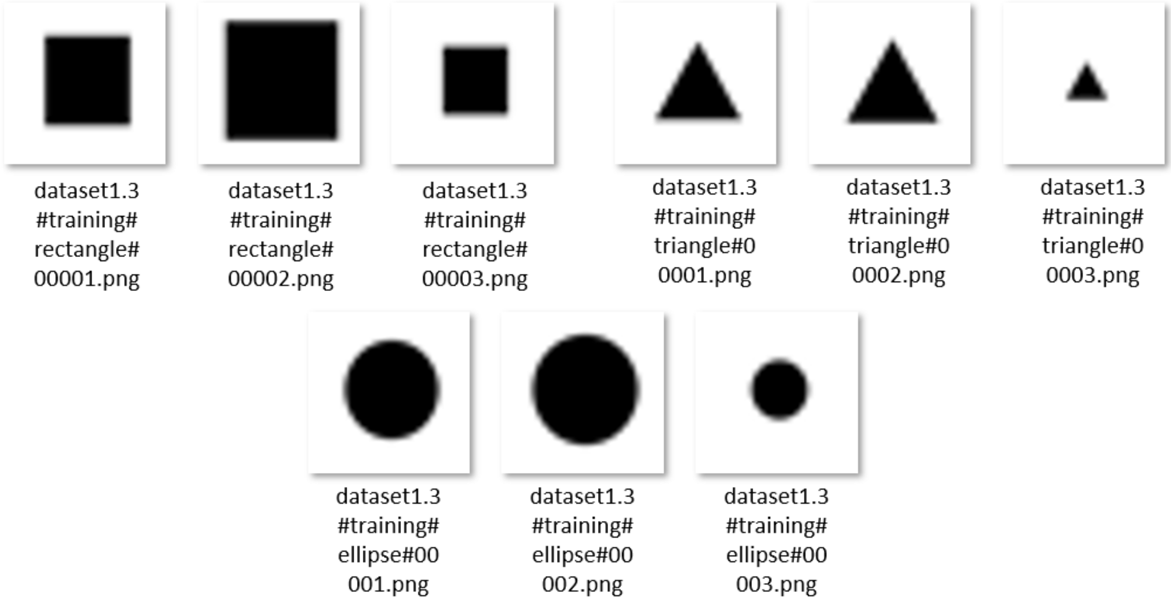

Figure 4.6. Examples of images from dataset 1.3 belonging to the three different classes.

- **Dataset 2.1**. It contains 28x28 images with centred geometric shapes keeping the size constant, with the radio equal to 0.7 but rotated a random amount of degrees (30°, 60°, 120°, etc). Figure 4.7 shows a sample of images extracted from this dataset. Notice that the rotation is performed over the figure's centroid, which in all cases is placed in the coordinate $(0, 0)$ except for the triangle, as it can be observed in Figure 4.8. The ellipse, as it is similar to a circle, is immutable to rotations. In this way, the rotation problem can be addressed. Similar to the dataset 1.3, figures are black on a white background, training dataset has 3000 images and validation dataset 6000 images, both of them with the three classes equally balanced.

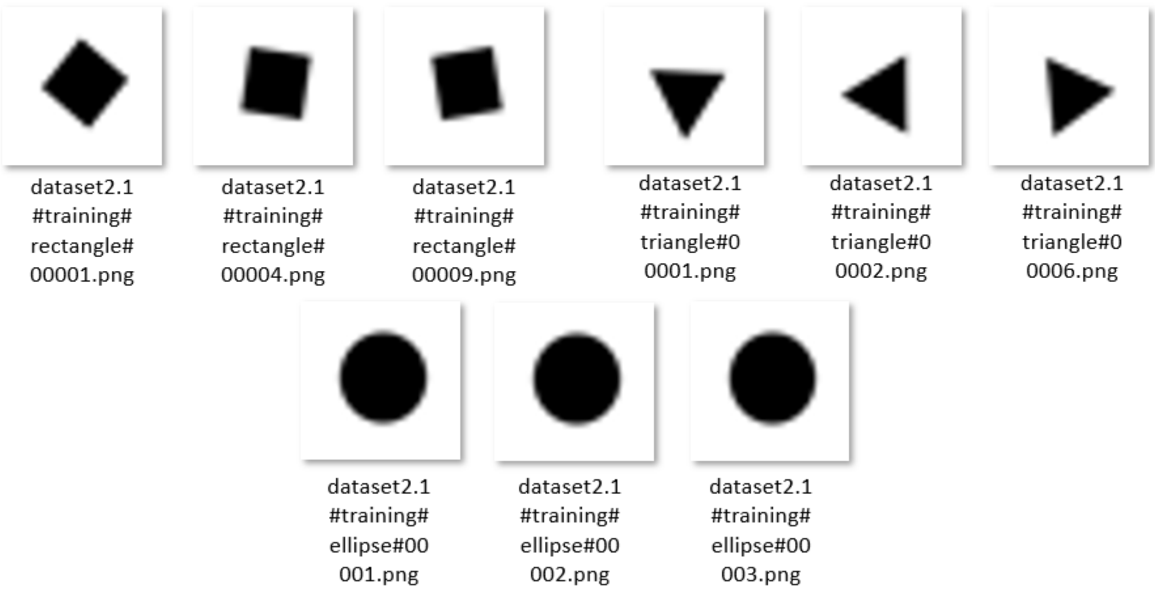

Figure 4.7. Examples of images from dataset 2.1 belonging to the three different classes.

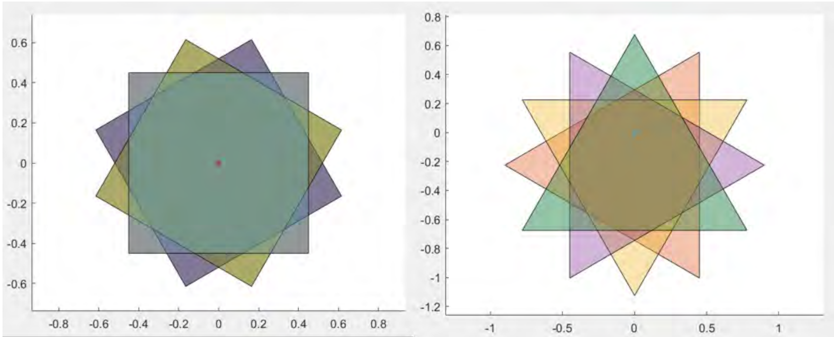

Figure 4.8. Rotation system for images in dataset 2.1.

4.1.2. Experiment 1.1

For the minimum model search, the first architectures proposed are comprised of a convolutional layer together with a ReLU layer as the activation function, pooling layer of size (2,2) and a flatten layer fully connected to a dense layer with three neurons (as there are three output classes), without any more fully connected layers between the flattening layer and the output nodes. Stride and padding values are the ones specified in the introduction of Chapter 4.

This architecture was considered one of the simplest ones to start training models. The hyperparameters being modified are the number of filters in the first (and only) convolutional layer and the size of these filters. For dataset 1.1, the number of filters takes the following values: 2, 4, 8, 16 and 32; and, for each one of these values, the kernel size varies among 3x3, 5x5 and 7x7. Each model is trained only during 10 epochs, in this way, it can be observed quicker if the model has a good tendency or not.

Results from these models are shown in Table 4.1. All models achieve a training and validation accuracy of one even with just a few epochs, since the magnitude of the number of parameters is of several thousands. The starting point for the model architectures is already very complex and the increase in the number of filters generates a marginal increase in the total number of parameters, hence, in the complexity. The classification problem has not such a high level of difficulty that it implies models with so many parameters, so, in this case, the number of filters must be reduced as well as its size.

For datasets 1.3 and 2.1 models with similar architectures are generated, though the hyperparameter representing the number of filters in the first convolutional layer takes the values 2, 4, 8 and 16, skipping 32. Tables 4.2 and 4.3 shows the results obtained from this grid search. Remember that images belonging to both of these datasets have a size of 28x28 pixels so the number of parameters is reduced significantly. Nevertheless, this reduction is not enough and models still have a high degree of complexity, the simpler one with one thousand parameters. Again, almost all models classify correctly 100% of images since they are oversized for the assigned task.

To sum up the results obtained from this experiment, although they all achieve results with high accuracy, the group of models generated are too complex, they all have a high amount of parameters, even the smallest architecture. So, in order to obtain the minimum model, grid values must be reduced so that the number of parameters reduces itself. Nevertheless, that number of parameters cannot be reduced just by modifying the hyperparameters from the convolutional layer, but it is necessary to add a pooling layer. This reason motivates the conduction of Experiment 1.2.

Model timestamp	nepochs	nfilters_1st	ksize_1st	number_param	tr_acc	val_acc
20200904T122040	10	2	3	75287	1	1
20200904T125728	10	2	5	75319	1	1
20200904T125946	10	2	7	75367	1	1
20200904T130205	10	4	3	150571	1	1
20200904T130422	10	4	5	150635	1	1
20200904T130641	10	4	7	150731	1	1
20200904T130859	10	8	3	301139	1	1
20200904T131116	10	8	5	301267	1	1
20200904T131331	10	8	7	301459	1	1
20200904T131548	10	16	3	602275	1	1
20200904T131803	10	16	5	602531	1	1
20200904T132018	10	16	7	602915	1	1
20200904T132235	10	32	3	1204547	1	1
20200904T132451	10	32	5	1205059	1	1
20200904T132709	10	32	7	1205827	1	1

Table 4.1. Model hyperparameters and results obtained after performing Experiment 1.1 over dataset 1.1. *Model timestamp* is the date and hour when the model was created (used as an identifier) with the format: *<year><month><day>T<hour><minute><second>*, *nepochs* stands for "number of epochs", *nfilters_1st* stands for "number of filters in the first convolutional layer", *ksize_1st* stands for "kernel size of the first layer", *number_param* stands for "number of parameters", *tr_acc* stands for "training accuracy" and *val_acc* stands for "validation accuracy". Model names follow the structure: <model timestamp>_dataset1.1_1L_nf1_<nfilters_1st>_ks1_<ksize_1st>_ne_<nepochs>.

Model timestamp	nepochs	nfilters_1st	ksize_1st	number_param	tr_acc	val_acc
20200917T184415	10	2	3	1199	0.967	0.965
20200917T193757	10	2	5	1231	1	1
20200917T193957	10	2	7	1279	0.999	1
20200917T194154	10	4	3	2395	0.998	0.998
20200917T194353	10	4	5	2459	0.997	0.996
20200917T194553	10	4	7	2555	1	1
20200917T194750	10	8	3	4787	1	1
20200917T194948	10	8	5	4915	1	1
20200917T195146	10	8	7	5107	1	1
20200917T195342	10	16	3	9571	1	1
20200917T195537	10	16	5	9827	1	1
20200917T195735	10	16	7	10211	1	1

Table 4.2. Model hyperparameters and results obtained after performing Experiment 1.1 over dataset 1.3. *Model timestamp* is the date and hour when the model was created (used as an identifier) with the format: *<year><month><day>T<hour><minute><second>*, *nepochs* stands for "number of epochs", *nfilters_1st* stands for "number of filters in the first convolutional layer", *ksize_1st* stands for "kernel size of the first layer", *number_param* stands for "number of parameters", *tr_acc* stands for "training accuracy" and *val_acc* stands for "validation accuracy". Model names follow the structure: <model timestamp>_dataset1.3_1L_nf1_<nfilters_1st>_ks1_<ksize_1st>_ne_<nepochs>.

Model timestamp	nepochs	nfilters_1st	ksize_1st	number_param	tr_acc	val_acc
20200917T195929	10	2	3	1199	1	1
20200917T204822	10	2	5	1231	1	1
20200917T205022	10	2	7	1279	1	1
20200917T205223	10	4	3	2395	1	1
20200917T205423	10	4	5	2459	1	1
20200917T205624	10	4	7	2555	1	1
20200917T205823	10	8	3	4787	1	1
20200917T210027	10	8	5	4915	1	1
20200917T210234	10	8	7	5107	1	1
20200917T210438	10	16	3	9571	1	1
20200917T210639	10	16	5	9827	1	1
20200917T210842	10	16	7	10211	1	1

Table 4.3. Model hyperparameters and results obtained after performing Experiment 1.1 over dataset 1.3. *Model timestamp* is the date and hour when the model was created (used as an identifier) with the format: *<year><month><day>T<hour><minute><second>*, *nepochs* stands for "number of epochs", *nfilters_1st* stands for "number of filters in the first convolutional layer", *ksize_1st* stands for "kernel size of the first layer", *number_param* stands for "number of parameters", *tr_acc* stands for "training accuracy" and *val_acc* stands for "validation accuracy". Model names follow the structure: <model timestamp>_dataset2.1_1L_nf1_<nfilters_1st>_ks1_<ksize_1st>_ne_<nepochs>.

4.1.3. Experiment 1.2

The aim of this experiment is to generate new models that contain a lower amount of parameters than the previous ones trained in Experiment 1.1. This is achieved with a new architecture that is similar to the one used in the previous experiment, with the difference that, apart from one convolutional layer varying the number and filter size, a pooling size is added. The size of the pooling layer changes also from bigger values that create simpler architectures to small values, which generate more complex models. In this case, models are trained firs with 10 epochs and then with 20 epochs, so that its evolution can be observed. In addition, with the reduction of the number of parameters, the time spent in training the models is decreased too.

For datasets 1.3 and 2.1, the pooling size takes the following values: 5x5, 4x4, 3x3, and 2x2. As dataset 1.1 had models with a higher amount of parameters, in addition to these values, there are also models generated with pooling size equal to 11x11 and 7x7. Tables 4.4, 4.5 and 4.6 show results from the models generated with datasets 1.1, 1.3. and 2.1 respectively. Hyperparameters related to filters from the convolutional layer take the same values that in the previous experiment, but in the tables only the architectures with lower hyperparameter values are shown, since with these ones, maximum levels of accuracy are already achieved.

For the three datasets, it can be observed that the increase in the number of epochs is noticeable, the more epochs, the more time the model is training and therefore, the higher values of accuracy are achieved. In the case of dataset 1.1, the number of hyperparameters is still very high and it takes a long time to train the models. As these models are very complex compared to the ones obtained from dataset 1.3 and they contain the same figures for the rest of experiments, dataset 1.1 is not used. For datasets 1.3 and 2.1, it can be said the best possible candidates to minimum architectures have been found. These models will be the ones with two filters of size three in the first convolutional layer. Related to the pooling size, it is still not clear which one is the best value so the best option is to continue testing.

Model timestamp	nepochs	nfilters_1st	ksize_1st	poolsize_1st	number_param	number_features	tr_acc	val_acc
20200912T231459	10	2	3	11	2423	800	0.797	0.788
20200912T231702	10	2	3	7	6167	2048	0.840	0.817
20200912T231905	10	2	3	5	11639	3872	0.831	0.821
20200912T232107	10	2	3	4	18839	6272	0.951	0.946
20200912T232310	10	2	3	3	32879	10952	1	1
20200912T232512	10	2	3	2	75287	25088	1	1
20200912T232715	10	2	5	11	2455	800	0.950	0.950
20200912T232917	10	2	5	7	6199	2048	0.989	0.988
20200912T233119	10	2	5	5	11671	3872	1	1
20200912T233321	10	2	5	4	18871	6272	1	1
20200912T233523	10	2	5	3	32911	10952	1	1
20200912T233726	10	2	5	2	75319	25088	1	1
20200913T010449	20	2	3	11	2423	800	0.869	0.855
20200913T010834	20	2	3	7	6167	2048	0.938	0.931
20200913T011218	20	2	3	5	11639	3872	0.990	0.990
20200913T011603	20	2	3	4	18839	6272	1	1
20200913T011949	20	2	3	3	32879	10952	1	1
20200913T012335	20	2	3	2	75287	25088	1	1
20200913T012721	20	2	5	11	2455	800	0.998	0.999
20200913T013107	20	2	5	7	6199	2048	0.999	1
20200913T013454	20	2	5	5	11671	3872	1	1
20200913T013840	20	2	5	4	18871	6272	1	1
20200913T014225	20	2	5	3	32911	10952	1	1
20200913T014610	20	2	5	2	75319	25088	1	1

Table 4.4. Model hyperparameters and results obtained after performing Experiment 1.1 over dataset 1.3. *Model timestamp* is the date and hour when the model was created (used as an identifier) with the format: *<year><month><day>T<hour><minute><second>*, *nepochs* stands for "number of epochs", *nfilters_1st* stands for "number of filters in the first convolutional layer", *ksize_1st* stands for "kernel size of the first layer", *poolsize_1st* stands for "pool size in the first layer", *number_param* stands for "number of parameters", *number_features* stands for "number of features", *tr_acc* stands for "training accuracy" and *val_acc* stands for "validation accuracy". Model names follow the structure: <model timestamp>_dataset1.1_1L_nf1_<nfilters_1st>_ks1_<ksize_1st>_ps1_<poolsize_1st>_ne_<nepochs>.

Note that there is a special case among the models, the one with timestamp equal to 20200919T075614 trained with the dataset 1.3 from Table 4.5 and the one with timestamp 20200919T115645 trained with dataset 2.1 from Table 4.6. They both have the same amount of parameters, nevertheless, the model trained with dataset 2.1 obtains an accuracy of 1 in training whereas the one trained with dataset 1.3 fails to classify 20% of the images. A priori, this could indicate that dataset 1.3 is more difficult to classify than dataset 2.1 of maybe some kind of bias has been introduced in dataset 2.1 unconsciously. Another possible explanation is that the network is not entirely well adjusted and it is necessary to iterate more times to complete the training, that is, to increase the number of epochs. This last option is more likely to be the correct one since it can be observed that with model

20200919T090921_dataset1.3_1L_nf1_2_ks1_3_ps1_3_ne_20 that have the same architecture except that is trained during the double of epochs, the accuracy reached is higher.

Model timestamp	**nepochs**	**nfilters_1st**	**ksize_1st**	**poolsize_1st**	**number_param**	**number_features**	**tr_acc**	**val_acc**
20200919T070324	10	2	3	5	173	50	0.190	0.203
20200919T075412	10	2	3	4	317	98	0.389	0.398
20200919T075614	10	2	3	3	509	162	0.826	0.823
20200919T075818	10	2	3	2	1199	392	0.967	0.965
20200919T080020	10	2	5	5	205	50	0.532	0.545
20200919T080222	10	2	5	4	349	98	0.791	0.787
20200919T080425	10	2	5	3	541	162	0.976	0.968
20200919T080628	10	2	5	2	1231	392	1	1
20200919T090248	20	2	3	5	173	50	0.546	0.567
20200919T090605	20	2	3	4	317	98	0.686	0.683
20200919T090921	20	2	3	3	509	162	0.929	0.926
20200919T091239	20	2	3	2	1199	392	1	1
20200919T091557	20	2	5	5	205	50	0.828	0.821
20200919T091915	20	2	5	4	349	98	0.974	0.973
20200919T092234	20	2	5	3	541	162	0.999	0.999
20200919T092552	20	2	5	2	1231	392	1	1

Table 4.5. Model hyperparameters and results obtained after performing Experiment 1.1 over dataset 1.3. *Model timestamp* is the date and hour when the model was created (used as an identifier) with the format: *<year><month><day>T<hour><minute><second>*, *nepochs* stands for "number of epochs", *nfilters_1st* stands for "number of filters in the first convolutional layer", *ksize_1st* stands for "kernel size of the first layer", *poolsize_1st* stands for "pool size in the first layer", *number_param* stands for "number of parameters", *number_features* stands for "number of features", *tr_acc* stands for "training accuracy" and *val_acc* stands for "validation accuracy". Model names follow the structure: <model timestamp>_dataset1.3_1L_nf1_<nfilters_1st>_ks1_<ksize_1st>_ps1_<poolsize_1st>_ne_<nepochs>.

As a summary, the best architectures to be candidates for minimum model are the ones with 2 filters of size 3x3 in the convolutional layer and different pooling sizes, from now on, this group of architectures is going to be referred to as 2-3-x in relation with the structure of Tables 4.5 and 4.6: *nfilters_1st-ksize_1st-poolsize_1st*, the "x" indicates that hyperparameter can take any value. In addition is necessary to increase the number of epochs.

Model timestamp	nepochs	nfilters_1st	ksize_1st	poolsize_1st	number_param	number_features	tr_acc	val_acc
20200919T110627	10	2	3	5	173	50	0.265	0.265
20200919T115453	10	2	3	4	317	98	0.971	0.971
20200919T115645	10	2	3	3	509	162	1	1
20200919T115835	10	2	3	2	1199	392	1	1
20200919T120024	10	2	5	5	205	50	0.624	0.624
20200919T120214	10	2	5	4	349	98	0.887	0.900
20200919T120403	10	2	5	3	541	162	1	1
20200919T120553	10	2	5	2	1231	392	1	1
20200919T130022	20	2	3	5	173	50	0.928	0.925
20200919T130345	20	2	3	4	317	98	0.999	0.999
20200919T130707	20	2	3	3	509	162	1	1
20200919T131027	20	2	3	2	1199	392	1	1
20200919T131347	20	2	5	5	205	50	0.853	0.847
20200919T131708	20	2	5	4	349	98	0.999	0.999
20200919T132028	20	2	5	3	541	162	1	1
20200919T132346	20	2	5	2	1231	392	1	1

Table 4.6. Model hyperparameters and results obtained after performing Experiment 1.1 over dataset 1.3. *Model timestamp* is the date and hour when the model was created (used as an identifier) with the format: *<year><month><day>T<hour><minute><second>*, *nepochs* stands for "number of epochs", *nfilters_1st* stands for "number of filters in the first convolutional layer", *ksize_1st* stands for "kernel size of the first layer", *poolsize_1st* stands for "pool size in the first layer", *number_param* stands for "number of parameters", *number_features* stands for "number of features", *tr_acc* stands for "training accuracy" and *val_acc* stands for "validation accuracy". Model names follow the structure: <model timestamp>_dataset2.1_1L_nf1_<nfilters_1st>_ks1_<ksize_1st>_ps1_<poolsize_1st>_ne_<nepochs>.

4.2. Group 2: Scale and rotation problem without noise

Once an exploratory analysis over the input hyperparameters space has been performed, an overall view of which are the minimum architectures is obtained, as well as, which are the best datasets to start testing. Therefore, the next step and the aim of these group of experiments is analysing a particular problem such as how geometric transformations on the input dataset influence the learning process of the CNN. "How many features does the model need to classify the transformed images?", "does the model learn different features depending on the type of transformation being applied?" or "how are the filters learnt in the convolutional layer?" will be some of the questions that these experiments will try to answer.

Specifically, the transformations that are going to be analysed are two: scaling and rotation. In order to study the first one, dataset 1.3 will be used; and dataset 2.1 for rotation problem. Both datasets are carefully explained in subsection 4.1.1. Images have the same characteristics, so results obtained from different experiments can be compared.

Three main experiments are carried on in this group: 2.1, 2.2 and 2.3, together with some sub-experiments derived from the main ones. Although the best candidates for minimum architectures have been detected in the group of preliminary experiments, more fine tuning is needed to adjust the hyperparameters completely. That is the reason why the first two experiments, 2.1 and 2.2 (sections 4.2.1 and 4.2.2), only perform until the second phase of experiments, i.e., minimum model search. It is in Experiment 2.3 where finally, the minimum models for both datasets are found and the analysis of both models and results is performed.

4.2.1. Experiment 2.1

From the previous group of experiments, the architectures known as 2-3-x were the ones that obtained the best balance between accuracy and complexity. These models comprise a convolutional layer with two filters of size 3x3 together with a ReLU layer and a pooling layer of variable size (x). After that there is a flattening layer fully connected with a three nodes output layer. In addition, in this experiment, it is going to be tested the architectures 1-3-x, which just contain a filter in the convolutional layer. These set of architectures are the smaller ones tested so, if models achieve high levels of accuracy, among them the minimum one will be found.

In this experiment, the pooling size is going to take the values: 5x5, 4x4, 3x3, 2x2 and 1x1, generating more complex to simpler models. In addition, the number of epochs is increased up to 50, so that the training process has more time to converge. Finally, it is important to add a new hyperparameter that had not been taken into account in the previous experiments: the random seed. Machine learning models makes use of randomness in several ways, being one of the most obvious the initialization point of all network parameters, that is, the starting point from which the model starts looking for the best solution (a minimum) in the input space. At a practical level, seeds are very important as they enable to reproduce the same results across runs of the models and even when the same model is trained from scratch with the same dataset. Moreover, to evaluate the robustness and the generalization power of an architecture, the same model has to be trained with different seeds, so that it can be checked if the hyperparameters are really the best ones or it is just that the starting point was one of the best for the model to start looking for the solution. Each architecture is tested with five different random seeds whose values are: 0, 4, 10, 550 and 1234.

Table 4.7 and Table 4.8 show the results obtained in this experiment for architectures trained with dataset 1.3 and dataset 2.1 respectively. In total, fifty models have been trained. Several conclusions can be extracted from the results shown in both tables. First of all, 1-3-x architectures seem to be better candidates for minimum architecture than 2-3-x architectures, since the first ones achieve almost the same levels of accuracy which a much lower amount of parameters. So, from now on, these are the architectures that should be tested exhaustively. The smaller one has 25 features and, taking into account the low complexity of the problem (differentiate between three polygons), the number of features seems excessive. Therefore, the next experiments must continue decreasing this number.

In order to verify the robustness and generalization performance of the network, it must be checked that, for one specific architecture, all the models (or the majority of them) that have been trained with different seeds have high levels of accuracy, for example, that they classify correctly over the 60% of the images. In this case, all architectures are robust enough, as they achieve almost they same levels of accuracy both for training and validation data through all the seeds.

On the other hand, the fact that all models but a few achieve 100% of accuracy is noteworthy. The reason behind this can be, once again, that the number of epochs is still not enough and it should be increased. Since with the raising in the number of epochs, the training time increases as well, the re-training of all the models to test if the accuracy increases is not viable, and that is why the Experiment 2.1.1 is carried on (Section 4.2.1.1).

To sum up, the most likely models to be the minimum architectures are the ones that have one convolutional layer with filter of size 3x3 and a pooling layer. The following experiments have to continue studying the 1-3-x architectures including more epochs to see if the overall accuracy raises and trying to reduce even more the number of parameters by increasing the pooling size.

Table 4.7. Model hyperparameters and results obtained after performing Experiment 2.1 over dataset 1.3. *Model timestamp* is the date and hour when the model was created (used as an identifier) with the format: *<year><month><day>T<hour><minute><second>*, *seed* is the seed used for fixing randomness in the model, *nepochs* stands for "number of epochs", *nfilters_1st* stands for "number of filters in the first convolutional layer", *ksize_1st* stands for "kernel size of the first layer", *poolsize_1st* stands for "pool size in the first layer", *number_param* stands for "number of parameters", *number_features* stands for "number of features", *tr_acc* stands for "training accuracy" and *val_acc* stands for "validation accuracy". Model names follow the structure: <model timestamp>_dataset1.3_1L_nf1_<nfilters_1st>_ks1_<ksize_1st>_ps1_<poolsize_1st>_ne_<nepochs>_s_<seed>.

Model timestamp	seed	nepochs	nfilters_1st	ksize_1st	poolsize_1st	number_param	number_features	tr_acc	val_acc
20200926T122535	0	50	1	3	5	88	25	0.837	0.826
20200926T132219	4	50	1	3	5	88	25	0.746	0.732
20200926T133000	10	50	1	3	5	88	25	0.948	0.949
20200926T133747	550	50	1	3	5	88	25	0.783	0.752
20200926T134528	1234	50	1	3	5	88	25	0.884	0.872
20200926T135311	0	50	1	3	4	160	49	0.905	0.889
20200926T140051	4	50	1	3	4	160	49	0.992	0.993
20200926T140832	10	50	1	3	4	160	49	0.954	0.949
20200926T141612	550	50	1	3	4	160	49	0.972	0.973
20200926T142353	1234	50	1	3	4	160	49	0.995	0.995
20200926T143132	0	50	1	3	3	256	81	0.999	0.999
20200926T143917	4	50	1	3	3	256	81	0.999	0.999
20200926T144703	10	50	1	3	3	256	81	1	1
20200926T145531	550	50	1	3	3	256	81	1	1
20200926T150330	1234	50	1	3	3	256	81	0.933	0.920
20200926T151115	0	50	1	3	2	601	196	1	1
20200926T151941	4	50	1	3	2	601	196	1	1
20200926T152734	10	50	1	3	2	601	196	1	1
20200926T153521	550	50	1	3	2	601	196	1	1
20200926T154301	1234	50	1	3	2	601	196	1	1
20200926T155044	0	50	1	3	1	2365	784	1	1
20200926T155839	4	50	1	3	1	2365	784	1	1
20200926T160702	10	50	1	3	1	2365	784	1	1
20200926T161531	550	50	1	3	1	2365	784	1	1
20200926T162334	1234	50	1	3	1	2365	784	1	1
20200926T163202	0	50	2	3	5	173	50	0.999	1
20200926T164012	4	50	2	3	5	173	50	1	1
20200926T164807	10	50	2	3	5	173	50	0.979	0.974
20200926T165550	550	50	2	3	5	173	50	1	1
20200926T170330	1234	50	2	3	5	173	50	0.998	0.998

Continued on next page

Table 4.7 – continued from previous page

Model timestamp	seed	nepochs	nfilters_1st	ksize_1st	poolsize_1st	number_param	number_features	tr_acc	val_acc
20200926T171110	0	50	2	3	4	317	98	0.981	0.978
20200926T171852	4	50	2	3	4	317	98	1	1
20200926T172631	10	50	2	3	4	317	98	1	1
20200926T173411	550	50	2	3	4	317	98	1	1
20200926T174149	1234	50	2	3	4	317	98	0.970	0.970
20200926T174927	0	50	2	3	3	509	162	1	1
20200926T175704	4	50	2	3	3	509	162	1	1
20200926T180441	10	50	2	3	3	509	162	1	1
20200926T181218	550	50	2	3	3	509	162	1	1
20200926T181955	1234	50	2	3	3	509	162	1	1
20200926T182733	0	50	2	3	2	1199	392	1	1
20200926T183511	4	50	2	3	2	1199	392	1	1
20200926T184249	10	50	2	3	2	1199	392	1	1
20200926T185028	550	50	2	3	2	1199	392	1	1
20200926T185807	1234	50	2	3	2	1199	392	1	1
20200926T190546	0	50	2	3	1	4727	1568	1	1
20200926T191324	4	50	2	3	1	4727	1568	1	1
20200926T192102	10	50	2	3	1	4727	1568	1	1
20200926T192840	550	50	2	3	1	4727	1568	1	1
20200926T193618	1234	50	2	3	1	4727	1568	1	1

Table 4.8. Model hyperparameters and results obtained after performing Experiment 2.1 over dataset 2.1. *Model timestamp* is the date and hour when the model was created (used as an identifier) with the format: *<year><month><day>T<hour><minute><second>*, *seed* is the seed used for fixing randomness in the model, *nepochs* stands for "number of epochs", *nfilters_1st* stands for "number of filters in the first convolutional layer", *ksize_1st* stands for "kernel size of the first layer", *poolsize_1st* stands for "pool size in the first layer", *number_param* stands for "number of parameters", *number_features* stands for "number of features", *tr_acc* stands for "training accuracy" and *val_acc* stands for "validation accuracy". Model names follow the structure: <model timestamp>_dataset1.3_1L_nf1_<nfilters_1st>_ks1_<ksize_1st>_ps1_<poolsize_1st>_ne_<nepochs> _s_<seed>.

Model timestamp	seed	nepochs	nfilters_1st	ksize_1st	poolsize_1st	number_param	number_features	tr_acc	val_acc
20200926T194356	0	50	1	3	5	88	25	1	1
20200926T203832	4	50	1	3	5	88	25	0.997	0.994
20200926T204619	10	50	1	3	5	88	25	1	1
20200926T205405	550	50	1	3	5	88	25	0.995	0.992
20200926T210150	1234	50	1	3	5	88	25	0.991	0.994
20200926T210933	0	50	1	3	4	160	49	1	1
20200926T211715	4	50	1	3	4	160	49	0.950	0.957
20200926T212456	10	50	1	3	4	160	49	0.995	0.995
20200926T213237	550	50	1	3	4	160	49	0.999	1

Continued on next page

Table 4.8 – continued from previous page

Model timestamp	seed	nepochs	nfilters_1st	ksize_1st	poolsize_1st	number_param	number_features	tr_acc	val_acc
20200926T214023	1234	50	1	3	4	160	49	1	1
20200926T214819	0	50	1	3	3	256	81	1	1
20200926T215618	4	50	1	3	3	256	81	1	1
20200926T220413	10	50	1	3	3	256	81	1	1
20200926T221213	550	50	1	3	3	256	81	1	1
20200927T095234	1234	50	1	3	3	256	81	1	1
20200927T105721	0	50	1	3	2	601	196	1	1
20200927T110453	4	50	1	3	2	601	196	1	1
20200927T111224	10	50	1	3	2	601	196	1	1
20200927T111956	550	50	1	3	2	601	196	1	1
20200927T112728	1234	50	1	3	2	601	196	1	1
20200927T113502	0	50	1	3	1	2365	784	1	1
20200927T114236	4	50	1	3	1	2365	784	1	1
20200927T115013	10	50	1	3	1	2365	784	1	1
20200927T115747	550	50	1	3	1	2365	784	1	1
20200927T120520	1234	50	1	3	1	2365	784	1	1
20200927T130129	0	50	2	3	5	173	50	1	1
20200927T130904	4	50	2	3	5	173	50	1	1
20200927T131637	10	50	2	3	5	173	50	1	1
20200927T132415	550	50	2	3	5	173	50	1	1
20200927T133152	1234	50	2	3	5	173	50	1	1
20200927T133926	0	50	2	3	4	317	98	1	1
20200927T134705	4	50	2	3	4	317	98	1	1
20200927T135445	10	50	2	3	4	317	98	1	1
20200927T140218	550	50	2	3	4	317	98	1	1
20200927T140951	1234	50	2	3	4	317	98	1	1
20200927T141725	0	50	2	3	3	509	162	1	1
20200927T142458	4	50	2	3	3	509	162	1	1
20200927T143231	10	50	2	3	3	509	162	1	1
20200927T144006	550	50	2	3	3	509	162	1	1
20200927T144745	1234	50	2	3	3	509	162	1	1
20200927T145526	0	50	2	3	2	1199	392	1	1
20200927T150308	4	50	2	3	2	1199	392	1	1
20200927T151052	10	50	2	3	2	1199	392	1	1
20200927T151836	550	50	2	3	2	1199	392	1	1
20200927T152619	1234	50	2	3	2	1199	392	1	1
20200927T153359	0	50	2	3	1	4727	1568	1	1
20200927T154134	4	50	2	3	1	4727	1568	1	1
20200927T154909	10	50	2	3	1	4727	1568	1	1
20200927T155643	550	50	2	3	1	4727	1568	1	1
20200927T160418	1234	50	2	3	1	4727	1568	1	1

4.2.1.1. Experiment 2.1.1

In order to verify if the number of epochs has to be increased and, if this is the case, find out which is the minimum number of epochs to train the models, the present experiment is conducted. Since increasing the number of epochs produces a marginal increase in the training time, this experiment is going to train a single architecture a few times, raising each time the number of epochs by fifty. In this way, keeping the rest of hyperparameters constant and just varying the number of epochs, changes in both training and test accuracy can be attributed to the increase in the number of epochs. This methodology enables to find out the optimal number of epochs and, at the same time, save time.

The model that is going to be re-trained is the one shown in Table ... , known as 20200926T133000_dataset1.3_1L_nf1_1_ks1_3_ps1_5_ne_50_s_10, whose architecture comprises a convolutional layer with a single filter of size 3x3 and a pooling layer of size 5x5. The seed is fixed to 10 for reproducibility of results, so when a new model is generated with more epochs, the previous iterations have the same value as the other models. As the rest of architectures that are going to be trained from this point on, have the same order of parameters, the number of epochs extracted from this experiment can be applied equally to all models.

Table 4.9 shows the results obtained from this experiment. It can be observed that indeed, as the number of epochs increases, the training and testing accuracy does too. One hundred and fifty epochs seems to ve a valid value to train the following models, as they reach the maximum accuracy.

Model timestamp	**seed**	**nepochs**	**nfilters_1st**	**ksize_1st**	**poolsize_1st**	**number_param**	**number_features**	**tr_acc**	**val_acc**
20200926T133000	10	50	1	3	5	88	25	0.948	0.949
20201004T195321	10	100	1	3	5	88	25	0.997	0.995
20201004T202610	10	150	1	3	5	88	25	1	1

Table 4.9. Model hyperparameters and results obtained after performing Experiment 2.1.1 over dataset 1.3. *Model timestamp* is the date and hour when the model was created (used as an idetifier) with the format: *<year><month><day>T<hour><minute><second>*, *seed* is the seed used for fixing randomness in the model, *nepochs* stands for "number of epochs", *nfilters_1st* stands for "number of filters in the first convolutional layer", *ksize_1st* stands for "kernel size of the first layer", *poolsize_1st* stands for "pool size in the first layer", *number_param* stands for "number of parameters", *number_features* stands for "number of features", *tr_acc* stands for "training accuracy" and *val_acc* stands for "validation accuracy". Model names follow the structure: <model timestamp>_dataset1.3_1L_nf1_<nfilters_1st>_ks1_<ksize_1st>_ps1_<poolsize_1st>_ne_<nepochs> _s_<seed>.

4.2.2. Experiment 2.2

As a result of performing Experiment 2.1 (Section 4.2.1), one of the models trained with 1-3-x architectures is meant to be the minimum model. Nevertheless, it is neccesary to find out if, increasing even more the pooling size, the model is still able to recognise the figures. Therefore, the aim of this experiment is to find the minimum model, discovering if with the minimum number of features possible, that is, one feature, the model can still perform the classification task.

The group of models trained are going to have one convolutional layer with three filters plus a pooling layer of variable size. As the number of parameters has to be reduced to the minimum, pooling size has to be increased, hence, the pooling size in this experiment takes highest values: 19x19, 17x17, 15x15, 13x13, 11x11, 9x9 and 7x7 for dataset 1.3; whereas for dataset 2.1, sizes 19x19 and 17x17 are not used. The reason behind is that after performing the experiment on dataset 1.3, it was notable that with those pooling values the models do not reach high accuracies. The number of epochs is kept equal to fifty. In this way, it does not take so long to train all models and obtain possible candidates. As it happened in the previous experiment, for a single architecture, five models are trained, each one of them with a different seed.

Tables 4.10 and 4.11 show both the models trained and results obtained. Looking at them, several conclusions can be extracted. First of all, in dataset 1.3 (Table 4.10), there are certain models that with just one parameter manage to reach high levels of accuracy. Specifically, four of the models with the architecture 1-3-15, achieve an accuracy over 60% and maybe all of them would manage to reach the 100% of accuracy if they were trained during more epochs. And the same happens with the rest of models that only contain one parameter, only some of them manage high accuracies. This fact motivates carrying out Experiment 2.2.1 (Section 4.2.2.1) which, in a similar way to Experiment 2.1.1 (Section 4.2.1.1) checks if models need more epochs to converge.

Nevertheless, for dataset 2.1 (Table 4.11), this does not happen since the models with just one feature do not reach accuracies over 60%. This fact go against an hypothesis that raised during Experiment 1.2 (Section 4.1.3) that dataset 2.1 could be easier to classify than dataset 1.3. Once again, the number of epochs is crucial for the models to converge to a stable accuracy level. Therefore, until the number of epochs is not increased, no final conclusions can be drawn.

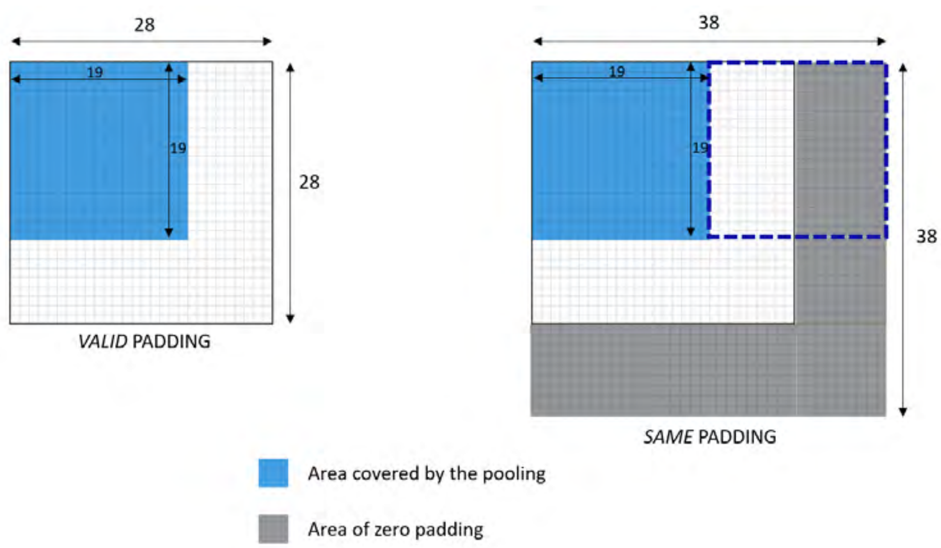

Figure 4.9. Valid versus same padding when performing pooling if the pooling size is not divisor of the image dimensions. *Valid* option is equal to no padding, skipping some areas of the image, whereas *same* option adds padding until the pooling can be performed over the complete original image, without skipping parts of the image.

On the other hand, it is not until this point, when models with one parameter are trained, that it is noticeable pooling sizes are not completely correct. This is due to the way in which pooling operation is performed by the library Tensor Flow. When setting padding value to “valid”, as it happens in the pooling layer, no padding is added to the input, so unless the pooling size is divisor of the image dimensions, some part of the image will be omitted. Figure 4.9 shows a example of both types of padding that help to understand better these concepts. In the example (as in these experiments), images have size of 28x28 and pooling stride value is equal to the pooling size that is used in each case. If the pooling size is not divisor of image dimensions and valid padding is used, some parts of the image will be skipped, since no padding is added. On the other hand, same padding adds pixels with value zero that are needed for the pooling to pass over the entire original image. Nevertheless, if pooling size is divisor of 28, performing pooling with either valid or same padding produces the same result, and the complete image is taking into account when performing the operation avoiding to add extra information to the image that has already been filtered.

Notice that although the pooling is not taking into account part of the image, there are models that are able to perform the classification correctly, reaching high levels of accuracy. It is striking that the model does not need the information present in the complete image to identify it, and the reason could be that some bias has been introduced unconsciously in the datasets when creating them. This kind of hypothesis can be better analysed once the minimum model is found and the feature maps are visualized, what will be finally done in Experiment 2.3 (Section 4.2.3).

Summarizing, the minimum model has an architecture that with a single feature is able to recognise all images. Nevertheless, in order to build models with just one feature that are valid, i.e. that consider the whole image to make a classification decision, a valid pooling size has to be used. Therefore, the architectures that are going to be tested in the following experiments must have a pooling size that is divisor of 28. Related to the number of epochs, Experiment 2.1.1 make clear that it has to be, at least, one hundred fifty and Experiment 2.2.1 is performed in order to confirm it.

Table 4.10. Model hyperparameters and results obtained after performing Experiment 2.2 over dataset 1.3. *Model timestamp* is the date and hour when the model was created (used as an identifier) with the format: *<year><month><day>T<hour><minute><second>*, *seed* is the seed used for fixing randomness in the model, *nepochs* stands for "number of epochs", *nfilters_1st* stands for "number of filters in the first convolutional layer", *ksize_1st* stands for "kernel size of the first layer", *poolsize_1st* stands for "pool size in the first layer", *number_param* stands for "number of parameters", *number_features* stands for "number of features", *tr_acc* stands for "training accuracy" and *val_acc* stands for "validation accuracy". Model names follow the structure: <model timestamp>_dataset1.3_1L_nf1_<nfilters_1st>_ks1_<ksize_1st>_ps1_<poolsize_1st>_ne_<nepochs> _s_<seed>.

Model timestamp	seed	nepochs	nfilters_1st	ksize_1st	poolsize_1st	number_param	number_features	tr_acc	val_acc
20201004T171157	0	50	1	3	19	16	1	0.340	0.343
20201004T191743	4	50	1	3	19	16	1	0.667	0.667
20201004T192537	10	50	1	3	19	16	1	0.333	0.334
20201004T193320	550	50	1	3	19	16	1	0.323	0.347
20201004T194101	1234	50	1	3	19	16	1	0.629	0.618
20201003T182536	0	50	1	3	17	16	1	0.405	0.393
20201003T192301	4	50	1	3	17	16	1	0.667	0.667
20201003T193121	10	50	1	3	17	16	1	0.333	0.334
20201003T193940	550	50	1	3	17	16	1	0.466	0.479
20201003T194804	1234	50	1	3	17	16	1	0.875	0.859
20201003T132834	0	50	1	3	15	16	1	0.333	0.334

Continued on next page

Table 4.10 – continued from previous page

Model timestamp	seed	nepochs	nfilters_1st	ksize_1st	poolsize_1st	number_param	number_features	tr_acc	val_acc
20201003T133636	4	50	1	3	15	16	1	0.667	0.667
20201003T134429	10	50	1	3	15	16	1	0.679	0.668
20201003T135238	550	50	1	3	15	16	1	0.943	0.945
20201003T140038	1234	50	1	3	15	16	1	0.735	0.736
20201003T140830	0	50	1	3	13	25	4	0.823	0.836
20201003T141651	4	50	1	3	13	25	4	0.809	0.794
20201003T142456	10	50	1	3	13	25	4	0.385	0.382
20201003T143304	550	50	1	3	13	25	4	0.652	0.661
20201003T144111	1234	50	1	3	13	25	4	0.991	0.991
20201003T072820	0	50	1	3	11	25	4	0.425	0.430
20201003T083037	4	50	1	3	11	25	4	0.891	0.880
20201003T083816	10	50	1	3	11	25	4	0.532	0.519
20201003T084554	550	50	1	3	11	25	4	0.866	0.866
20201003T085332	1234	50	1	3	11	25	4	0.953	0.956
20201003T090108	0	50	1	3	9	40	9	0.763	0.752
20201003T090843	4	50	1	3	9	40	9	0.829	0.816
20201003T091619	10	50	1	3	9	40	9	0.638	0.644
20201003T092353	550	50	1	3	9	40	9	0.689	0.662
20201003T093130	1234	50	1	3	9	40	9	0.881	0.882
20201003T093907	0	50	1	3	7	61	16	0.689	0.662
20201003T094645	4	50	1	3	7	61	16	0.968	0.972
20201003T095422	10	50	1	3	7	61	16	0.743	0.720
20201003T100203	550	50	1	3	7	61	16	0.836	0.816
20201003T100944	1234	50	1	3	7	61	16	0.737	0.729

Table 4.11. Model hyperparameters and results obtained after performing Experiment 2.2 over dataset 2.1. *Model timestamp* is the date and hour when the model was created (used as an identifier) with the format: *<year><month><day>T<hour><minute><second>*, *seed* is the seed used for fixing randomness in the model, *nepochs* stands for "number of epochs", *nfilters_1st* stands for "number of filters in the first convolutional layer", *ksize_1st* stands for "kernel size of the first layer", *poolsize_1st* stands for "pool size in the first layer", *number_param* stands for "number of parameters", *number_features* stands for "number of features", *tr_acc* stands for "training accuracy" and *val_acc* stands for "validation accuracy". Model names follow the structure: <model timestamp>_dataset2.1_1L_nf1_<nfilters_1st>_ks1_<ksize_1st>_ps1_<poolsize_1st>_ne_<nepochs> _s_<seed>.

Model timestamp	seed	nepochs	nfilters_1st	ksize_1st	poolsize_1st	number_param	number_features	tr_acc	val_acc
20201003T144910	0	50	1	3	15	16	1	0.390	0.387
20201003T145710	4	50	1	3	15	16	1	0.408	0.393
20201003T150508	10	50	1	3	15	16	1	0.317	0.393
20201003T151307	550	50	1	3	15	16	1	0.306	0.305
20201003T152103	1234	50	1	3	15	16	1	0.392	0.392

Continued on next page

Table 4.11 – continued from previous page

Model timestamp	seed	nepochs	nfilters_1st	ksize_1st	poolsize_1st	number_param	number_features	tr_acc	val_acc
20201003T152901	0	50	1	3	13	25	4	0.586	0.600
20201003T153704	4	50	1	3	13	25	4	0.582	0.576
20201003T154506	10	50	1	3	13	25	4	0.134	0.124
20201003T155317	550	50	1	3	13	25	4	0.210	0.209
20201003T160132	1234	50	1	3	13	25	4	0.730	0.718
20201003T101722	0	50	1	3	11	25	4	0.282	0.282
20201003T111922	4	50	1	3	11	25	4	0.942	0.942
20201003T112701	10	50	1	3	11	25	4	0.433	0.428
20201003T113441	550	50	1	3	11	25	4	0.218	0.230
20201003T114217	1234	50	1	3	11	25	4	0.720	0.719
20201003T114953	0	50	1	3	9	40	9	0.754	0.758
20201003T115729	4	50	1	3	9	40	9	0.666	0.669
20201003T120506	10	50	1	3	9	40	9	0.472	0.480
20201003T121244	550	50	1	3	9	40	9	0.732	0.708
20201003T122021	1234	50	1	3	9	40	9	0.951	0.947
20201003T122758	0	50	1	3	7	61	16	0.795	0.804
20201003T123535	4	50	1	3	7	61	16	0.856	0.857
20201003T124312	10	50	1	3	7	61	16	0.987	0.991
20201003T125049	550	50	1	3	7	61	16	0.885	0.882
20201003T125826	1234	50	1	3	7	61	16	0.920	0.929

4.2.2.1. Experiment 2.2.1

The aim of this experiment is to check if models with just one single feature are able to classify correctly the 100% of the training samples if the number of epochs is increased. In this is the case, the basic hyperparameters for the minimum model will be setted, because although the models tested in this experiment are omitting part of the image when pooling, if even with this condition they are able to classify the images, a model that takes all the information from the image into account to make the decision, will perform equally well.

The group of models being tested in this experiment are the ones with the architecture 1-3-15, that is, with one convolutional layer that has one filter of size 3x3 plus a pooling layer of size 15x15. Looking at Table 4.10 extracted from the previous experiment, it is important to highlight that, from the models that have just one feature, the ones with the pooling size closer to a divisor of 28 (in this case 15 is closer to 14 than 17 or 19) are the ones that are able to reach the highest levels of accuracy. This indicator supports the theory that apparently seems to be obvious: the model takes better decisions when it has more information about the problem (in this case, when he takes all the image into account). All models will be trained during 200 epochs, to make sure they all reach their maximum level of accuracy and only dataset 1.3 will be used.

Table 4.12 contains the results obtained from this experiment and it can be observed that, from a total of five models, four of them manage to reach over a 65% of accuracy in both training and validation datasets and, two out of the four, achieve a 100%. Therefore, this experiment ratifies that with one single feature the model is able to classify the complete dataset correctly and the minimum model has to have an architecture similar to these ones. The three models that are unable to reach the maximum accuracy, are an indication that those seeds are not a good random initialization point in this case.

Model timestamp	seed	nepochs	nfilters_1st	ksize_1st	poolsize_1st	number_param	number_features	tr_acc	val_acc
20201006T180741	0	200	1	3	15	16	1	0.333	0.333
20201006T193105	4	200	1	3	15	16	1	0.694	0.694
20201006T200156	10	200	1	3	15	16	1	1	1
20201006T203317	550	200	1	3	15	16	1	1	1
20201006T210401	1234	200	1	3	15	16	1	1	1

Table 4.12. Model hyperparameters and results obtained after performing Experiment 2.2.1 over dataset 1.3. *Model timestamp* is the date and hour when the model was created (used as an idetifier) with the format: *<year><month><day>T<hour><minute><second>*, *seed* is the seed used for fixing randomness in the model, *nepochs* stands for “number of epochs”, *nfilters_1st* stands for “number of filters in the first convolutional layer”, *ksize_1st* stands for “kernel size of the first layer”, *poolsize_1st* stands for “pool size in the first layer”, *number_param* stands for “number of parameters”, *number_features* stands for “number of features”, *tr_acc* stands for “training accuracy” and *val_acc* stands for “validation accuracy”. Model names follow the structure: <model timestamp>_dataset1.3_1L_nf1_<nfilters_1st>_ks1_<ksize_1st>_ps1_<poolsize_1st>_ne_<nepochs> _s_<seed>.

4.2.3. Experiment 2.3

After having carried out all the previous experiments, there are some facts that have been discovered and need to be taken into account while performing the last experiment of this group. First of all, the best candidates to be the minimum architecture are the ones with one convolutional layer that have a single filter of size 3x3 plus a pooling layer (1-3-x). The classification part of the network does not have any hidden layers, just the input flattening layer and the output layer with three nodes. Among these architectures, the ones that only have one feature, are able to perform the classification task reaching high levels of accuracy, so those are the ones that will be explored in detain in this experiment. Secondly, it is important to take into account that the CNN are executing *valid* padding over the images, hence in order no to leave out some parts of the image, the pooling sizes has to be a divisor of the image dimensions, in this case, a divisor fo 28. That gives three possible options for pooling size: 28x28, 14x14 and 7x7. One is also a divisor of 28 but a pooling of size 1x1 does not make sense since it will give as a result the same image, contrary to the aim of pooling itself, which is summarise the feature information. Finally, the number of epochs has to be of the order of hundreds, for the model training to converge and reach its maximum level of accuracy avoiding overfitting.

As stated before, these are key points to build the minimum model and once it has been found out, the results obtained have to be analysed. Therefore the aim of this experiment is to find out how the minimum model is learning to classify the images when they are either scaled or rotated, comparing the behaviour of the models in both cases. Following the nomenclature used in this chapter, three kinds of architectures have been trained in this experiment: 1-3-28, 1-3-14 and 1-3-7. The number of epochs is set to 200, as it has been observed in 2.2.1 (Section 4.2.2.1) that these values are enough for the models to converge.

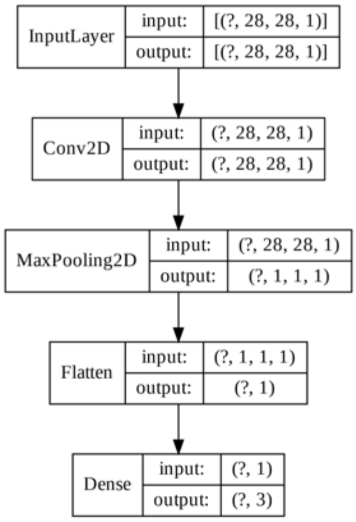

Figure 4.10. Architecture of the minimum model selected for datasets 1.3 and 2.1.

Model timestamp	**seed**	**nepochs**	**nfilters_1st**	**ksize_1st**	**poolsize_1st**	**number_param**	**number_features**	**tr_acc**	**val_acc**
20201008T190742	0	200	1	3	28	16	1	0.667	0.667
20201008T190742	4	200	1	3	28	16	1	0.667	0.667
20201008T211004	10	200	1	3	28	16	1	1	0.9998
20201008T214602	550	200	1	3	28	16	1	0.814	0.820
20201008T221959	1234	200	1	3	28	16	1	0.869	0.871
20201008T225221	0	200	1	3	14	25	4	0.935	0.940
20201008T232412	4	200	1	3	14	25	4	1	1
20201008T235751	10	200	1	3	14	25	4	1	1
20201009T003114	550	200	1	3	14	25	4	0.998	0.999
20201009T010543	1234	200	1	3	14	25	4	1	1
20201009T014056	0	200	1	3	7	61	16	0.981	0.978
20201009T021420	4	200	1	3	7	61	16	1	1
20201009T024819	10	200	1	3	7	61	16	1	1
20201009T032330	550	200	1	3	7	61	16	0.979	0.979
20201009T035902	1234	200	1	3	7	61	16	1	1

Table 4.13. Model hyperparameters and results obtained after performing Experiment 2.3 over dataset 1.3. Shaded cells show the model selected as the minimum one. *Model timestamp* is the date and hour when the model was created (used as an identifier) with the format: *<year><month><day>T<hour><minute><second>*, *seed* is the seed used for fixing randomness in the model, *nepochs* stands for "number of epochs", *nfilters_1st* stands for "number of filters in the first convolutional layer", *ksize_1st* stands for "kernel size of the first layer", *poolsize_1st* stands for "pool size in the first layer", *number_param* stands for "number of parameters", *number_features* stands for "number of features", *tr_acc* stands for "training accuracy" and *val_acc* stands for "validation accuracy". Model names follow the structure: <model timestamp>_dataset1.3_1L_nf1_<nfilters_1st>_ks1_<ksize_1st>_ps1_<poolsize_1st>_ne_<nepochs>_s_<seed>.

Both Tables 4.13 and 4.14 show results obtained from the experiment for dataset 1.3 and dataset 2.1. It is highlighted in grey the minimum models that have been selected, nevertheless some general comments can be made before start analysing the minimum model for each dataset. The selected architecture can be visualized in Figure 4.10.

Looking at Table 4.13 it can be observed that there are several models either with a pooling size of 14x14 or 7x7 that achieve the same level of accuracy, that is the maximum (100%). Models with the same architecture and same levels of accuracy result in questioning if they all are the exactly the same model, if starting from different initialization points the all have learnt the same features/filters. In order to answer this question, filters from models 20201008T232412_dataset1.3_1L_nf1_1_ks1_3_ps1_14_ne_200_s_4, 20201008T232412_dataset1.3_1L_nf1_1_ks1_3_ps1_14_ne_200_s_4 and 20201008T232412_dataset1.3_1L_nf1_1_ks1_3_ps1_14_ne_200_s_4 are plotted in Figure 4.11.

It can be observed that filters obtained from the three models are different, what means that the features leant will also differ among them: they will be similar, but not exactly the same. Therefore, the same result can be obtain in different ways. This conclusion emphasizes the fact that CNN and neural networks in general are search methods, whose aim is to examine the input space given by the data (images in the case of CNNs) and search for a point where the error is minimal. But there are different ways of achieving that point of minimum error and that is why networks that have extract different features achieve the same levels of accuracy.

Model timestamp	**seed**	**nepochs**	**nfilters_1st**	**ksize_1st**	**poolsize_1st**	**number_param**	**number_features**	**tr_acc**	**val_acc**
20201009T132043	0	200	1	3	28	16	1	0.598	0.601
20201009T155101	4	200	1	3	28	16	1	0.648	0.642
20201009T164248	10	200	1	3	28	16	1	0.630	0.628
20201009T173501	550	200	1	3	28	16	1	0.610	0.616
20201009T182406	1234	200	1	3	28	16	1	0.333	0.332
20201009T191243	0	200	1	3	14	25	4	0.667	0.668
20201009T195419	4	200	1	3	14	25	4	0.859	0.863
20201009T203516	10	200	1	3	14	25	4	0.620	0.609
20201009T211451	550	200	1	3	14	25	4	0.709	0.731
20201009T215247	1234	200	1	3	14	25	4	0.809	0.809
20201009T223027	0	200	1	3	7	61	16	0.891	0.889
20201010T083756	4	200	1	3	7	61	16	0.883	0.883
20201010T100449	10	200	1	3	7	61	16	1	1
20201010T103735	550	200	1	3	7	61	16	0.994	0.994
20201010T111028	1234	200	1	3	7	61	16	1	1

Table 4.14. Model hyperparameters and results obtained after performing Experiment 2.3 over dataset 2.1. Shaded cells show the model selected as the minimum one. *Model timestamp* is the date and hour when the model was created (used as an identifier) with the format: *<year><month><day>T<hour><minute><second>*, *seed* is the seed used for fixing randomness in the model, *nepochs* stands for "number of epochs", *nfilters_1st* stands for "number of filters in the first convolutional layer", *ksize_1st* stands for "kernel size of the first layer", *poolsize_1st* stands for "pool size in the first layer", *number_param* stands for "number of parameters", *number_features* stands for "number of features", *tr_acc* stands for "training accuracy" and *val_acc* stands for "validation accuracy". Model names follow the structure: <model timestamp>_dataset2.1_1L_nf1_<nfilters_1st>_ks1_<ksize_1st>_ps1_<poolsize_1st>_ne_<nepochs> _s_<seed>.

20201008T232412_dataset1.3_1L_nf1_1 _ks1_3_ps1_14_ne_200_s_4

20201008T235751_dataset1.3_1L_nf1_1 _ks1_3_ps1_14_ne_200_s_10

20201008T232412_dataset1.3_1L_nf1_1 _ks1_3_ps1_14_ne_200_s_4

0.389	0.036	-0.638
-0.577	-0.427	-0.373
0.189	-0.452	0.462

-0.014	0.167	0.274
-0.101	-0.069	0.395
-0.758	-0.747	-0.172

-0.038	-0.294	-0.020
-0.388	-0.012	0.405
-0.202	0.236	0.271

Figure 4.11. Filters from three different models that have the same architecture but different initialization seeds and obtain the maximum accuracy in both training and validation. First row of images shows the luminance of the filter whereas in the second the exact numeric values are shown. They represent three different solutions to the same problem. The models are extracted from Table 4.13.

Finally, the minimum models for each dataset can be selected. In the case of dataset 1.3 (Table 4.13), it is very clear that the best option for minimum model is 20201008T211004_dataset 1.3_1L_nf1_1_ks1_3_ps1_28_ne_200_s_10. With just one feature is able to obtain a 1 of accuracy in training and a 0.998 in validation, which means that it does not have overfitting. Among the five models with the same architecture (1-3-28), the selected model reaches the highest accuracy levels, so seed 10 seems to be the best initialization point. Moreover, the five models, and indicator of the high robustness of the model. On the other hand, the results obtained for dataset 2.1 (Table 4.14) are in general worse, as most of the models do not manage to achieve at least 0.8 of accuracy, as it happens in the case of dataset 1.3. Nevertheless, a model with one single feature can be chosen as the minimum one, 20201009T155101_dataset2.1_1L_nf1_1_ks1_3_ps1_28_ne_200_s_4, since it manages to classify correctly around the 65% of the images. Although this model does not manage to achieve such high levels of accuracy it is selected to attempt to understand what the model is doing to classify rotated figures. In addition, the choice of this models enables to compare better the learning process as both cases have a similar architecture. Related to the robustness of the model, this architecture has also a good generalization power, since all the models trained with it using different seeds reach almost the same accuracy, except for the one with seed value equal to 1234.

It is important to notice that in these minimal architectures, the classification part of the network is not performing almost any calculation. It only comprises the flattening input layer which has one single neuron since there is only one feature extracted by the network plus an output layer with three neurons, since the CNN has to differentiate between three classes. The classifier that has been built is so simple that at first glance, it seems that the model is not taking advantage of the power that multilayer perceptrons have. Nevertheless, since the minimum models only have one feature that they can use to perform the classification, this single feature cannot be combined with others to improve the decision-making process of the classifier, so adding hidden layers to the perceptron is not really going to help that much. The implications of using this classifier is a key point that has to be remembered during the whole analysis that is presented next.

Once the minimum models have been selected, a detailed study of its learning process is presented in the following sections. The analysis has been divided by dataset into two different sections. In this way, Scale problem linked to dataset 1.3 will be studied in first place and, after that, rotation problem which is associated to dataset 2.1.

4.2.3.1. Scale problem without noise

Two different approaches have been developed to analyse how the CNN is learning to differentiate among figures when its size changes, each one of them making use of a different set of techniques. The aim of the first approach is to understand in general the features that have been learnt: what they are exactly, how they enable the model to perform the classification or how they have been extracted. The second approach is focused on trying to obtain the physical meaning of the model: which parts of the images are more important and why that areas help the model distinguish a class from one another.

For the first approach, the techniques used are other Machine Learning classification algorithms which ease the interpretability of the decision making process, together with basic feature representations; whereas in the second approach filter representations, feature maps and a different version of heat maps are used. Therefore, the next two sections provide a complete analysis of the minimum models learning process, which enables to understand better the results given by the network and how it is able to achieve them.

Related to the feature analysis, it is important to remember how exactly that feature has been extracted and what does it mean in terms of the CNN. First of all, the filter is passed through the entire image giving as a result a feature map, where the parts of the image that are considered the most important (the ones that contain the features) are activated. After that, a pooling operation is performed. In the case of the minimum model, the pooling size is equal to 28x28, which means that only the pixel with the strongest activation is selected and inputted to the classifier. Hence, what it is called feature, actually is only a value of luminance level.

For analysing how the model 20201008T211004_dataset1.3_1L_nf1_1_ks1_3_ps1_28_ne _200_s_10 is using the single feature it has learnt, an option is to substitute the classification part of the network which uses a multi-layer perceptron for another algorithm of classification that has a higher level of interpretability, such as a decision tree. Observing how the tree is performing the classification enables by analogy, find out how the network is working as well as dealing with the trade-off between complexity and accuracy. Figure 4.12 shows the decision tree that has been built taking the feature learnt by the network, *feature_1*, as input.

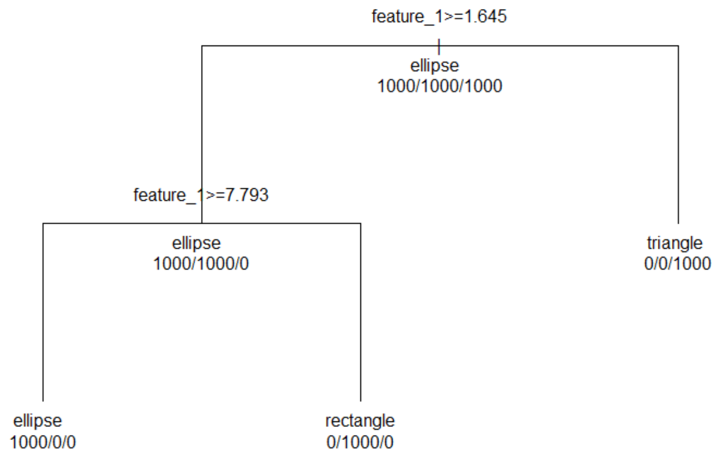

Figure 4.12. Decision tree using the feature extracted by the minimum model for dataset 1.3: 20201008T211004_dataset1.3_1L_nf1_1_ks1_3_ps1_28_ne_200_s_10.

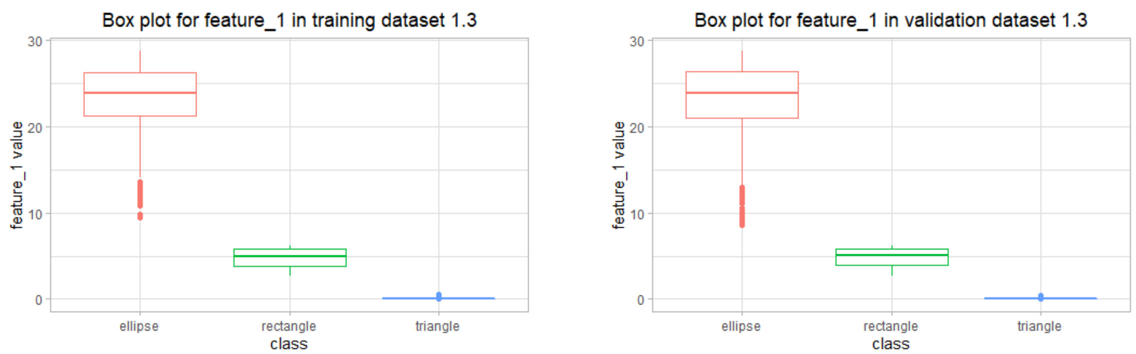

Figure 4.13. Boxplots showing values of the feature extracted depending on the class by the minimum model for dataset 1.3, 20201008T211004_dataset1.3_1L_nf1_1_ks1_3_ps1_28_ne_200_s_10, in both training and validation.

As it can be seen the tree is able to reach the same levels of accuracy than the network in a very simple way, what leads to confirm that the network is working in a similar way. Figure 4.13 shows two boxplots with the values taken by the feature in both training and validation datasets. It can be observed how the classes are completely separable by the luminance value of that feature. For ellipses, feature_1 takes values between seven and twenty-eight approximately, for rectangles, around five and for triangles almost zero. That is the reason why the tree, without increasing its number of parameters excessively, just with two "cuts" of the input space, is able to classify the images correctly. If the value of feature, that is, the luminance value of the selected pixel, is less or equal than 1.645, the image is a triangle. If the luminance value is between 1.645 and 7.793, the image is a rectangle, whereas if the value is higher than 7.793, the figure is an ellipse.

To go in depth in the study of features meaning, it is interesting to build some synthetic features that are thought to be useful for the network to classify the images. These are the features that intuitively a human will consider that the network can be extracting, some of them are complementary. It is important to take into account that black pixels are the ones that have a luminance value equal to 0 and white pixels the ones that have a luminance value equal to 1:

- sum_white_gray_pixels: sum of the luminance values of white and gray pixels.
- white_gray_pixels: count of the number of pixels that are either white or gray.
- black_pixels: count of the number of black pixels.
- white_pixels: count of the number of white pixels.
- gray_pixels: count of the number of gray pixels.
- pct_gray_vs_black_pixels: percentage of the number of gray pixels over the number of black pixels.
- pct_gray_vs_white_pixels: percentage of the number of gray pixels over number the of white pixels.
- pct_white_gray_pixels: percentage of the number of white pixels over the number of gray pixels.
- pct_black_pixels: percentage of number of black pixels over the total number of pixels in the image, that is 784.
- pct_white_pixels:percentage of number of white pixels over the total number of pixels in the image, that is 784.
- pct_gray_pixels: percentage of number of gray pixels over the total number of pixels in the image, that is 784.

With these features, a new decision tree is built, trying to find an alternate solution to the problem. This method imitates the traditional way of performing image classification that was used before the development of Deep Learning, where features from images that were considered important by the data scientist were extracted manually using techniques like SIFT. Nevertheless, in this case with the features that have been built the tree is not able to perform the classification correctly. This gives an insight of the power that the feature extraction part of the CNN has, being able to find features that go unnoticed by humans. Figure 4.14 shows the tree that has been built, using a pruning level of 0.01. This tree achieves a training accuracy of 0.893 with a confidence interval of 95% equal to (0.8817, 0.902) and a validation accuracy of 0.8812 with a confidence interval of 95% equal to (0.873, 0.889). The features more important for the model are pct_gray_vs_black_pixels and gray_pixels, which can be an indicator that the tree is focusing on the borders of figures to make its decisions.

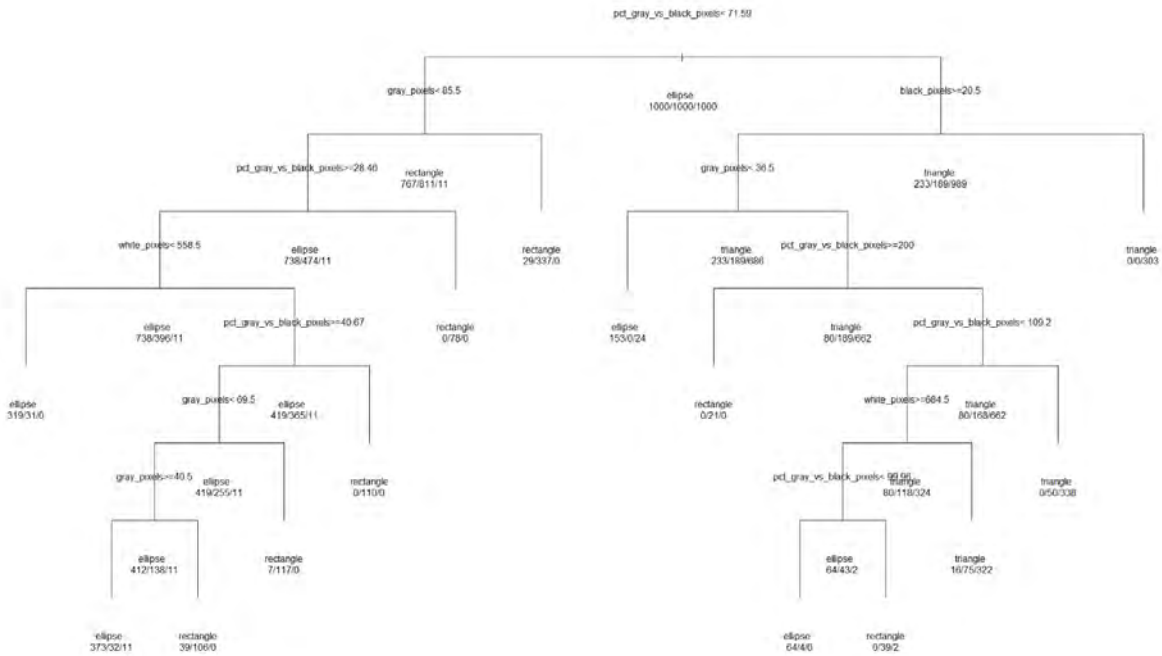

Figure 4.14. Decision tree using the sythetic features extracted manually for dataset 1.3.

In addition, some regression trees are built in order to check if the feature extracted by the model can be explained by the rest of synthetic features. Nevertheless, the results obtained are not good, which means that the feature that the CNN has learnt is unlike any other feature that intuitively a human would expect. All in all, with the luminance information of one single pixel, the CNN is able to differentiate among the three classes. Now, it is turn to see where exactly is that pixel placed in the images, which leads to the second approach used in this analysis: the study of the physical meaning of the model.

Since the values taken by the feature have been already studied, analysing the position where that pixel is located in the image, can help to gain a deeper understanding of how the model extracts that specific feature for each class and how it is crucial to differentiate among them. For this reason, two kinds of representations are developed. On the one hand, feature maps are plotted, so that the parts of the image that are considered most important by the network, are highlighted. In addition, feature maps give valuable insight into the the kind of kernel that the CNN is applying (for detection of color, border, texture, objects parts, etc.). On the other hand, a special kind of representation is built. Following the learning process of the CNN, once the feature map has been extracted, the pixel (or pixels) that obtains the maximum luminance values are marked over the original image. In this way, the location of the specific pixel that has been selected by the pooling layer can be examined. These representations are included at the end of this section.

There is no need to compare different pixels within an image since only one feature is selected by the model, contrary to what happens in a common heat map, where areas of the image are highlighted in a different colour depending on the importance it has to the CNN. That is the reason why in maximum activation pixel representations a single random colour has been assigned to each class: blue for ellipses, red for rectangles and yellow for triangles.

Both feature maps and maximum pixel representations are performed over several different samples so the results can be compared not only among different classes but within images from the same class. Thus, patterns can be extracted. Figures 4.19 and 4.20 show these representations for the ellipses. It can be observed in the feature maps (Figure 4.19) that the filter is performing border detection, extracting a very specific border: the top border of the ellipse. In addition, the border is not strictly horizontal but it has a certain slope, as if the filter was extracting a diagonal border. Looking at the maximum pixel activation, it is always located in the top right border of the ellipse.

In Figure 4.25 the same kind of plots but with more detail of several images can be observed. In this way, the exactly position of the maximum pixel can be extracted and, as it has been seen in the other representations, this pixel is always located in the top right part of the image, for example in the coordinates: (18,6), (18,5), (16,9), (16,10) and so on. And that specific pixel from the border of the ellipse has an activation value different from the features extracted in rectangles and triangle images, hence it is characteristic of the ellipses.

0.005	0.121	-0.094
-0.020	-0.213	0.187
-0.745	-0.629	-0.380

20201008T211004_dataset1.3_1L_nf1_1_ks1_3_ps1_28_ne_200_s_10

Figure 4.15. Filter learnt by the CNN minimum model selected for dataset 1.3. Luminance representation on the left and the exact luminance values on the left.

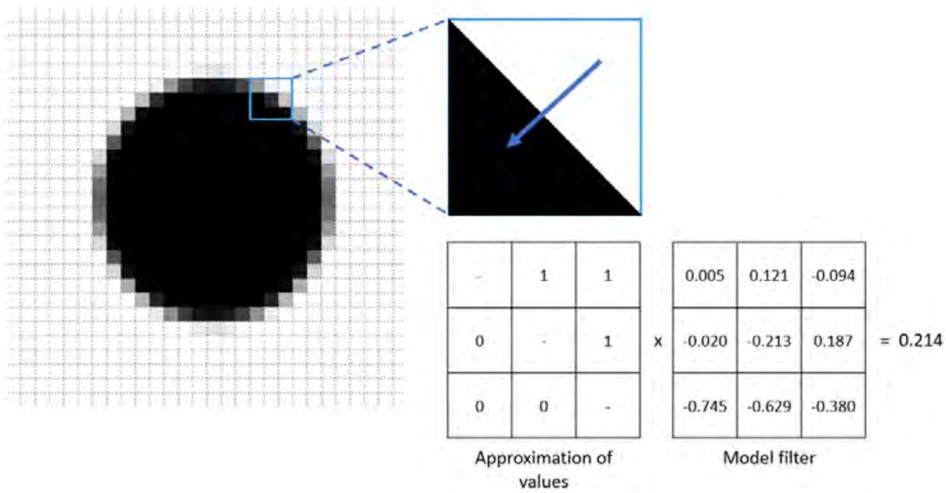

Figure 4.16. Example of dataset 1.3 minimum model filter applied to an ellipse image. The result of the convolution is an approximation, not the real one. The arrow shows the specific direction of colour change that activates the filter and the hyphen represents a value of gray.

Looking at the filter representation shown in Figure 4.15 it can be confirmed that the border that the filter extracts is slightly diagonal, from the top left part of the image to the bottom right. When there is a change from white to black colour where the white colour is on the top right part of the image and black colour on the bottom left, the neurons in that part of the image activate. This is due to the multiplication (convolution) of black pixels (whose luminance value is 0) by negative values of the filter plus the multiplication of white pixels (with luminance value equal to 1) by positive values of the filter, giving as a result a positive value (a neuron activation). Figure 4.16 shows an example of this filter behaviour.

In the case of rectangles, it happens something similar. Figure 4.21 shows the activation maps for different rectangle samples and it can be observed that the filter is focusing on the top border of the figure, just as with the ellipses, but in this case the border is completely horizontal and it does not have a slight slope. Looking at representation shown in Figure 4.21 it can be observed that several pixels from the image take the maximum activation value, more specifically, the ones in the border of the rectangle. The same fact can be observed in Figure 4.26 with more detail.

If the behaviour of the filter is analysed, as in the example of Figure **??**, it can be observed that there is also a change of colour from white to black but in this case without slope in another direction. It also activates the neurons, giving a positive value as result, but not as strong as the one obtained for the ellipses. That is the reason why the feature extracted is stronger for ellipses than it is for rectangles.

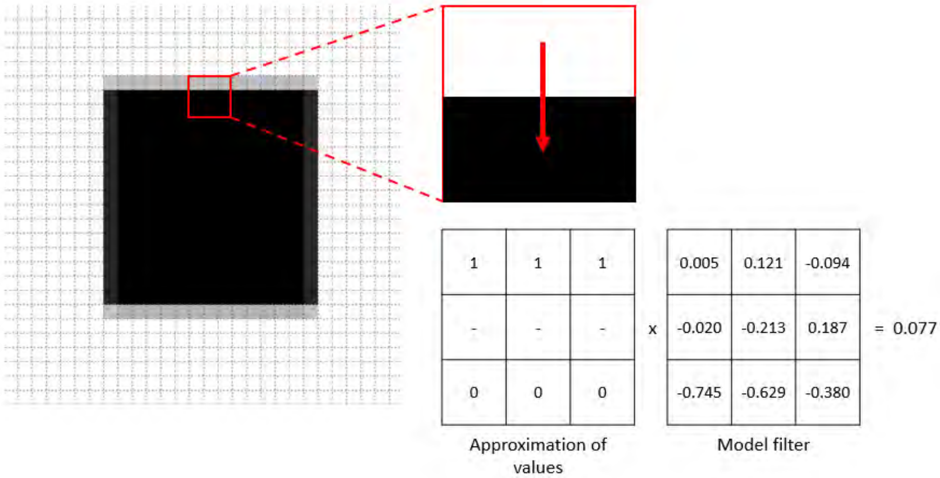

Figure 4.17. Example of dataset 1.3 minimum model filter applied to a rectangle image. The result of the convolution is an approximation, not the real one. The arrow shows the specific direction of colour change that activates the filter and the hyphen represents a value of gray.

Finally, in the case of triangle shapes, almost all feature maps resulting after applying the filter have their values set to zero. This means that there is not any neuron that activates when the figure is a triangle. The fewer feature maps in which all the values are not zero are shown in Figure 4.23. Even in this images it can be appreciated that only one pixel is activated, meaning that, unlike the other cases, the filter is not able to recognise a border. Looking at images from Figure 4.24, it seems that the model is still trying to find a border, even if it is not clear for the filter that is being used, since the pixels that are activated are located in the edges of the triangles.

Figure 4.27 shows with more detail that there is not a clear pattern in the pixels that are activated for triangles, since they are located in a different place for each case. Nevertheless, it seems that for triangles with a big size, the filter always recognise a pixel in the left border whereas for triangles of small size it recognise a pixel in the right border. The first case can be something similar to what happens with ellipses, a more slight slope is located in the bottom right corner of the triangle. Whereas in the second case, it happens something similar as with rectangles, since triangles are so little that its middle part can be considered a horizontal line. However, as this behaviour is not repeated for a considerable amount of images, it cannot be considered a valid general conclusion.

There is no change from color white to black neither in an horizontal border or a slightly diagonal border. Nevertheless, there is change from black to white in an horizontal border, that is, the base of the triangle, and change from white to black in a diagonal border but with an abrupt slope: the right side of the triangle; but it is not what the filter is detecting. Figure 4.18 shows an example in which it can be observed why the feature values for the rectangle are the lower ones, and in most of the cases, the feature will be equal to zero. Nothing seems to activate the neurons for triangle images.

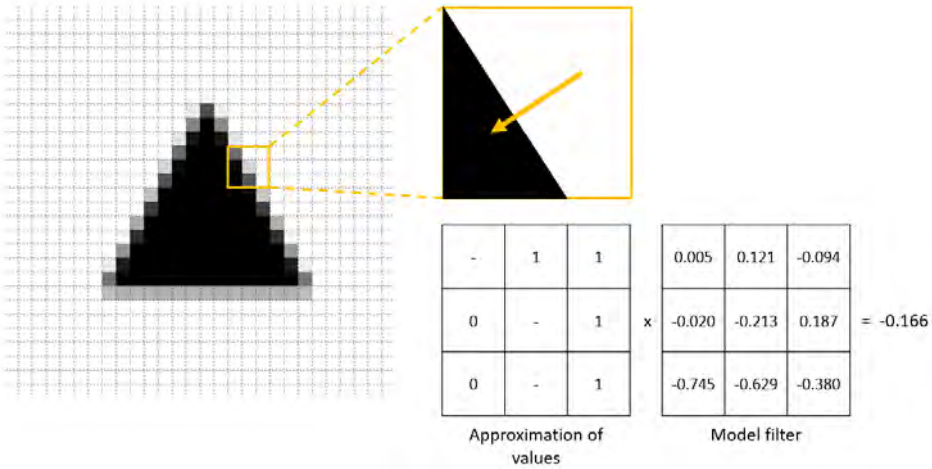

Figure 4.18. Example of dataset 1.3 minimum model filter applied to a triangle image. The result of the convolution is an approximation, not the real one. The arrow shows the specific direction of colour change that activates the filter and the hyphen represents a value of gray.

All in all, the filter, with its negative and positive values, is extracting a horizontal border and a specific slope in a determined direction. It is detecting a border not a single pixel, transition from white to black; in that specific order, not from black to white. Due to this fact, in both ellipses and rectangles only the top border of the figures is activated. In the same way, as triangles do not have this kind of transition with the same slope as either the ellipse of the rectangle, in almost all the images there is no strong activations. In this way the CNN has managed to obtain a feature with which is able to classify the images correctly but that is not intuitive for the human eye.

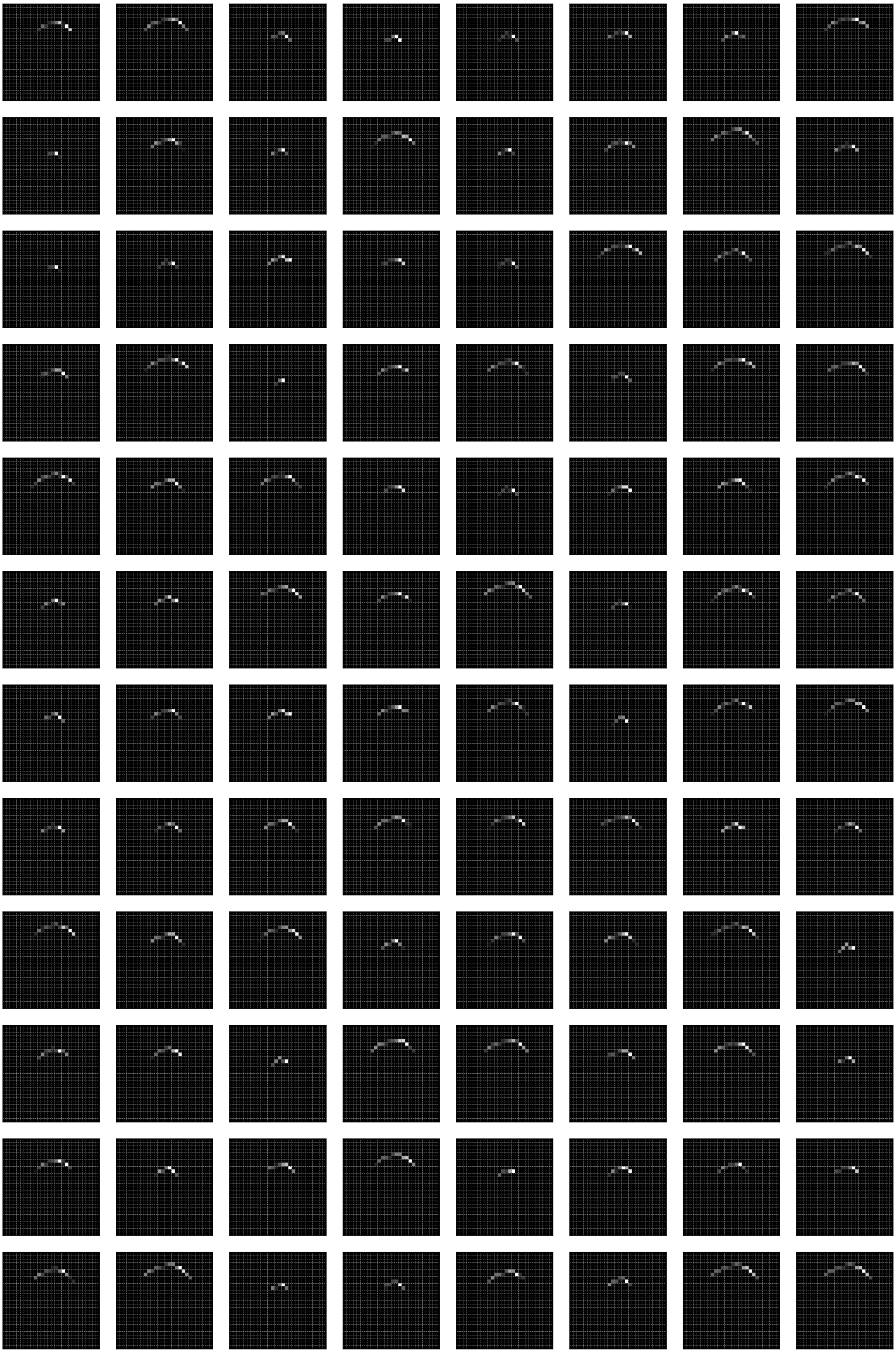

Figure 4.19. Feature maps of several ellipses samples from dataset 1.3. Best viewed in electronic form, zoom in to distinguish the pixel grid.

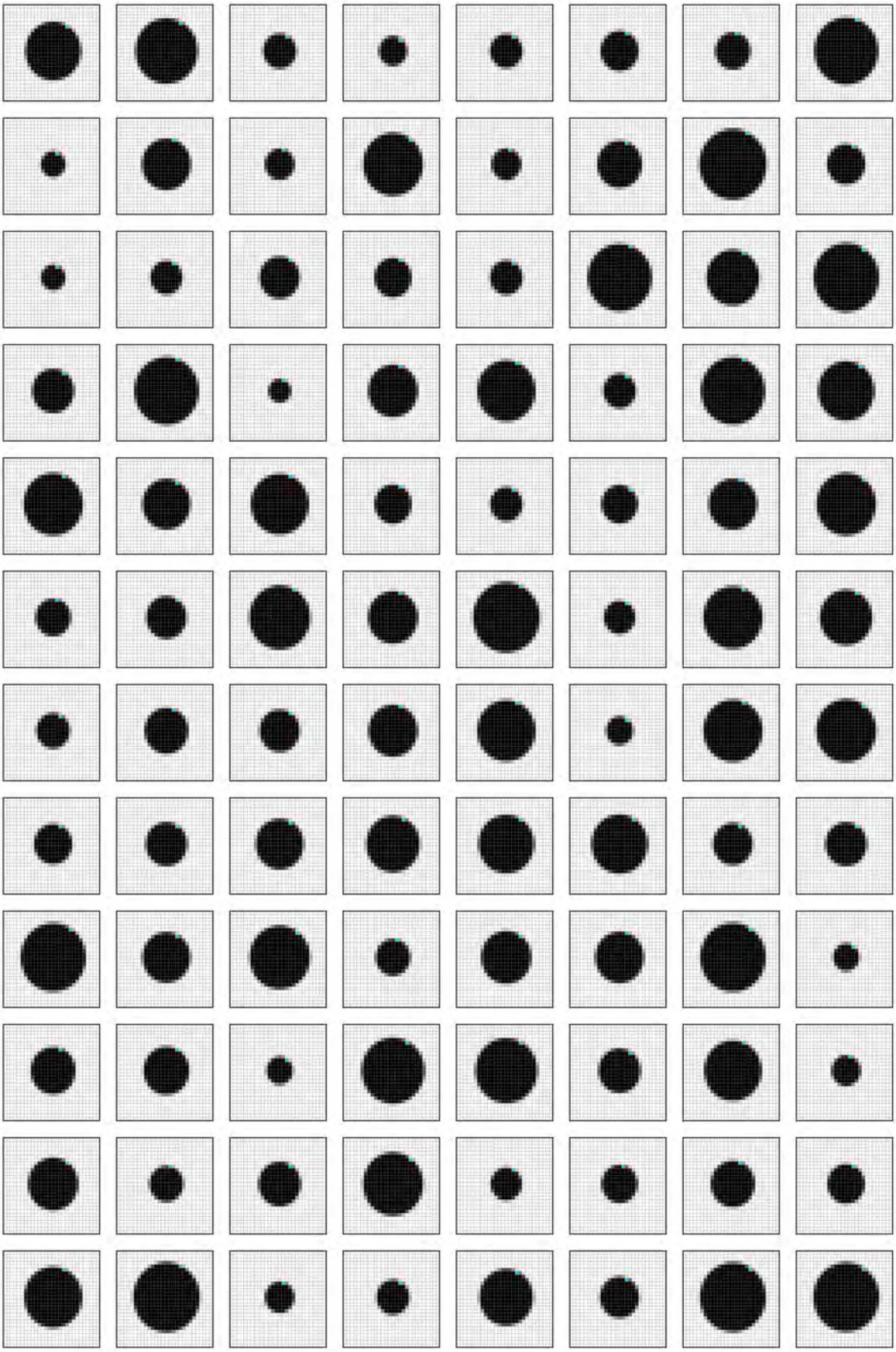

Figure 4.20. Original images of several ellipses from dataset 1.3 with maximum activation pixel in colour blue. Best viewed in electronic form, zoom in to distinguish the pixel grid.

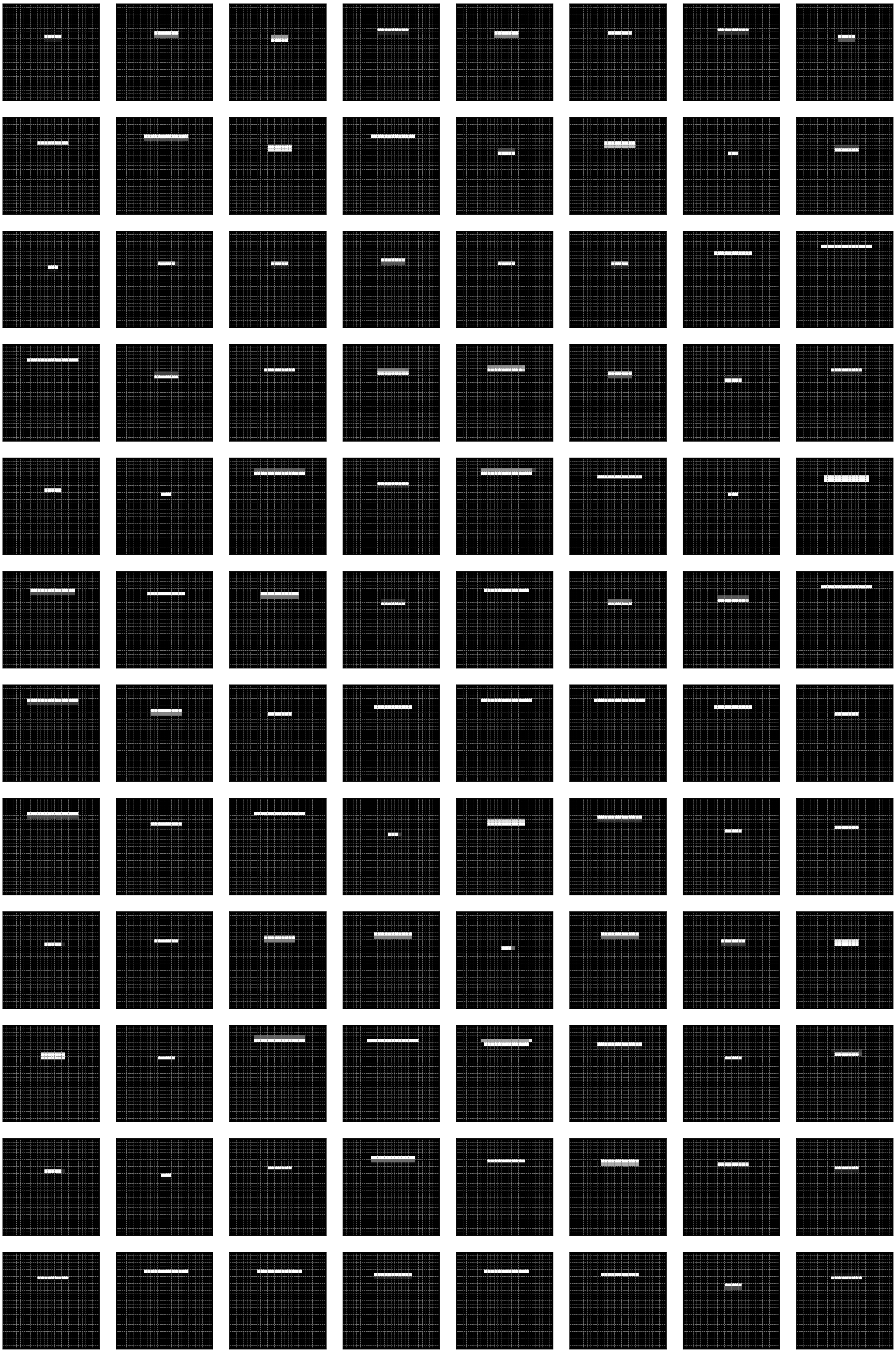

Figure 4.21. Feature maps of several rectangle samples from dataset 1.3. Best viewed in electronic form, zoom in to distinguish the pixel grid.

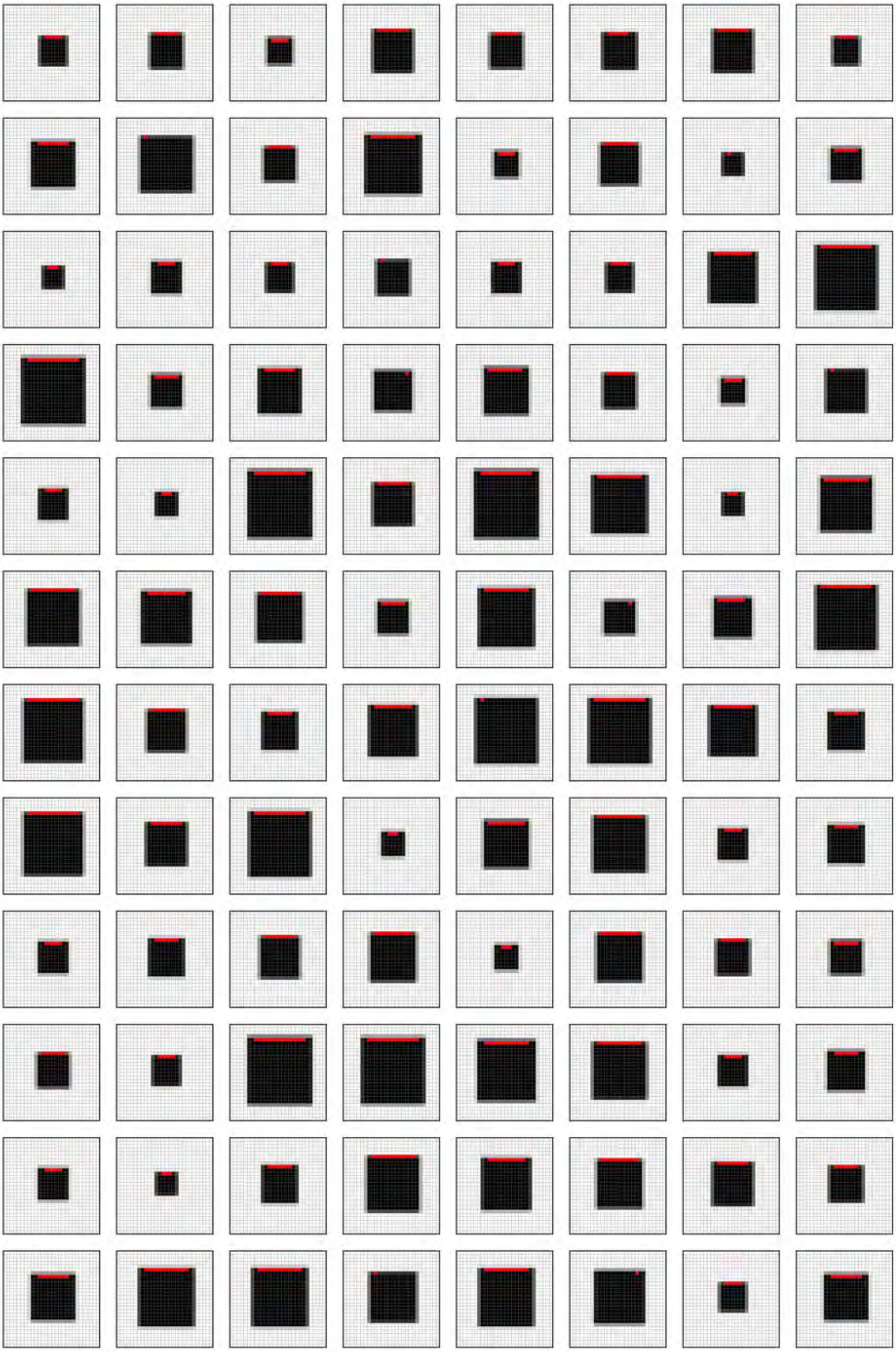

Figure 4.22. Original images of several rectangles from dataset 1.3 with maximum activation pixel in colour red. Best viewed in electronic form, zoom in to distinguish the pixel grid.

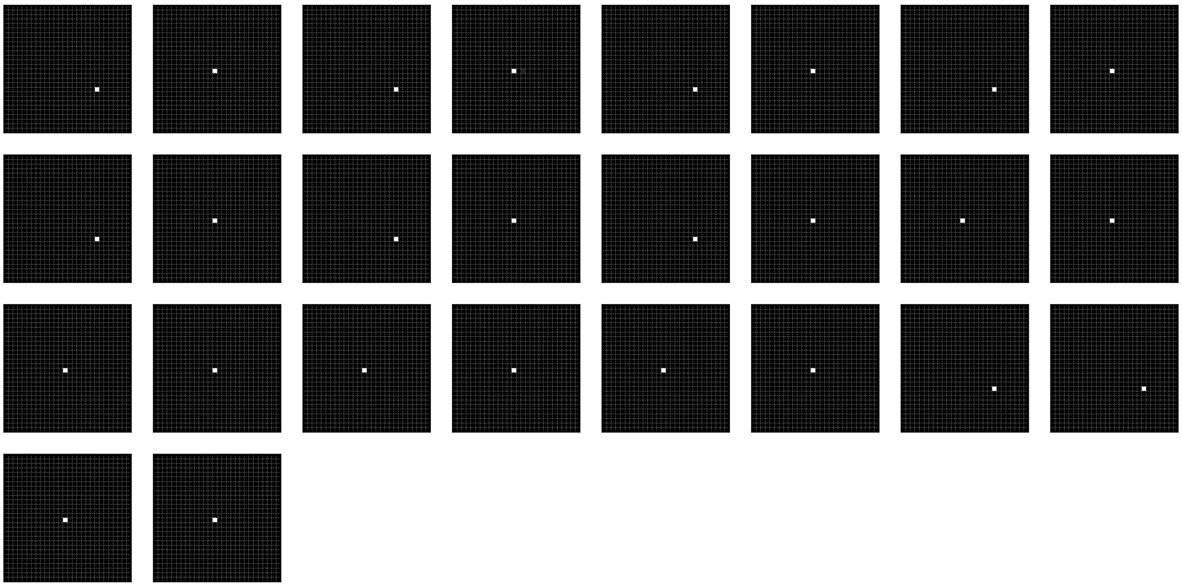

Figure 4.23. Feature maps of several triangle samples from dataset 1.3. Best viewed in electronic form, zoom in to distinguish the pixel grid.

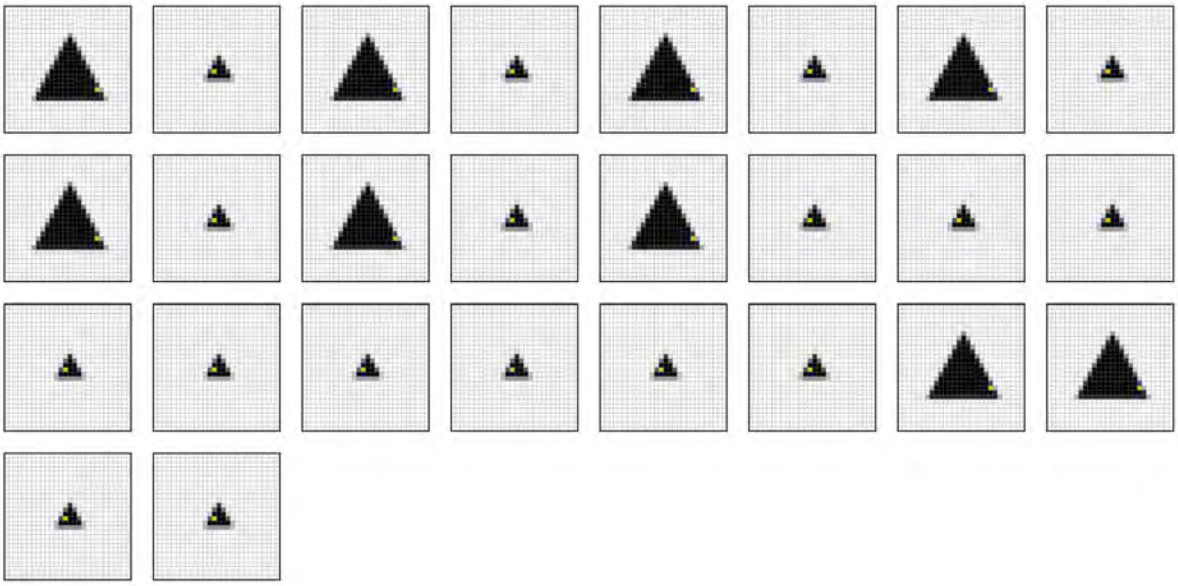

Figure 4.24. Original images of several triangles from dataset 1.3 with maximum activation pixel in colour yellow. Best viewed in electronic form, zoom in to distinguish the pixel grid.

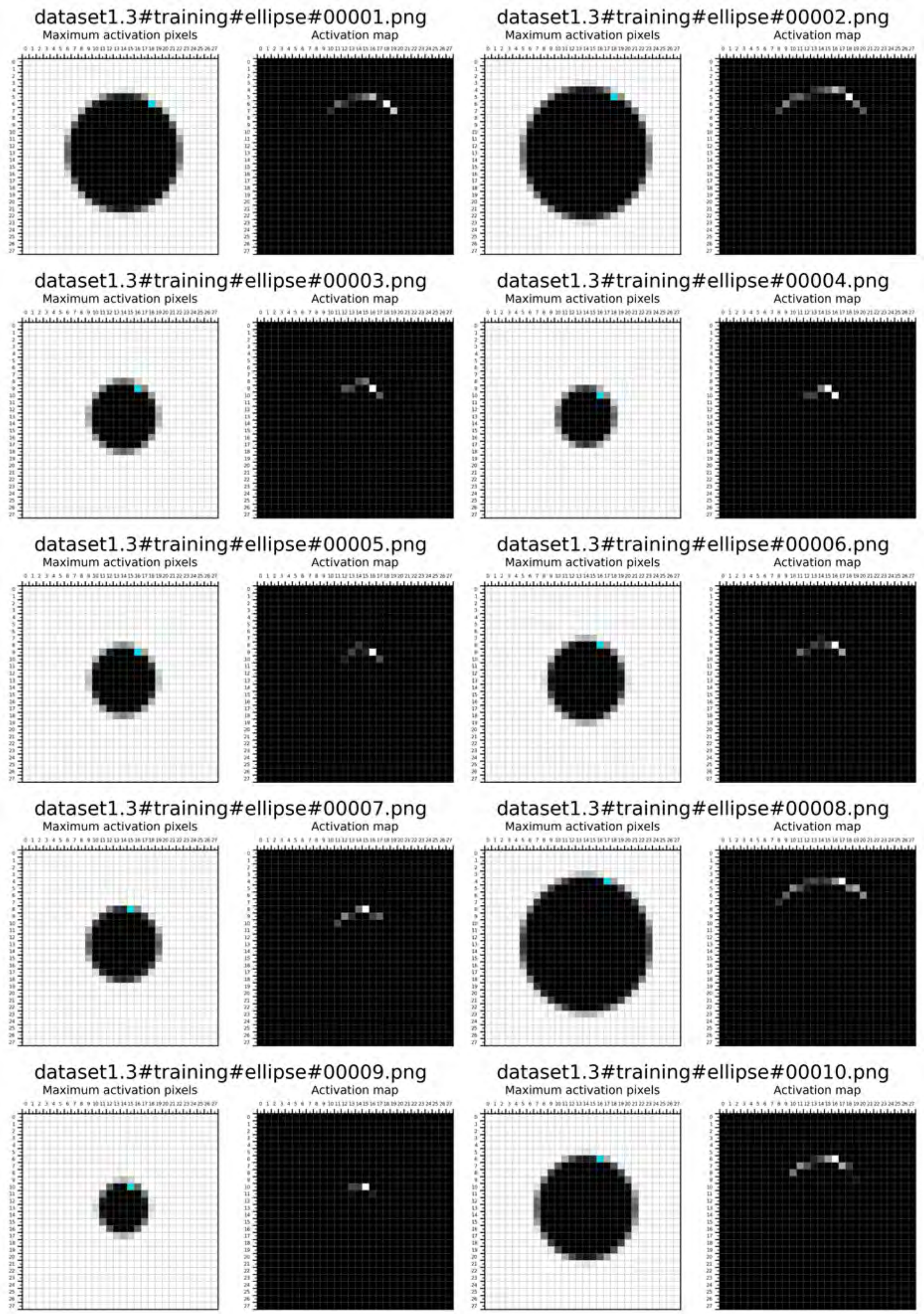

Figure 4.25. Both feature map and maximum activation pixel representations of several ellipses samples from dataset 1.3. in a detailed form. Best viewed in electronic form, zoom in to distinguish the pixel grid.

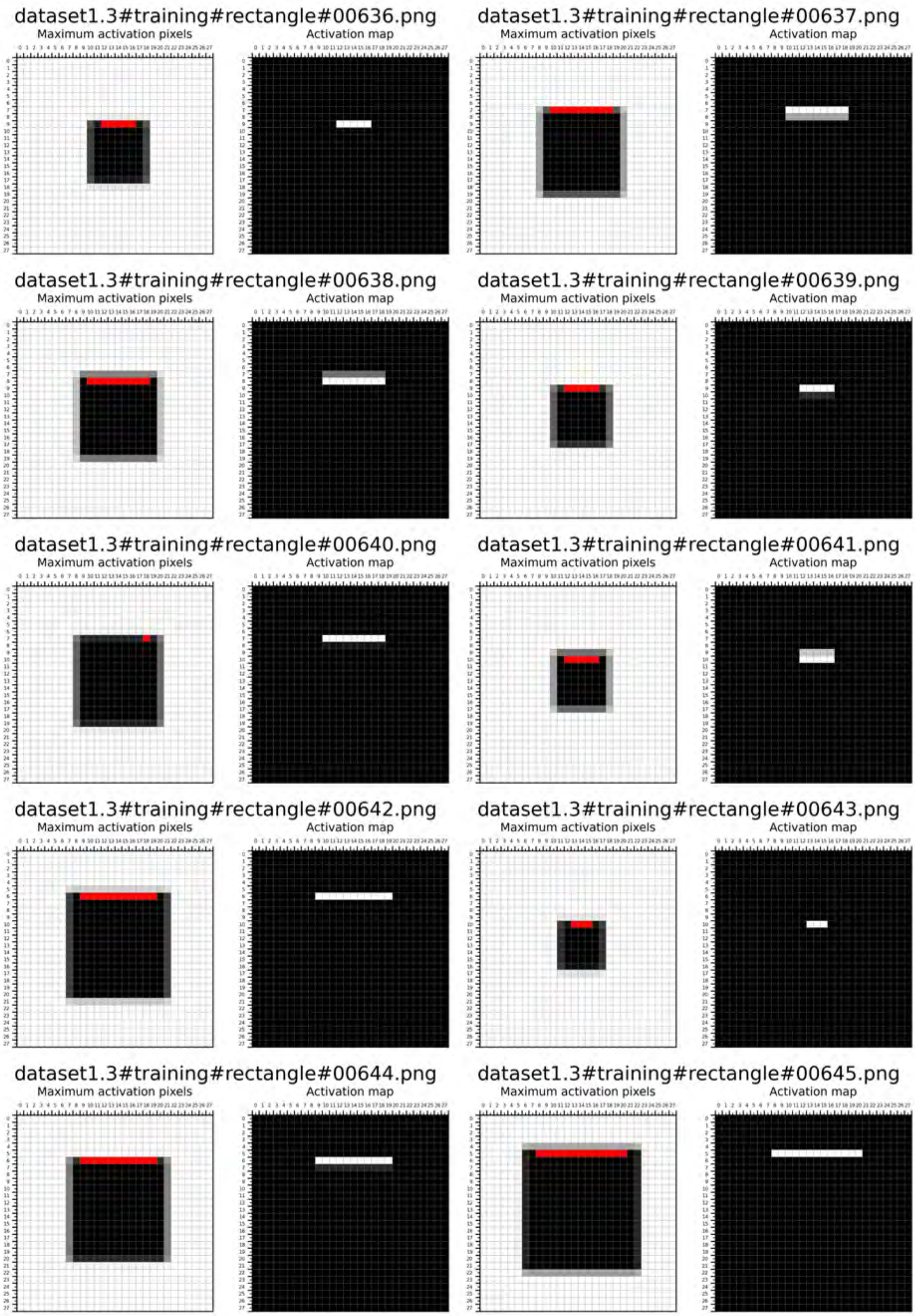

Figure 4.26. Both feature map and maximum activation pixel representations of several rectangles samples from dataset 1.3. in a detailed form. Best viewed in electronic form, zoom in to distinguish the pixel grid.

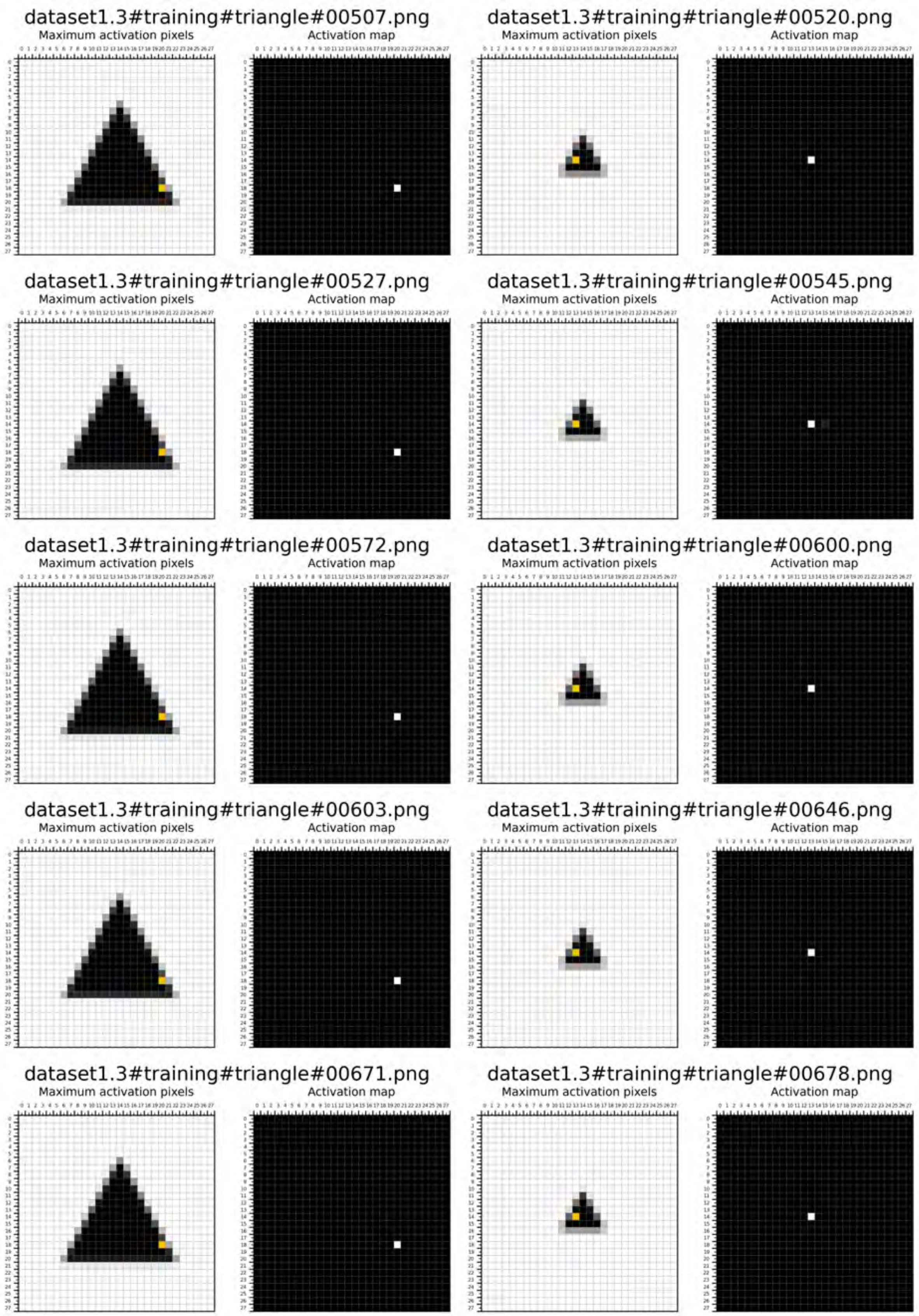

Figure 4.27. Both feature map and maximum activation pixel representations of several triangles samples from dataset 1.3. in a detailed form. Best viewed in electronic form, zoom in to distinguish the pixel grid.

4.2.3.2. Rotation problem without noise

Just as the analysis of the scale problem has been performed in the previous section (Section 4.2.3.1), for the rotation problem the two approaches are used again. In the first place, features are deeply analysed making use of decision trees to understand how the classification part of the network is working. After that, a more visual study is carried out, plotting activation/feature maps and the specific feature for several input samples together with representation of filters.

Related to the feature analysis, in this case, the model is also extracting one single feature, what means that the network classifier has to be able to perform the classification with only a luminance level as input. Initially, it would be logical to think that the minimum model selected in this case, 20201009T155101_dataset2.1_1L_nf1_1_ks1_3_ps1_28_ne_200_s_4, is working in the same way as the minimum model selected for dataset 1.3 is working (Section 4.2.3.1). Nevertheless, the decision tree built for the feature extracted by the model in this case shows that the classifier is unable to classify correctly all the images, as it can be seen in Figure 4.28. Final leaves of the tree are not totally pure, since there is no clear separation between values of the feature for the three classes or, what is the same, given the luminance value, the model is not able to tell with total certainty to which class the sample belongs. The tree reaches a training accuracy of 0.727 within the confidence interval of (0.7101, 0.7429) and a validation accuracy of 0.717 within the confidence interval of (0.705, 0.728).

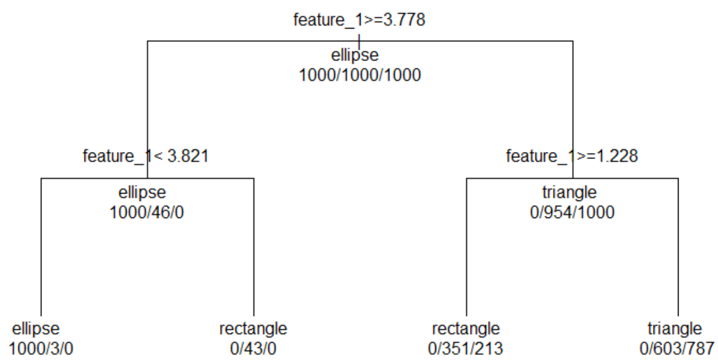

Figure 4.28. Decision tree using the feature extracted by the minimum model for dataset 2.1: 20201009T155101_dataset2.1_1L_nf1_1_ks1_3_ps1_28_ne_200_s_4.

This fact is better observed in Figure 4.29 which shows training and validation feature values in boxplots. The only class that the tree is able to differentiate better from the others is the ellipse since, as it can be seen in the boxplots, its luminance level always takes the same value regardless the sample. It is important to take into account this fact, since it can be an indicator that dataset 1.2 is biased. The decision tree manages to separate these images from the rest but there are three images that are considered to be ellipses when they actually are rectangles. This can be due to the fact that there are some images of rectangles that have similar values as the one in the ellipse as it can be seen in Figure 4.29. Both rectangles and triangles are the classes that the classifier finds more difficult to separate, since the feature takes almost the same range of values. The CNN behaves in a similar way to the decision tree, being unable to achieve a

100% level of accuracy, reaching a 65%, even a 5% less than the decision tree (remember that it is not being used all the multilayer perceptron classification power). Similar to the decision tree, the CNN model classifies correctly all ellipses.

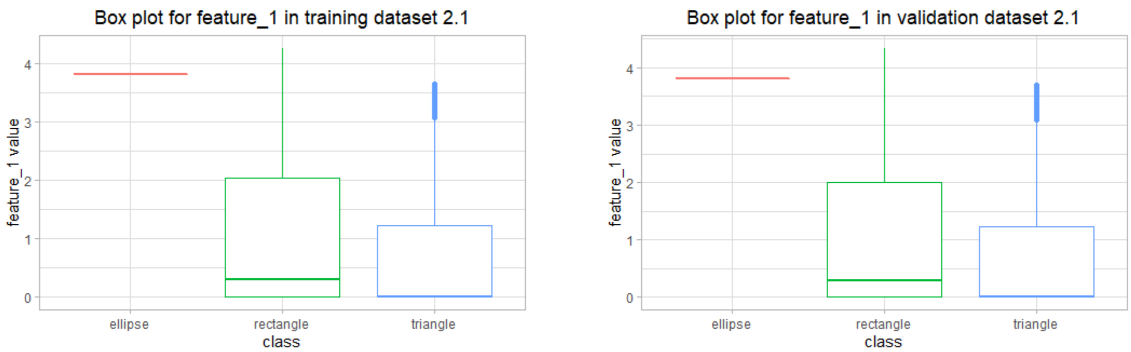

Figure 4.29. Boxplots showing values of the feature extracted depending on the class by the minimum model for dataset 2.1, 20201009T155101_dataset2.1_1L_nf1_1_ks1_3_ps1_28_ne_200_s_4, in both training and validation.

Once a decision tree has been made using the feature extracted by the model, other trees are trained with the synthetic features built from the model. In this way, it can be observed if there are alternative solutions to the problem, comparing them to the one that only uses the feature extracted. The features that have been generated synthetically are the same ones as the ones used in the previous section (Section 4.2.3.1) but they are included here to avoid the reader returning to previous pages:

- sum_white_gray_pixels: sum of the luminance values of white and gray pixels.
- white_gray_pixels: count of the number of pixels that are either white or gray.
- black_pixels: count of the number of black pixels.
- white_pixels: count of the number of white pixels.
- gray_pixels: count of the number of gray pixels.
- pct_gray_vs_black_pixels: percentage of the number of gray pixels over the number of black pixels.
- pct_gray_vs_white_pixels: percentage of the number of gray pixels over number the of white pixels.
- pct_white_gray_pixels: percentage of the number of white pixels over the number of gray pixels.
- pct_black_pixels: percentage of number of black pixels over the total number of pixels in the image, that is 784.
- pct_white_pixels:percentage of number of white pixels over the total number of pixels in the image, that is 784.
- pct_gray_pixels: percentage of number of gray pixels over the total number of pixels in the image, that is 784.

Figure 4.30 shows two of the trees that have been trained. Contrary to what happens with the scaling problem, in this case the decision trees that use the synthetic features are able to classify perfectly the images. It is important to take into account that these two trees are equivalent since the features are complementary, that is, if to the total number of pixels, 784 the number of black pixels is subtracted, the result is the sum of the number of white and gray pixels. These features in fact are counting the area of figures, and, since they does not change its size in this case, the models manage to achieve a 100% of accuracy. Therefore, there are alternate features that are able to differentiate among classes, which does not mean that they are the same that have been extracted by the network but there are many possible solutions.

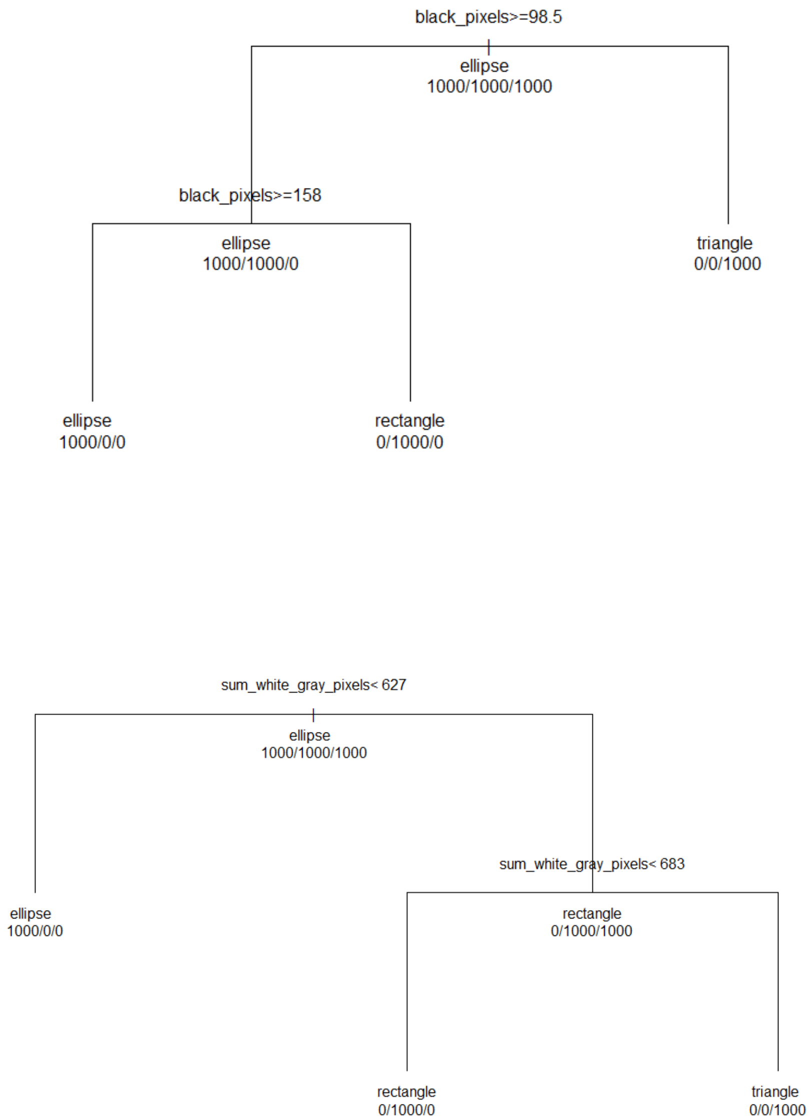

Figure 4.30. Decision trees using the sythetic features extracted manually for dataset 2.1.

In a similar way as it was performed with dataset 1.3, some regression trees were built trying to explain the model's feature with the synthetic ones. None of them give good results, so the conclusion is that the model is focusing on a feature that for humans would be intuitively very hard to detect. All in all, with the feature extracted by the minimum model it is not possible to classify correctly the total number of images, but the majority of them. In addition, both the features representation and the model's behaviour are indicators that a bias may have been introduced when creating the dataset 2.1. This fact will be studied more in detail with the second approach: the analysis of the physical meaning of the model.

The same kind of representations as the ones showed in the previous section are performed for dataset 2.1: both activation maps and maximum activation value. However, in this case, it is important to take into account that not all images of rectangles and triangles are classified correctly by the model. Therefore, in these classes plots are generated both for images correctly and wrongly classified so the differences between them can be observed.

Figures 4.36 and 4.37 show the representations for the ellipses. It can be observed in the feature maps (Figure 4.36) that always the same two pixels are activated. Moreover, looking at the maximum activation pixel (Figure 4.37), it can be appreciated that the pixel is always located on the border of the figure. It seems like the network has learnt a specific pixel from the border because it is characteristic of the ellipses, i.e. that pixel only appears when the image is an ellipse. This representations are consistent with the boxplots from Figure 4.29 in which it can be seen that the feature for ellipses always takes the same value, no matter the input sample. In particular, the exact location of that pixel can be observed in Figure 4.46 which shows more detailed representations: (10,8).

This fact is related to one of the questions that arose in previous sections: "is dataset 2.1 biased?" and the answer is yes. It can be observed that the ellipses class contain a bias since, keeping the size constant and just rotating the ellipse gives as a result always the same image as this ellipse has infinite symmetry axes. In fact, this model has taken the pixel (10,8) as the characteristic one for ellipses but there are many more, provided that the pixel is located on the border and does not appear in neither rectangles nor triangles. In order to check, the same representations are built for a model that achieves a similar level of accuracy: 20201009T164248_dataset2.1_1L_nf1_1_ks1_3_ps1_28_ne_200_s_10. Figure 4.47 show that the pixel that is activated in this case is (19,8).

The filter representation is shown in Figure 4.31 and it can be observed that it tries to extract a diagonal border with positive slope, as it has positive values in the left part, contrary to what happens in the case of dataset 1.3. This fact can be better understood with the example shown in Figure 4.32 where the only pixel that activates after filter convolution is the one in the top left corner with an undetermined level of gray.

0.040	-0.241	-0.642
-0.745	-0.491	-0.471
0.004	-0.782	-0.120

20201009T155101_dataset2.1_1L_nf1_1_ks1_3_ps1_28_ne_200_s_4

Figure 4.31. Filter learnt by the minimum model selected for dataset 2.1. Luminance representation on the left and the exact luminance values on the left.

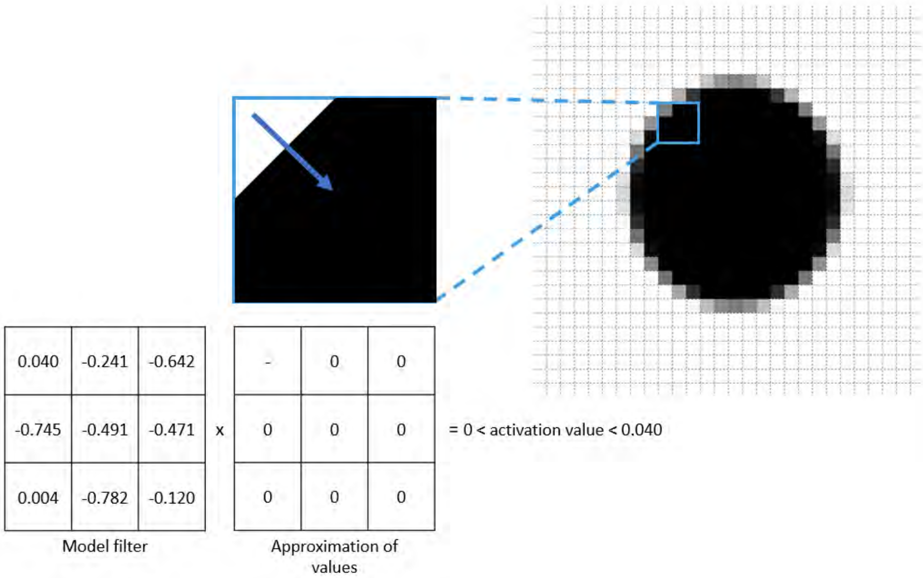

Figure 4.32. Example of dataset 2.1 minimum model filter applied to an ellipse image. The result of the convolution is an approximation, not the real one. The arrow shows the specific direction of colour change that activates the filter and the hyphen represents a value of gray.

In the case of rectangles, some samples that are correctly classified by the model are shown in Figure 4.38. It is more obvious in these images that the filter is extracting diagonal borders of the figures, from the bottom left part of the image to the top right. Since the rectangles in this case are rotating, the slope of the border changes but approximately it has 45°. Looking at the same samples but with the representation of its maximum activation feature (Figure 4.39), it can be observed that, contrary to what happens in dataset 1.3, a single pixel of the border is activated. There is no a clear pattern of the pixel that activates, in some cases it is located in a corner of the rectangle, whereas in other samples it is found in the middle of the border. The locations of the maximum activation pixel are usually (11,10), (13,9) and (9,15), all located in the left half of the image as it can be seen in Figure 4.48. Moreover, focusing on the images that the models classifies incorrectly, shown in Figure 4.40, 4.41, and 4.49 it can be appreciated that there is not a clear pattern that enables to differentiate between correct and wrong classified images.

	PREDICTED		
KNOWN	ellipse	rectangle	triangle
ellipse	1000	0	0
rectangle	235	156	609
triangle	139	74	787

Training confusion matrix

	PREDICTED		
KNOWN	ellipse	rectangle	triangle
ellipse	2000	0	0
rectangle	454	309	1237
triangle	316	144	1540

Validation confusion matrix

Figure 4.33. Confusion matrix both for training and validation images for the minimum model selected for dataset 2.1.

Therefore, the model is classifying rectangle samples randomly. This means that when the image is a rectangle, the behaviour of the model it is similar to the tossing of a coin. This behaviour can be better understood looking at the confusion matrix for both training and validation data shown in Figure 4.33: when the image is a rectangle, the model the most part of the time classifies it as a triangle, and if not, as an ellipse; so the model is not learning to classify the rectangles correctly.

Looking at the example shown in Figure 4.34 it can be observed that a border with positive a positive slope of 45º approximately is activating the filter, since the positive value of 0.040 is added to two negative values that are very small (those two pixels in the image are dark gray so they have a value next to zero). The problem is that this feature can take a wide range of gray values as it can be appreciated in the rectangle boxplot (4.29) as it happens with triangles, which makes it very difficult to distinguish from one another.

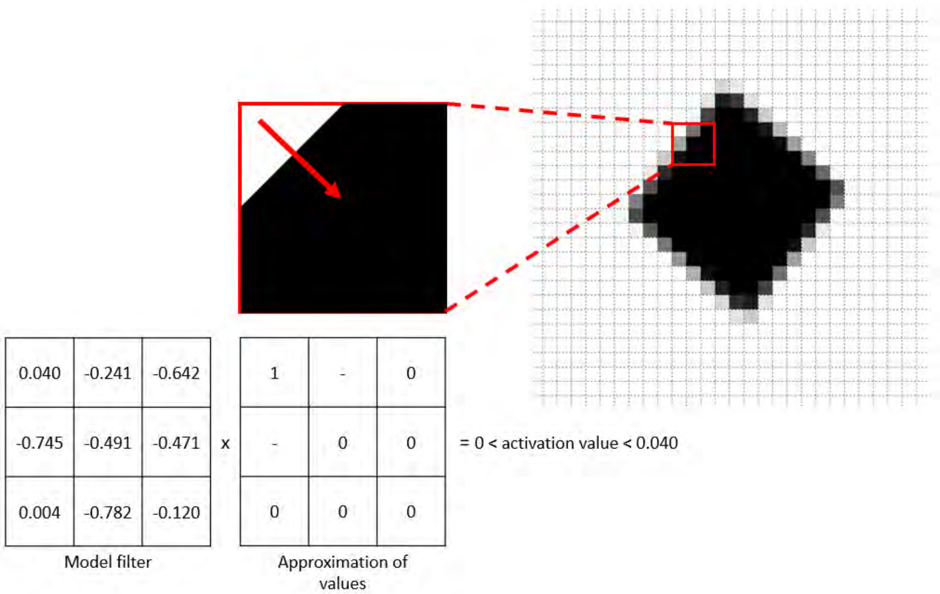

Figure 4.34. Example of dataset 2.1 minimum model filter applied to a rectangle image. The result of the convolution is an approximation, not the real one. The arrow shows the specific direction of colour change that activates the filter and the hyphen represents a value of gray.

In the case of triangles, the classification is performed by the model in a similar way as with the rectangles class. Figures 4.42 and 4.43 show both feature maps and maximum activation pixel representations in which it can be appreciated that the borders with a positive slope of

45°approximately are being extracted and only one pixel of that border is selected. In a similar way to what happened with rectangles, sometimes that specific pixel is located on a vertex of the shape and for other samples is located on the middle of a side. In order to observe the locations of the maximum pixel, Figure 4.50 shows that it is usually located in the coordinates (12,14), (15,12) and (9,14), which are located in the left half of the image as well. Observing the representations of the images that the model classifies wrongly, shown in Figures 4.44, 4.45, and 4.51. There is no difference from the samples that are classified correctly, so the model is again predicting a class for the triangles randomly.

The fact that the feature is always located in the left part of the image is due to the change that the filter is detecting. As shown in Figure 4.35 the filter is being activated when a diagonal border from bottom left to top right part of the image is detected, and this kind of borders are located in this part of the image. When there is a change of white to black along this direction, the convolution outputs a positive value of gray, as it is shown in the exaple of the triangle, similar to the rectangle example.

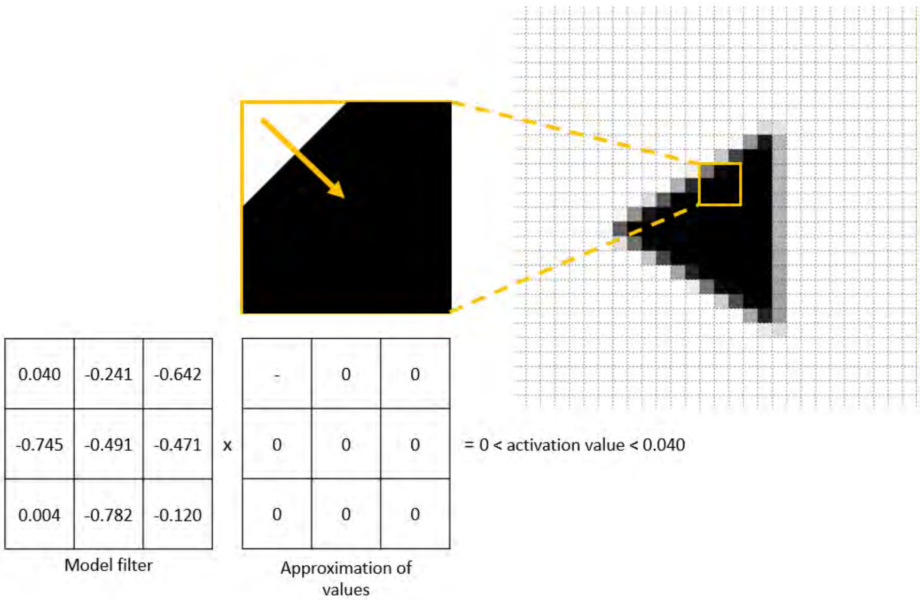

Figure 4.35. Example of dataset 2.1 minimum model filter applied to a triangle image. The result of the convolution is an approximation, not the real one. The arrow shows the specific direction of colour change that activates the filter and the hyphen represents a value of gray.

As a conclusion, a bias has been introduced unconsciously into dataset 2.1, because rotating the ellipses does not give as a result different images but the same one: ellipses are immutable to rotation. Therefore, the model has learnt to detect very well a single ellipse and, in this way, it is able to classify correctly the 33% of the samples. For the other two classes, the model is not able to find a specific feature so it classifies the images randomly, like the tossin of a coin, in this way, by probability, it manages to achieve another 33% of accuracy (since it is the half of 66%, the proportion of images that are left). For the next groups of experiments, new datasets are generated to avoid the bias and try to guide the training process so that the model learns actual features.

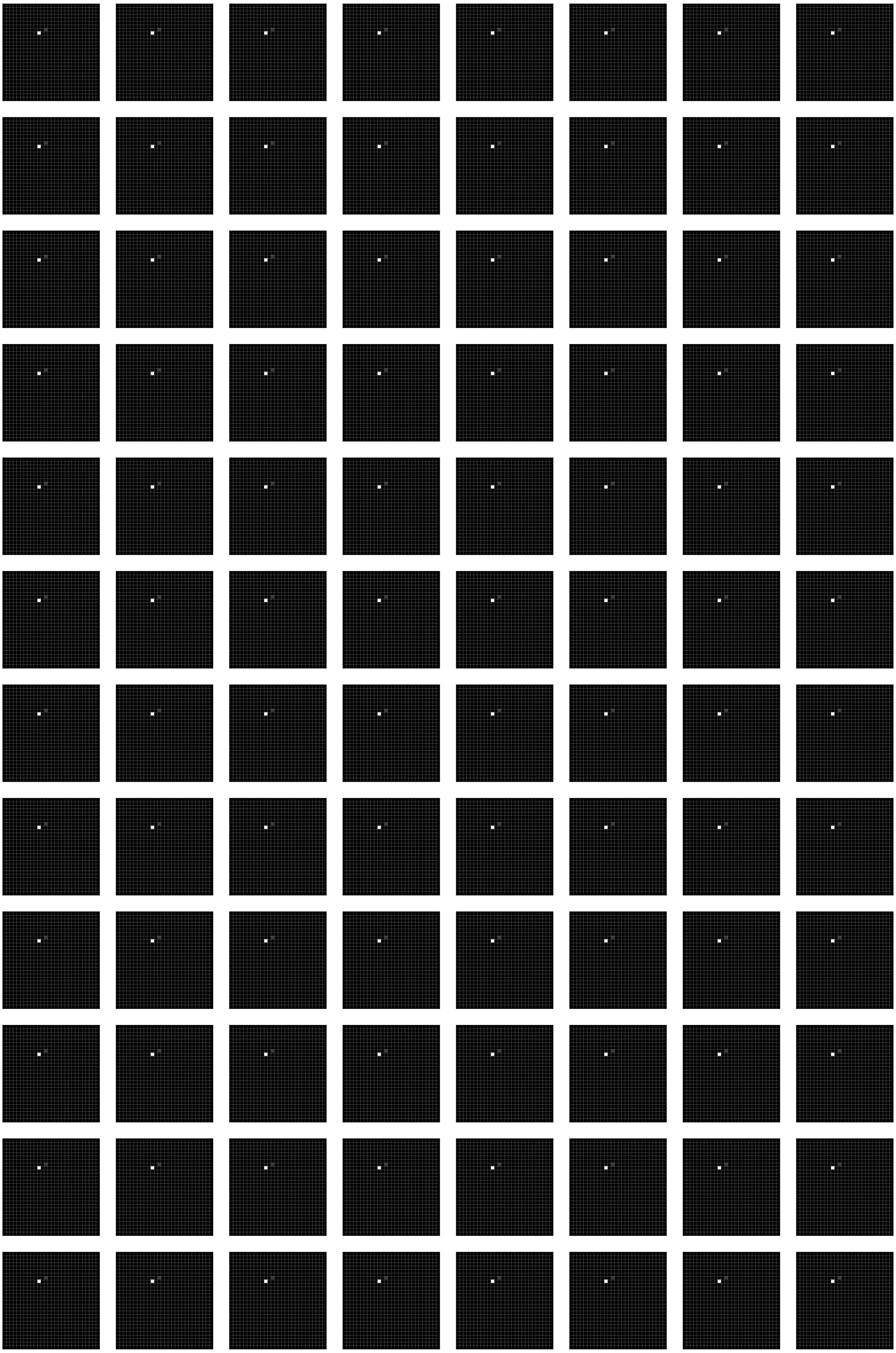

Figure 4.36. Feature maps of several ellipse samples from dataset 2.1 correctly classified. Best viewed in electronic form, zoom in to distinguish the pixel grid.

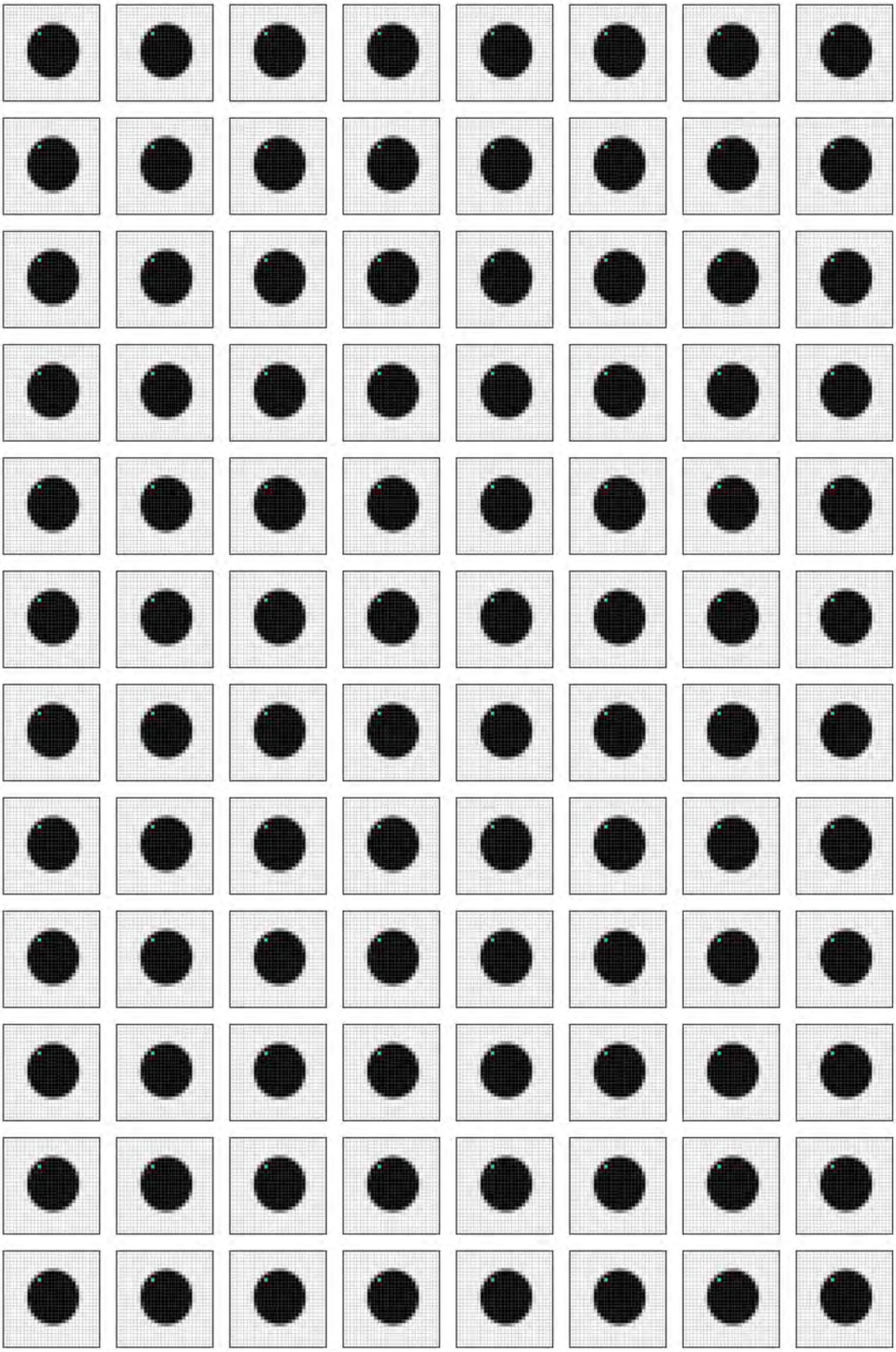

Figure 4.37. Original images of several ellipses from dataset 2.1 correctly classified with maximum activation pixel in colour blue. Best viewed in electronic form, zoom in to distinguish the pixel grid.

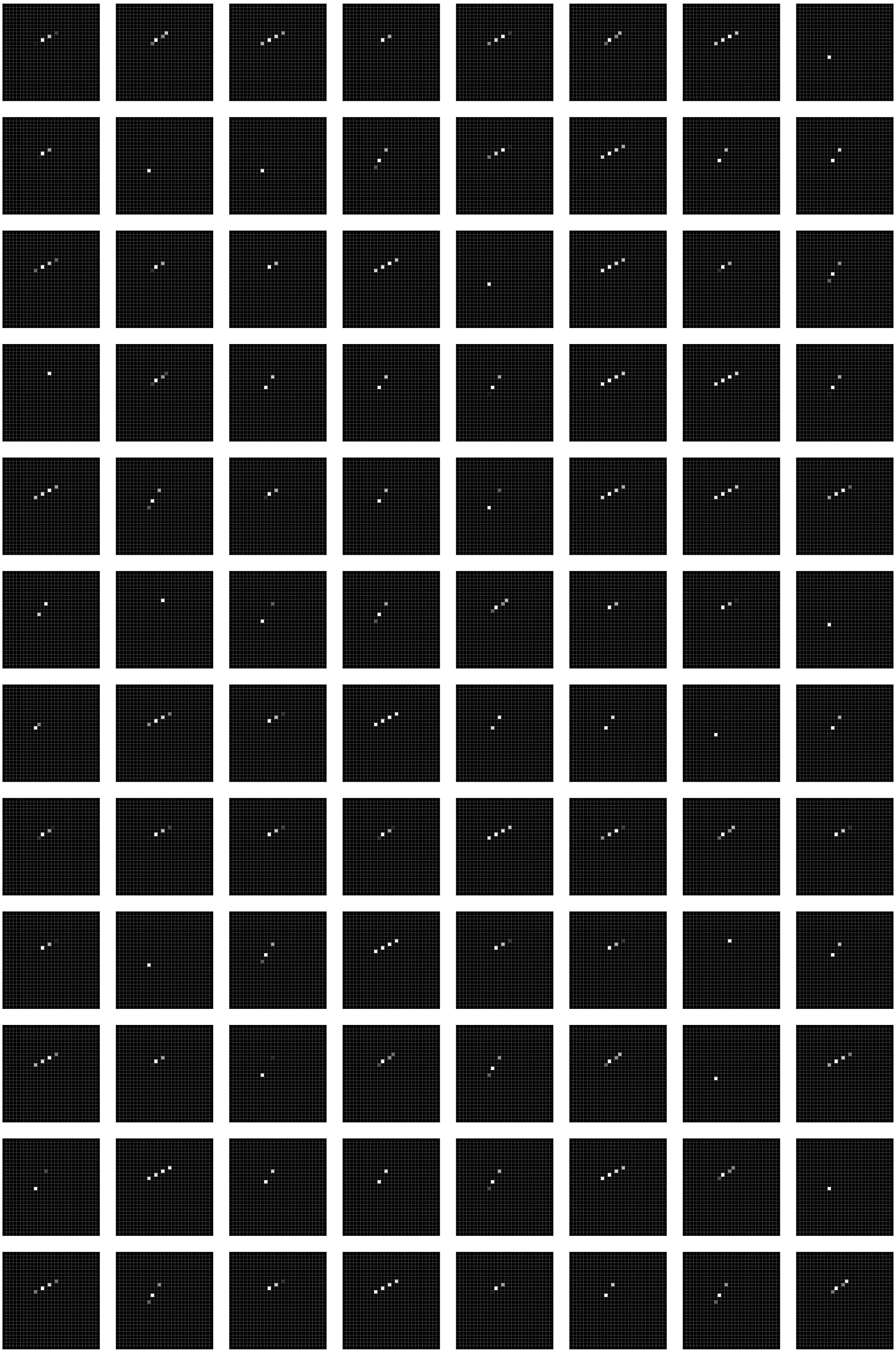

Figure 4.38. Feature maps of several rectangle samples from dataset 1.3 correctly classified. Best viewed in electronic form, zoom in to distinguish the pixel grid.

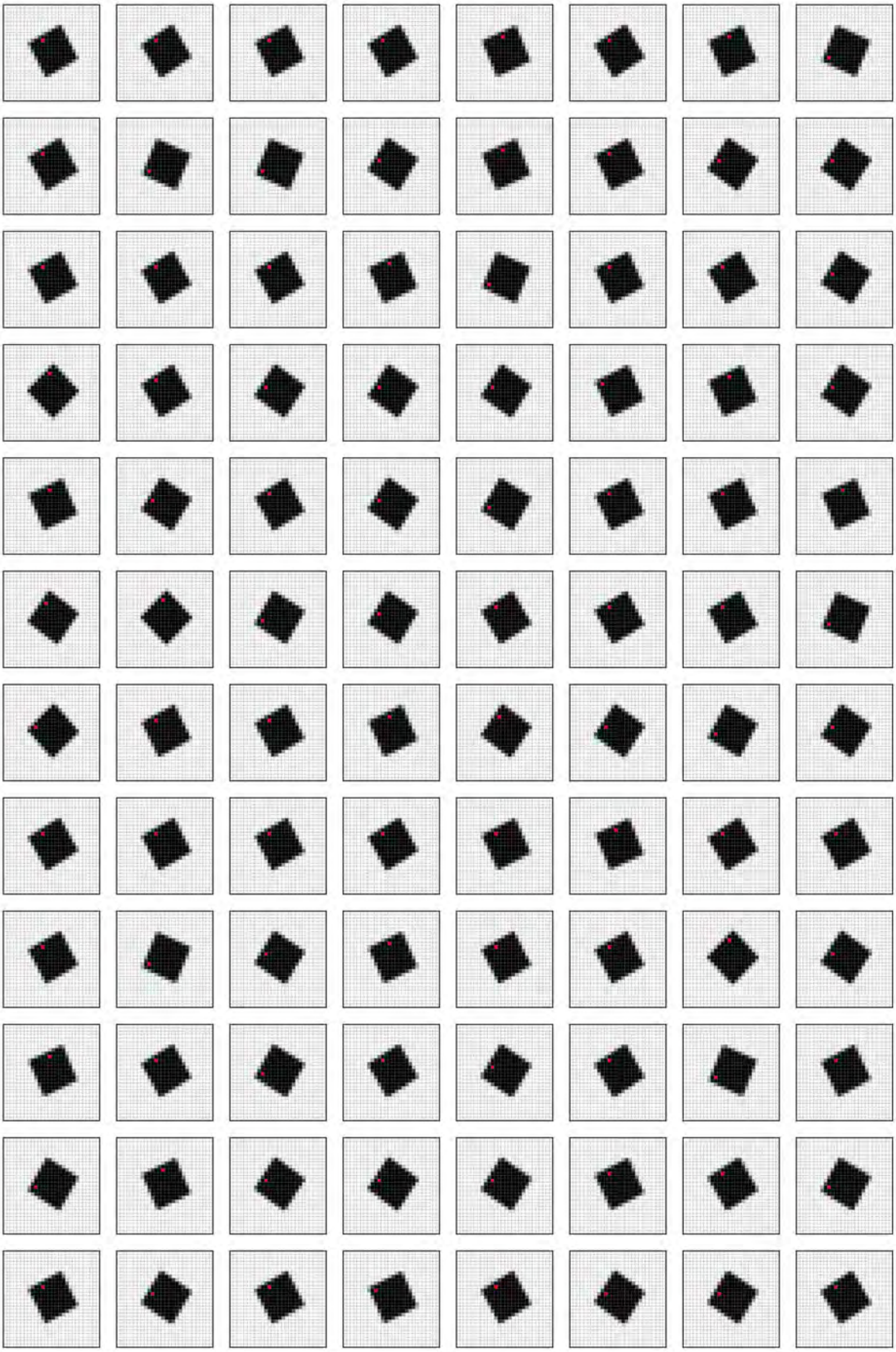

Figure 4.39. Original images of several rectangles from dataset 1.3 correctly classified with maximum activation pixel in colour red. Best viewed in electronic form, zoom in to distinguish the pixel grid.

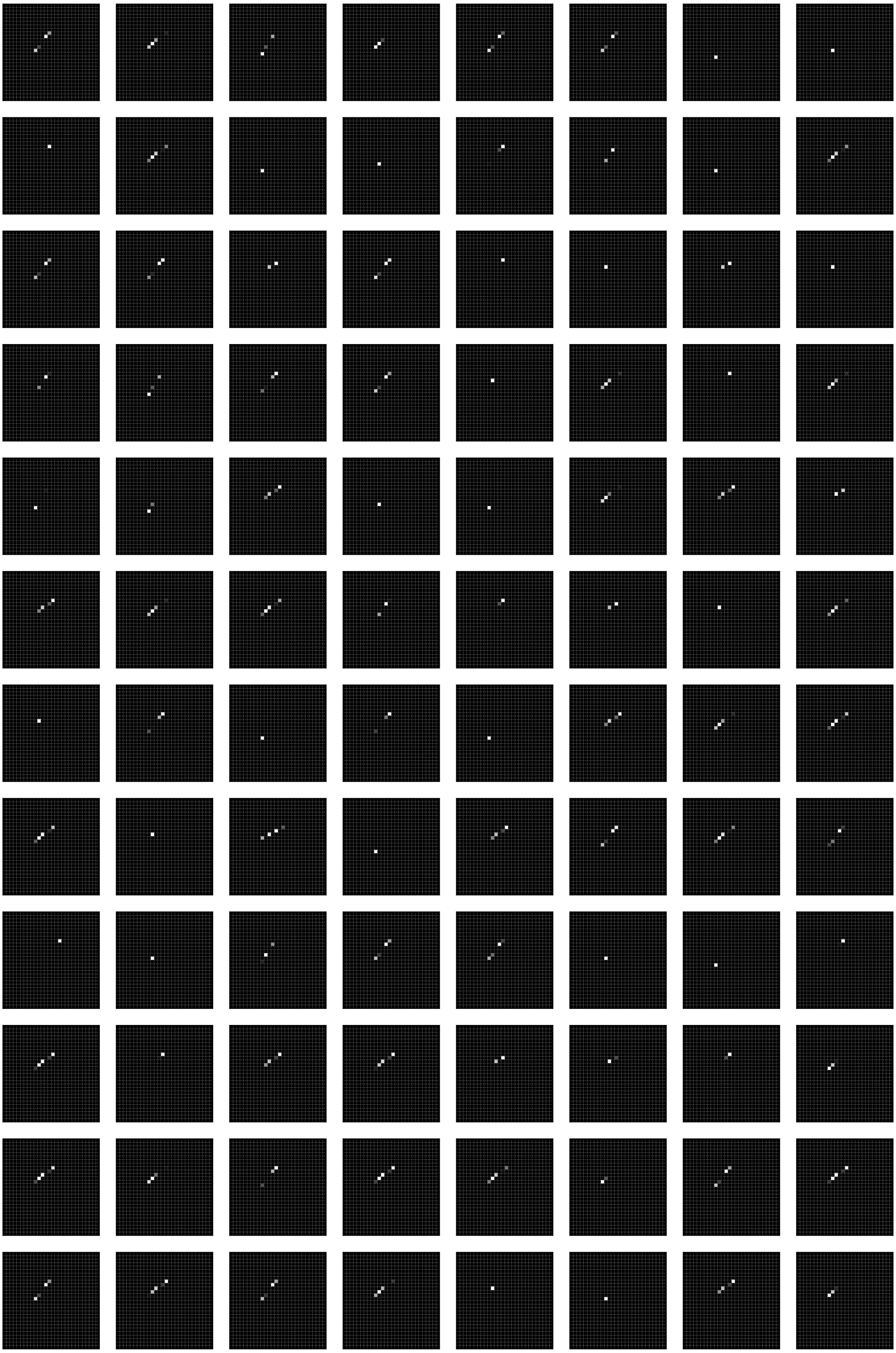

Figure 4.40. Feature maps of several rectangles samples from dataset 1.3 wrongly classified. Best viewed in electronic form, zoom in to distinguish the pixel grid.

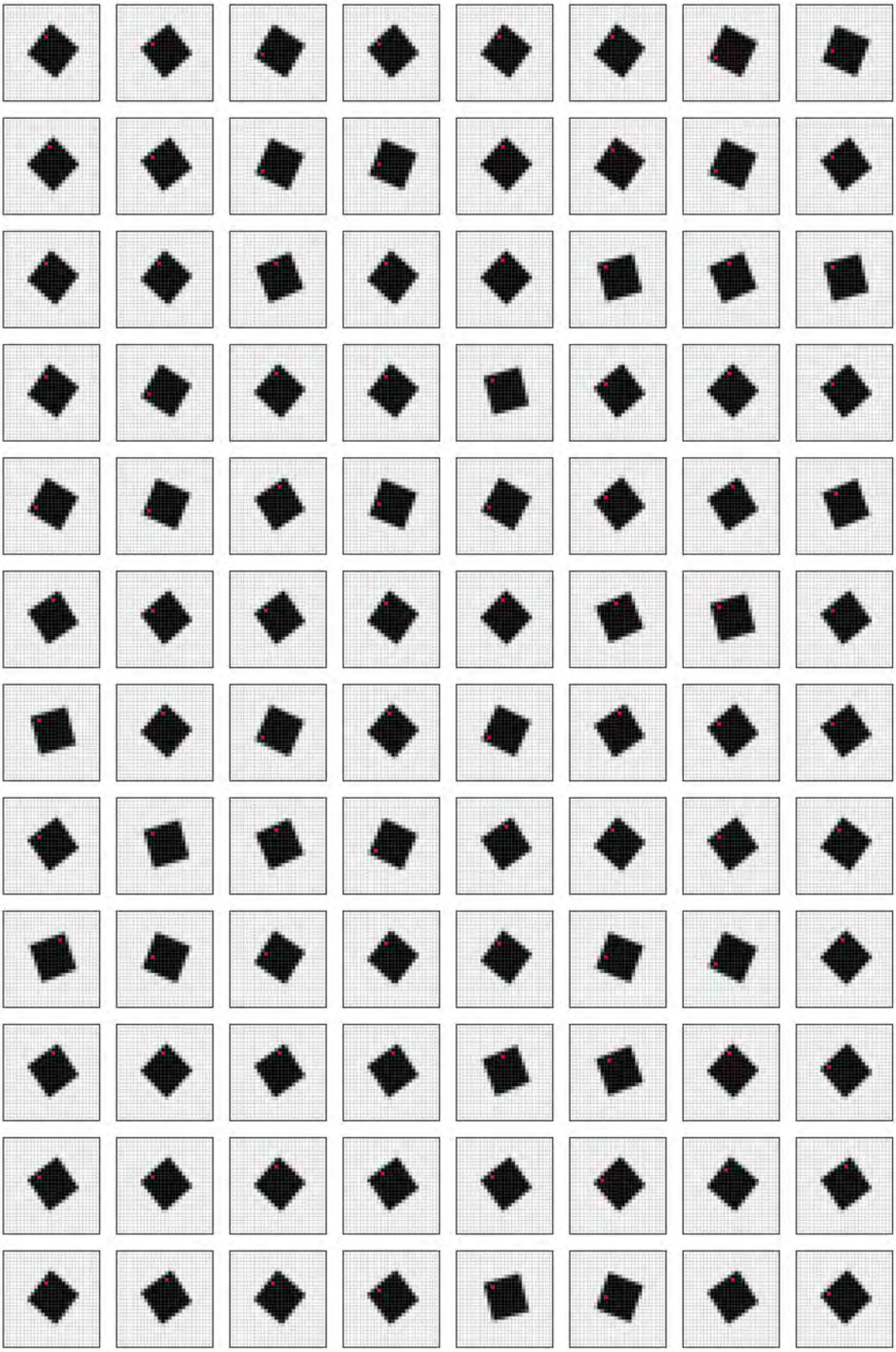

Figure 4.41. Original images of several rectangles from dataset 2.1 wrongly classified with maximum activation pixel in colour red. Best viewed in electronic form, zoom in to distinguish the pixel grid.

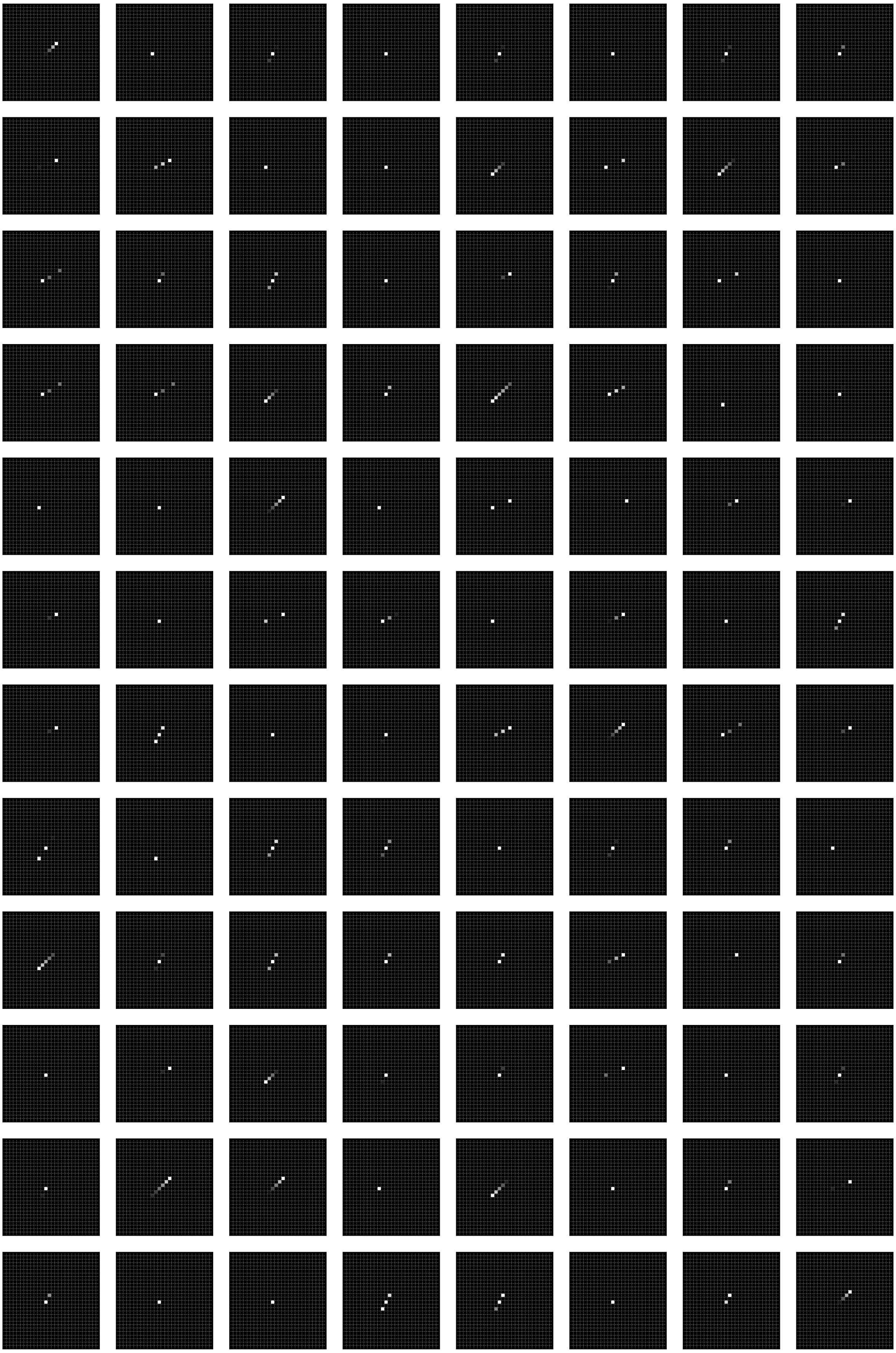

Figure 4.42. Feature maps of several triangles samples from dataset 2.1 correctly classified. Best viewed in electronic form, zoom in to distinguish the pixel grid.

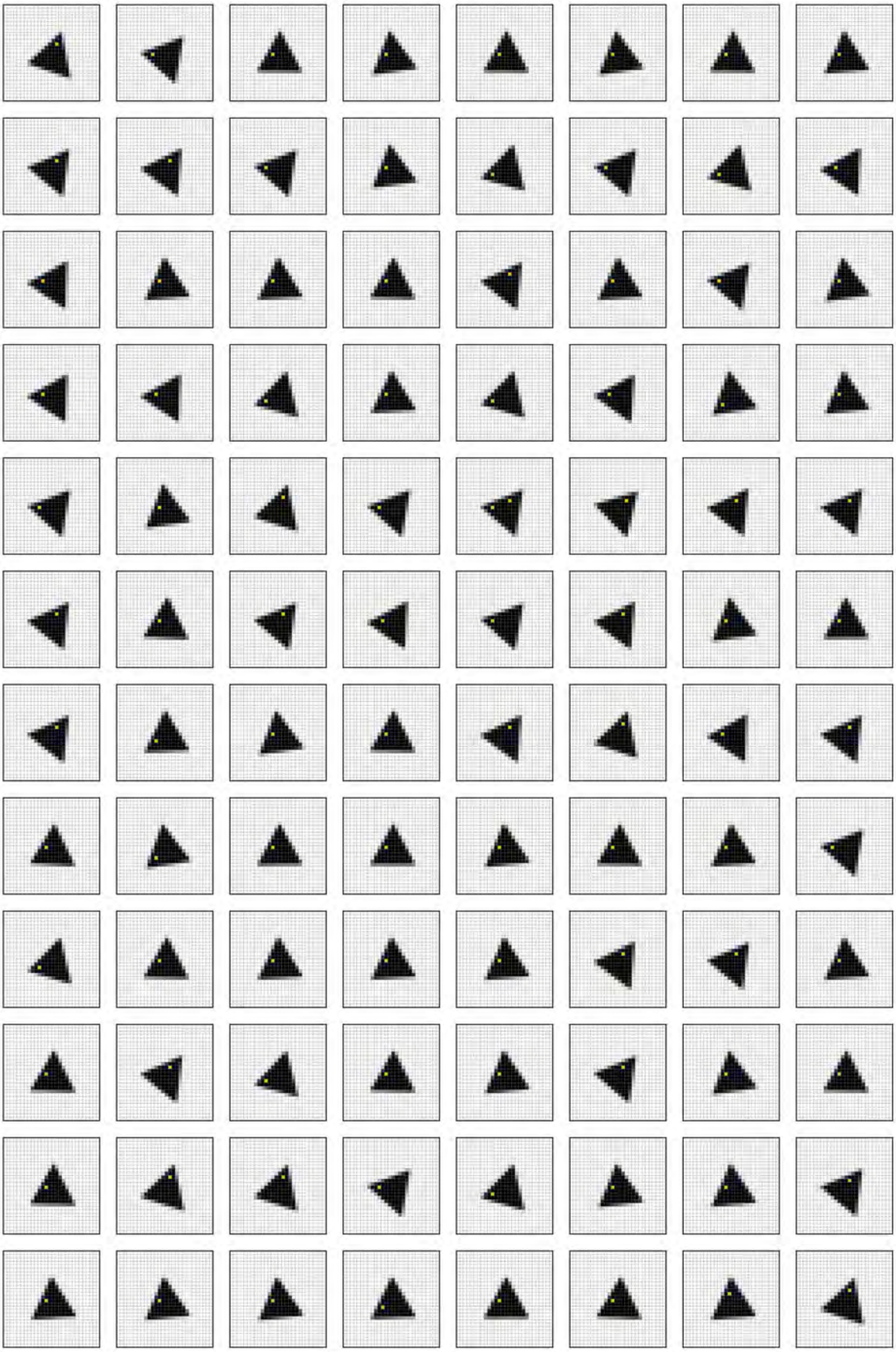

Figure 4.43. Original images of several triangles from dataset 2.1 correctly classified with maximum activation pixel in colour yellow. Best viewed in electronic form, zoom in to distinguish the pixel grid.

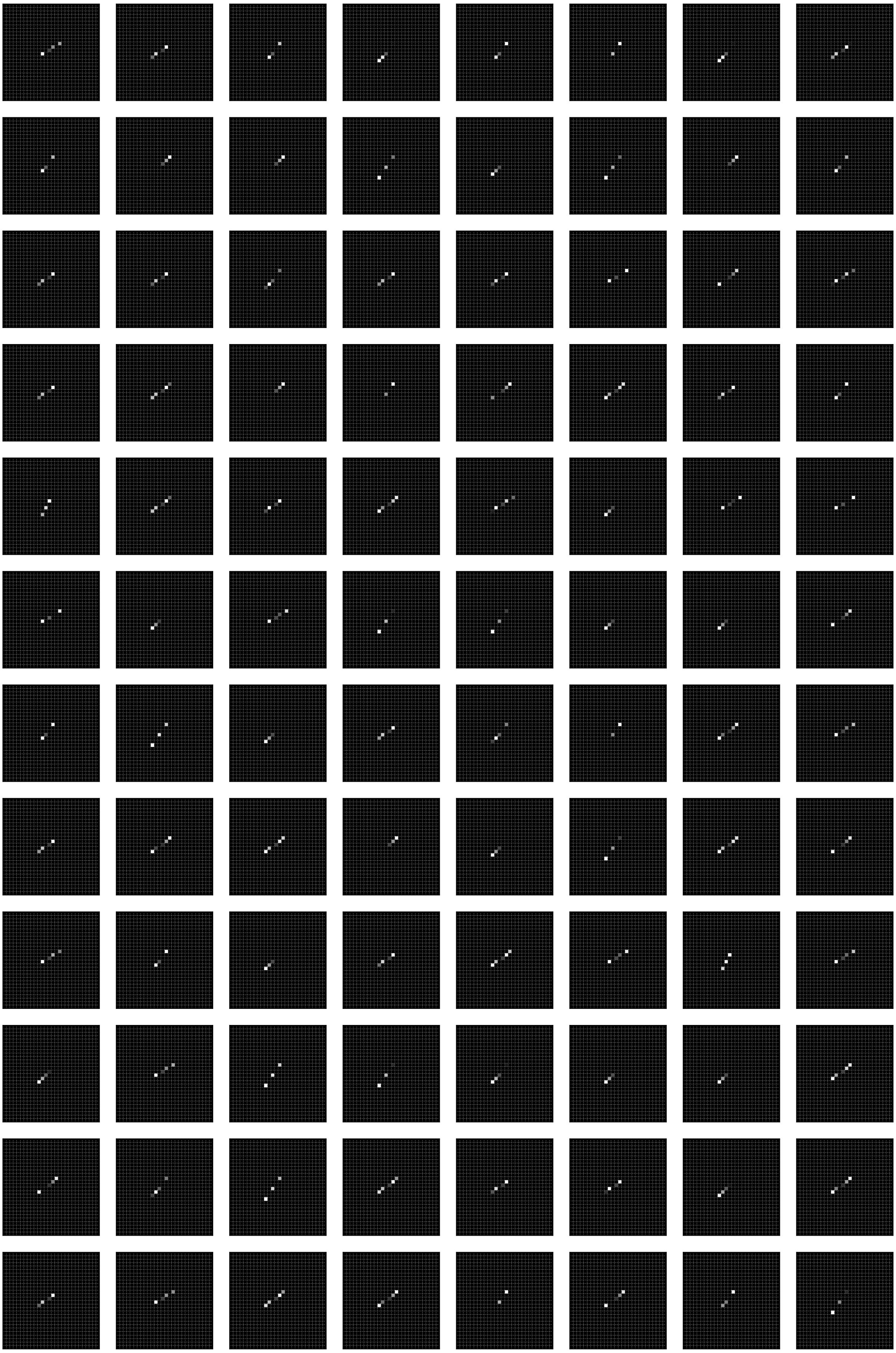

Figure 4.44. Feature maps of several triangles samples from dataset 2.1 wrongly classified. Best viewed in electronic form, zoom in to distinguish the pixel grid.

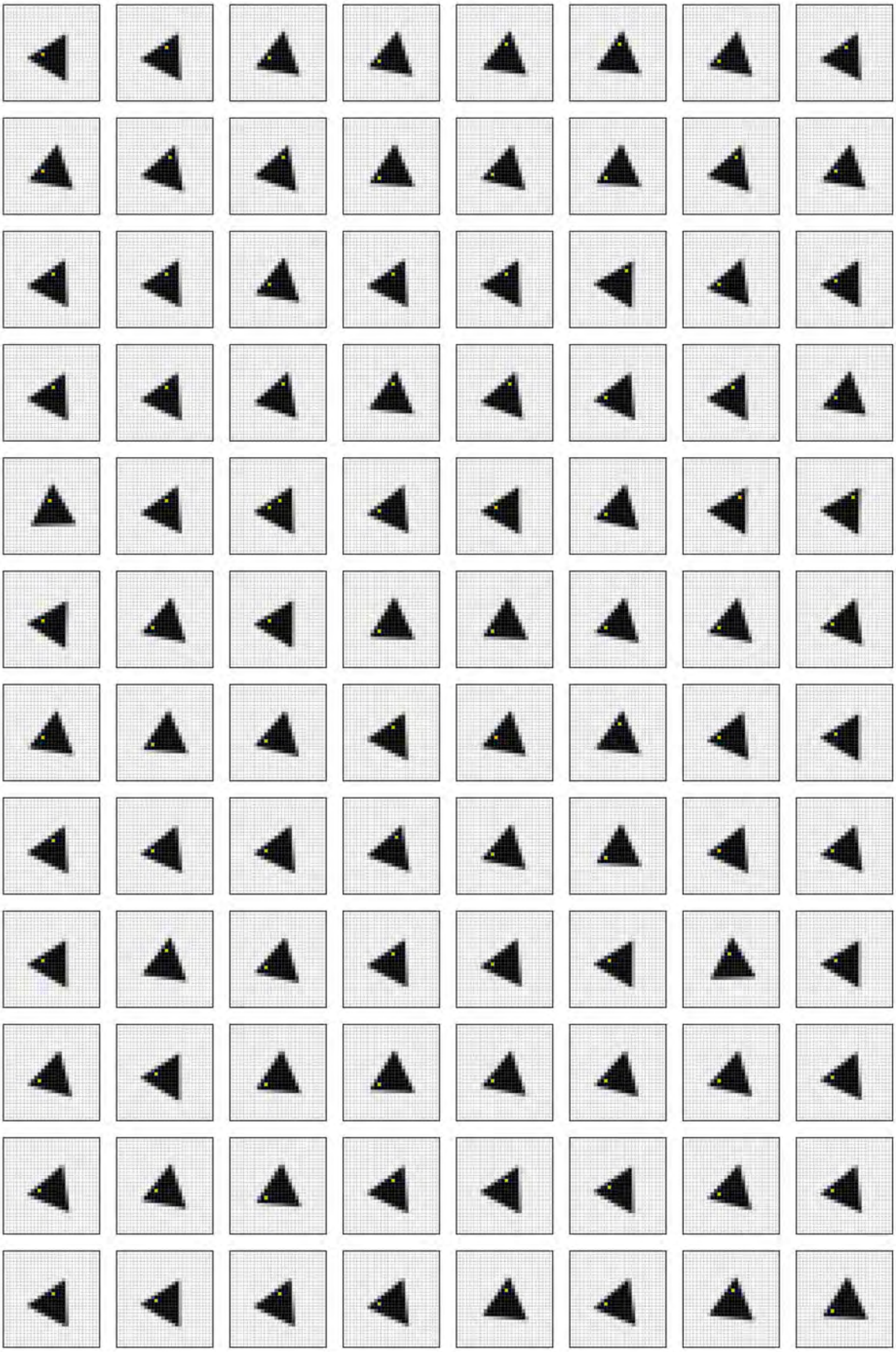

Figure 4.45. Original images of several triangles from dataset 2.1 wrongly classified with maximum activation pixel in colour yellow. Best viewed in electronic form, zoom in to distinguish the pixel grid.

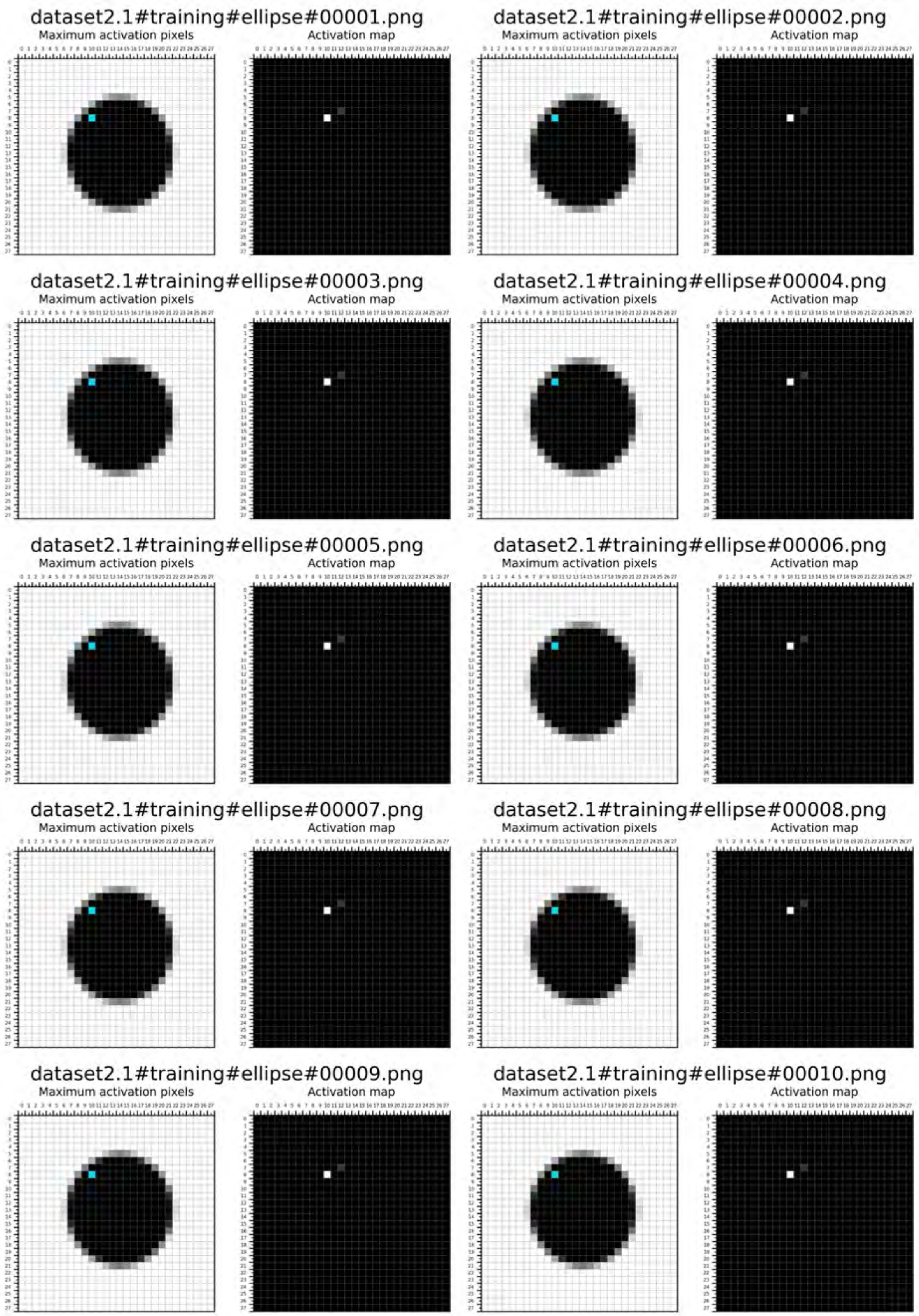

Figure 4.46. Both feature map and maximum activation pixel representations of several ellipses samples that are correctly classified from dataset 2.1. in a detailed form. Best viewed in electronic form, zoom in to distinguish the pixel grid.

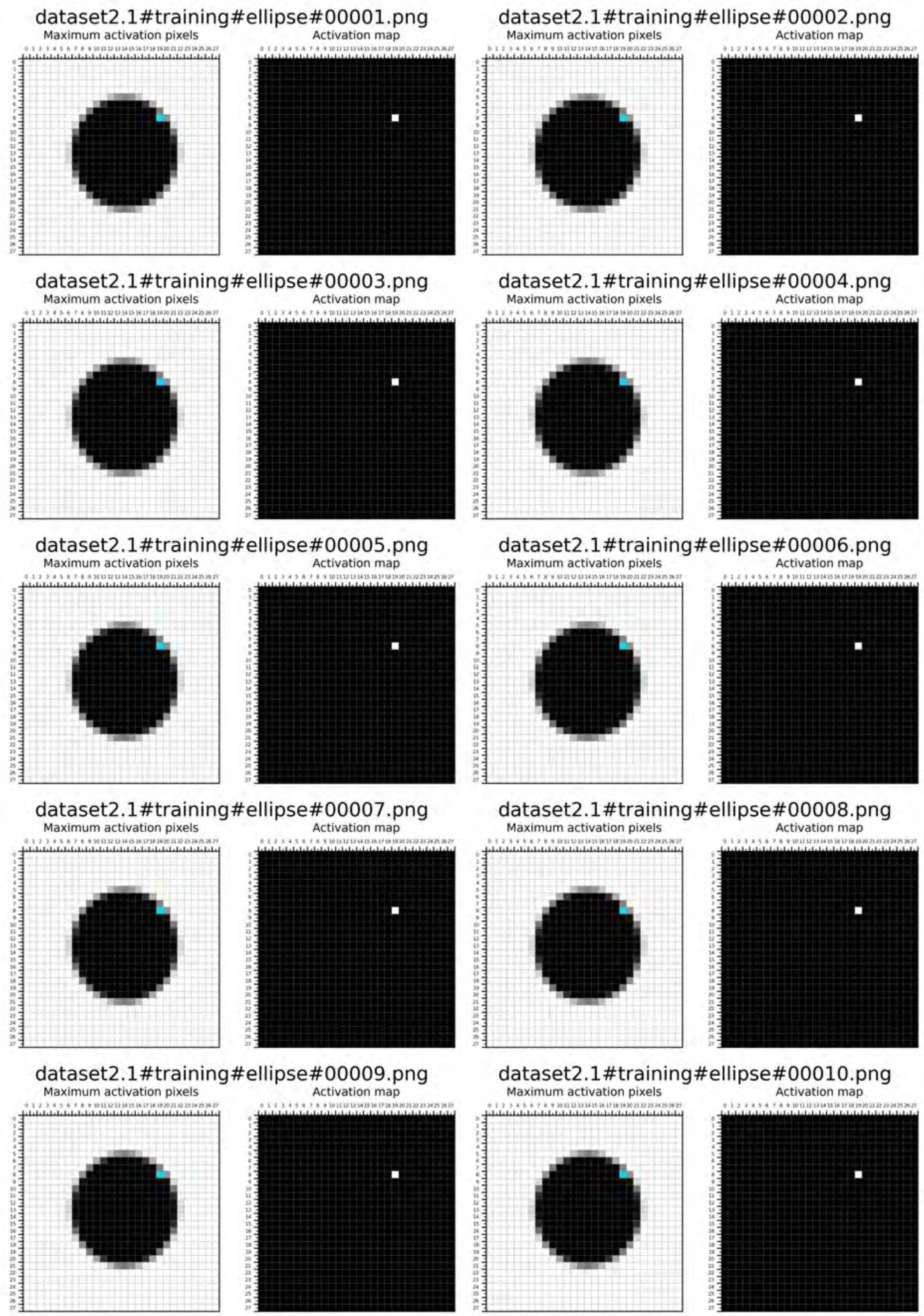

Figure 4.47. Both feature map and maximum activation pixel representations of several ellipses samples that are correctly classified by model another model (20201009T164248_dataset2.1_1L_nf1_1_ks1_3_ps1_28_ne_200_s_10) from dataset 2.1. in a detailed form. Best viewed in electronic form, zoom in to distinguish the pixel grid.

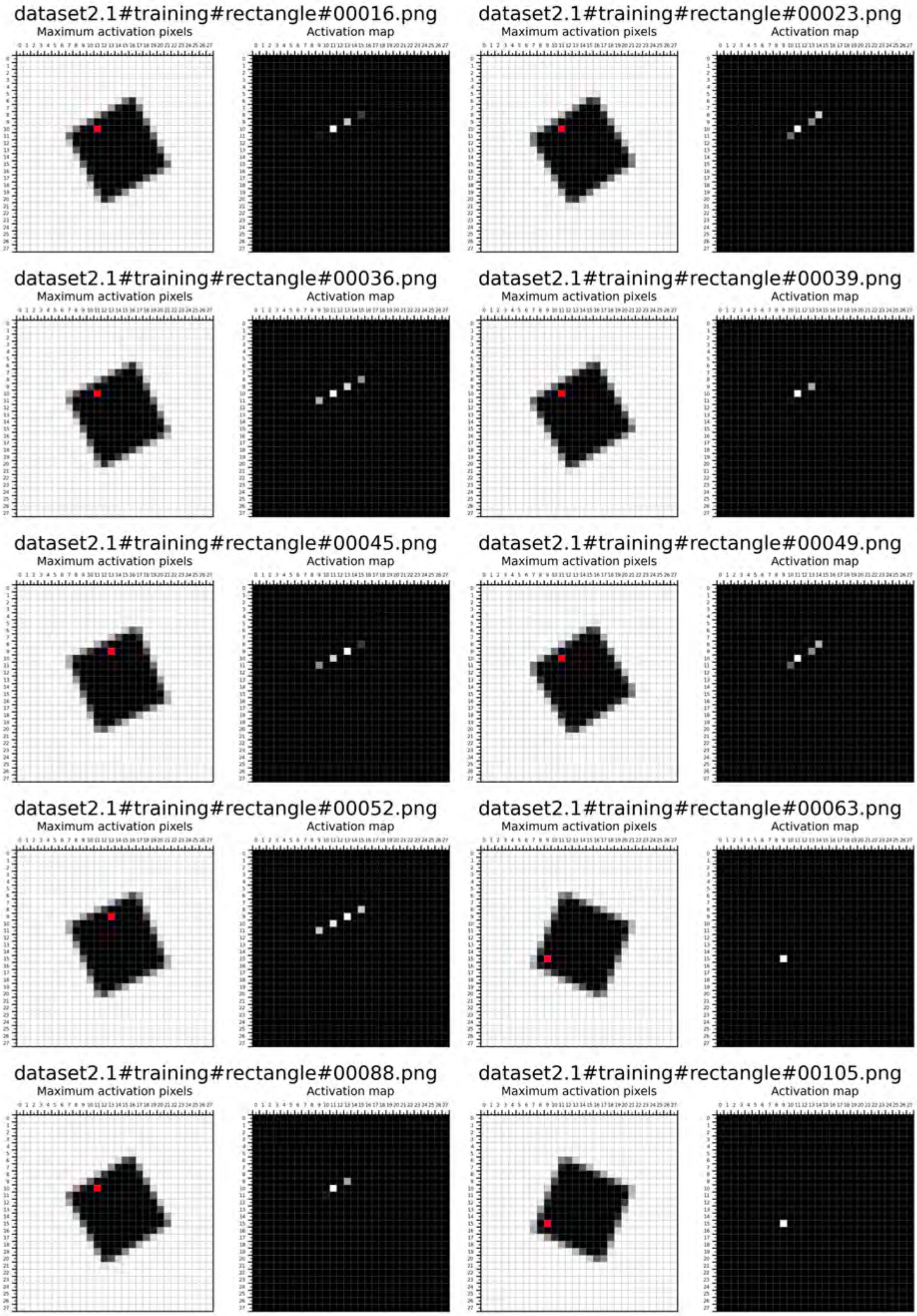

Figure 4.48. Both feature map and maximum activation pixel representations of several rectangles samples that are correctly classified from dataset 2.1 in a detailed form. Best viewed in electronic form, zoom in to distinguish the pixel grid.

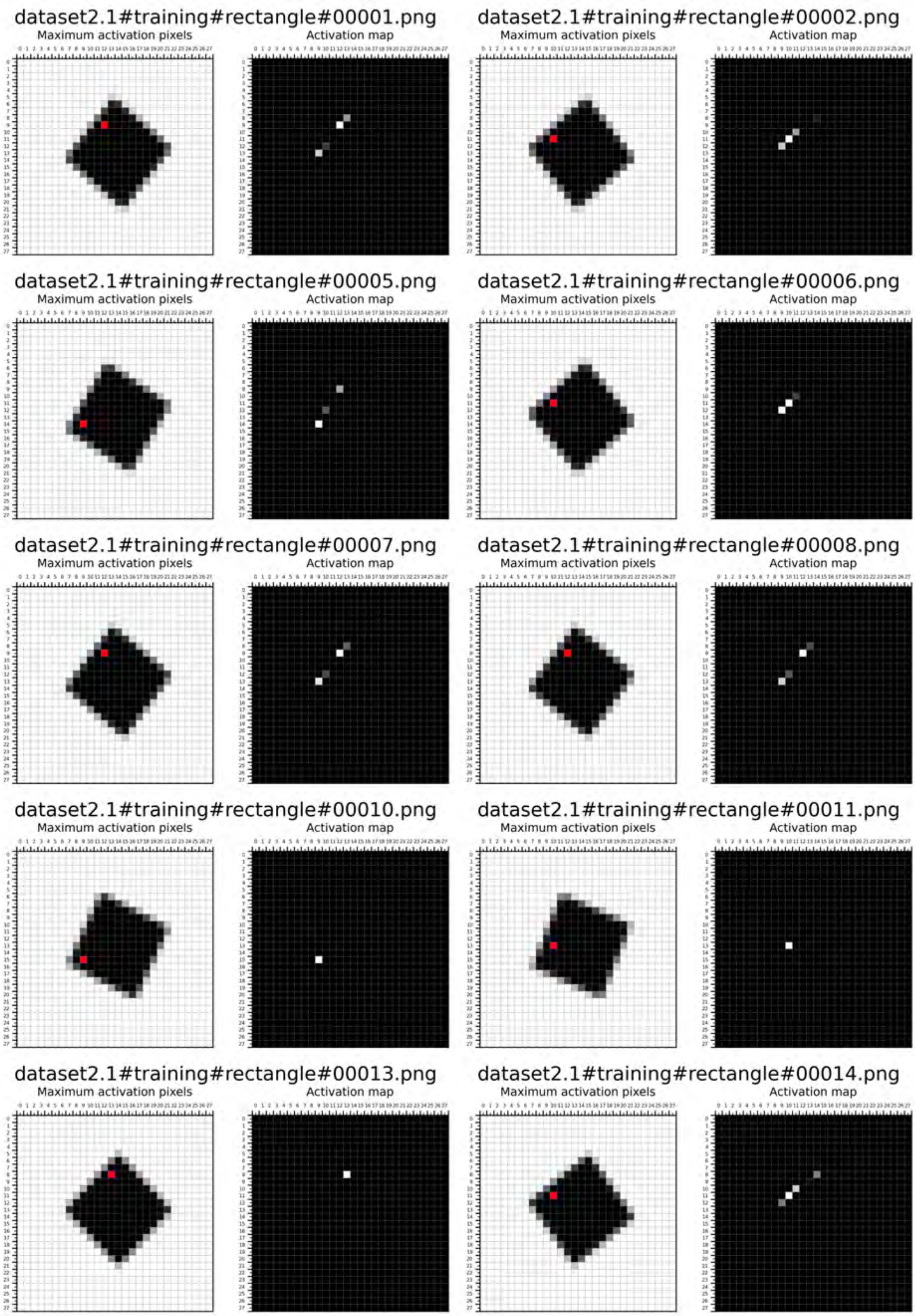

Figure 4.49. Both feature map and maximum activation pixel representations of several rectangles samples that are wrongly classified from dataset 2.1 in a detailed form. Best viewed in electronic form, zoom in to distinguish the pixel grid.

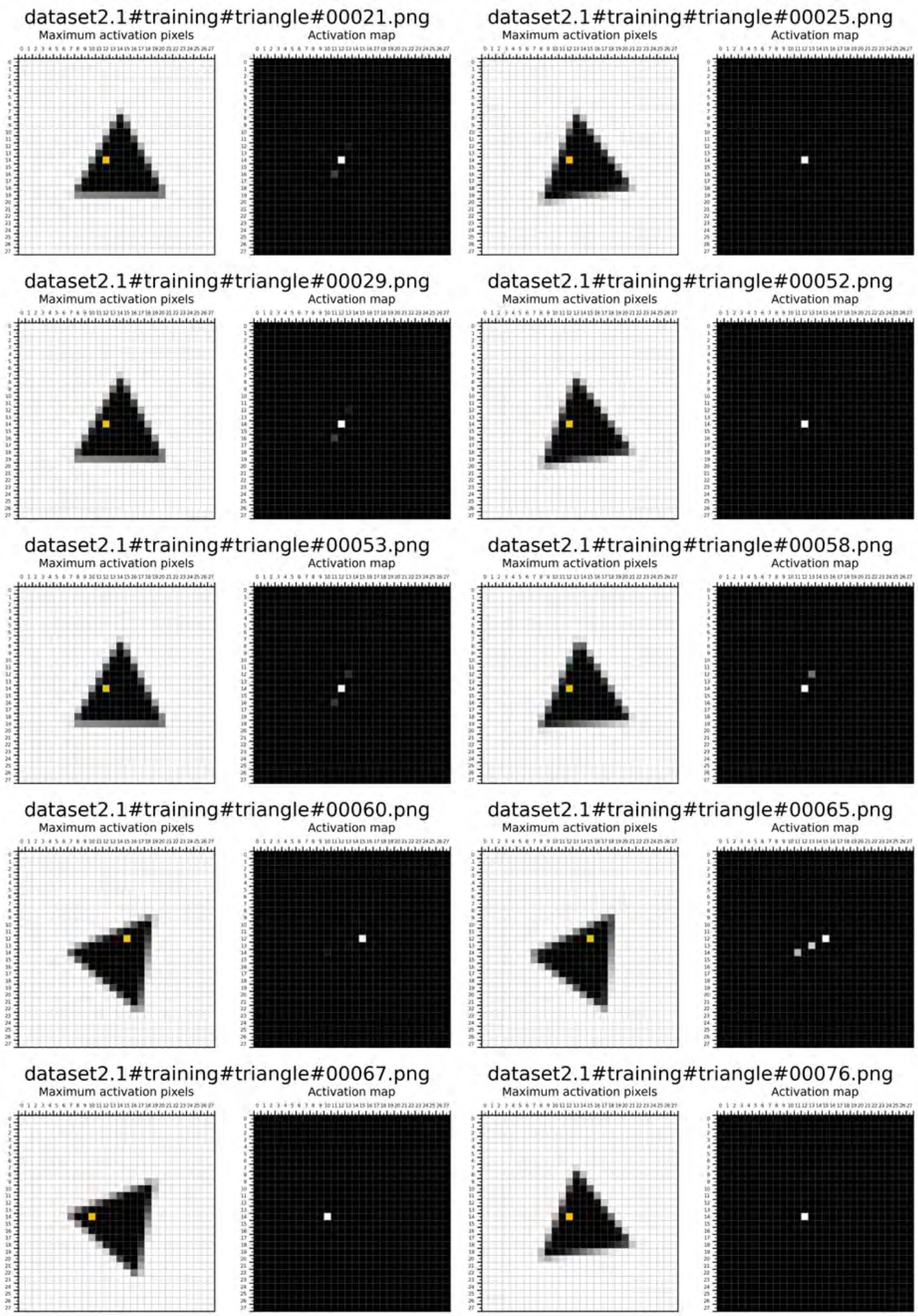

Figure 4.50. Both feature map and maximum activation pixel representations of several triangles samples that are correctly classified from dataset 2.1 in a detailed form. Best viewed in electronic form, zoom in to distinguish the pixel grid.

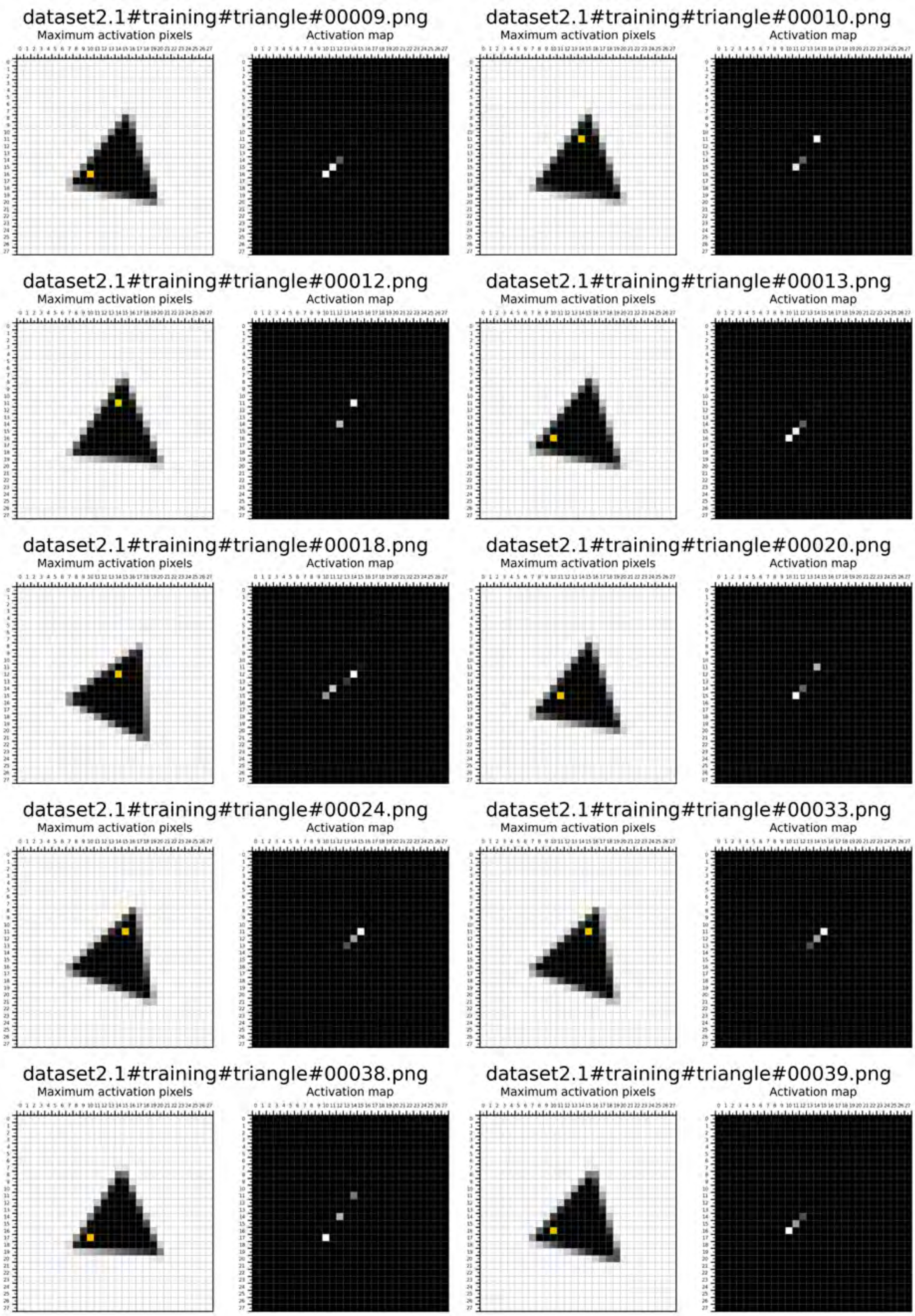

Figure 4.51. Both feature map and maximum activation pixel representations of several triangles samples that are wrongly classified from dataset 2.1 in a detailed form. Best viewed in electronic form, zoom in to distinguish the pixel grid.

4.3. Group 3: Scale and noise problem

Once the minimum model for scaling problem has been found and analysed deeply, the same problem but adding noise can be studied. In order to perform the analysis, a new dataset is generated based on the images from dataset 1.3, called dataset 3.0. This dataset not only contains images with added noise but some of them have exchange the background colour for the figure colour and vice-versa. In this way, the task is even more difficult for the CNN since it cannot relate the information of a class with colour information.

In the case of experiment 2.3, the minimum model only needs a feature to solve the classification task, reaching the highest level of accuracy for both training and validation datasets. It is interesting to study if an increase on the complexity of the problem, makes the model's complexity increase as well. Only an experiment has been performed to obtain the minimum model since once basic models for dataset 1.3 have been trained, building the minimum architecture for dataset 3.0 is straightforward: it is just a matter of increasing gradually the number of parameters until a high level of accuracy is achieved.

4.3.1. Dataset tested

The new dataset generated contains images of geometric figures with different sizes and centred in the image, similar to dataset 1.3, except by two new added features. The first one is salt-and-pepper noise with only gray values, avoiding white and black so that images are not distorted. Several datasets are built, each one of them with a different level of noise which are n01, n02, n05, n10 and n20. This means that, for example, in n20 case, 2000 (20×100) pixels are randomly selected and its value is swapped to black if it is white and vice versa. Figure 4.52 shows two sample images, one with the lowest level of noise, n01, and the other the highest one, n20. In addition, for each class, some of the images contain a white figure on a black background, that is, the colour is being inverted for some samples.

Specifically, training dataset 3.0 contains a total of 3000 images, 1000 for each class. Within each class 500 images have a black figure on a white background and 500 have a white figure over a black background. For the validation dataset it happens something similar but with the double of images: a total of 6000 samples, 2000 for each class and within the class half of the images have inverted colours. It is important to take into account that the experiment is carried out only with the dataset that has the highest noise level, n20, so that the problem is as difficult and different from the Experiment 2.3 (Section 4.2.3) as possible. Figure 4.53 shows images of different samples taken from the dataset 3.0 n20.

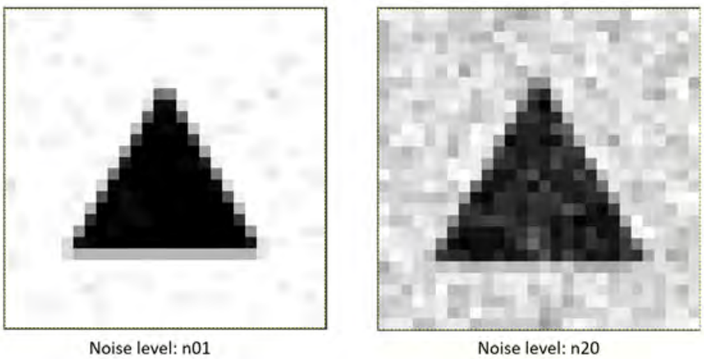

Figure 4.52. Detail of two images from dataset 3.0, one with the lowest level of noise (n01) and the other one with the highest level of noise (n20) so it can be compared.

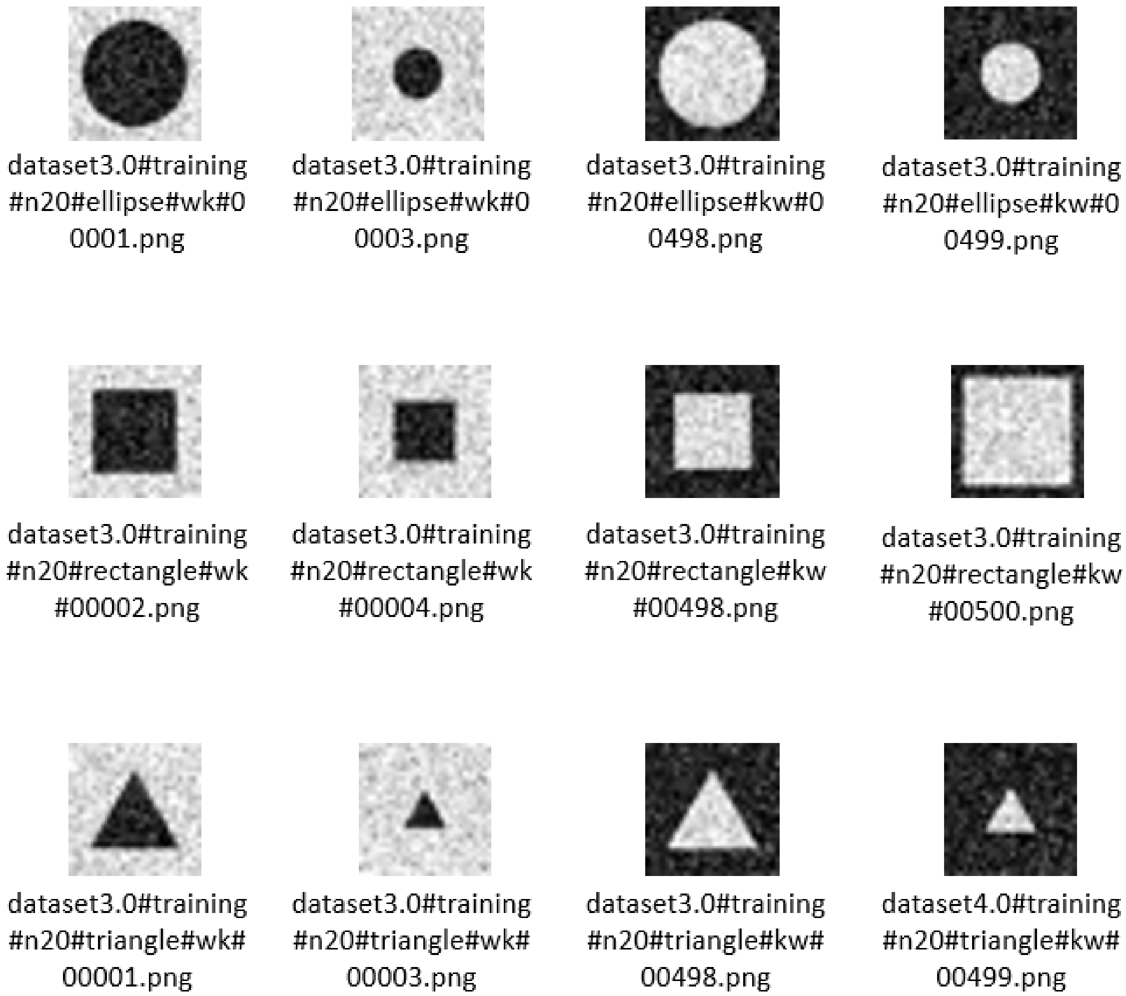

Figure 4.53. Examples of images from dataset 3.0 n20 belonging to the three different classes.

4.3.2. Experiment 3.1

Before starting with the analysis of the minimum model, it is important to remember that in all the previous experiments the classification part of the network was reduced to the minimum, since it only comprised the flattening layer and the output layer with three neurons. For dataset 1.3, this simple classifier was enough to solve the task as it was very simple (with just one feature it reached a minimum of 65% of accuracy). Nevertheless, with the increase in the level of complexity of dataset 3.0 n20, the power of the classifier has to be used. That is the reason why a new layer is added to the architectures tested in this experiment, a hidden layer for the classifier. Therefore, new hyperparameter is presented in this section too, known as *hlneurons*, which indicates the number of neurons that this hidden layer contains.

Therefore, the architectures that have been trained in this experiment have the following architecture: a convolutional layer comprised of one filter of size 3x3, a pooling layer which size varies among the values 28x28, 14x14 and 7x7; a flattening layer, a hidden layer and an output layer with three neurons (since that is the number of classes in the dataset). For the hidden layer the number of neurons has been set to the number of features of the model multiplied by 1, 2 or 3. In this way for the architectures 1-3-28, the parameter *hlneurons* takes the values 1, 2 or 3; for the architectures 1-3-14, it takes the values 4, 8 and 12 and for the architectures 1-3-7, it takes the values 16, 32 and 48. That is a method for estimating the number of neurons of the hidden layer depending on the amount of features that are inputted to the classifier.

Table 4.15 shows all the architectures that have been trained together with the results of accuracy obtained. It draws the attention that a lot of models, in particular the simplest ones, obtain such poor levels of accuracy that with the tossing of a coin, the results would be similar. These architectures are too simple, so they are not able to solve the problem and hence they all obtain a 33.33% level of accuracy, both for training and validation datasets, assigning a class to the images randomly.

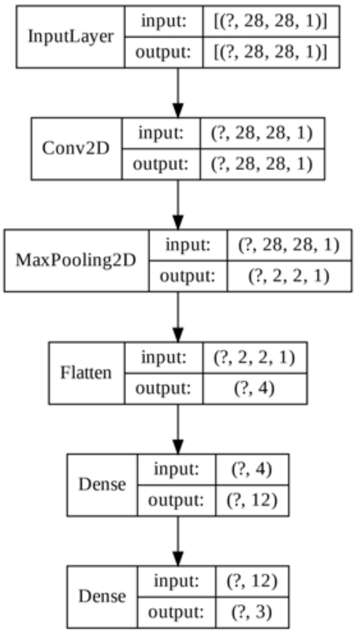

Figure 4.54. Architecture of the minimum model selected for dataset 3.0 n20.

The minimum model selected in this case is the one known as 20201205T015639_dataset3.0 n20_1L_nf1_1_ks1_3_ps1_14_ne_200_s_4 and it is shown in Figure 4.54. It is comprised of one convolutional layer with a single filter of size 3x3, a pooling layer of size 14x14, a flattening layer plus a hidden layer with twelve neurons and an output layer with three nodes. Its total number of parameters is 109 and it has four features. Among the five models with the same architecture, this is the one that reaches the highest accuracy although the five of them have high levels of accuracy except for the one with seed 0, which makes the architecture robust. In particular, the model reaches 94% in training and a 93% in validation, meaning that there is no overfitting.

It can be observed in Table 4.15 that there are certain models that achieve levels of accuracy slightly higher than the ones of the minimum model. Nevertheless, the difference is not very significant but the complexity increases exponentially. That is the reason why it has been selected that minimum model, it has the optimum trade-off between complexity and accuracy.

Table 4.15. Model hyperparameters and results obtained after performing Experiment 3.1 over dataset 3.0. Shaded cells show the model selected as the minimum one. *Model timestamp* is the date and hour when the model was created (used as an identifier) with the format: *<year><month><day>T<hour><minute><second>*, *seed* is the seed used for fixing randomness in the model, *nepochs* stands for "number of epochs", *nfilters_1st* stands for "number of filters in the first convolutional layer", *ksize_1st* stands for "kernel size of the first layer", *poolsize_1st* stands for "pool size in the first layer", *hlneurons* for "neurons in the hidden layer", *number_param* stands for "number of parameters", *number_features* stands for "number of features", *tr_acc* stands for "training accuracy" and *val_acc* stands for "validation accuracy". Model names follow the structure (no number of neurons in the hidden layer included to avoid very long names): <model timestamp>_dataset3.0n20_1L_nf1_<nfilters_1st>_ks1_<ksize_1st>_ps1_<poolsize_1st>_ne_<nepochs>_s_<seed>.

Model timestamp	seed	nepochs	nfilters_1st	ksize_1st	poolsize_1st	hlneurons	number_param	number_features	tr_acc	val_acc
20201203T200506	0	200	1	3	28	1	18	1	0.333	0.333
20201203T211825	4	200	1	3	28	1	18	1	0.333	0.333
20201203T215104	10	200	1	3	28	1	18	1	0.333	0.333
20201203T222410	550	200	1	3	28	1	18	1	0.333	0.333
20201203T225748	1234	200	1	3	28	1	18	1	0.333	0.333
20201203T233050	0	200	1	3	28	2	23	1	0.333	0.333
20201204T000322	4	200	1	3	28	2	23	1	0.333	0.333
20201204T003459	10	200	1	3	28	2	23	1	0.333	0.333
20201204T010655	550	200	1	3	28	2	23	1	0.333	0.333
20201204T013841	1234	200	1	3	28	2	23	1	0.333	0.333
20201204T021034	0	200	1	3	28	3	28	1	0.333	0.333
20201204T024300	4	200	1	3	28	3	28	1	0.333	0.333
20201204T031458	10	200	1	3	28	3	28	1	0.333	0.333
20201204T034657	550	200	1	3	28	3	28	1	0.333	0.333
20201204T041923	1234	200	1	3	28	3	28	1	0.333	0.333
20201204T185102	0	200	1	3	14	4	45	4	0.638	0.632
20201204T202346	4	200	1	3	14	4	45	4	0.627	0.623
20201204T205657	10	200	1	3	14	4	45	4	0.573	0.568
20201204T213129	550	200	1	3	14	4	45	4	0.501	0.501
20201204T220450	1234	200	1	3	14	4	45	4	0.500	0.500
20201204T223835	0	200	1	3	14	8	77	4	0.333	0.333
20201204T231241	4	200	1	3	14	8	77	4	0.773	0.772
20201204T234623	10	200	1	3	14	8	77	4	0.679	0.676
20201205T001931	550	200	1	3	14	8	77	4	0.591	0.591
20201205T005150	1234	200	1	3	14	8	77	4	0.664	0.660
20201205T012412	0	200	1	3	14	12	109	4	0.333	0.333
20201205T015639	4	200	1	3	14	12	109	4	0.940	0.930
20201205T023010	10	200	1	3	14	12	109	4	0.785	0.782
20201205T030522	550	200	1	3	14	12	109	4	0.801	0.795
20201205T034107	1234	200	1	3	14	12	109	4	0.934	0.933
20201205T095653	0	200	1	3	7	16	333	16	0.737	0.728
20201205T112851	4	200	1	3	7	16	333	16	0.953	0.942
20201205T120333	10	200	1	3	7	16	333	16	0.879	0.861
20201205T123802	550	200	1	3	7	16	333	16	0.868	0.851
20201205T131126	1234	200	1	3	7	16	333	16	0.834	0.821
20201205T134445	0	200	1	3	7	32	653	16	0.968	0.964
20201205T141756	4	200	1	3	7	32	653	16	0.978	0.962

Continued on next page

Table 4.15 – continued from previous page

Model timestamp	seed	nepochs	nfilters_1st	ksize_1st	poolsize_1st	hlneurons	number_param	number_features	tr_acc	val_acc
20201205T145056	10	200	1	3	7	32	653	16	0.928	0.912
20201205T152401	550	200	1	3	7	32	653	16	0.910	0.896
20201205T155619	1234	200	1	3	7	32	653	16	0.900	0.899
20201205T162831	0	200	1	3	7	48	973	16	0.975	0.966
20201205T170041	4	200	1	3	7	48	973	16	0.989	0.970
20201205T173252	10	200	1	3	7	48	973	16	0.975	0.967
20201205T180512	550	200	1	3	7	48	973	16	0.969	0.956
20201205T183714	1234	200	1	3	7	48	973	16	0.976	0.970

The minimum model for the scaling plus noise problem needs four features to perform the classification correctly, which means that this problem is more difficult that the one analysed in Experiment 2.3 (Section 4.2.3), where the images do not have noise. Following the same structure as for that experiment, first of all these features are going to be studied deeply in order to understand better how the CNN is able to perform the classification with the information that they provide. Basically, an exploratory data analysis is performed for the dataset of features that is inputted to the classification part of the CNN. It is important to take into account during most part of the analysis, only training representations are shown. The reason is that the minimum model achieves almost equal accuracy values for both training and validation dataset, hence the representations are very similar and in this way information is not repeated.

Figure 4.55 shows boxplots for each feature extracted by the model depending on the class. In this way, it can be observed that, for example, feature 2 takes very particular values for triangles which can be useful for distinguish this figure from the other two; and both features 2 and 3 could be use to distinguish between ellipses and rectangles. In general, the CNN is using a combination of the four features in order to perform the classification. In addition, Figure 4.56 shows the correlation existing among the four features and it can be observed that there are relations between pairs of features: feature 1 and 3 have direct correlation whereas both have negative or inverse correlation with features 2 and 4. Feature 2 and 4 have positive correlation too but its value is insignificant.

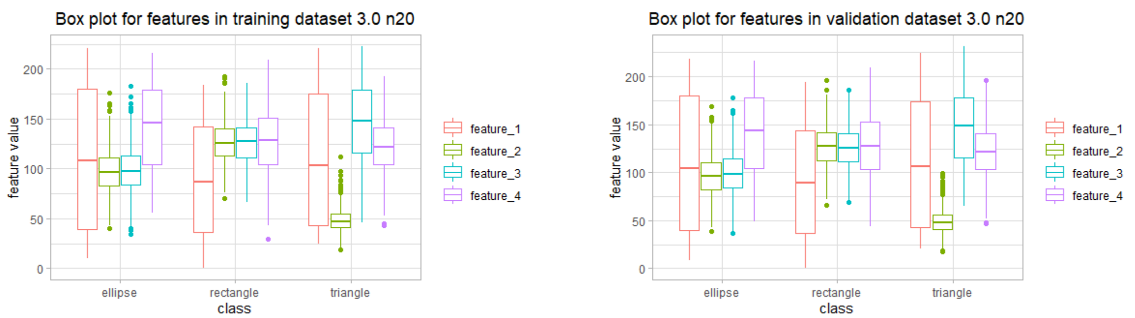

Figure 4.55. Boxplots showing values of the features extracted depending on the class by the minimum model for dataset 3.0 n20, 20201205T015639_dataset3.0n20_1L_nf1_1_ks1_3_ps1_14_ne_200_s_4, in both training and validation.

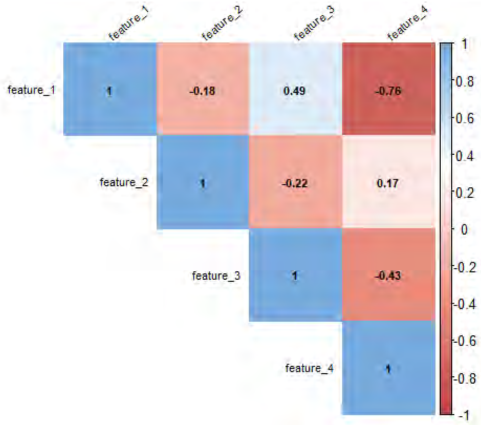

Figure 4.56. Linear correlation plot between features extracted by the minimum model for dataset 3.0 n20.

A new analysis that is carried out in this experiment in order to analyse deeply the features is the Principal Component Analysis or PCA. This technique enables to obtain the information that the features extracted by the model contain and how that input space they generate can be explained. Figure 4.57 shows the amount of variance explained by each one of the components built and it can be observed that with the first three the model is able to explain approximately a 95% of the variance. In the same figure another plot can be observed, with the coefficients from each new variable and it can be seen that a different feature is the most representative of each principal component.

Figure 4.58 shows the projections of the training dataset on the plane formed by the intersection of the three principal components, since these three are the ones that contain more information about the data. It can be observed in the first projection that triangle samples are located on the top part of the plot, ellipses in the middle part and rectangles at the bottom. In the other two projections the classes are located in different areas of the plane but note that the samples are always grouped together by class. Therefore, in the space created by the features that have been extracted by the CNN is possible to classify correctly samples belonging to the three categories since they are quite separated. In addition, looking at the direction and magnitude of the arrows representing the features, it can be observed what has been analysed in the previous plot, feature 1 has a great influence on the first principal component and so on.

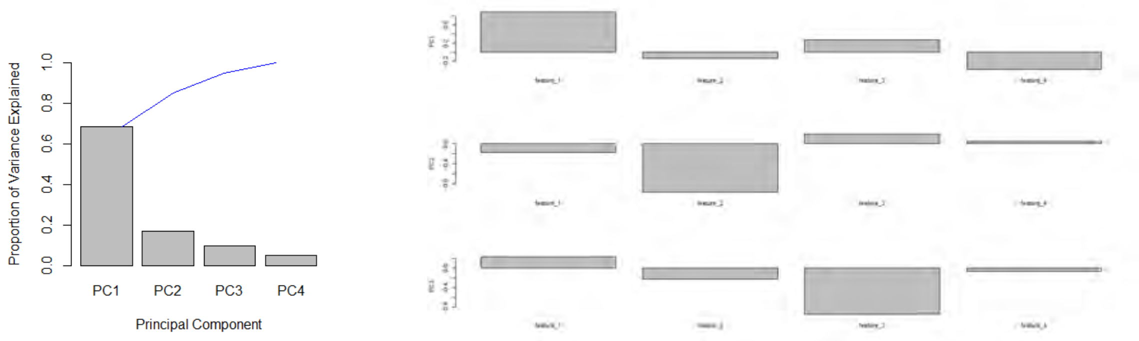

Figure 4.57. Proportion of variance explained by each principal component (left) and composition of each principal component (right) for features extracted by dataset 3.0 n20 minimum model.

It is important to highlight that when the dataset is projected on the plane formed by the first and second principal components or the first and the third principal components, two well defined groups of samples appear in the image: one is placed on the left part of the image, with samples taking negative values of the first principal component and the other on the right, with samples taking positive values. The explanation to this is found in Figure 4.59 which shows the same projection representations but indicating the colours of the samples together with the class. As it can be observed on the new projections, samples placed on the left part of the image are the ones with black background and white figures whereas the ones in the right part have white background and black figures.

To sum up the PCA analysis, the feature space generated by the CNN eases the distinction among the three kinds of images, or, what is the same, the classification task. In addition, there are different areas for images with each combination of background and figure colours and this can be observed in the three dimensional space formed by the three first principal components. Once the feature space has been analysed, and it has been observed that is feasible to separate, it is interesting to input it to some classification techniques. In this way, the information given by the features is trying to be understood. Each classification model will try to extract that information using its particular methodology but the input information is the same for all the models.

The first classifier that has been trained with the features space is a decision tree, following the analysis performed in Experiment 2.3. Figure 4.60 shows one of the tree models that has been trained. Regarding the structure of the tree, it can be seen that with the first cut of the space a lot of triangles are isolated from the rest of the samples and correctly classified. It is important to take into account that this tree is one of the possible solutions to perform the task but not the only one.

Figure 4.58. Projections of feature dataset generated for each sample from dataset 3.0 n20 with its three classes on the planes formed by the first three principal components: PCA 1 vs. PCA 2 (first plot), PCA 2 vs. PCA 3 (second plot) and PCA 1 vs. PCA 3 (third plot). The component that explains the largest amount of information is placed on the horizontal axis.

Figure 4.59. Projections of feature dataset generated for each sample from dataset 3.0 n20 with six classes (taking into account color of the figure and the background) on the planes formed by the first three principal components: PCA 1 vs. PCA 2 (first plot), PCA 2 vs. PCA 3 (second plot) and PCA 1 vs. PCA 3 (third plot). The component that explains the largest amount of information is placed on the horizontal axis.

The trained tree gives a clue about the logic of the network, that may not be so different from the logic of the tree but not equal. In addition, this tree achieves a training accuracy of 0.92 within the confidence interval of (0.9178, 0.9367) which is again an indicator of how powerful the features extracted by the CNN are, since with a much simple classification algorithm, the levels of accuracy reached are very high.

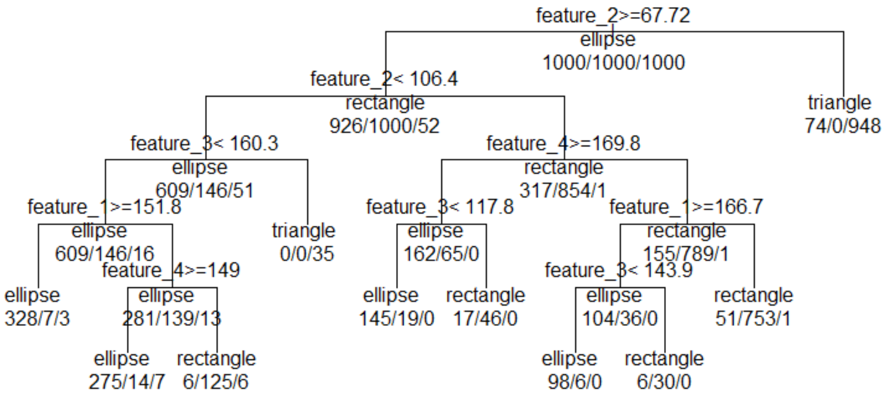

Figure 4.60. Decision tree using feature extracted by the minimum model for dataset 3.0 n20: 20201205T015639_dataset3.0n20_1L_nf1_1_ks1_3_ps1_14_ne_200_s_4.

The next algorithm that is tested is K-means. This is not a classification algorithm but a clustering one. In this way, it can be checked if the groups of samples that are visually appreciated, actually exist. According to the elbow plot shown in Figure 4.61, the optimum number of clusters that appear in the features space is seven. The scatter plot from Figure 4.62 shows the projection over the first two principal components when the number of clusters is equal to seven, where it can be observed that clusters 1, 2 and 3 are, in fact, separated from each other. The rest of them will be probably separated too but is not very well appreciated in this projection.

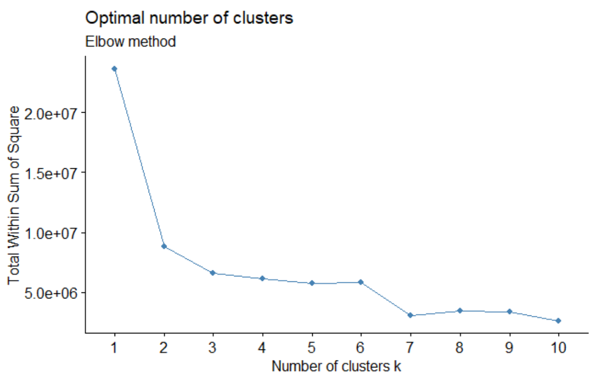

Figure 4.61. Elbow plot for K-means model trained with feature dataset generated by the minimum model for dataset 3.0 n20.

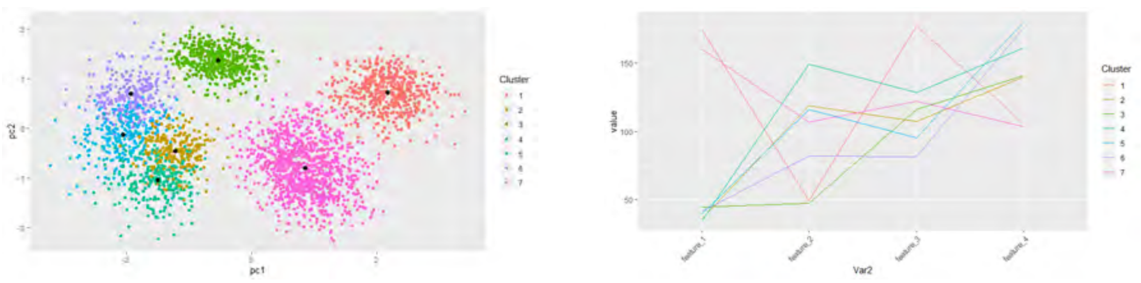

Figure 4.62. Projection of features generated by the minimum model seleted for dataset 3.0 n20 with the clusters built by K-means model (left) and centroids for each cluster (right).

	ellipse	rectangle	triangle
1	10	0	497
2	13	217	2
3	3	0	490
4	15	246	0
5	245	33	0
6	224	4	8
7	490	500	3

Figure 4.63. Number of samples from the three classes assigned to each cluster by the K-means model trained with the feature dataset generated by the minimum model selected for dataset 3.0 n20.

	PREDICTED		
KNOWN	**ellipse**	**rectangle**	**triangle**
ellipse	907	76	17
rectangle	67	930	3
triangle	11	6	983

Training confusion matrix

	PREDICTED		
KNOWN	**ellipse**	**rectangle**	**triangle**
ellipse	1782	166	52
rectangle	161	1837	2
triangle	31	9	1960

Validation confusion matrix

Figure 4.64. Confusion matrix both for training and validation images for the minimum model selected for dataset 3.0.

Looking at the second plot shown in Figure 4.62 it can be seen that each group of samples is different from each other since each centroids, which are the representatives points for each cluster, differ. It makes sense the fact that the K-means model has identify seven clusters since, for what it has been studied previously, there are six clear groups in the data, attending at the combination of the image class, the background and the figure colour. The left cluster probably contains all the samples located in the boundary between classes, which are more difficult to classify. Figure 4.63 shows the number of samples from each class that have been assigned to the different clusters. This table confirms what it has been already stated, there are two clusters that contain images from the same class and then another cluster with a mixture of samples belonging to different clusters, the ones that are in the limit between classes. In this way, cluters 1 and 3 contain triangles, 2 and 4 are rectangles, 5 and 6 ellipses, whereas cluster 7 contains both ellipses and rectangles. This makes sense since the ellipse and the rectangle are the most similar geometric shapes and, therefore, the hardest to separate. Figure 4.64 shows the confusion matrices generated by the CNN minimum model for both training and validation datasets and they confirm that the two classes that the model finds more difficult to separate are ellipses and rectangles, since it is where more mistakes is making.

All in all, PCA, decision trees and K-means algorithms confirm that in the feature space created by the CNN the images are grouped in regions that can be clearly separated depending on its type. This methodology of extracting first the input space of features and then try different classification algorithms imitates the traditional way of performing image classification that was used before the development of Deep Learning. Features from images that were considered important by the data scientist are extracted manually using techniques like SIFT and each different feature extracted is part of a dictionary (similar as it happens with words inside a language). Then, a model widely known as Bag of Words (BoW) describes the occurrence of features in each image (measures the presence of known features, that is, the ones that are in the dictionary). Finally, the BoW representation for each image is inputted into a classification algorithm such as a decision tree (like in this case) or a Support Vector Machine (SVM). In this case, the power of the CNN is replacing the manual extraction of features, which could be similar to the BoW, and combined with different classification methods too.

0.815	0.244	-0.281
-0.063	-0.506	-0.204
0.160	-0.602	0.084

20201205T015639_dataset3.0n20_1L_nf1_1_ks1_3_ps1_14_ne_200_s_4

Figure 4.65. Filter learnt by the minimum model selected for dataset 3.0 n20. Luminance representation on the left and the exact luminance values on the left.

For the analysis of the physical meaning of the model, once again, instead of plotting heat maps that give information about all of the image, only the specific pixels extracted by the model are highlight over the original images. Nevertheless, as this model extract four different features, instead of plotting them with the same colour, a different colour is assigned to each pixel depending on the value of the feature: red for the higher values and blue for the lower ones. In addition, feature maps are also plotted and both representations together for some images in a more detailed way.

To begin with this analysis, the filter learnt by the CNN has to be analysed. Figure 4.65 shows the representation of the filter together with its specific numerical values and it can be seen that the network is trying to obtain again borders but it seems that the filter can extract two kind of borders at the same time. This is due to the location of positive pixels, which are both on the top part of the filter and at the bottom part. This is better observed in the feature maps representations. The filter only has four pixels with a positive value that will activate when they are convolved with white or gray pixels from the image.

First of all, for the ellipses analysis, feature maps are shown in Figures 4.75 and 4.77. It can be observed that for images with white figures on a black background the border that is activated is the top left, whereas for the images where the colors are inverted, the border detected by the filter is the one on the bottom right corner. The filter seems to be extracting borders that create a color change placed in a diagonal direction as it can be better understood in Figures 4.66 and 4.67, which show two approximations of the convolution operation.

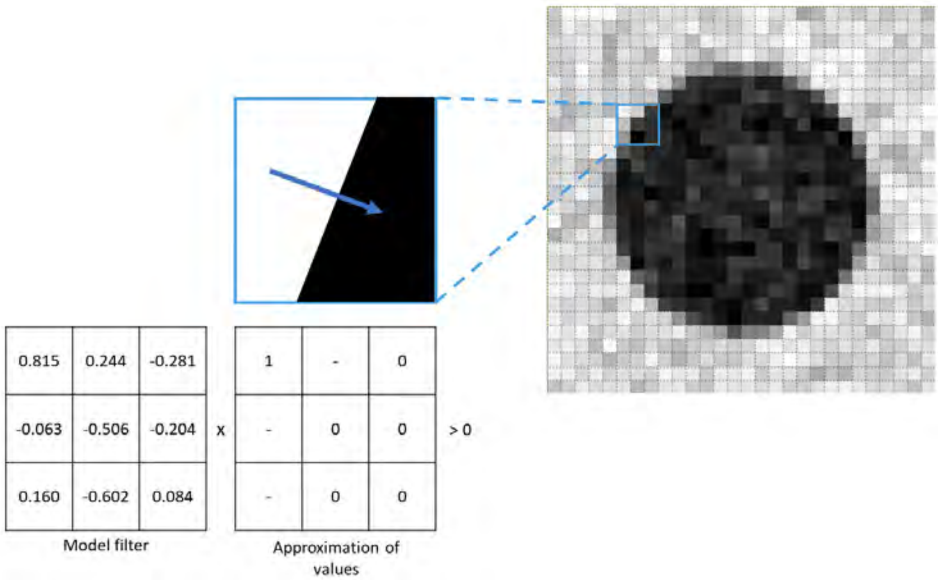

Figure 4.66. Example of dataset 3.0 n20 minimum model filter applied to an ellipse image with the figure in black on a white background. The result of the convolution is an approximation, not the real one. The arrow shows the specific direction of colour change that activates the filter and the hyphen represents a value of gray.

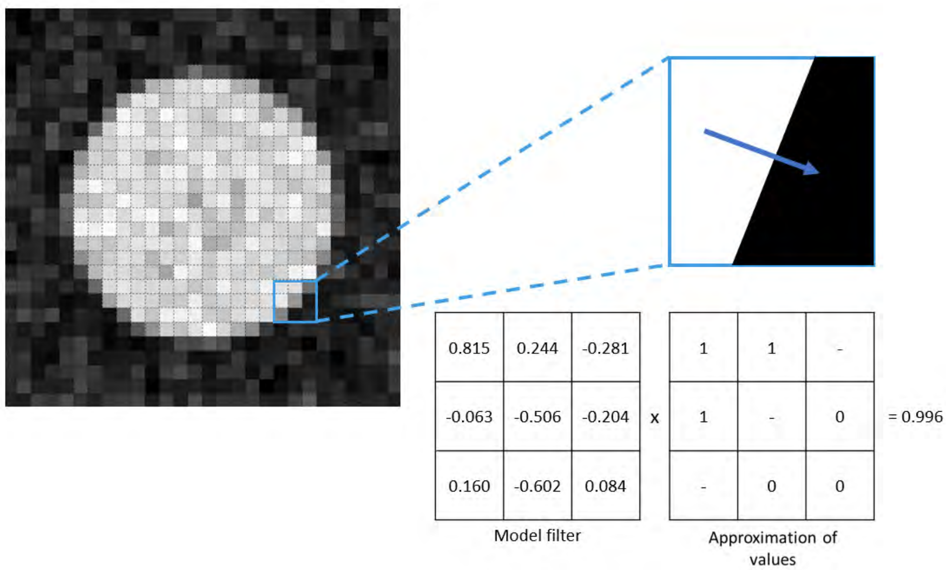

Figure 4.67. Example of dataset 3.0 n20 minimum model filter applied to an ellipse image with the figure in white on a black background. The result of the convolution is an approximation, not the real one. The arrow shows the specific direction of colour change that activates the filter and the hyphen represents a value of gray.

Looking at the plots of the maximum activation pixels it can be observed that in all images there are four pixels coloured as expected: the model is extracting four features by the pooling operation. Figure 4.76 shows the activations for black ellipses on a white background. It can be observed that the pixel that activates always with the strongest value is always on the top left quarter of the image, that is, where the strongest diagonal change from white to black colour is produced. In particular, Figure 4.87 shows a few representations of this kind in more detail, and it can be observed that this pixel varies its location depending on the ellipse size, but remains in the first quarter, such as: (8,11), (2,10) or (6,10). Its values are lower than (14,14). Something similar happens to white ellipses on a black background as it can be seen in Figure 4.78. In this case, the pixel with the strongest activation is placed on the last quarter of the image, the one located on the right bottom part of the image since that is the area where there is a diagonal change from white to black for this images. If the positions are carefully examined in Figure 4.88 their coordinates are higher than (14,14), such as: (20,23), (19,14) or (19,18).

Secondly, feature maps of rectangle samples are shown in Figures 4.79 and 4.81. It can be observed that the filter is behaving for rectangles just as for ellipses, obtaining the two sides that form the top left vertex of the rectangle in the first case, and the ones that form the bottom right corner in the second case. Therefore, examples from Figures 4.66 and 4.67 can be applied also to the rectangles samples. Is in those areas of the image where there the strongest changes from white to black colour are detected. Moreover, the filter is not only extracting changes in diagonal directions but also on vertical and horizontal directions, as the borders being detected are parallel to the borders of the image.

Regarding representations of the maximum activation pixels, it can be observed in Figure 4.80 that generally, there are three pixels activated with high values and located on the top and right borders of the rectangle. In particular, the features are located on the first three quarters of the image (values lower than 14 for both coordinates at the same time), as it can be observed in detail in Figure 4.89: (5,13), (5,18), (14,3), (9,11), (9,16, (14,9) and so on. For the other combination of figure and background colour, Figure 4.82 shows also a high activation of three pixels but on the right and bottom borders, which means positions like (23,8), (22,23), (8,21), (11,26) or (12,20), as it can be observed in Figure 4.90. The fact that for samples of rectangles there are more features that take high values, can be used by the CNN to perform the classification and differentiate between rectangles and ellipses.

Finally, in the case of triangles Figure 4.83 shows the activation maps for images with a black triangle on a white background and it can be observed that the left side of the rectangle is detected perfectly by the filter. These images have clear examples of a diagonal change from white to black, therefore the filter is being activated as it happened in the example of ellipses from Figure 4.66. In the case of images that have a white triangle with a black background, Figure 4.85, the filter is only able to recognise the base of the triangles since that is where there is a change from white to black, not on the diagonal but vertical direction, as it happened in the case of rectangles. The change from white to black that generates the right sides of the triangles is not detected since the direction is diagonal from the left bottom to the top right area of the image, that is, the opposite direction that the filter is detecting.

Looking at pixels with maximum activation, Figure 4.84 there are always two pixels with high activations on the left side of the triangle. This fact can be used also to distinguish triangles from the rest of the classes, since for ellipses usually only one pixel activates with high values, and for rectangles there are normally three pixels white high activations. In particular, Figure 4.91 shows that the pixels, among others, take the positions (10,12), (9,14), (11,10) or (5,20). For images with white triangles on a black background two pixels are activated too, but in this case the pixels are placed on the triangle's base. Figure 4.86 shows these representations and in Figure 4.92 it can be observed the exact positions, for example: (17,18), (12,18), (21,23) or (7,23).

As a general observation regarding all black figures with a white background, in their feature maps it can be observed that the latest pixel, that is the one located in position (28,28), is always activated. The reason is that the kind of padding applied by default by Keras library is zero padding. As the filter is detecting diagonal changes from white colour to black, if the background is white but the image has black padding, the filter activates in that corner, since it detects the change in the specific direction.

Finally, after having observed the positions of each one of the four features and the values they take in general for all images, the values obtained in the correlation plot from Figure 4.56 make sense. For example, features 1 and 3 have positive correlation and they both are placed on the left part of the image and they are usually activated by the filter at the same time. On the contrary, feature 1 and 4 have a high negative correlation, which means that when the former is activated the latter is not, and that is exactly what happens, since they are located on opposite areas of the image.

To sum up, for images with black figures on a white background, the filter is extracting borders located on the top left part of the image whereas for images with white figures on a black background the filter detects the borders placed on the opposite area on the image, the bottom left area. That is how with one single filter the model is able to recognise the classes regardless of its colours. A combination of the luminance values extracted for each class, enables the CNN to distinguish among ellipses, triangles and rectangles.

Apart from the analysis of features and the physical meaning of the model, in this experiment, mistakes made by the model are also analysed. In order to perform this study, first of all, samples that are incorrectly classified are displayed on the PCA plots, so in a visual form the mistakes locations can be analysed. Figure 4.68 shows the three projections on the planes formed by pairs of the first three principal components with the mistakes in black colour.

Looking at the plots it can be confirmed what has been previously stated, the majority of mistakes appear at the limit between classes. There are no mistakes in areas where there is a high and well defined group of samples belonging to the same class. For example, in the first projection, triangles are placed on the top part of the plot and there are only samples incorrectly classified on the zone where the samples are closer to the group of ellipses, around the value 0.02 for PC2.

Once the mistakes have been analysed in a general form, it is interesting to study deeply some samples that are in the space next to others that are correctly classified. In order to do this, the samples that have been considered to be the most noteworthy are selected. Some samples stand out because they are "inside" a cloud of samples that belongs to another different class. Other samples stand out because the model has failed classifying them, even though they belong to the same class as those in their region. To perform the analysis, two kinds of representations are built. On the one hand, both detailed maximum activation pixels plot and the feature map is shown for the specific image, so the features can be observed visually. On the other hand, a new kind of visualizations is developed. They are bar diagrams which show the mean of the image features values by class and combination of colours (so that the representation is something similar to K-means centroid) plotted together with the features of the mistaken samples. The means are not weighted by the distance to the centroid, but are pure means. In this way, it can be compared the feature values that the wrongly classified image has with the values that approximately should have. It can be observed that each image has one or several features quite different from the average, which make the CNN fail.

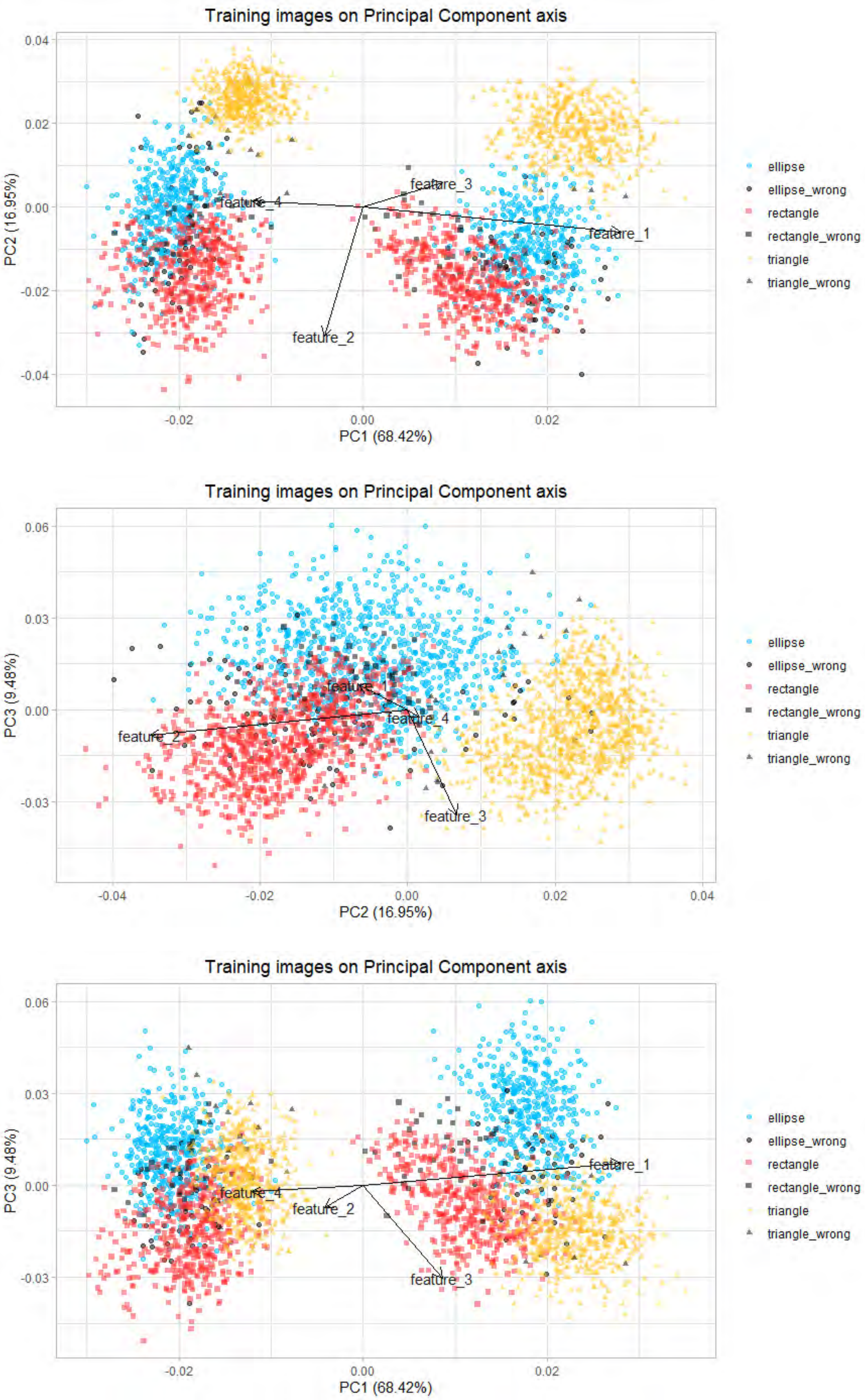

Figure 4.68. Projections of feature dataset generated for each sample from dataset 3.0 n20 with its three classes on the plane formed by the first three principal components: PCA 1 vs. PCA 2 (first plot), PCA 2 vs. PCA 3 (second plot) and PCA 1 vs. PCA 3 (third plot). Samples incorrectly classified by the minimum CNN model appear in black colour. The component that explains the largest amount of information is placed on the horizontal axis.

A brief analysis of each one of the specific samples selected is carried out next:

- Image *dataset3.0#training#n20#ellipse#kw#00169*: the image that is wrongly classified has a feature 3 with a value higher than the average and a value for feature 1 lower than the average (Figure 4.69).

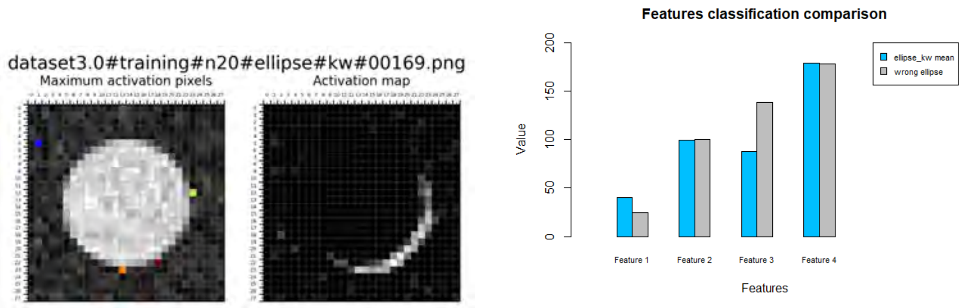

Figure 4.69. For an specific image: detailed maximum activation pixels representation together with feature map plot (left) and bar plot for the comparison of features (right).

- Image *dataset3.0#training#n20#ellipse#wk#00003*: the image that is wrongly classified has a feature 3 with a value higher than the average and a value for features 1 adn 4 lower than the average (Figure 4.70).

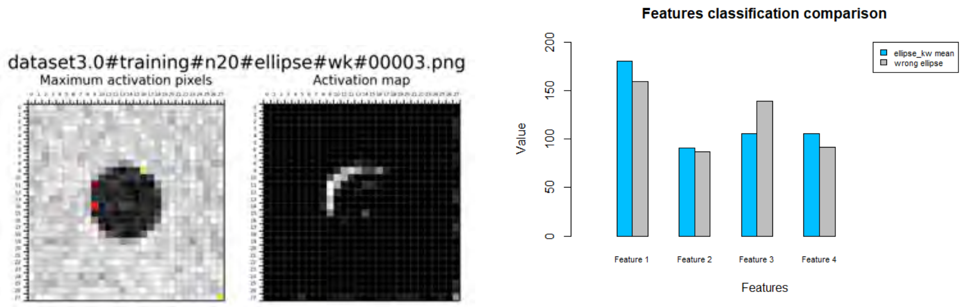

Figure 4.70. For an specific image: detailed maximum activation pixels representation together with feature map plot (left) and bar plot for the comparison of features (right).

- Image *dataset3.0#training#n20#rectangle#kw#00156*: the image that is wrongly classified has a value for features 1, 2 and 3 lower than the average (Figure 4.71).

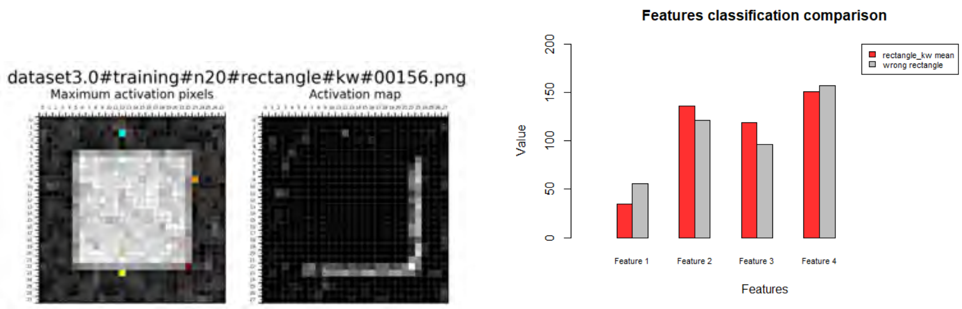

Figure 4.71. For an specific image: detailed maximum activation pixels representation together with feature map plot (left) and bar plot for the comparison of features (right).

- Image *dataset3.0#training#n20#rectangle#wk#00076*: the image that is wrongly classified has feature 3 with a lower value value than the average (Figure 4.72).

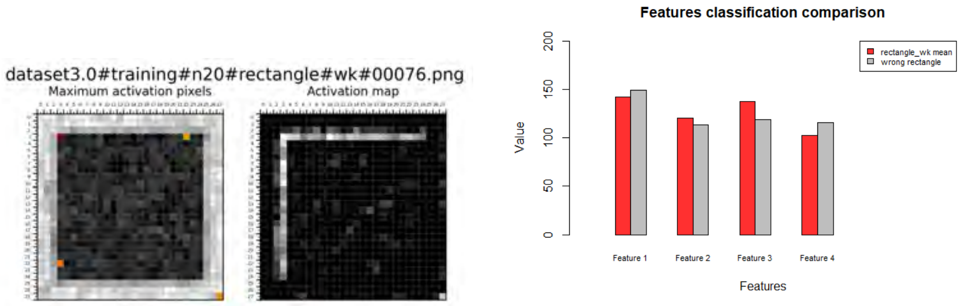

Figure 4.72. For an specific image: detailed maximum activation pixels representation together with feature map plot (left) and bar plot for the comparison of features (right).

- Image *dataset3.0#training#n20#triangle#kw#00345*: the image that is wrongly classified has a value for feature 2 higher than the average and for features 1, 3 and four lower than the average (Figure 4.73).

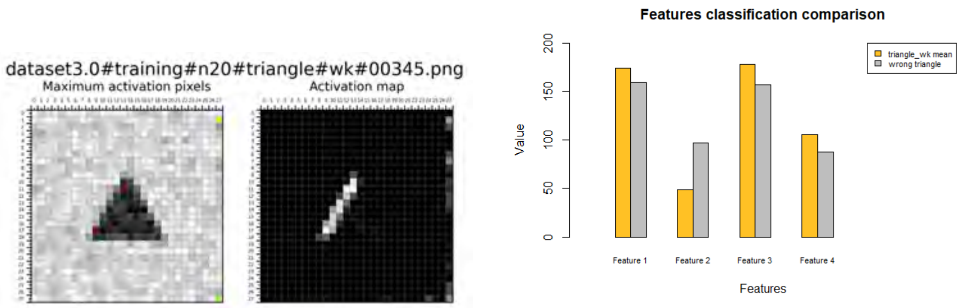

Figure 4.73. For an specific image: detailed maximum activation pixels representation together with feature map plot (left) and bar plot for the comparison of features (right).

- Image *dataset3.0#training#n20#triangle#wk#00451*: the image that is wrongly classified has a value for feature 2 higher than the average (Figure 4.74).

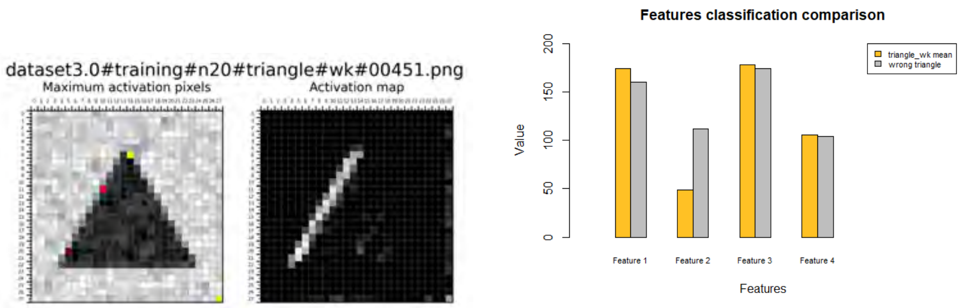

Figure 4.74. For an specific image: detailed maximum activation pixels representation together with feature map plot (left) and bar plot for the comparison of features (right).

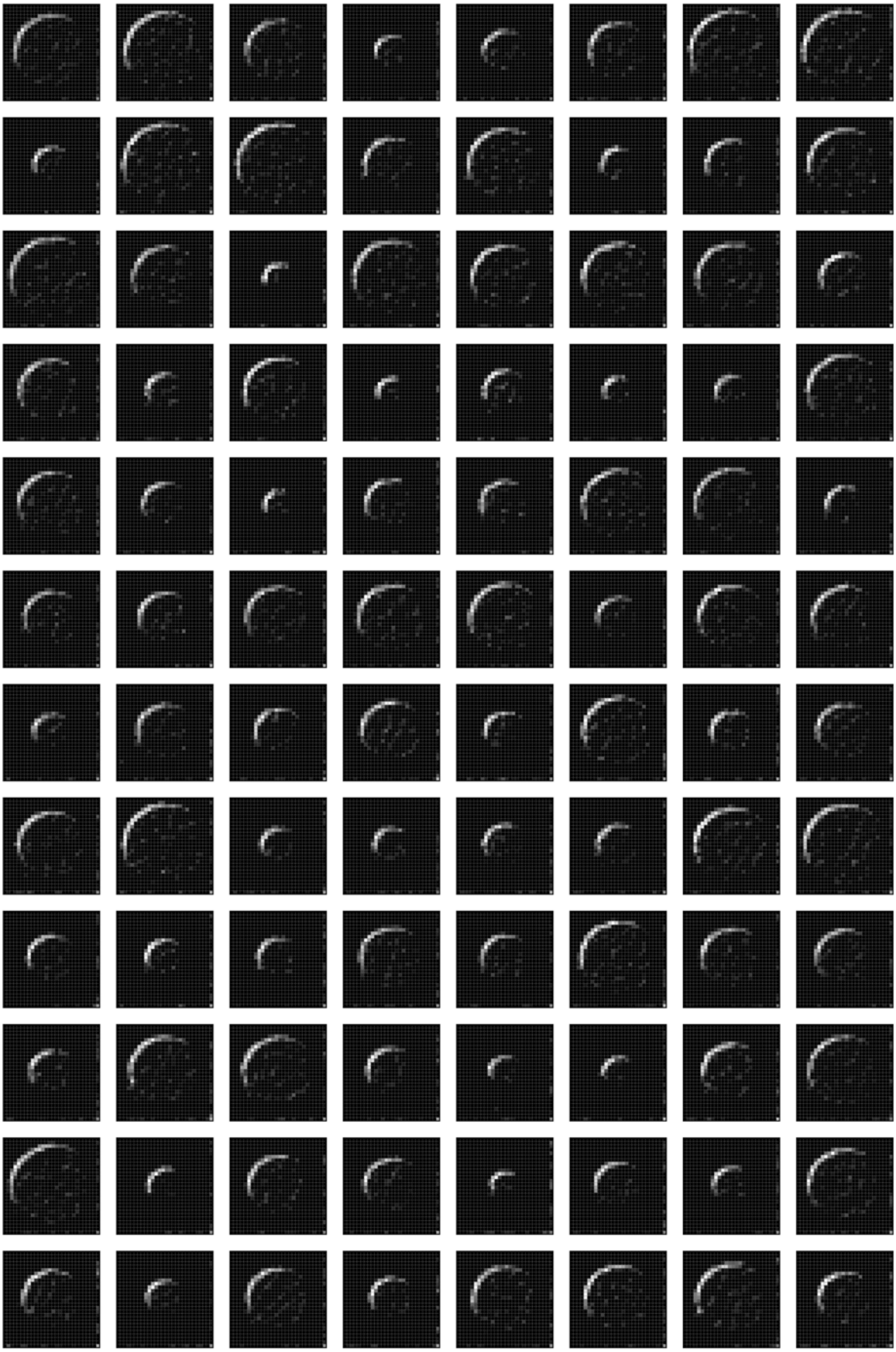

Figure 4.75. Feature maps of several ellipse samples with the figure in black on a white background from dataset 3.0 n20. Best viewed in electronic form, zoom in to distinguish the pixel grid.

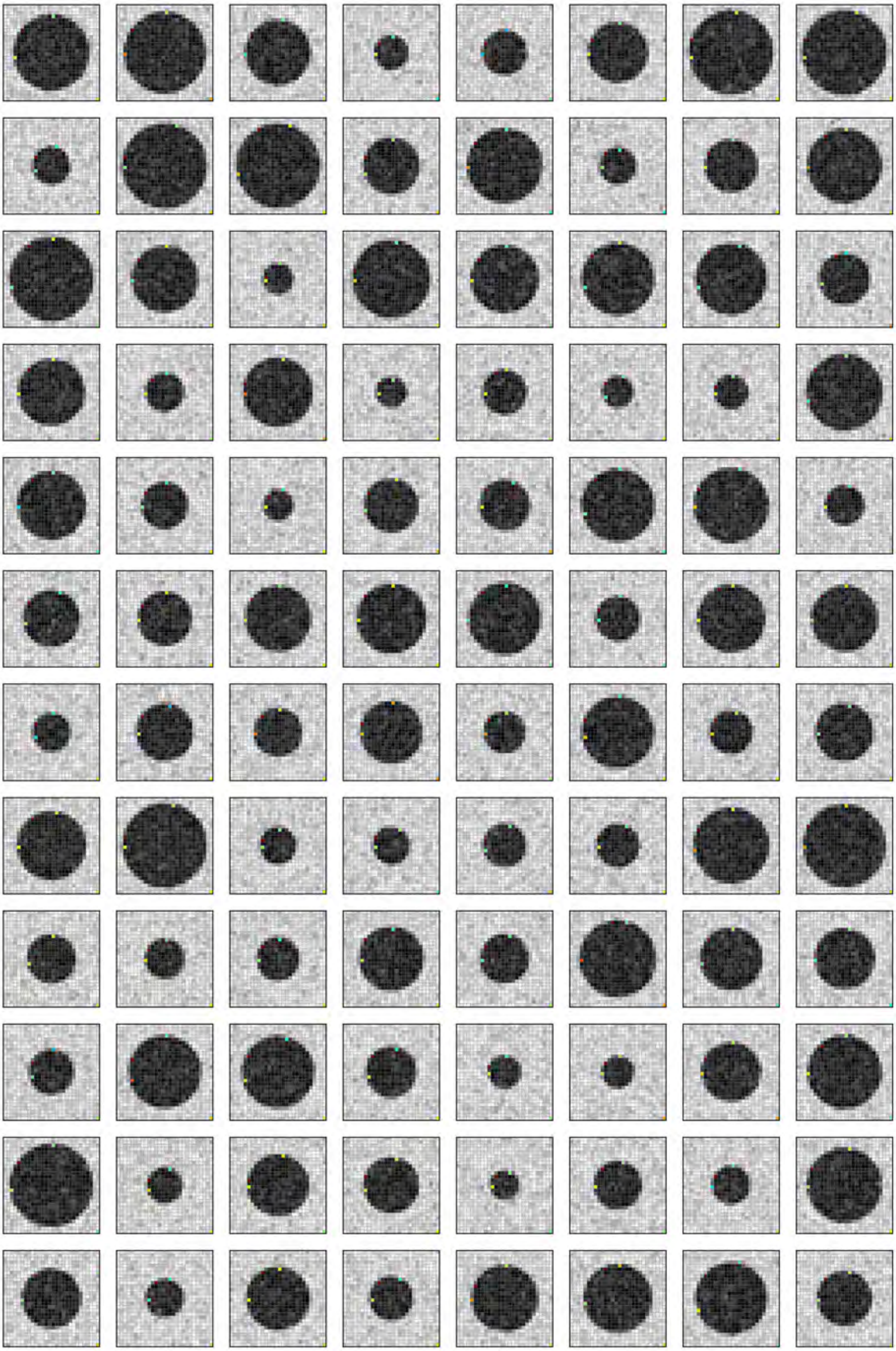

Figure 4.76. Original images of several ellipses from dataset 3.0 n20 with the figure in black on a white background and with maximum activation pixels in different colours according to each value (red representing the highest value and blue the lowest one). Best viewed in electronic form, zoom in to distinguish the pixel grid.

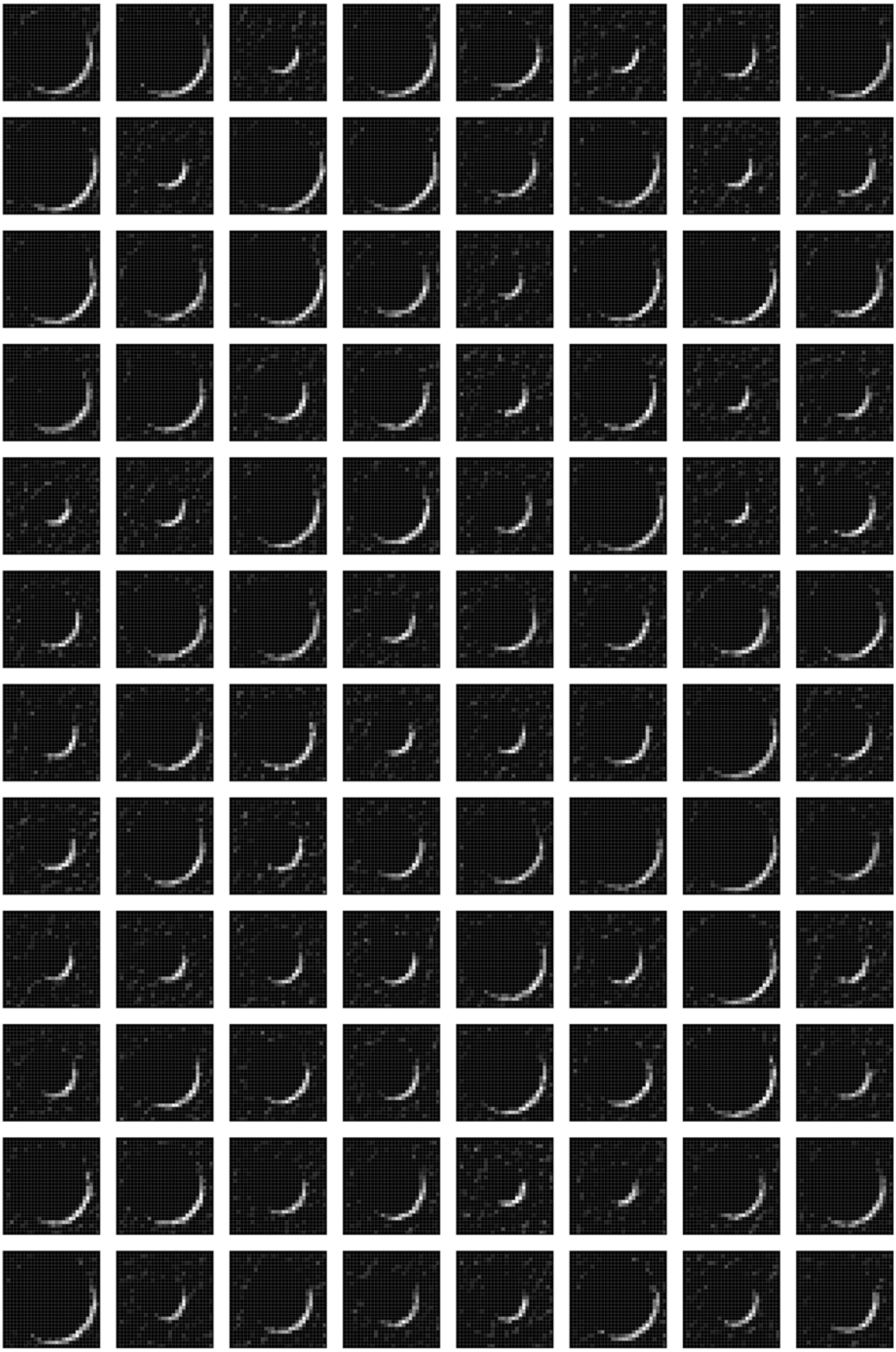

Figure 4.77. Feature maps of several ellipse samples with the figure in white on a black background from dataset 3.0 n20. Best viewed in electronic form, zoom in to distinguish the pixel grid.

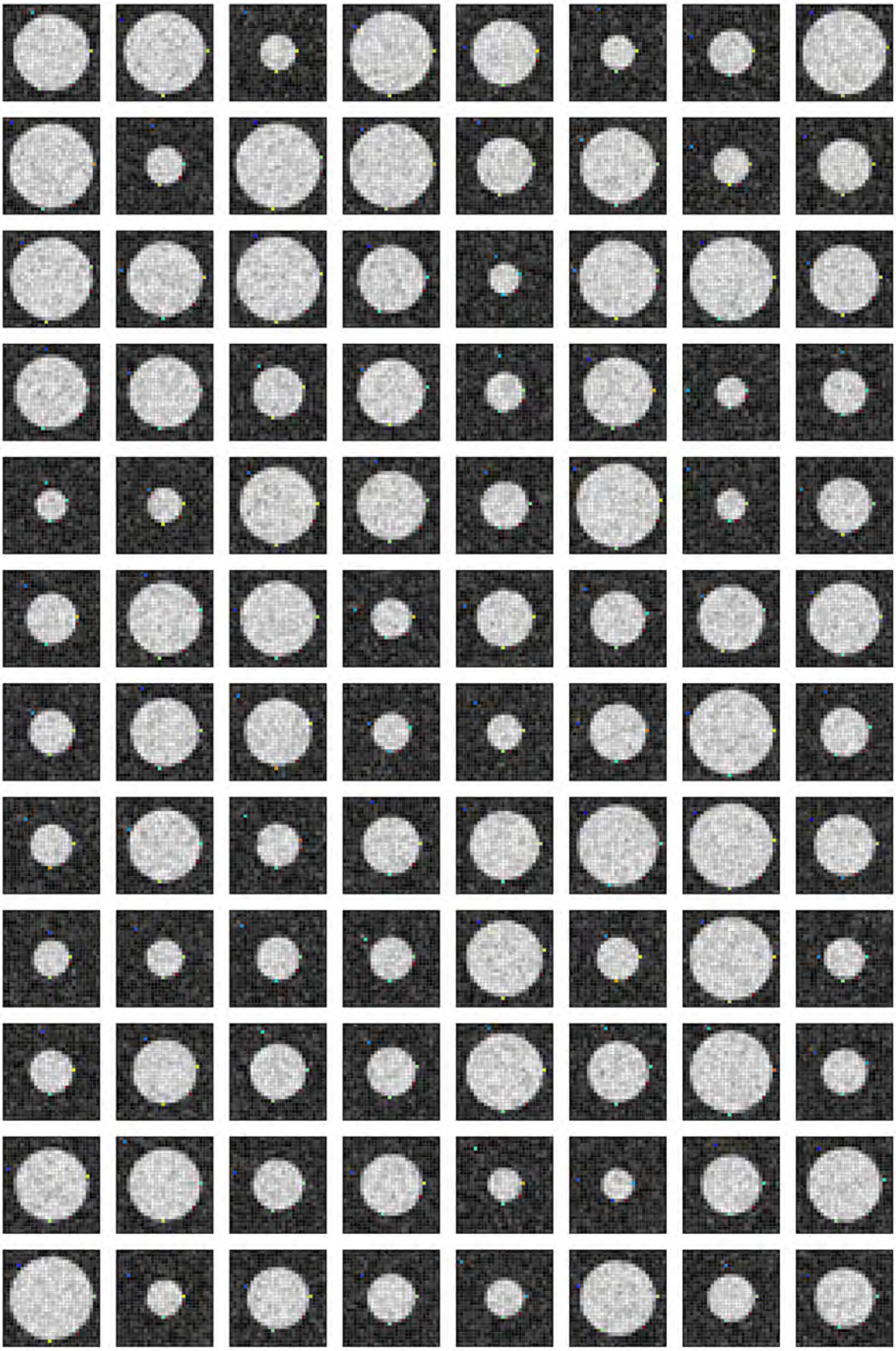

Figure 4.78. Original images of several ellipses from dataset 3.0 n20 with the figure in white on a black background and with maximum activation pixels in different colours according to each value (red representing the highest value and blue the lowest one). Best viewed in electronic form, zoom in to distinguish the pixel grid.

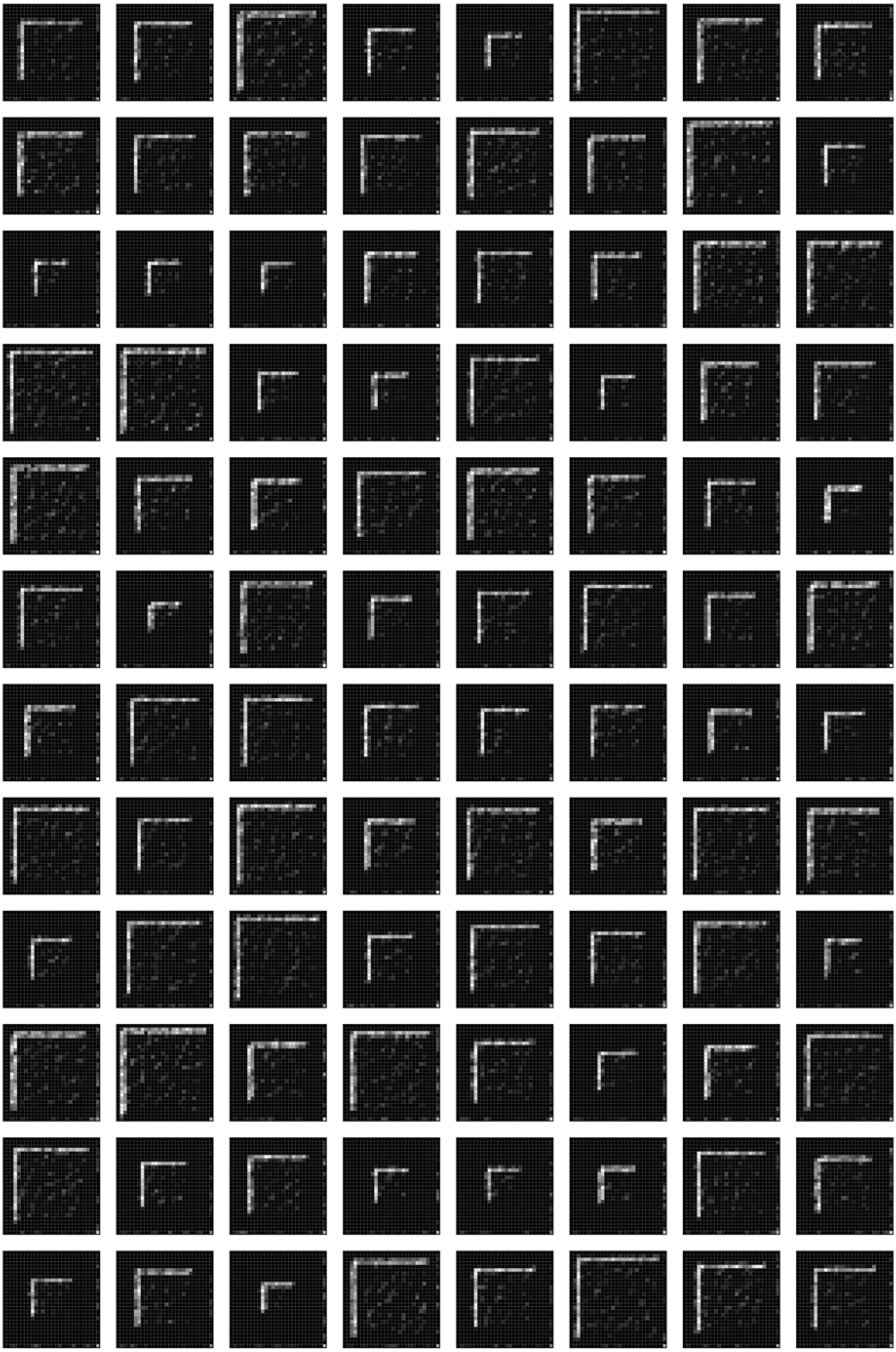

Figure 4.79. Feature maps of several rectangle samples with the figure in black on a white background from dataset 3.0 n20. Best viewed in electronic form, zoom in to distinguish the pixel grid.

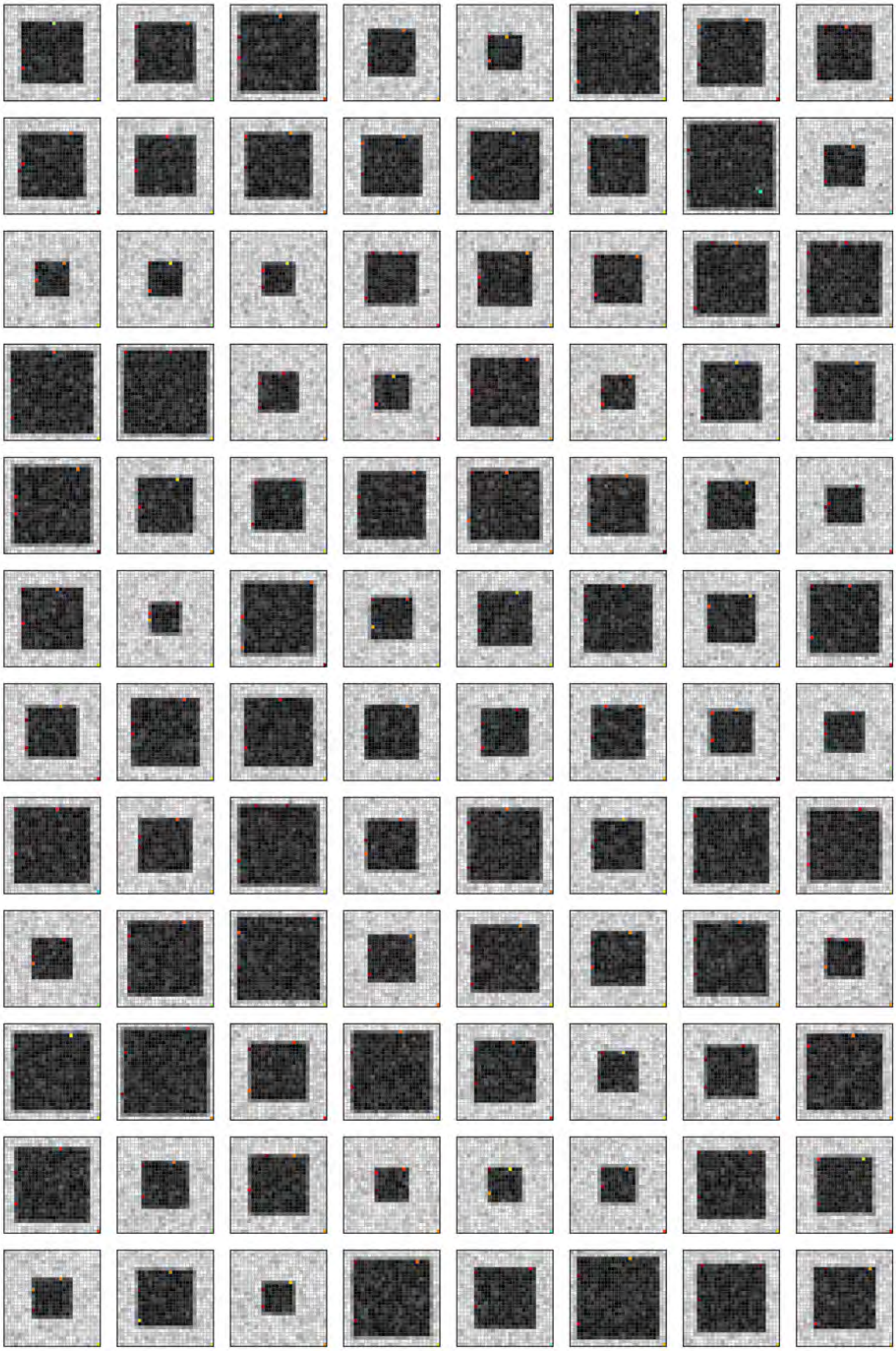

Figure 4.80. Original images of several rectangles from dataset 3.0 n20 with the figure in black on a white background and with maximum activation pixels in different colors according to each value (red representing the highest value and blue the lowest one). Best viewed in electronic form, zoom in to distinguish the pixel grid.

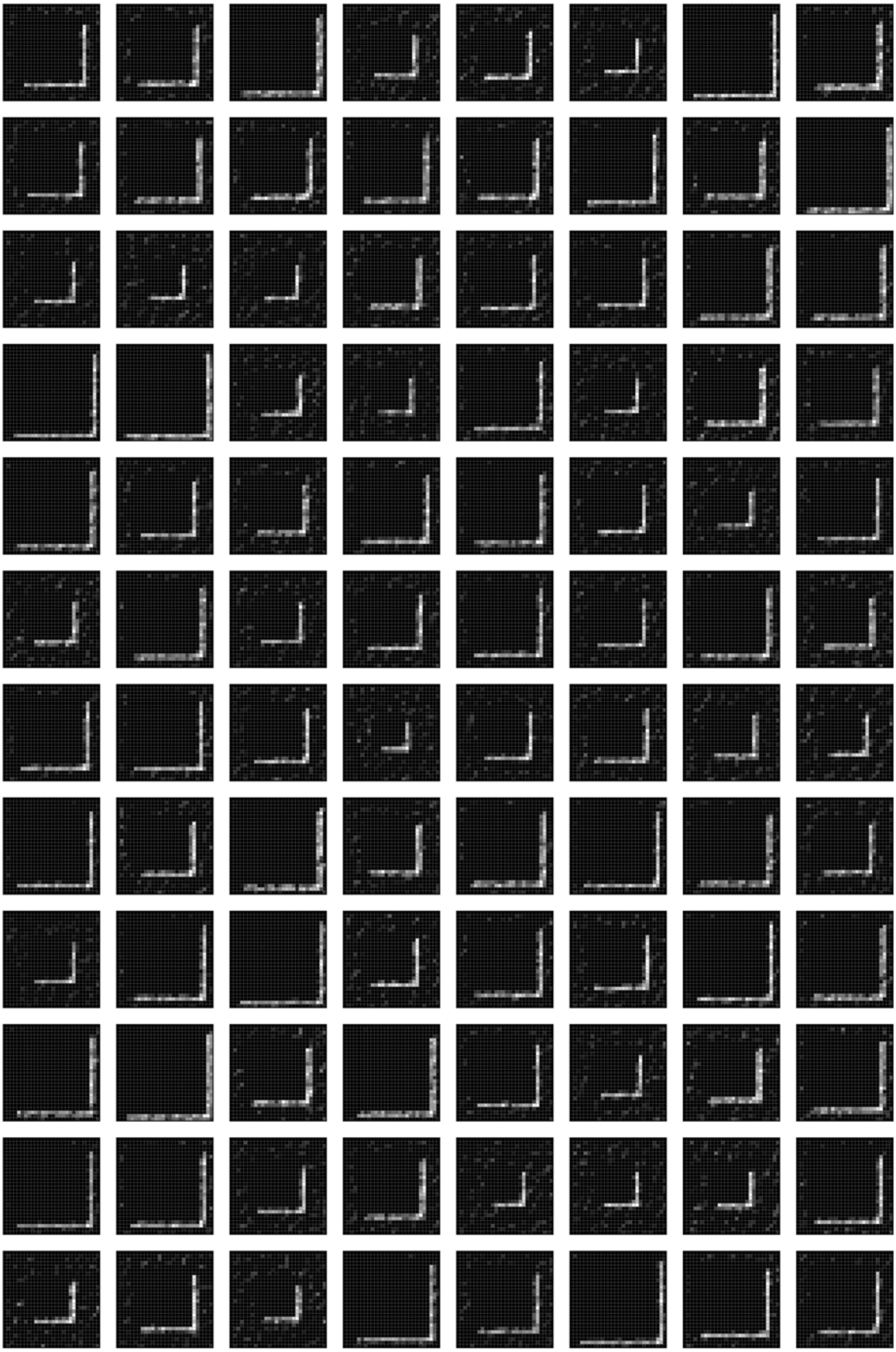

Figure 4.81. Feature maps of several rectangle samples with the figure in white on a black background from dataset 3.0 n20. Best viewed in electronic form, zoom in to distinguish the pixel grid.

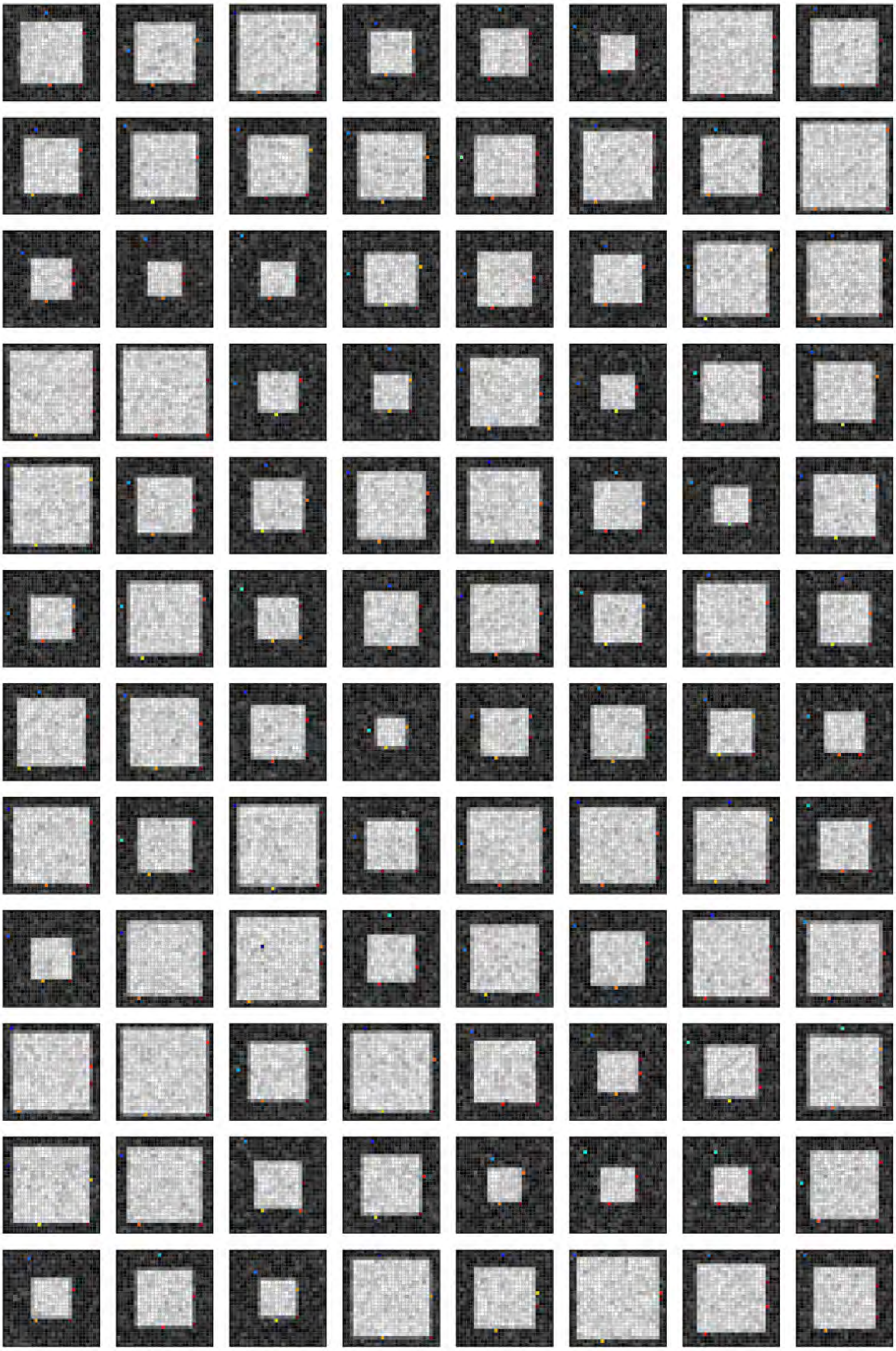

Figure 4.82. Original images of several rectangles from dataset 3.0 n20 with the figure in white on a black background and with maximum activation pixels in different colors according to each value (red representing the highest value and blue the lowest one). Best viewed in electronic form, zoom in to distinguish the pixel grid.

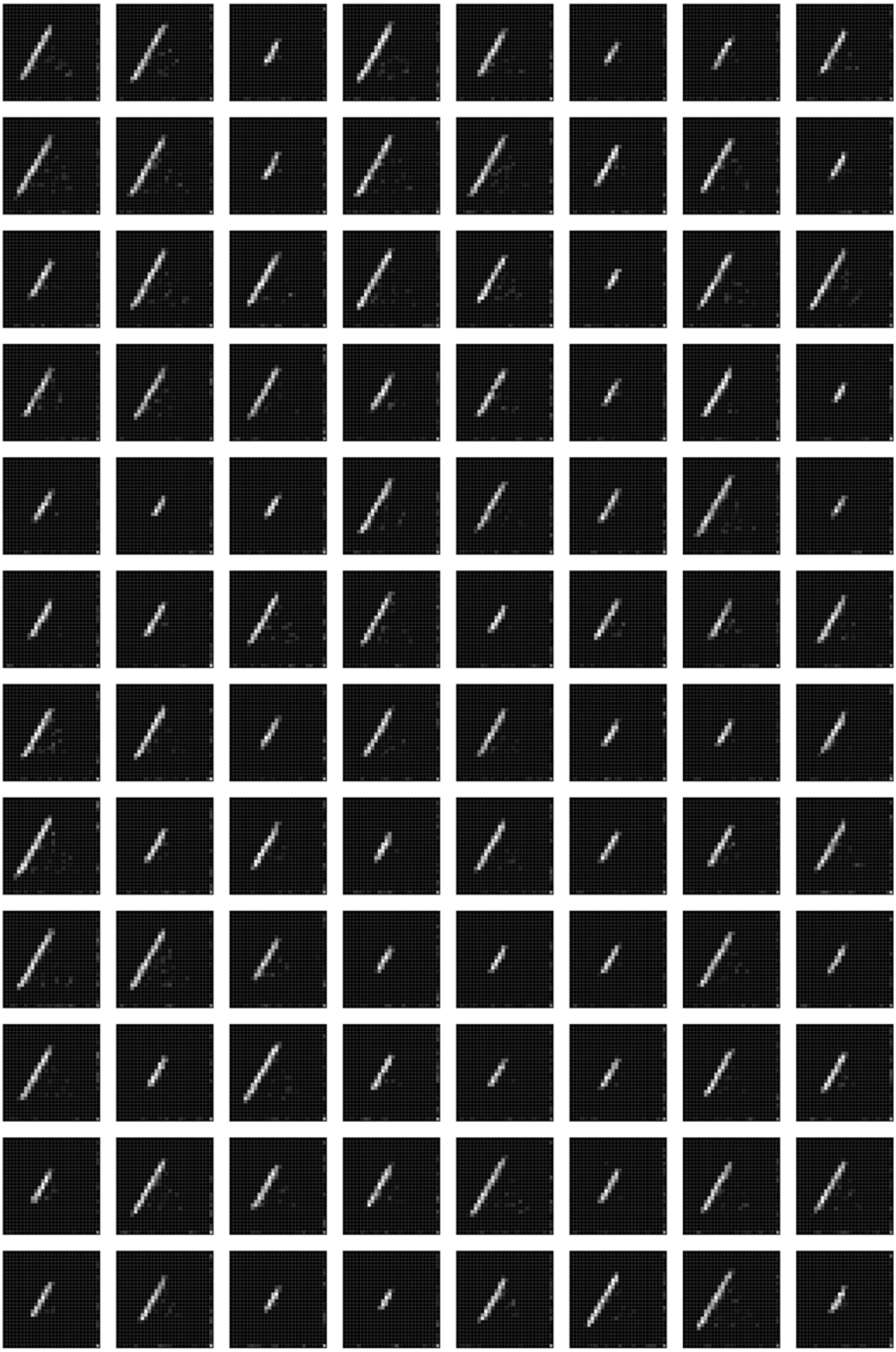

Figure 4.83. Feature maps of several triangles samples with the figure in black on a white background from dataset 3.0 n20. Best viewed in electronic form, zoom in to distinguish the pixel grid.

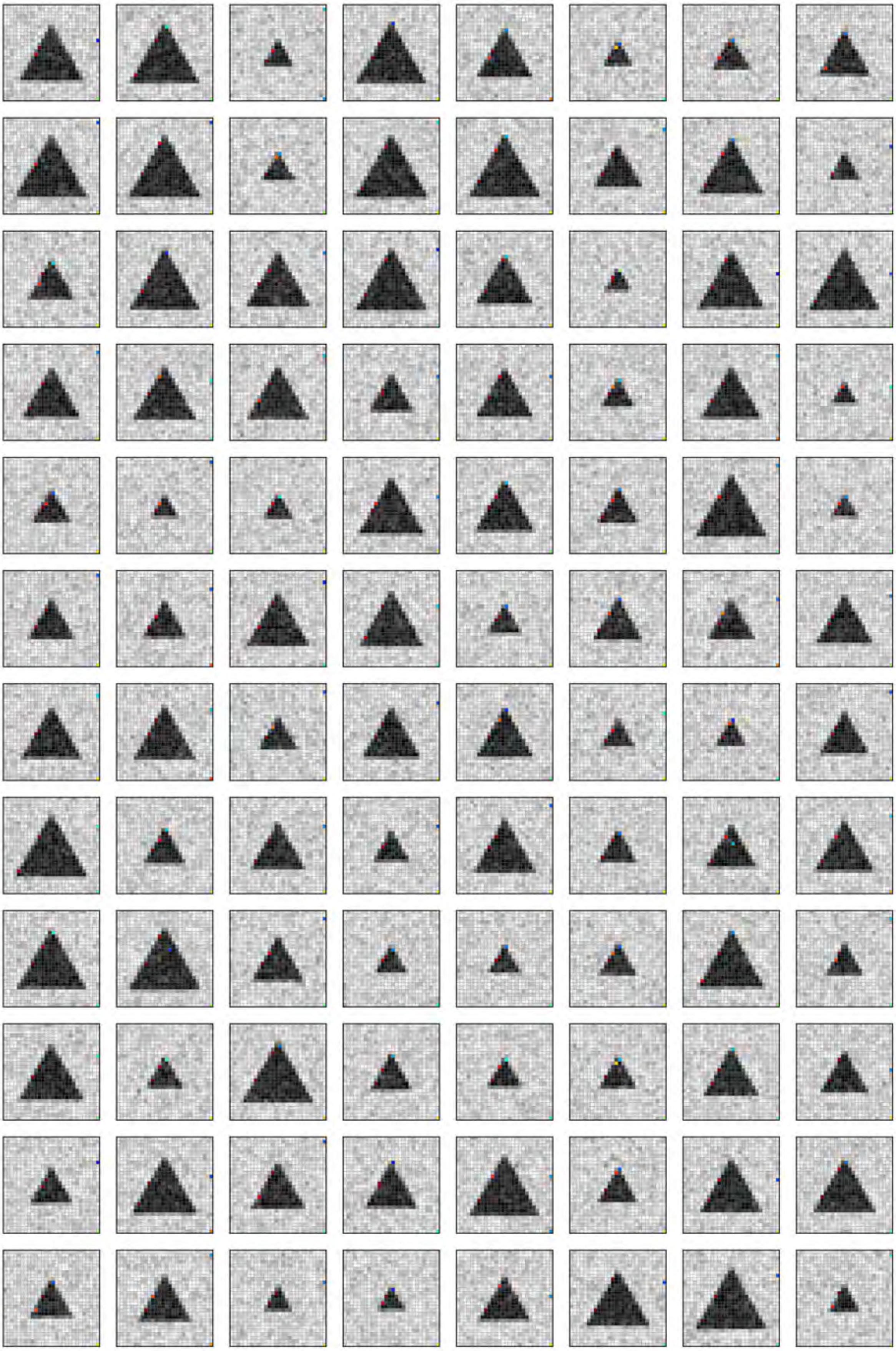

Figure 4.84. Original images of several triangles from dataset 3.0 n20 with the figure in black on a white background and with maximum activation pixels in different colours according to each value (red representing the highest value and blue the lowest one). Best viewed in electronic form, zoom in to distinguish the pixel grid.

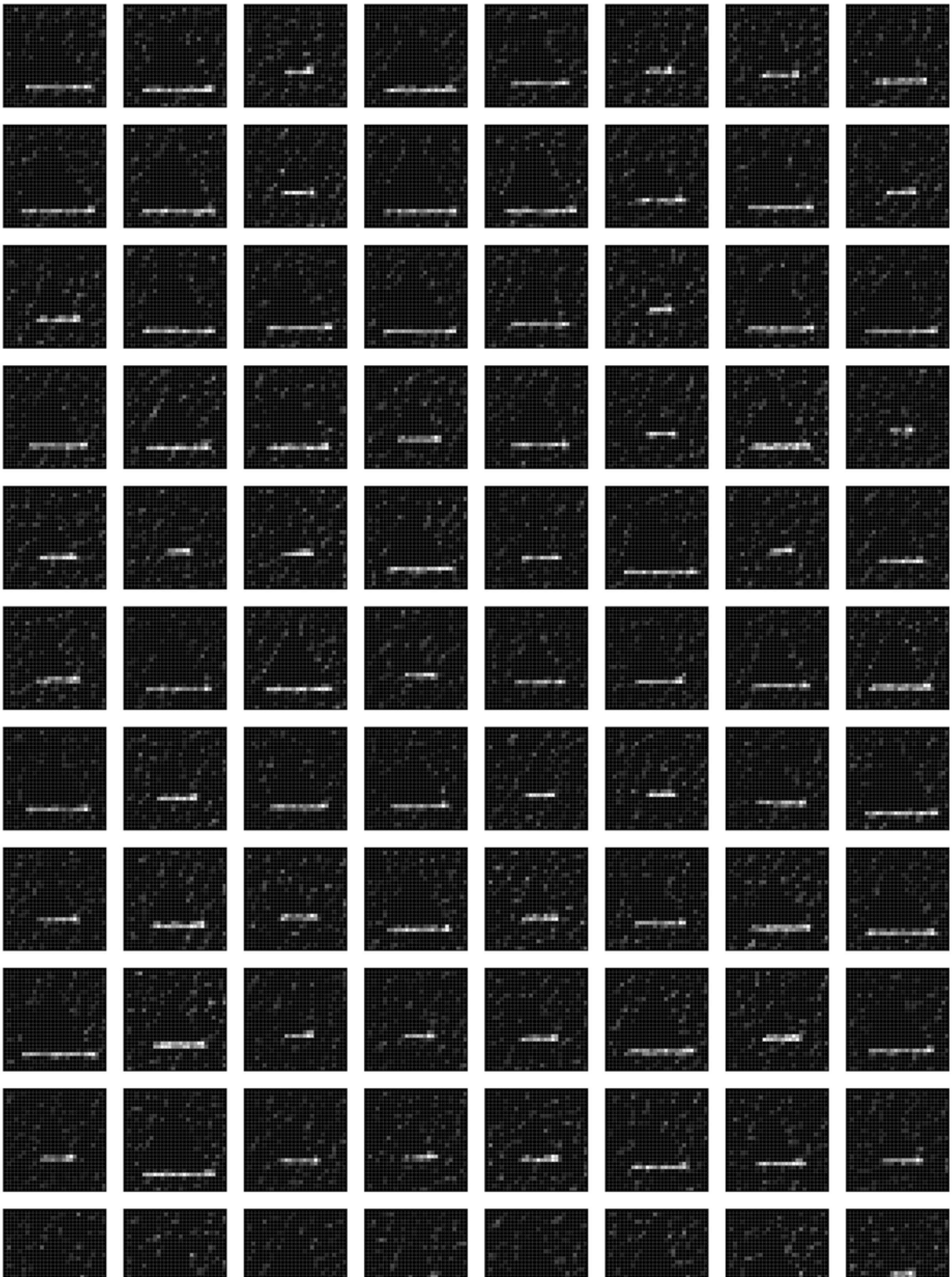

Figure 4.85. Feature maps of several triangles samples with the figure in white on a black background from dataset 3.0 n20. Best viewed in electronic form, zoom in to distinguish the pixel grid.

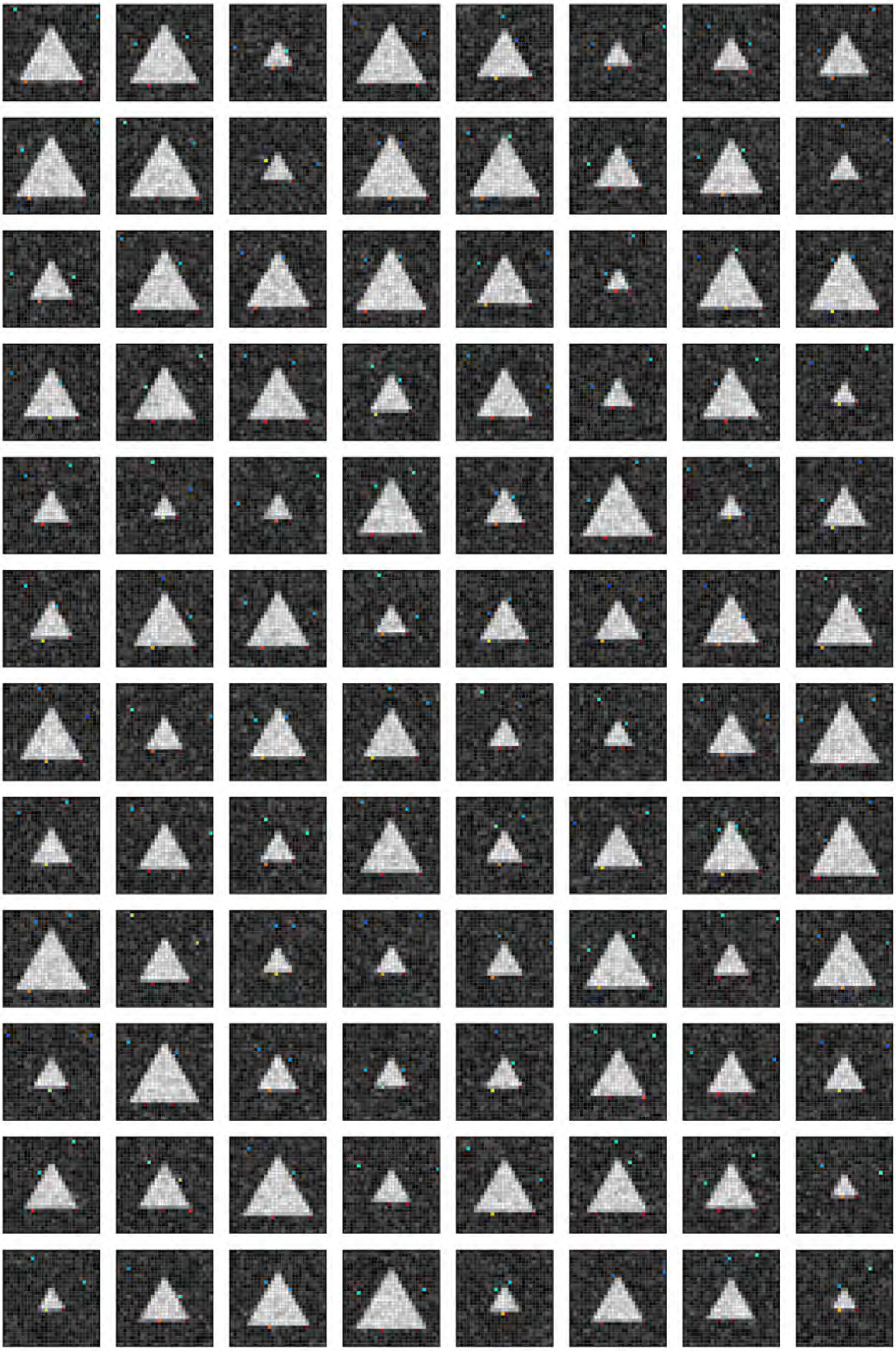

Figure 4.86. Original images of several triangles from dataset 3.0 n20 with the figure in white on a black background and with maximum activation pixels in different colours according to each value (red representing the highest value and blue the lowest one). Best viewed in electronic form, zoom in to distinguish the pixel grid.

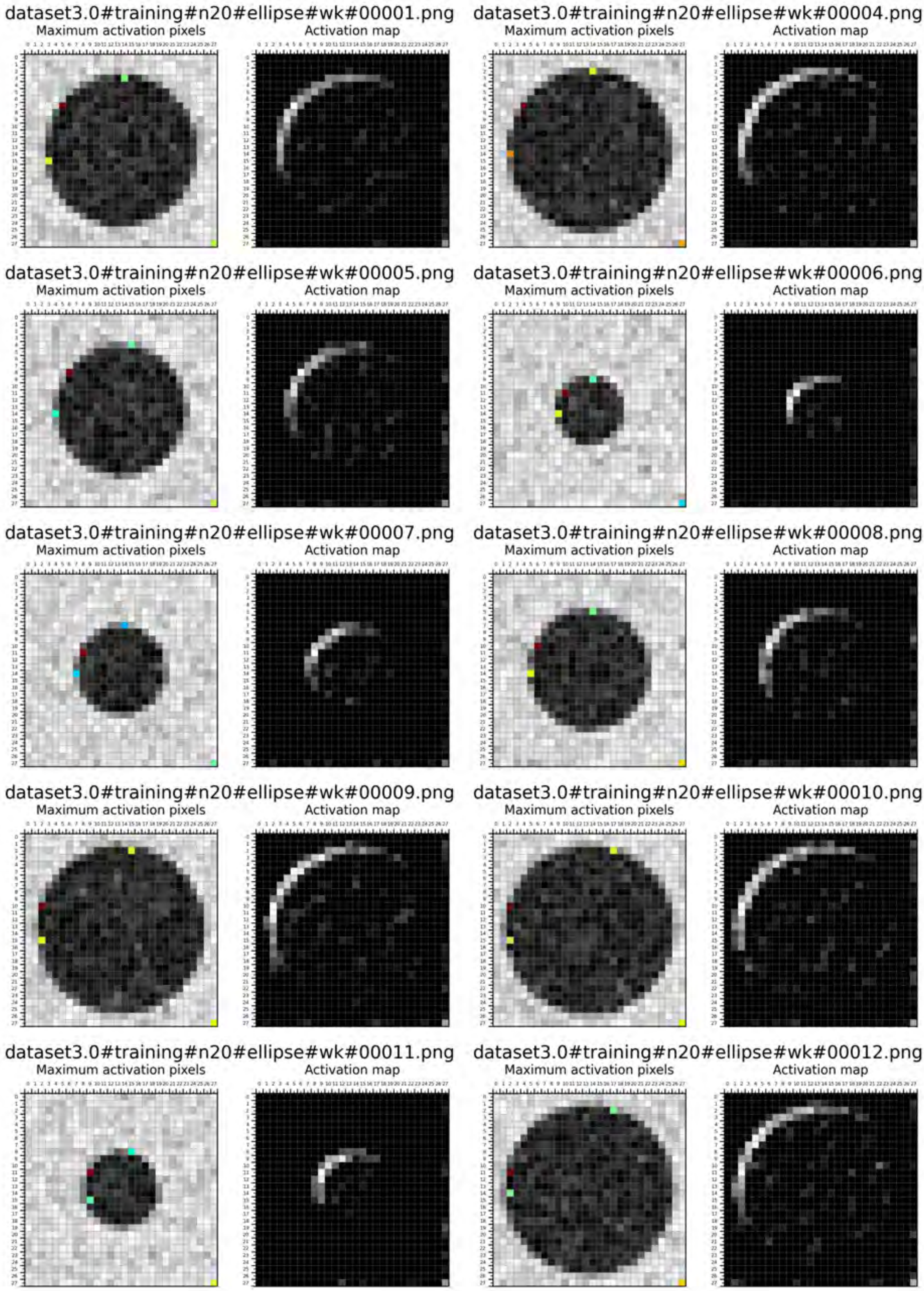

Figure 4.87. Both feature map and maximum activation pixel representations of several ellipses samples with the figure in black on a white background from dataset 3.0. n20 in a detailed form. Best viewed in electronic form, zoom in to distinguish the pixel grid.

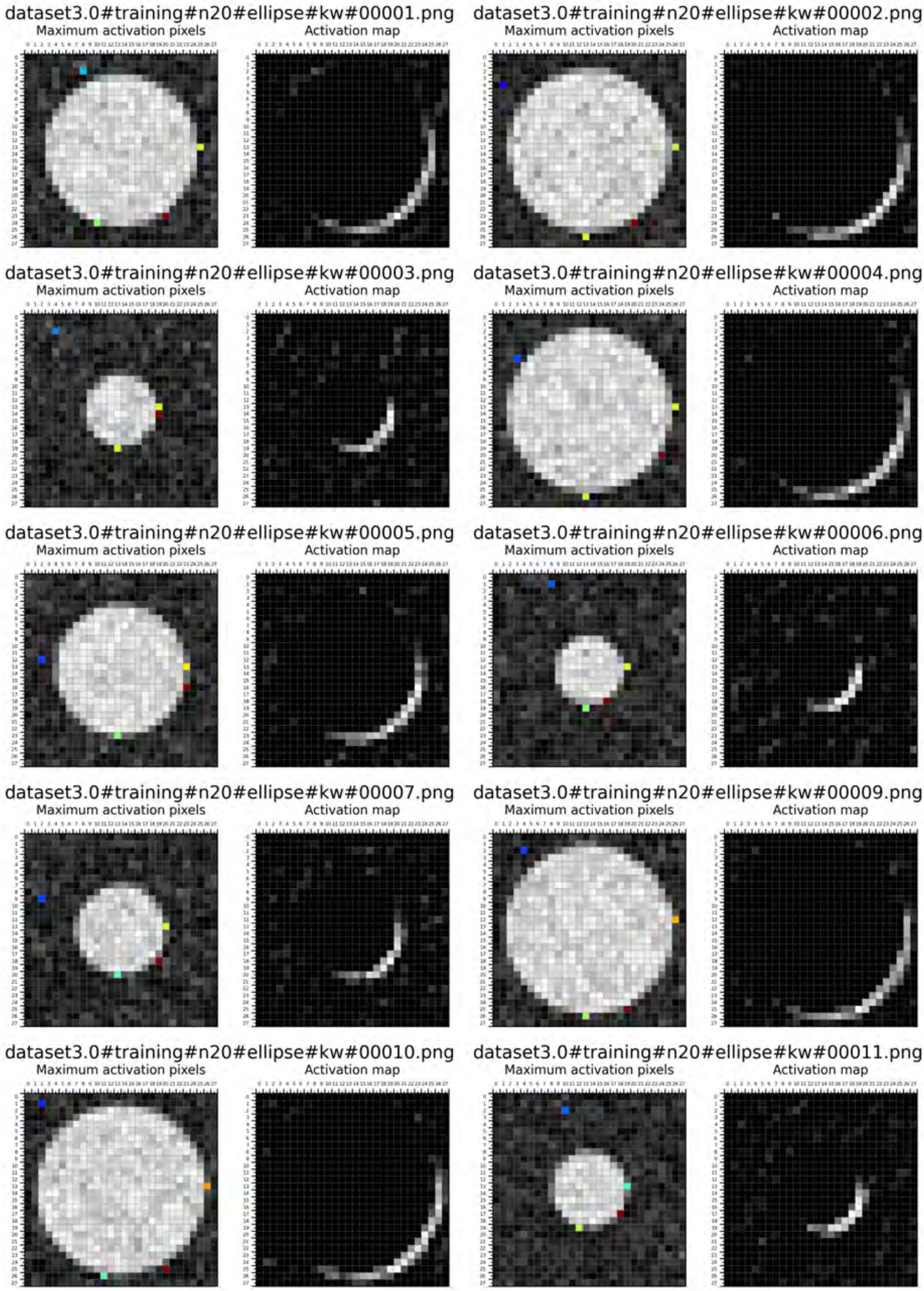

Figure 4.88. Both feature map and maximum activation pixel representations of several ellipses samples with the figure in white on a black background from dataset 3.0. n20 in a detailed form. Best viewed in electronic form, zoom in to distinguish the pixel grid.

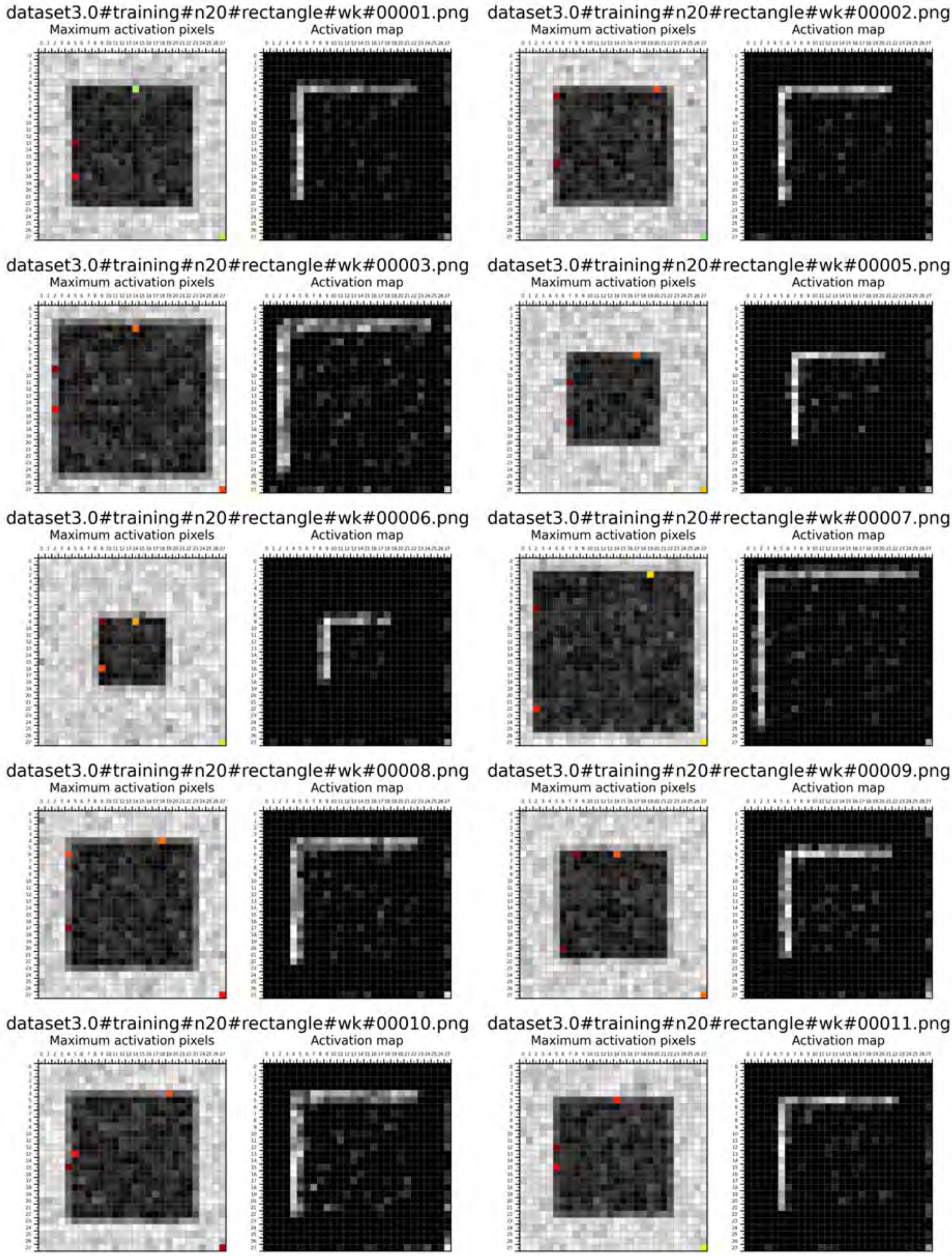

Figure 4.89. Both feature map and maximum activation pixel representations of several rectangles samples with the figure in black on a white background from dataset 3.0. n20 in a detailed form. Best viewed in electronic form, zoom in to distinguish the pixel grid.

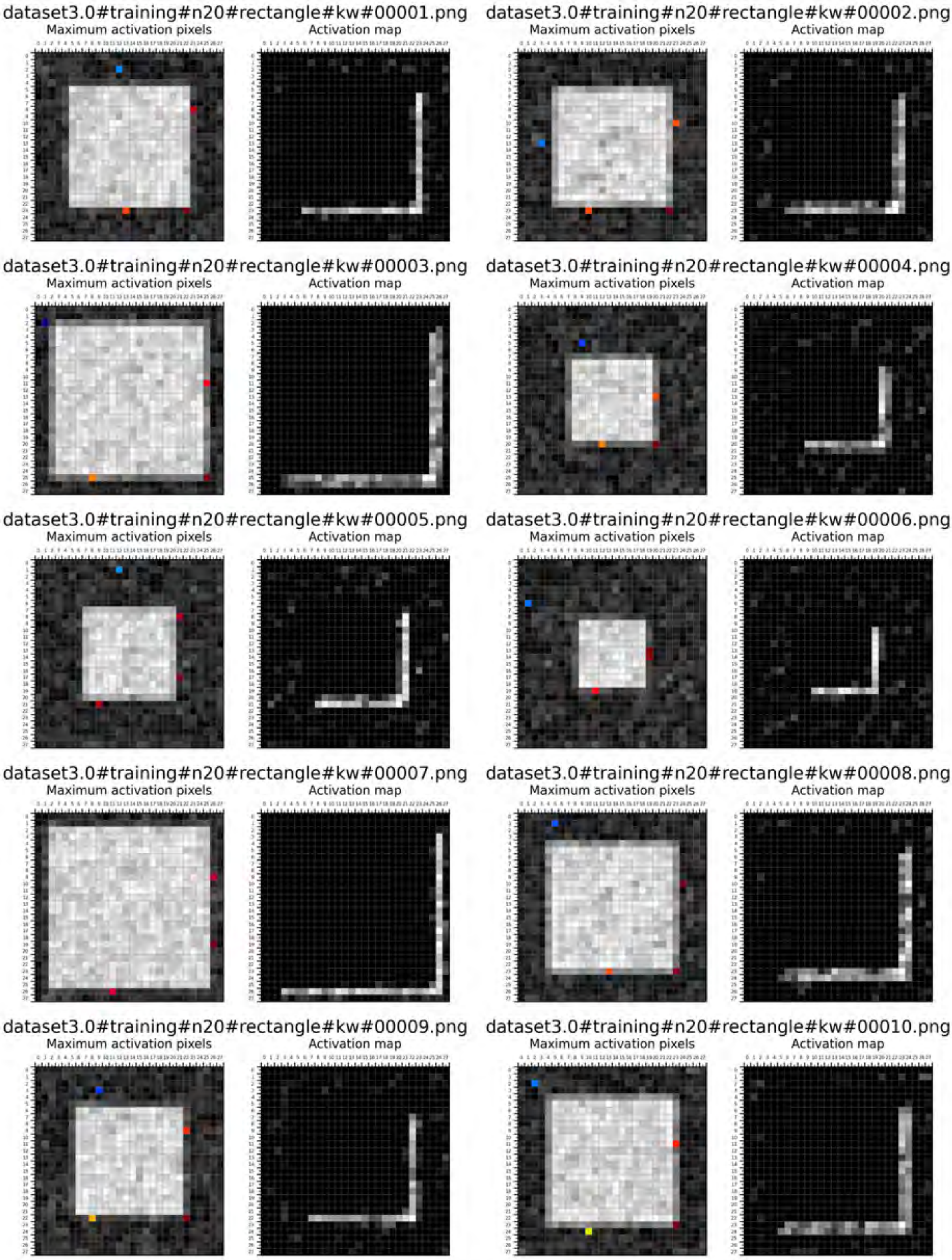

Figure 4.90. Both feature map and maximum activation pixel representations of several rectangles samples with the figure in white on a black background from dataset 3.0. n20 in a detailed form. Best viewed in electronic form, zoom in to distinguish the pixel grid.

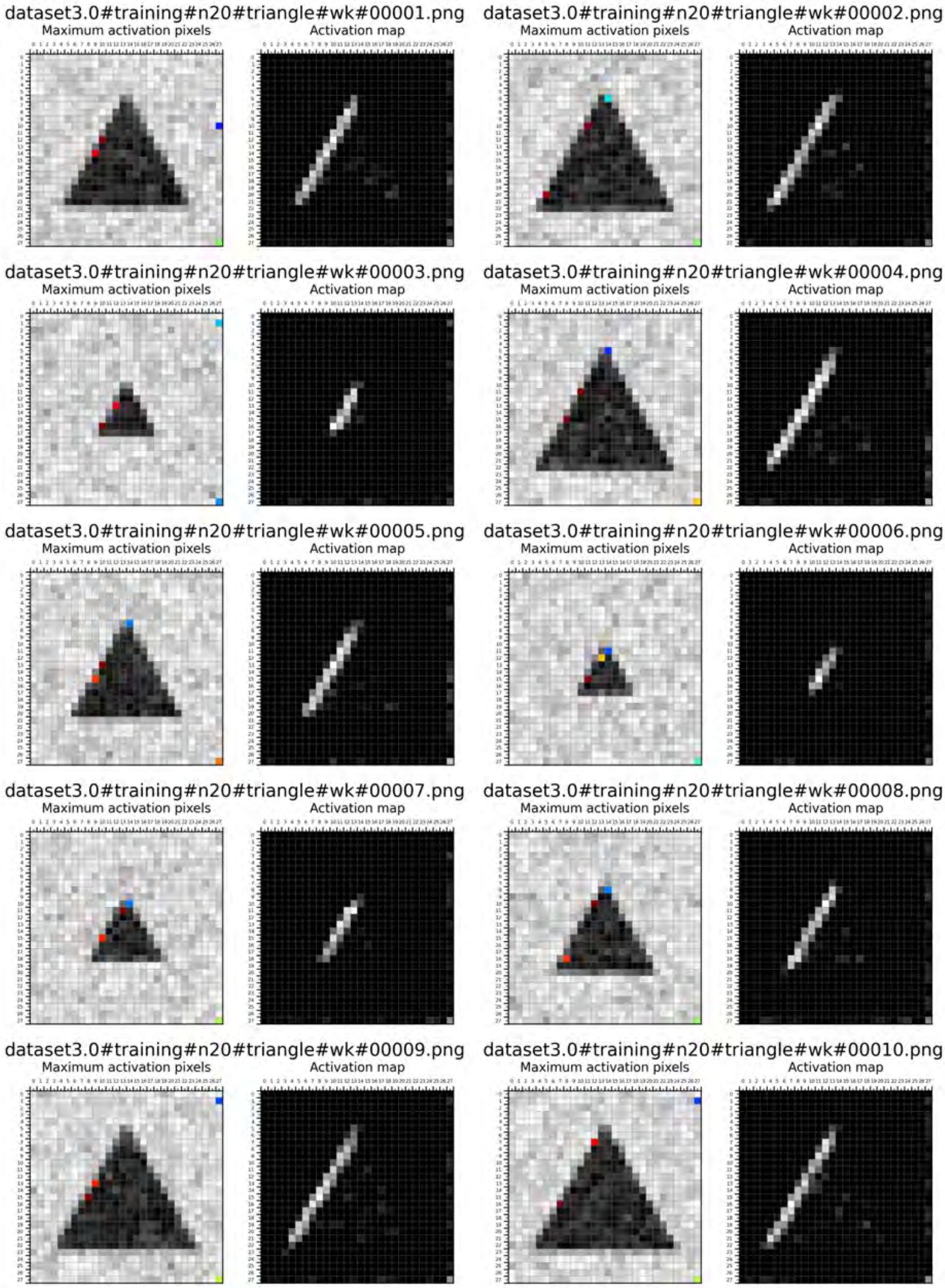

Figure 4.91. Both feature map and maximum activation pixel representations of several triangles samples with the figure in black on a white background from dataset 3.0. n20 in a detailed form. Best viewed in electronic form, zoom in to distinguish the pixel grid.

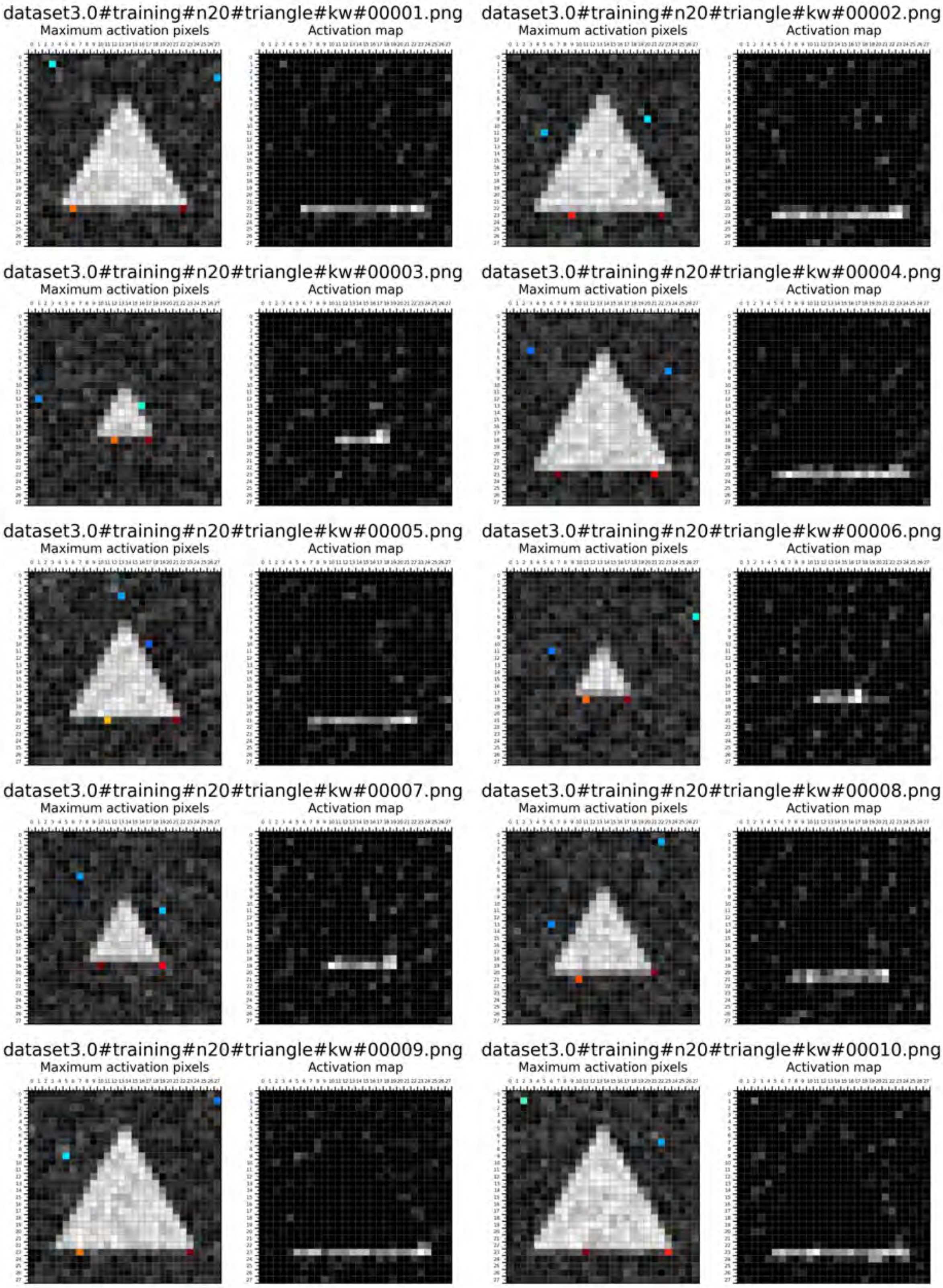

Figure 4.92. Both feature map and maximum activation pixel representations of several triangles samples with the figure in white on a black background from dataset 3.0. n20 in a detailed form. Best viewed in electronic form, zoom in to distinguish the pixel grid.

4.4. Group 4: Rotation and noise problem

Once the rotation problem has been analysed in a simple way, it is time to complicate the task and observe how CNN behave when there is noise added to the input images. Therefore, a new dataset is built taking as base dataset 2.1, adding noise to the images and swapping figure color and background color for some samples. One of the main reasons for building this dataset is the fact that dataset 2.1 was biased so the model did not learnt strong invariant features.

In addition, it is important to remember that with the minimum model obtained for dataset 2.1, the level of accuracy that was achieved was high enough but the model was not behaving as expected, since it learns to classify the ellipses correctly and then The model with one feature was not able to perform the classification correctly for the rotation problem but it managed to achieve 100% of accuracy for the scaling problem. This can be an indicator of the fact that the scaling problem is easier to solve, whereas the CNN finds the rotation problem more difficult. This hypothesis will be checked in the experiment that is performed in this section.

Similar to scaling plus noise problem, only one experiment is going to be performed to analyse the rotation with noise problem since the minimum model is easier to extract due to the fact that the input space has already been explored in a way, so there is a clear starting point. First, the test of the previous minimum architectures where be performed and after that, more parameters will be added to them until the model with the optimal trade-off between complexity and accuracy is found.

4.4.1. Dataset tested

The noise in this case is generated in a similar way that dataset 3.0. It is a salt-and-pepper noise that does not take white or black colour, just different values of gray; so that the geometric shapes are not distorted. Remember that in the previous experiment with dataset 2.1 (4.2.3.2) it was found out that the dataset was biased. With dataset 4.0 this problem is avoided since randomly generated noise is added to the images, what makes all ellipses different from each other. In this way, the model will be forced to understand better the content of the images and hence, learn strong features invariant to noise.

In addition, another bias was detected when applying ML classification algorithms to dataset 2.1: they are able to classify correctly the images attending to the amount of black pixels of each class. Or, what is the same, calculating the area of each geometric shape because its size is not changing. In order to avoid the model learning to count the number of black pixels, for each class half of the images are generated with a white figure on a black background and the other half is composed by images with a black figure on a white background. In this way, the model cannot use colour information to identify a single class.

The dataset is generated following an equal process as the one used for creating dataset 2.1, so the geometric shapes are the same, only with the added noise and the change of colours. As happened in the case of dataset 3.0, several levels of noise have been generated for the same images (n01, n02, n05, n10 and n20), but only the one the highest one is going to be analysed so the most difficult case is presented to the CNN. In Figure 4.52 from the previous section, the difference between two noise levels can be observed. Figure 4.93 shows some examples of images belonging to this dataset, called dataset 4.0 n20 (because of the level of noise selected). It can be better observed that although ellipses are still invariant to the rotation, the images generated in the dataset are not exactly equal since the noise added changes for each sample.

Training dataset 4.0 contains 3000 images, 1000 for each class and within a class 500 with white background and a black figure and 500 with black background and a white figure. For validation, dataset 4.0 contains 6000 images, 2000 for each class and within a class 1000 with white background and a black figure and 1000 with black background and a white figure.

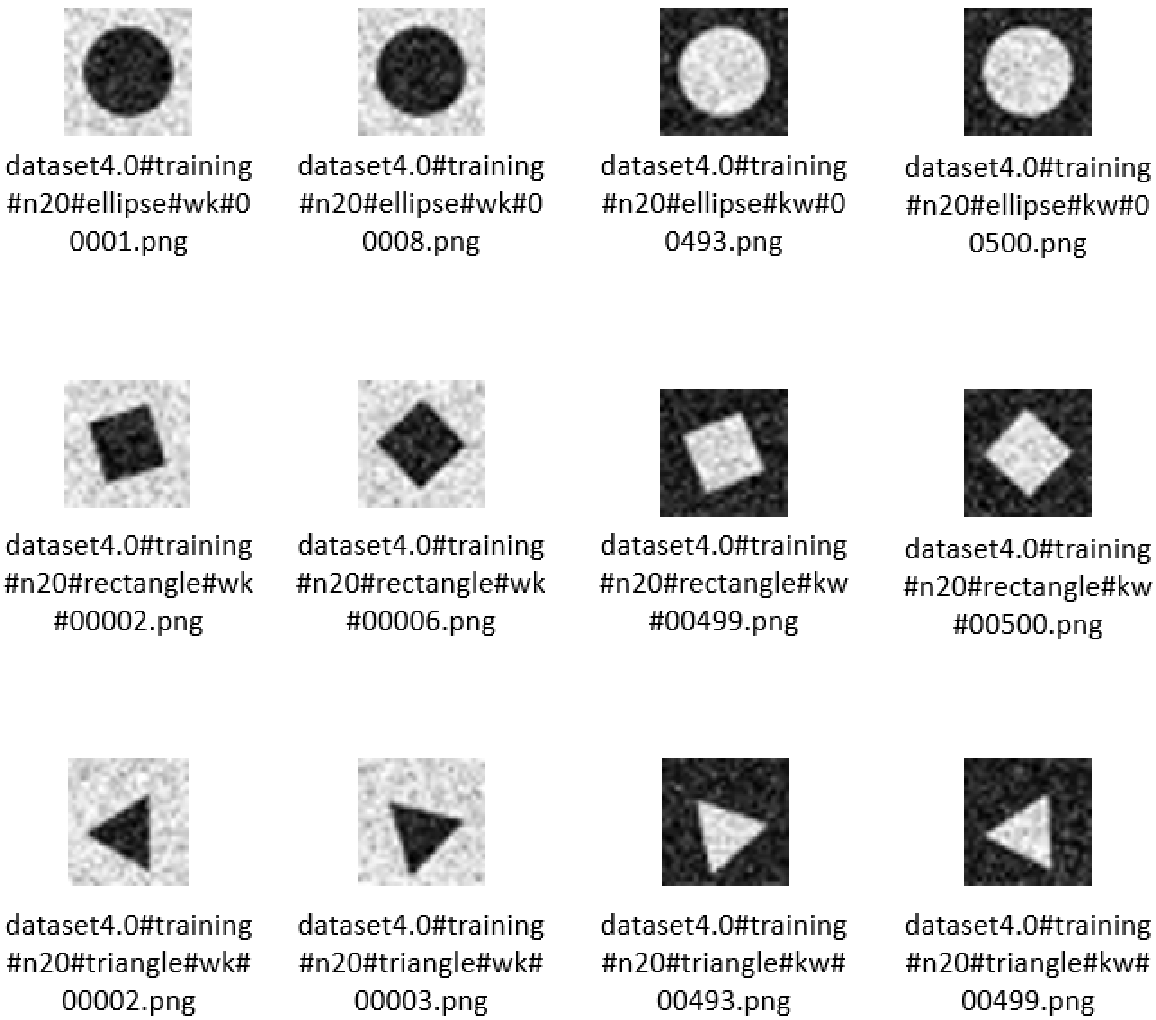

Figure 4.93. Examples of images from dataset 4.0 n20 belonging to the three different classes.

4.4.2. Experiment 4.1

Once the training and validation datasets have been built, it is time to search for the minimum model. The architectures that have been tested in this experiment are the same ones as in Experiment 3.1 (Section 4.3.2), so a hidden layer is added to the classifier too. Just to remember, the architectures are comprised of one convolutional layer with a single filter of size 3x3 together with a ReLU activation function, a pooling layer and a multilayer perceptron with a flattening layer, a hidden layer and a three-node output layer. The pooling size varies its size taking the following values: 28x28, 14x14 and 7x7. For the number of neurons in the hidden layer, the same estimation as in Experiment 3.1 has been done, that is multiplying the number of features of the model by 1, 2 or 3. This parameter is not very crucial as long as the classifier has enough complexity to extract all the information from the features.

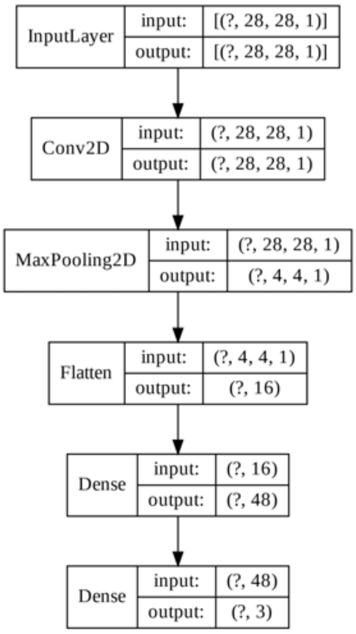

Figure 4.94. Architecture of the minimum model selected for dataset 4.0 n20.

Table 4.16. Model hyperparameters and results obtained after performing Experiment 4.1 over dataset 4.0. Shaded cells show the model selected as the minimum one. *Model timestamp* is the date and hour when the model was created (used as an identifier) with the format: *<year><month><day>T<hour><minute><second>*, *seed* is the seed used for fixing randomness in the model, *nepochs* stands for "number of epochs", *nfilters_1st* stands for "number of filters in the first convolutional layer", *ksize_1st* stands for "kernel size of the first layer", *poolsize_1st* stands for "pool size in the first layer", *hlneurons* for "neurons in the hidden layer", *number_param* stands for "number of parameters", *number_features* stands for "number of features", *tr_acc* stands for "training accuracy" and *val_acc* stands for "validation accuracy". Model names follow the structure (no number of neurons in the hidden layer included to avoid very long names): <model timestamp>_dataset3.0n20_1L_nf1_<nfilters_1st>_ks1_<ksize_1st>_ps1_<poolsize_1st>_ne_<nepochs>_s_<seed>.

Model timestamp	seed	nepochs	nfilters_1st	ksize_1st	poolsize_1st	hlneurons	number_param	number_features	tr_acc	val_acc
20201205T234006	0	200	1	3	28	1	18	1	0.333	0.333
20201206T005242	4	200	1	3	28	1	18	1	0.379	0.384
20201206T012609	10	200	1	3	28	1	18	1	0.333	0.333
20201206T015909	550	200	1	3	28	1	18	1	0.333	0.333
20201206T023208	1234	200	1	3	28	1	18	1	0.333	0.333
20201206T030509	0	200	1	3	28	2	23	1	0.333	0.333
20201206T033735	4	200	1	3	28	2	23	1	0.333	0.333
20201206T040903	10	200	1	3	28	2	23	1	0.333	0.333
20201206T044205	550	200	1	3	28	2	23	1	0.333	0.333
20201206T051636	1234	200	1	3	28	2	23	1	0.333	0.333
20201206T054850	0	200	1	3	28	3	28	1	0.333	0.333
20201206T062452	4	200	1	3	28	3	28	1	0.333	0.333
20201206T065944	10	200	1	3	28	3	28	1	0.333	0.333
20201206T073112	550	200	1	3	28	3	28	1	0.333	0.333
20201206T080236	1234	200	1	3	28	3	28	1	0.333	0.333
20201206T211238	0	200	1	3	14	4	45	4	0.404	0.402

Continued on next page

Table 4.16 – continued from previous page

Model timestamp	seed	nepochs	nfilters_1st	ksize_1st	poolsize_1st	hlneurons	number_param	number_features	tr_acc	val_acc
20201206T225003	4	200	1	3	14	4	45	4	0.437	0.416
20201206T232510	10	200	1	3	14	4	45	4	0.603	0.595
20201206T235820	550	200	1	3	14	4	45	4	0.507	0.502
20201207T003142	1234	200	1	3	14	4	45	4	0.333	0.333
20201207T010446	0	200	1	3	14	8	77	4	0.335	0.335
20201207T013724	4	200	1	3	14	8	77	4	0.567	0.561
20201207T020916	10	200	1	3	14	8	77	4	0.506	0.498
20201207T024047	550	200	1	3	14	8	77	4	0.461	0.456
20201207T031212	1234	200	1	3	14	8	77	4	0.408	0.406
20201207T034329	0	200	1	3	14	12	109	4	0.473	0.471
20201207T041444	4	200	1	3	14	12	109	4	0.662	0.659
20201207T044601	10	200	1	3	14	12	109	4	0.538	0.534
20201207T051703	550	200	1	3	14	12	109	4	0.682	0.677
20201207T054811	1234	200	1	3	14	12	109	4	0.611	0.613
20201207T091350	0	200	1	3	7	16	333	16	0.954	0.946
20201207T102511	4	200	1	3	7	16	333	16	0.994	0.988
20201207T105736	10	200	1	3	7	16	333	16	0.816	0.818
20201207T113005	550	200	1	3	7	16	333	16	0.958	0.956
20201207T201446	1234	200	1	3	7	16	333	16	0.986	0.988
20201207T212829	0	200	1	3	7	32	653	16	0.948	0.949
20201207T220041	4	200	1	3	7	32	653	16	0.999	0.995
20201207T223315	10	200	1	3	7	32	653	16	0.996	0.995
20201207T230557	550	200	1	3	7	32	653	16	0.977	0.973
20201207T233946	1234	200	1	3	7	32	653	16	0.999	0.999
20201208T001340	0	200	1	3	7	48	973	16	0.977	0.975
20201208T004739	4	200	1	3	7	48	973	16	1	0.996
20201208T012025	10	200	1	3	7	48	973	16	0.998	0.998
20201208T015312	550	200	1	3	7	48	973	16	0.990	0.991
20201208T022527	1234	200	1	3	7	48	973	16	1	1

Table 4.16 shows results obtained for each of the architectures tested. Similar to the previous experiment, simpler architectures with just one feature are not able to perform the classification so they reach only a 33% of accuracy for both training and validation datasets. Moreover, in this case, architectures with four features are neither capable of perform the classification correctly and they achieve a 60% of accuracy at most. Therefore, in this case, the minimum architecture has sixteen features. In particular, the architecture that has been selected as the minimum one is 20201207T102511_dataset4.0n20_1L_nf1_1_ks1_3_ps1_7_ne_200_s_4 (Figure 4.94). It has one hundred thirty-three parameters, a pooling layer of size 7x7 and a hidden layer with sixteen neurons. It manages to reach high levels of accuracy: 99.4% in training and 98.8% in validation. In addition, this architecture has a high level of robustness since all the models trained with different seeds achieve levels of accuracy of minimum 80%.

On the other hand, there are other models among the ones with sixteen features that achieve higher levels of accuracy, reaching even 100%, but they have the double or even the triple amount of parameters, what makes them too complex to be the minimum model. The selected model has the best balance between accuracy and complexity.

The minimum model for the rotation plus noise problem needs sixteen features to perform the classification correctly, which means that this problem is more difficult that the one analysed in Experiment 2.3 (Section 4.2.3) where the images do not have noise; and the most complex from the ones presented in this dissertation. Following the same structure as for the rest of experiments, first of all, these features are going to be studied deeply in order to understand better how the CNN is able to perform the classification with the information that they provide. An exploratory data analysis is performed for the dataset of features that is inputted to the classification part of the CNN. It is important to take into account during most part of the analysis, only training representations are shown. The reason is that the minimum model achieves almost equal accuracy values for both training and validation dataset, hence the representations are very similar and in this way information is not repeated.

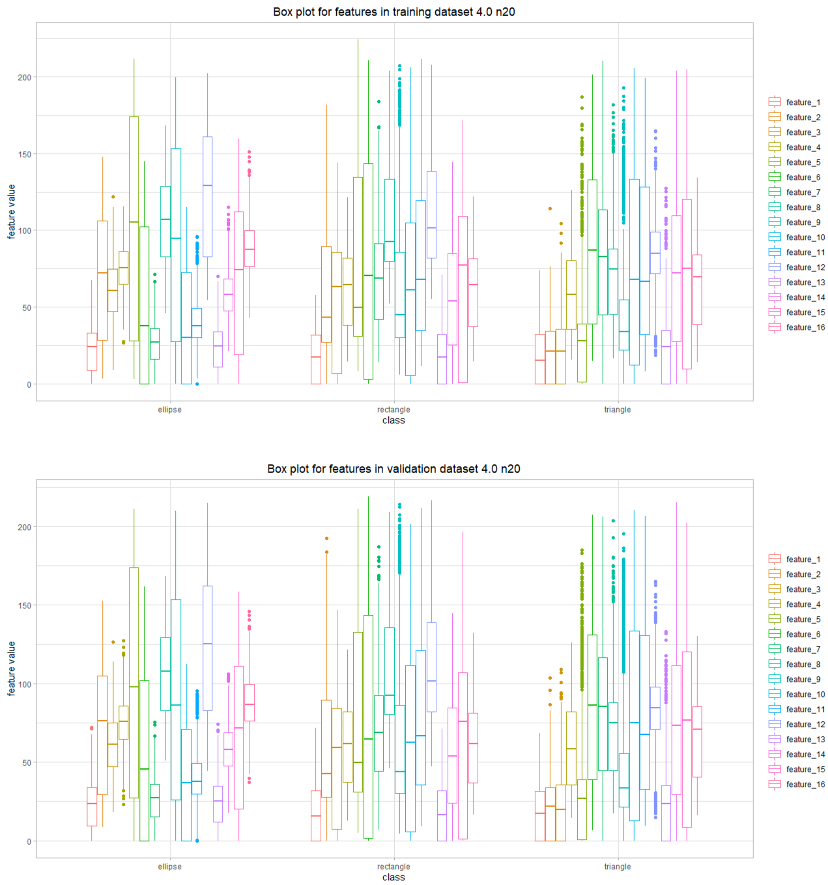

Figure 4.95. Boxplots showing values of the features extracted depending on the class by the minimum model for dataset 4.0, 20201207T102511_dataset4.0 n20_1L_nf1_1_ks1_3_ps1_7_ne_200_s_4, in both training and validation.

Figure 4.95 shows boxplots with the values for each one of the sixteen features extracted by the dataset depending on the class. Since there are many features, it is very difficult to identify at a glance which features take specific luminance values for each class, so that the network is able to perform the classification task correctly. Nevertheless, the neywork is not using just a few features to perform the classification but a combination of all of them since they all contain important information about the image. Figure 4.96 shows the linear correlation existing among the features, and it can be observed that some pair of features have a high direct correlation such as features 1 and 15; and features 10 and 6. On the other hand, features such as 6 and 15 or 10 and 15 have inverse correlation.

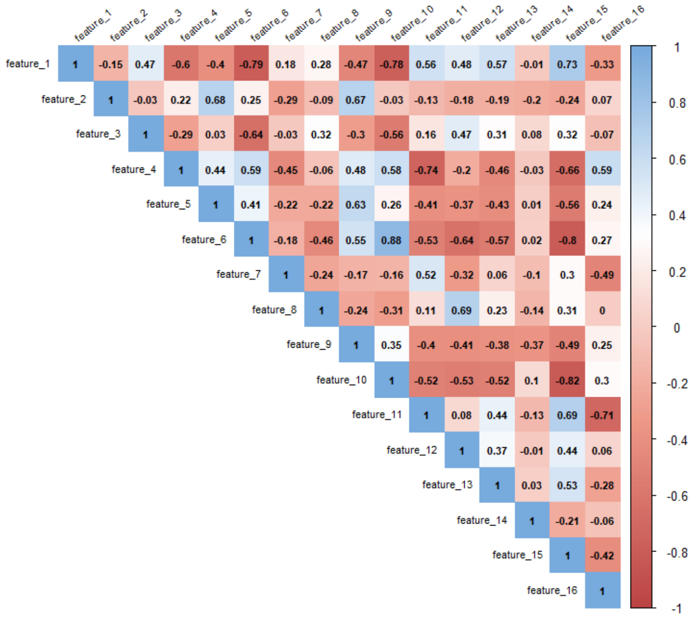

Figure 4.96. Linear correlation plot between features extracted by the minimum model for dataset 4.0 n20.

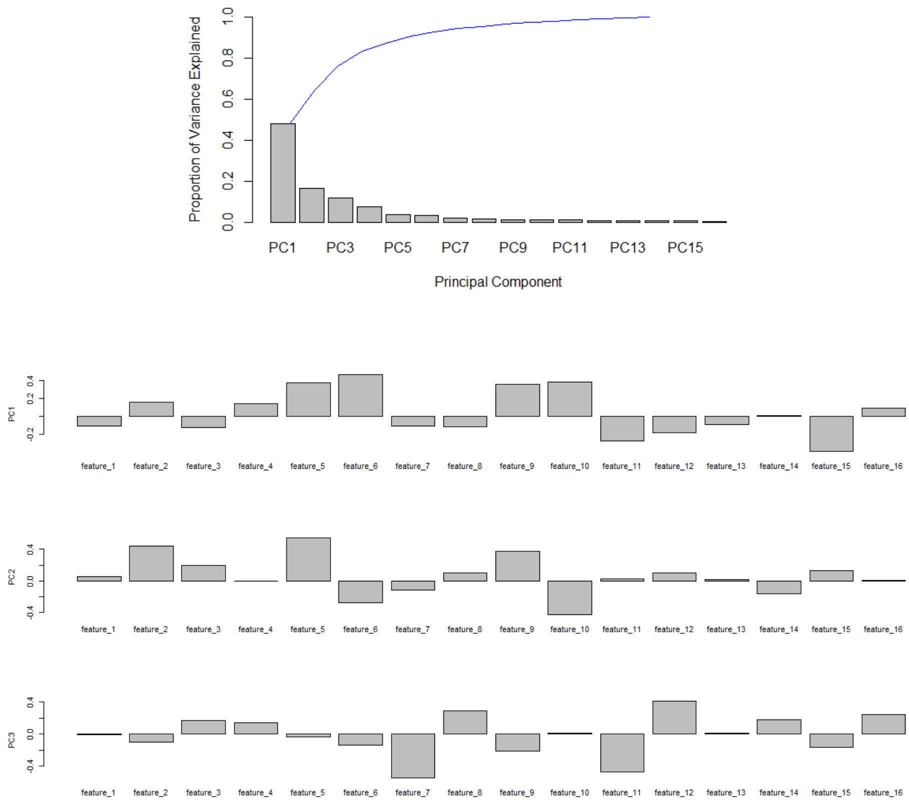

Figure 4.97. Proportion of variance explained by each principal component (top) and composition of each principal component (botom) for features extracted by dataset 4.0 minimum model.

The PCA analysis is also carried in this experiment in order to study the feature space given by CNN. Figure 4.97 shows the proportion of variance explained by each one of the principal components and it can be appreciated that with the first three principal components, an 80% of the dataset variance is explained. These components will be used to project the data. The figure also shows a plot with the coefficients assign to each feature in the principal components but there is not any remarkable feature.

Projections of the training dataset over the three main principal components are shown in Figure 4.98. As it happened in Experiment 3.1 (Section 4.3.2), the model has been able to extract features that create a space in which the images can be easily separated since the samples belonging to the same class are grouped together. To appreciate this in detail, the three projections have to be looked at, because each of them manages to separate certain groups of samples. For example in the first projection, samples placed in the right part of the image are crearly separated by class: ellipses on top, rectangles in the middle part and triangles at the bottom of the graph; whereas in the third projection, the images that are better separated are the ones on the left part of the graph.

Figure 4.99 shows the same projections but with the additional information of the combination of background and figure colour for each sample. Similar to what happens in experiment 3.1, images with a white figure on a black background are placed on the right of the feature space whereas samples with a black figure on a white background are placed on the left part of the image. Therefore, the minimum model in this case also manages to perform the separation between combinations of background and figure colour.

Figure 4.98. Projections of feature dataset generated for each sample from dataset 4.0 n20 with its three classes on the plane formed by the first three principal components: PCA 1 vs. PCA 2 (first plot), PCA 2 vs. PCA 3 (second plot) and PCA 1 vs. PCA 3 (third plot). The component that explains the largest amount of information is placed on the horizontal axis.

Figure 4.99. Projections of feature dataset generated for each sample from dataset 4.0 n20 with six classes (taking into account color of the figure and the background) on the planes formed by the first three principal components: PCA 1 vs. PCA 2 (first plot), PCA 2 vs. PCA 3 (second plot) and PCA 1 vs. PCA 3 (third plot). The component that explains the largest amount of information is placed on the horizontal axis.

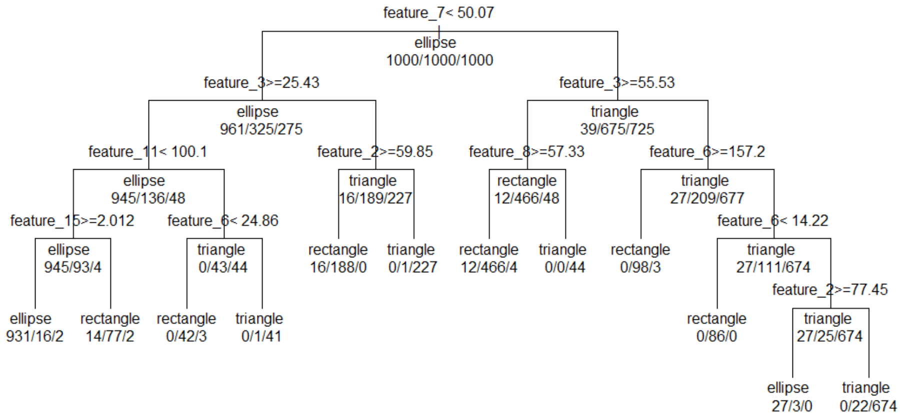

Figure 4.100. Decision tree using feature extracted by hte minimum model for dataset 4.0 n20: 20201207T102511_dataset4.0n20_1L_nf1_1_ks1_3_ps1_7_ne_200_s_4.

As a conclusion, the features extracted by the model give as a result a dataset which contains a clear separation of the classes. That dataset is inputted to the classification part of the CNN so it is able to perform the task correctly. In the dataset appears also a clear differentiation between images *kw* and *wk*. Once the feature space has been analysed, and it has been observed that is feasible to separate, it is interesting to input it to some classification techniques. In this way, the information given by the features is trying to be understood. Each classification model will try to extract that information using its particular methodology but the input information is the same for all the models.

The first classification algorithm that is going to be trained is a decision tree, as it has been done in previous experiments. Figure 4.100 shows the plot of one of the trees that have been trained. This tree achieves a training accuracy of 0.967 within the confidence interval of (0.96, 0.9731) which means that with the feature space generated by the CNN, it is easy to perform the classification. Once again there are many different solutions to this problem and this tree is just one of them and a tree model behaves different from a multilayer perceptron but the logic behind each algorithm can be similar.

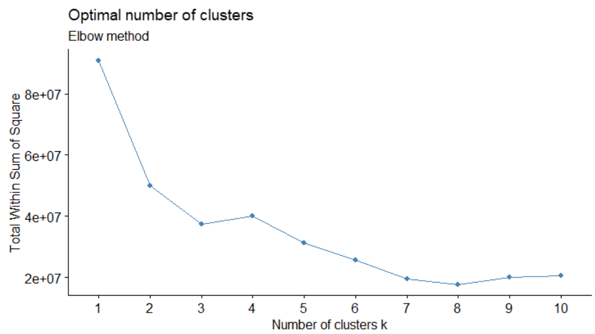

Figure 4.101. Elbow plot for K-means model trained with feature dataset generated by the minimum model for dataset 4.0 n20.

The next algorithm that is used to continue the features analysis is K-means, which is not a classification technique but a clustering one. From the plot shown in Figure 4.101 it is deduced that the same number of clusters as in Experiment 3.1 are extracted by the model. Once again, the samples are clearly separated in different regions of the space according to the combination of three factors: class, background colour and figure colour. This separation is better appreciated in the scatter plot from Figure 4.102 and also, in the existing differences among cluster centroids observed in the same figure.

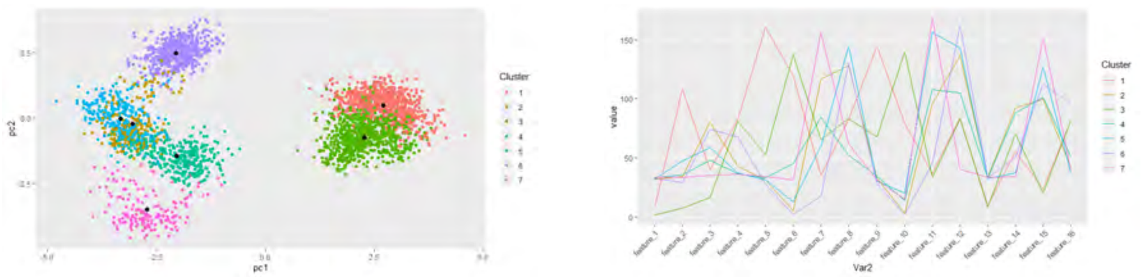

Figure 4.102. Projection of features generated by the minimum model selected for dataset 4.0 n20 with the clusters built by K-means model (left) and centroids for each cluster (right).

	ellipse	rectangle	triangle
1	500	304	1
2	0	214	0
3	0	196	499
4	0	61	279
5	0	222	51
6	500	3	0
7	0	0	170

Figure 4.103. Number of samples from the three classes assigned to each cluster by the K-means model trained with the feature dataset generated by the minimum model selected for dataset 4.0 n20.

	PREDICTED		
KNOWN	**ellipse**	**rectangle**	**triangle**
ellipse	998	2	0
rectangle	10	985	5
triangle	0	2	998

Training confusion matrix

	PREDICTED		
KNOWN	**ellipse**	**rectangle**	**triangle**
ellipse	1988	12	0
rectangle	29	1956	15
triangle	1	14	1985

Validation confusion matrix

Figure 4.104. Confusion matrix both for training and validation images for the minimum model selected for dataset 4.0.

In this case, ellipses there is only a cluster that identifies ellipses correctly and that is the number 6; for rectangles, clusters 2 and 5 and finally, for triangles, clusters 3, 4 and 7. Ellipses and rectangles are once again mixed up since they are very similar. It also draws the attention that there are three clusters for triangles, although one of them is a mixture of both rectangles and triangles. This fact also makes sense since with the rotation, a triangle can be confused with a rectangle since it also has marked angles and straight sides. Figure 4.104 shows the confusion matrices generated by the CNN minimum model for both training and validation datasets and they confirm that the two classes that the model finds more difficult to separate are ellipses and rectangles, since it is where more mistakes is making. It also can be observed that this model has less error than the one selected for the scaling plus noise problem, although this does not mean that the rotation plus noise problem is easier. In fact, the rotation problem is harder to solve since the minimum model is more complex than the one selected for Experiment 3.1, as it has been previously noted.

After performing this specific feature analysis it can be concluded that the features extracted by the CNN may seem simply a series of luminance values but, in fact, they are key to generate a new space where the images can be easily classified. The space is so well built that any ML algorithm, no matter how simple, is able to perform the classification correctly, as it has been observed in this study. As discussed earlier in Experiment 3.1, the most powerful part of the CNN is the feature extraction part which gives as result a “bag of features” that, similar to what happens in the BoW algorithm, can be inputted to any classifier (it does not have to be a multilayer perceptron) and it will be able to perform the classification.

For the analysis of the physical meaning of the model, once again, instead of plotting heat maps that give information about all of the image, only the specific pixels extracted by the model are highlight over the original images. Nevertheless, as this model extract sixteen different features, instead of plotting them with the same colour, a different colour is assigned to each pixel depending on the value of the feature, in the same way as the analysis carried out in Experiment 3.1.

First of all, from looking at the filter representation shown in Figure 4.105 it can be deduced that the filter in this case is also extracting borders in the image. Nevertheless, the filters seems to be identifying borders located on the right or left part of the image, since the highest positive values of the filter are placed in the positions (0,0) and (0,3)). The filter is able to recognise corners too, as it has a positive value in (1,0). This is better appreciated in the feature maps.

0.615	0.088	-0.229
-0.181	-0.380	-0.262
0.620	-0.478	0.004

20201207T102511_dataset4.0n20_1L_nf1_1_ks1_3_ps1_7_ne_200_s_4

Figure 4.105. Filter learnt by the minimum model selected for dataset 4.0 n20. Luminance representation on the left and the exact luminance values on the left.

In this way, for ellipses, active maps in Figures 4.114 and 4.116 show that the filter is detecting, in fact, the left border of the figure for the images with a black figure on a white background and the right border for the ones with a white figure on a black background. In addition, it can be observed that the filter is not detecting strictly straight borders but it is able to obtain also certain slopes. Looking at maximum activation pixel representations shown in Figures 4.115 and 4.117 that pixels which almost always are activated are the ones located in the centre of the border of the ellipse, either left or right. In particular, observing Figure

4.126, the pixels are located in positions (4,14) or (4,13) and for Figure 4.127 pixels are located in (24,13) or (22,17). Since for ellipses, the figures does not change its shape neither they are afected by the rotation, the locations of pixels with highest activations remain almost constant.

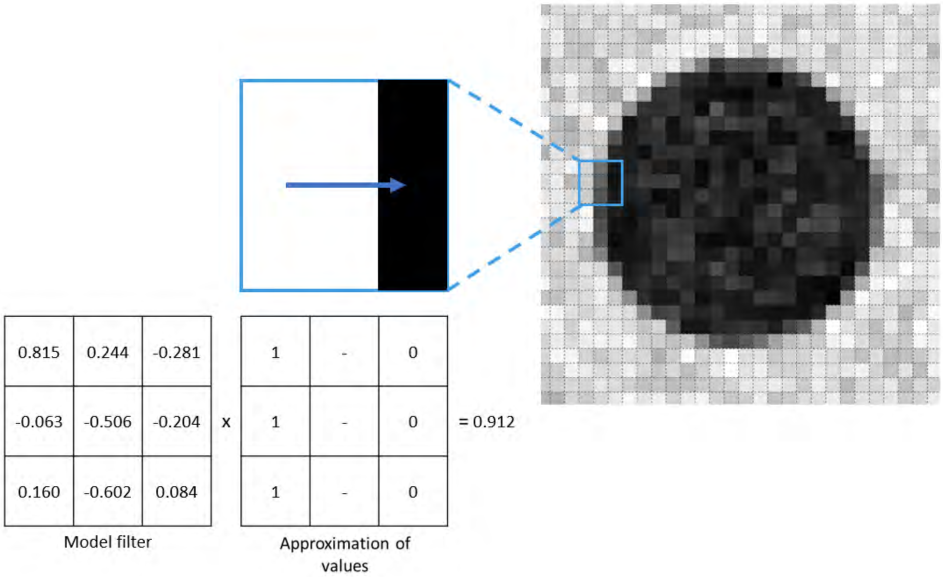

Figure 4.106. Example of dataset 4.0 minimum model filter applied to an ellipse image with the figure in black on a white background. The result of the convolution is an approximation, not the real one. The arrow shows the specific direction of colour change that activates the filter and the hyphen represents a value of gray.

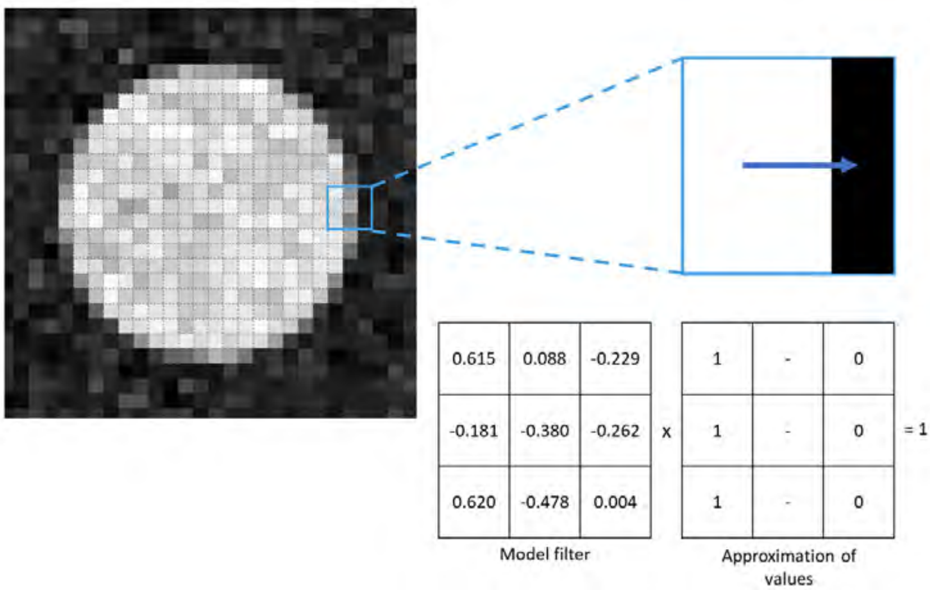

Figure 4.107. Example of dataset 4.0 minimum model filter applied to an ellipse image with the figure in white on a black background. The result of the convolution is an approximation, not the real one. The arrow shows the specific direction of colour change that activates the filter and the hyphen represents a value of gray.

Related to rectangles, activation maps are shown in Figure 4.118 for images with a white background and in Figure 4.120. In the first case, it can be appreciated that borders placed on the left part of the image are the ones that are being detected by the filter, whereas in the second case, are the ones placed on the right area of the image. The exactly filter functioning and activation it is equal for the three classes so it can be better appreciated in the ellipse examples showed in Figures 4.106 and 4.107. For rectangles, the filter activates in the same way, detecting a clear transition from white to black colour in directions from left to right, no matter whether the side of the rectangle is parallel to the border of the image or if it is diagonal. In particular, for rectangles, two borders are usually activated. Looking at Figure 4.128 it can be observed that the pixels with the strongest activations are placed in positions like (5,13), (8,9) and (9,20), on the left part of the image. Figure **??** shows that the strongest activations are located in positions like (22,11), (20,14) and (17,21).

Feature maps for triangles are shown in Figures 4.122 for images with white background and 4.124 for images with black background. It can be observed again that, for the first set of images, the borders that are being detected are the ones placed on the left part of the image, as it is in this part where there is a change from white to black in the direction left to right. For the second set of images, the borders are detected on the right part of the triangles, since in that part is the change from white to black.

Looking closely at the exact pixels that are selected by the pooling layer during training in Figure 4.119, for black figures on a white background the strongest activations are observed on the border that is placed on the left of the image and sometimes even on a vertex. In particular, in Figure 4.130 it can be observed that pixels with the highest activations are placed in locations such as (7,12), (9,17) or (11,22). All of them with its x value lower than fourteen, which represents the left half of the image.

For white triangles on a black background shown in Figure 4.121, the side that contains the strongest activations is the one located on the right part of the image. In Figure 4.131, pixels with the highest activations are in the positions (21,13), (20,15) or (20,20), that is, on the right area of the image. The rest of pixels in both sets of images have lower activations, so, in general, there are less strong activations for triangles than for rectangles, what can be a way to distinguish between them.

The functioning of the filter for triangles is similar as for ellipses and rectangles, so it can be observed better in Figures 4.106 and 4.107. In this case, activations are similar to the ones obtained for rectangles: depending on the degree of rotation, the filter is able to recognise one or two sides of the triangle, if there is a strong change from white to black in the direction indicated by the arrow shown in the representations (in horizontal direction from left to right).

It is important to highlight the fact that for all black figures with a white background, feature maps have several activations on the right border of the image. This effect was also previously observed in Experiment 3.1 where images with white background have the pixel in position (28,28) always activated. The reason behind it is the same one, the padding added to the image when performing the convolution is black and since the filter is detecting changes from white on the left to black on the right, the filter activates in this border of the image. Once again, there is recognition of the need for a proper padding method when performing operations through the CNN. Taking this into account, clamping seems to be one of the best options since it replicates edge pixels indefinitely. In this case it also would manage to replicate the noise without biasing the images.

Finally, as it happened in Experiment 3.1, after having observed the positions of each one of the sixteen features and the values they take in general for all images, the values obtained in the correlation plot from Figure 4.96 make sense. For example, features 5, 6, 9 and 10 have negative correlation with features 7, 8, 11 and 12, since they are placed in opposed areas of the images, the left half and the right half. When the filter activates for one of the halfs, is not activated for the other one and vice-versa, which means that when the value of one group of features is high, for the others is low.

In order to conclude this analysis of the physical meaning of the model is important to highlight that with just one single filter the CNN is able to differentiate among the three classes, no matter the combinations of figure and background colour. The combination of the values of features extracted for each class is what makes the model able to perform the classification correctly.

An additional analysis that has been performed in this experiment is the study of mistakes made by the minimum CNN model. In Figure 4.113 it can be confirmed that the model finds more difficult to classify the samples that are located at the limits between classes. Nevertheless, there are some samples that seem to be near the centre of the cluster but they are wrongly classified, some of these samples will be analysed next.

There are not many images that the CNN classifies wrongly, so a few of the most representative are chosen. These are samples which location is “inside” a cloud of samples that belongs to another different class. Other samples stand out because the model has failed classifying them, even though they belong to the same class as those in their region. Just as Experiment 3.1, to carried out this study two kinds of plots are built: one showing the maximum activation pixels representation and the feature map and the other a bar diagram that enables to compare between representative feature values of the class with the mistaken image.

A brief analysis of each one of the specific samples selected is carried out next:

- Image *dataset4.0#training#n20#ellipse#wk#00476*: the image that is wrongly classified has features 3, 5 and 14 with lower values than the average and features 6, 7, 13 and 15 with values higher than the average (Figure 4.108).

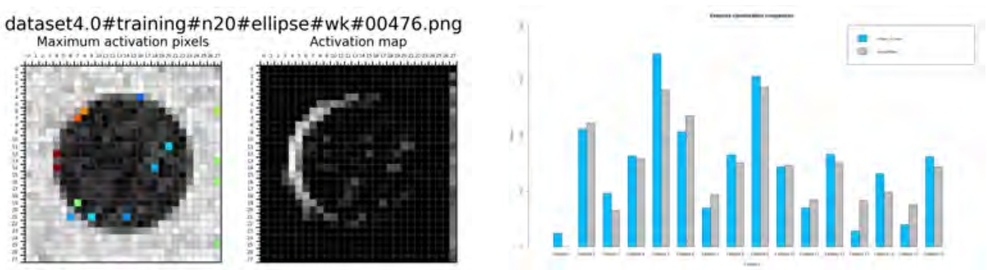

Figure 4.108. For an specific image: detailed maximum activation pixels representation together with feature map plot (left) and bar plot for the comparison of features (right).

- Image *dataset4.0#training#n20#rectangle#wk#00163*: the image that is wrongly classified has features 6, 7, 10 and 15 with lower values than the average and features 2, 3, 4, 5, 8, 12 and 14 with values higher than the average (Figure 4.109).

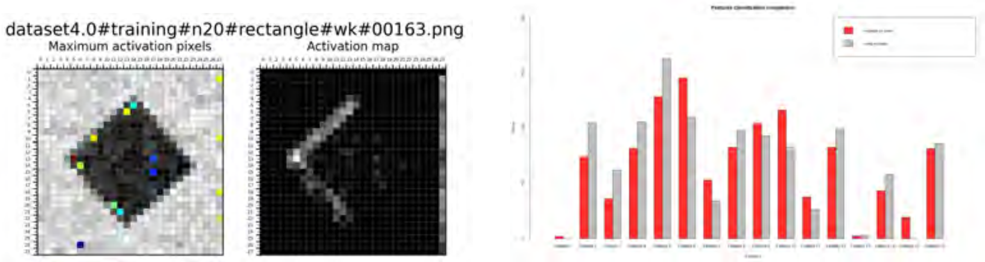

Figure 4.109. For an specific image: detailed maximum activation pixels representation together with feature map plot (left) and bar plot for the comparison of features (right).

- Image *dataset4.0#training#n20#rectangle#wk#00304*: the image that is wrongly classified has features 7, 8 and 10 with lower values than the average and features 2, 3, and 13 with values higher than the average (Figure 4.110).

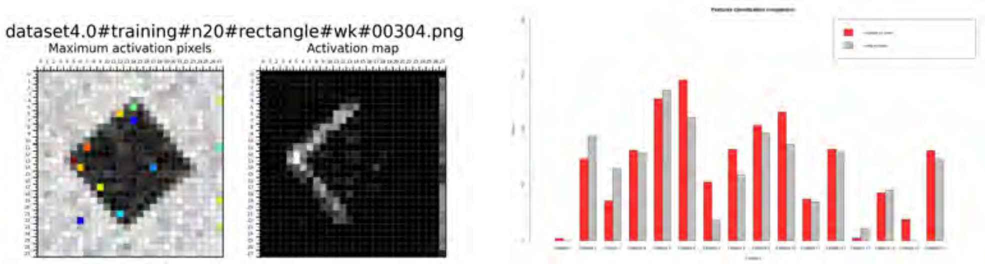

Figure 4.110. For an specific image: detailed maximum activation pixels representation together with feature map plot (left) and bar plot for the comparison of features (right).

- Image *dataset4.0#training#n20#triangle#kw#00101*: the image that is wrongly classified has features 6, 7, 9, 10 and 14 with lower values than the average and features 8, 12, 13 and 15 with values higher than the average (Figure 4.111).

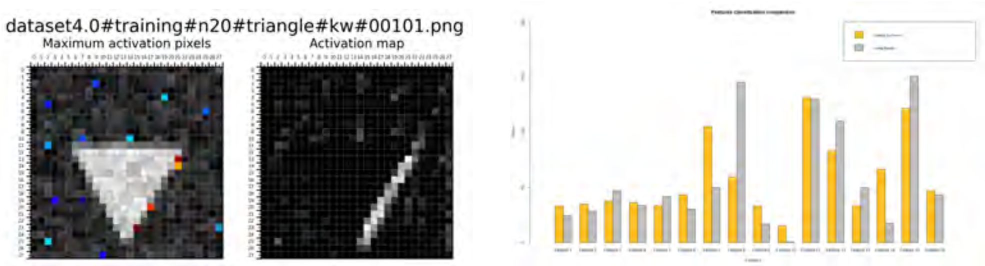

Figure 4.111. For an specific image: detailed maximum activation pixels representation together with feature map plot (left) and bar plot for the comparison of features (right).

- Image *dataset4.0#training#n20#triangle#wk#00314*: the image that is wrongly classified has features 5, 14 and 15 with lower values than the average and features 3, 6 and 9 with values higher than the average (Figure 4.112).

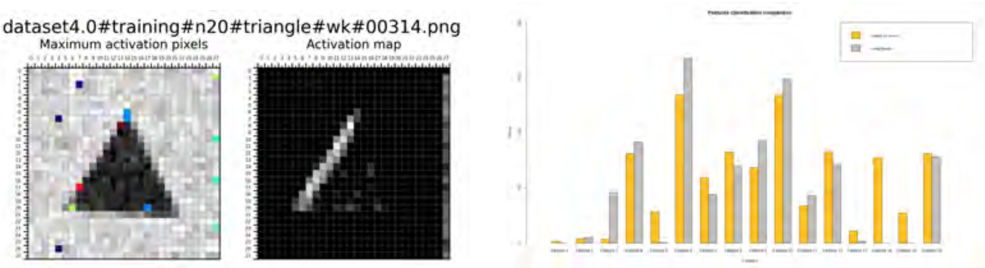

Figure 4.112. For an specific image: detailed maximum activation pixels representation together with feature map plot (left) and bar plot for the comparison of features (right).

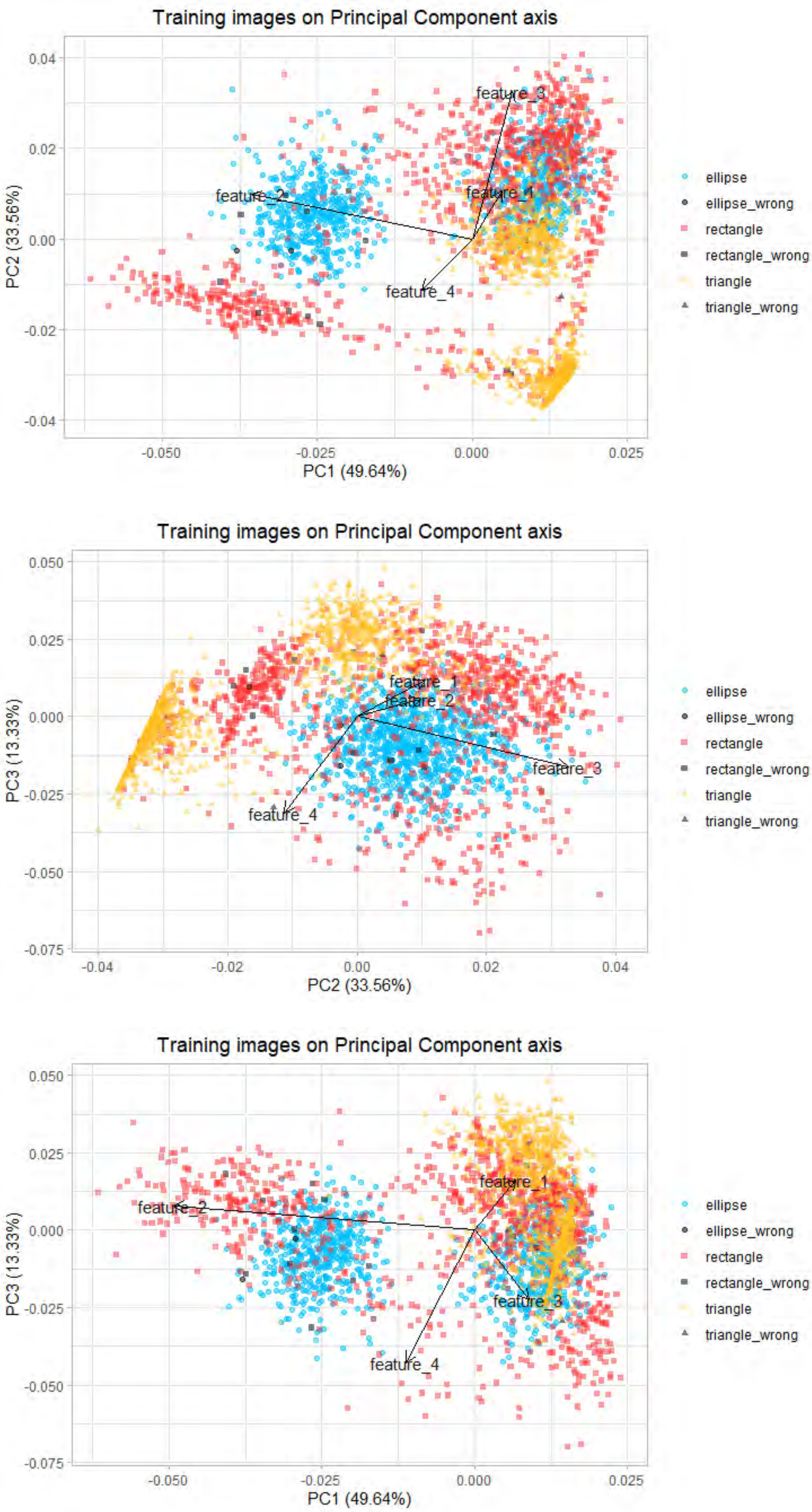

Figure 4.113. Projections of feature dataset generated for each sample from dataset 4.0 n20 with its three classes on the planes formed by the first three principal components: PCA 1 vs. PCA 2 (first plot), PCA 2 vs. PCA 3 (second plot) and PCA 1 vs. PCA 3 (third plot). Samples incorrectly classified by the minimum CNN model appear in black colour. The component that explains the largest amount of information is placed on the horizontal axis.

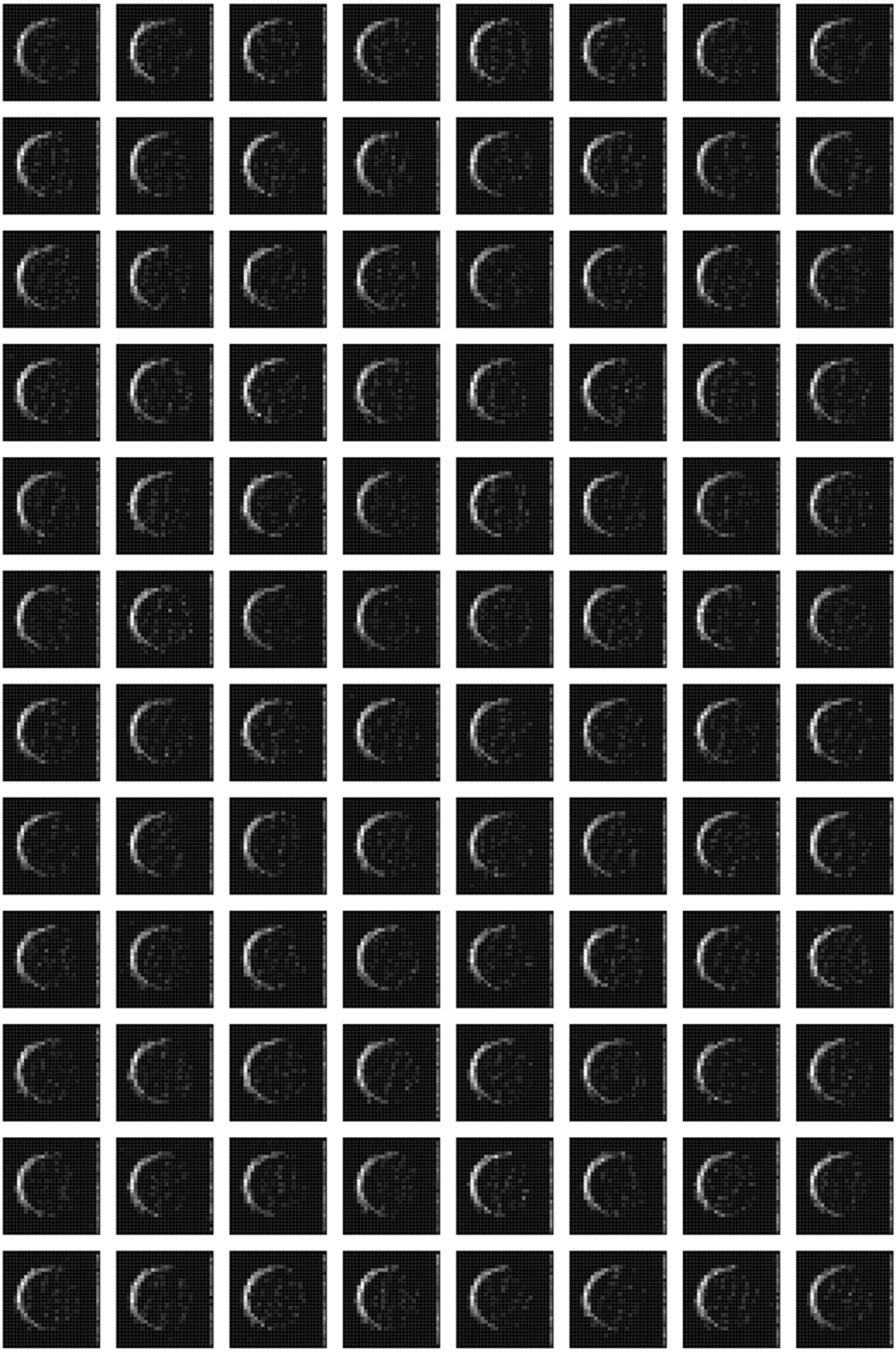

Figure 4.114. Feature maps of several ellipse samples with the figure in black on a white background from dataset 4.0 n20. Best viewed in electronic form, zoom in to distinguish the pixel grid.

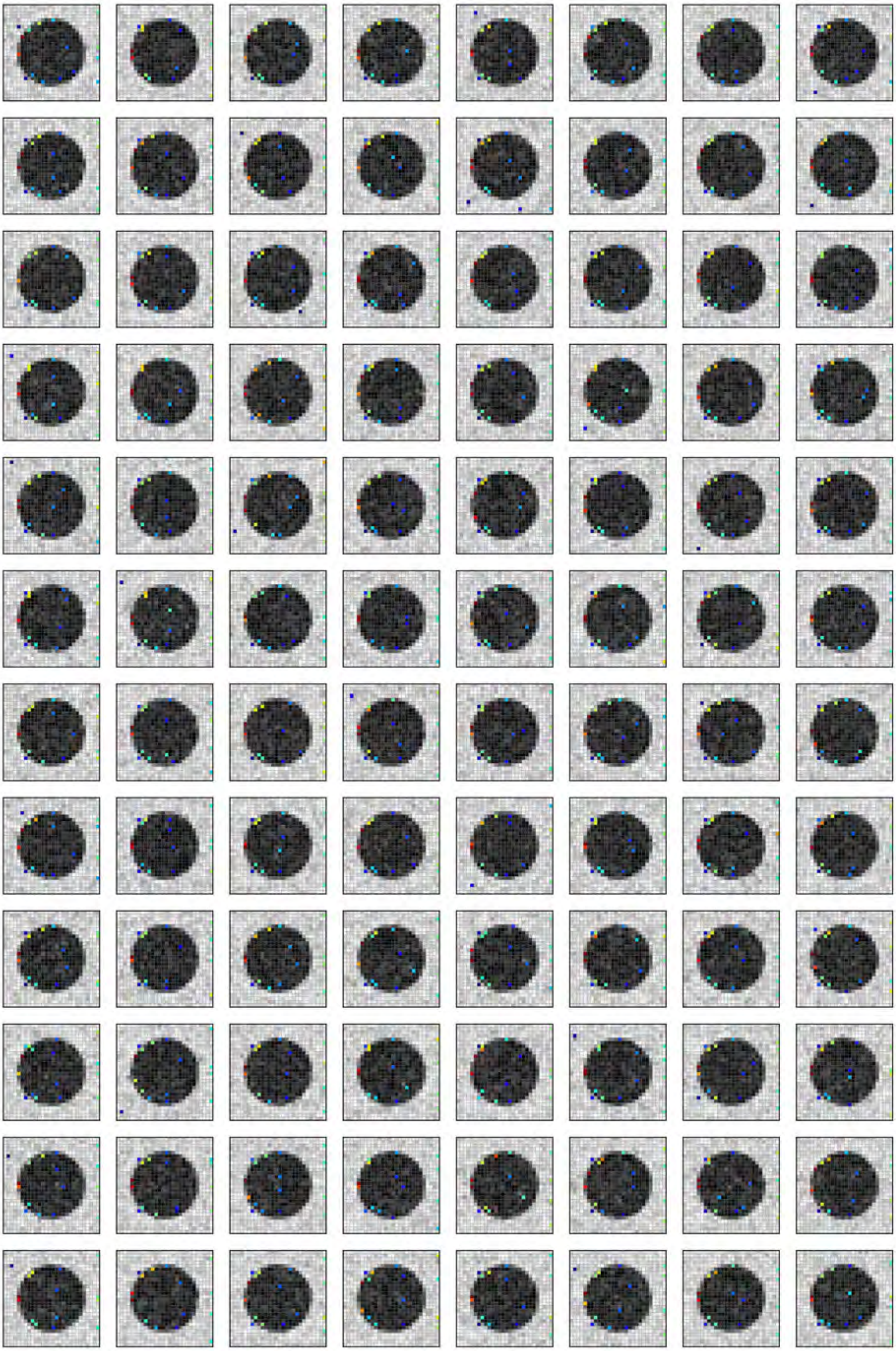

Figure 4.115. Original images of several ellipses from dataset 4.0 n20 with the figure in black on a white background and with maximum activation pixels in different colours according to each value (red representing the highest value and blue the lowest one). Best viewed in electronic form, zoom in to distinguish the pixel grid.

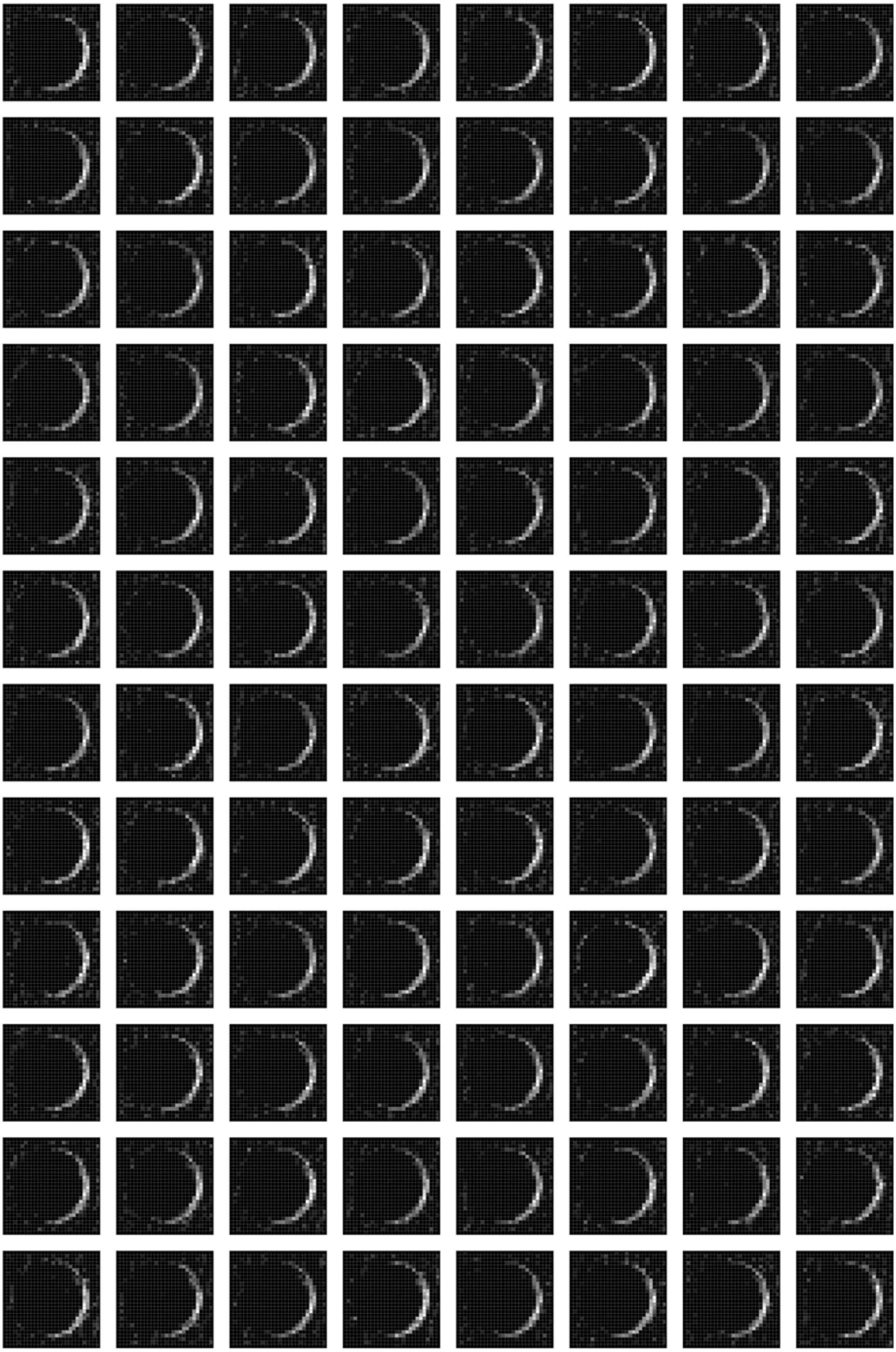

Figure 4.116. Feature maps of several ellipses samples with the figure in white on a black background from dataset 4.0 n20. Best viewed in electronic form, zoom in to distinguish the pixel grid.

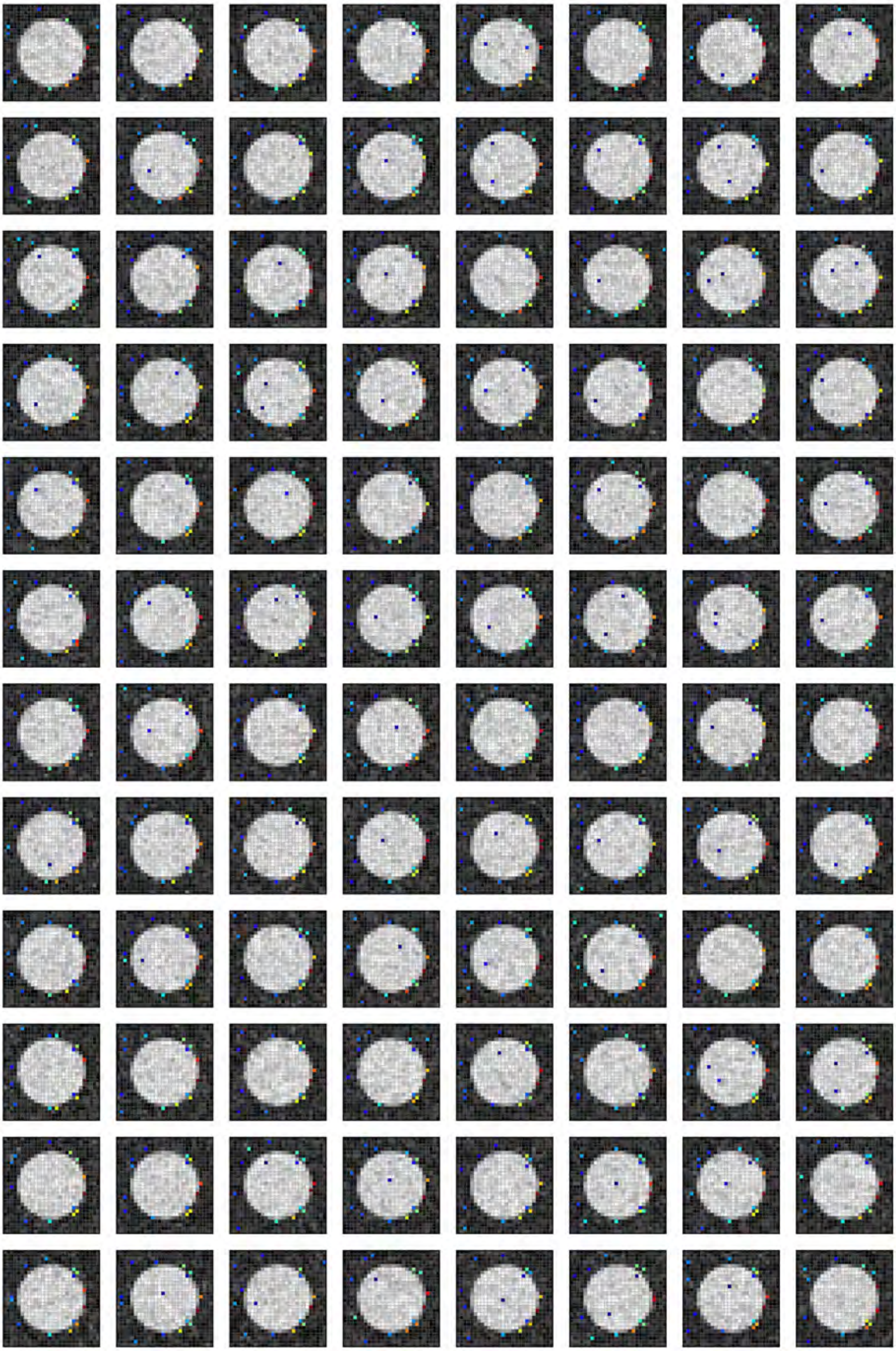

Figure 4.117. Original images of several ellipses from dataset 4.0 n20 with the figure in white on a black background and with maximum activation pixels in different colours according to each value (red representing the highest value and blue the lowest one). Best viewed in electronic form, zoom in to distinguish the pixel grid.

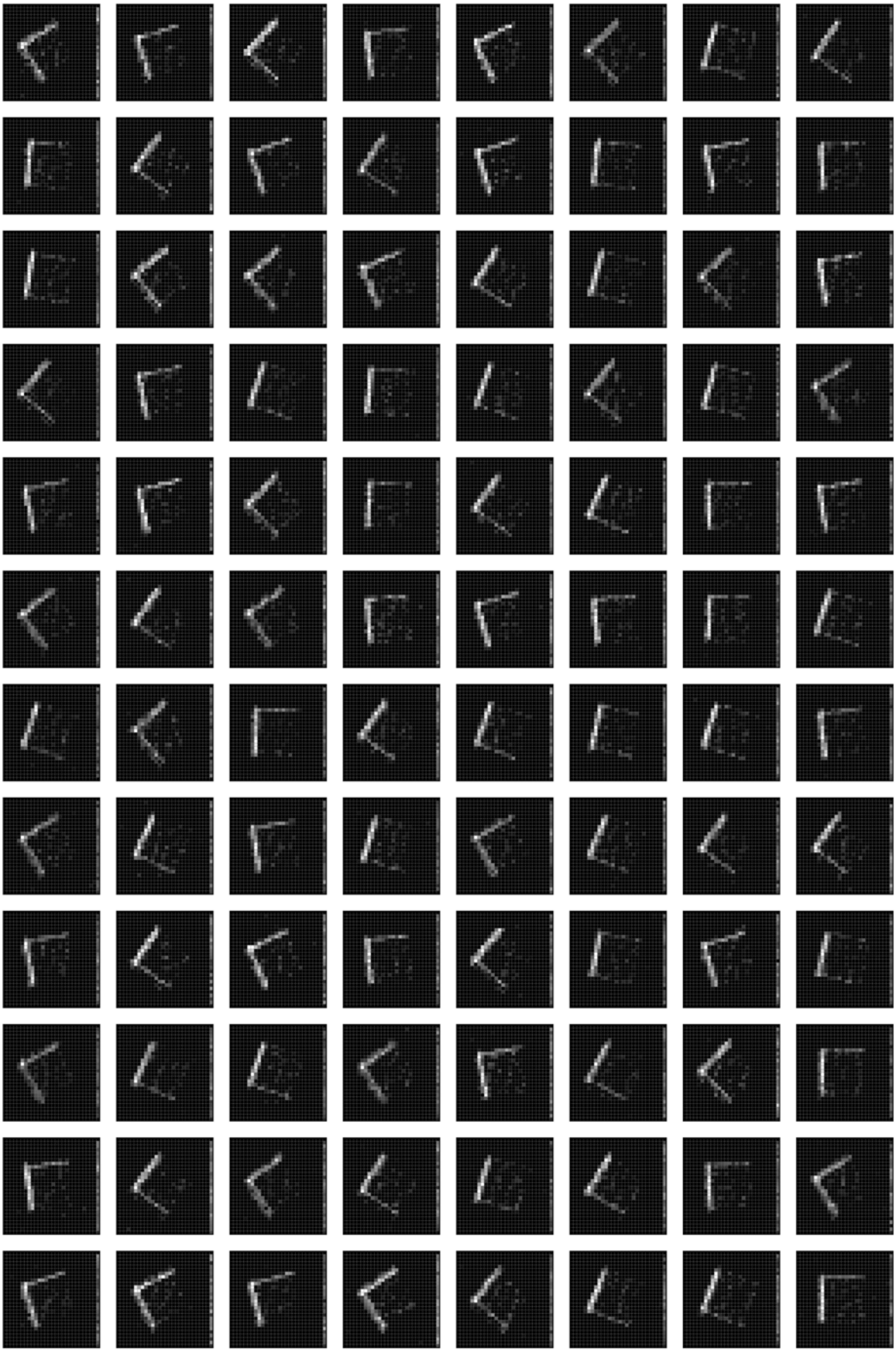

Figure 4.118. Feature maps of several rectangle samples with the figure in black on a white background from dataset 4.0 n20. Best viewed in electronic form, zoom in to distinguish the pixel grid.

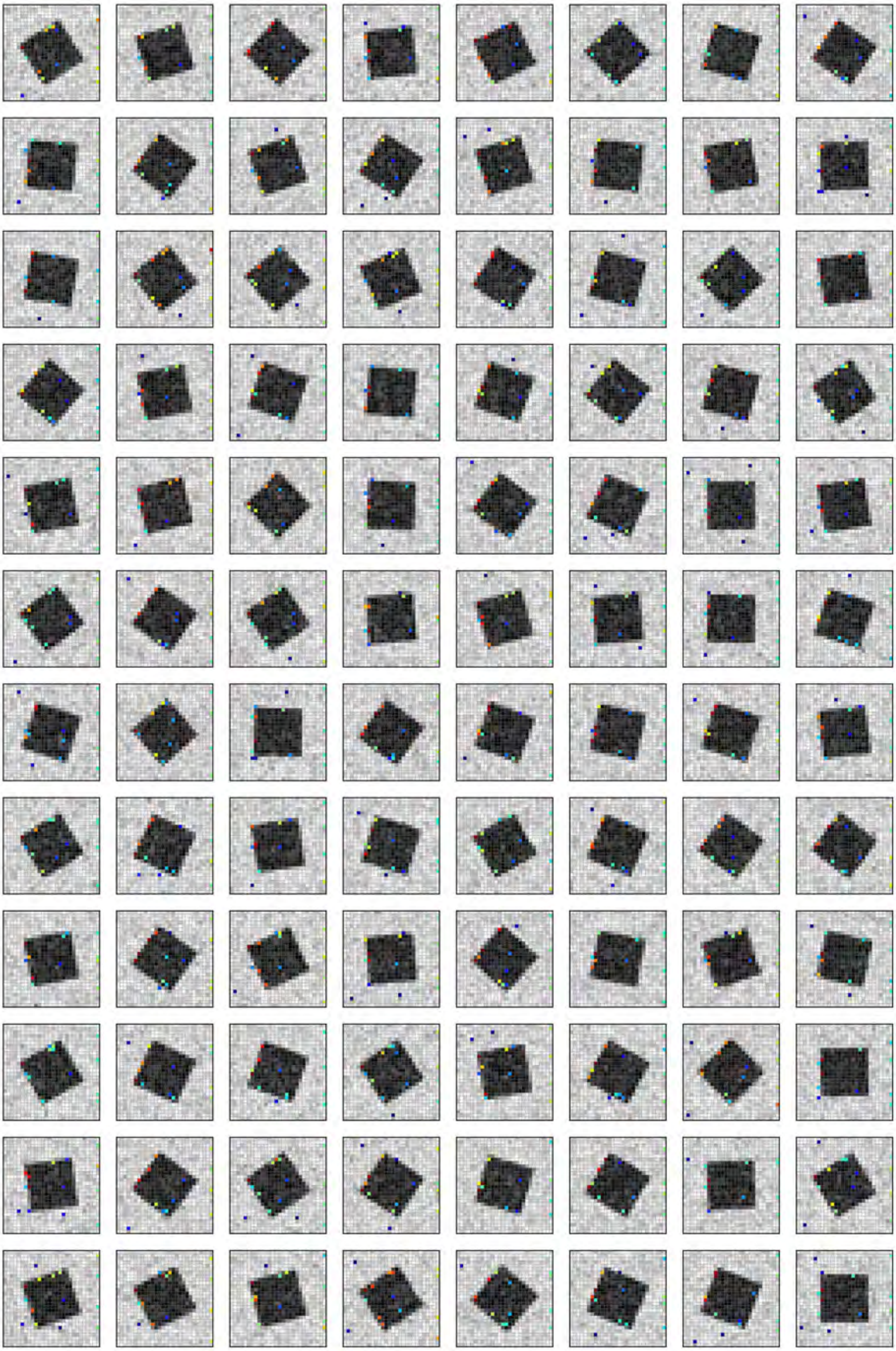

Figure 4.119. Original images of several rectangles from dataset 4.0 n20 with the figure in black on a white background and with maximum activation pixels in different colors according to each value (red representing the highest value and blue the lowest one). Best viewed in electronic form, zoom in to distinguish the pixel grid.

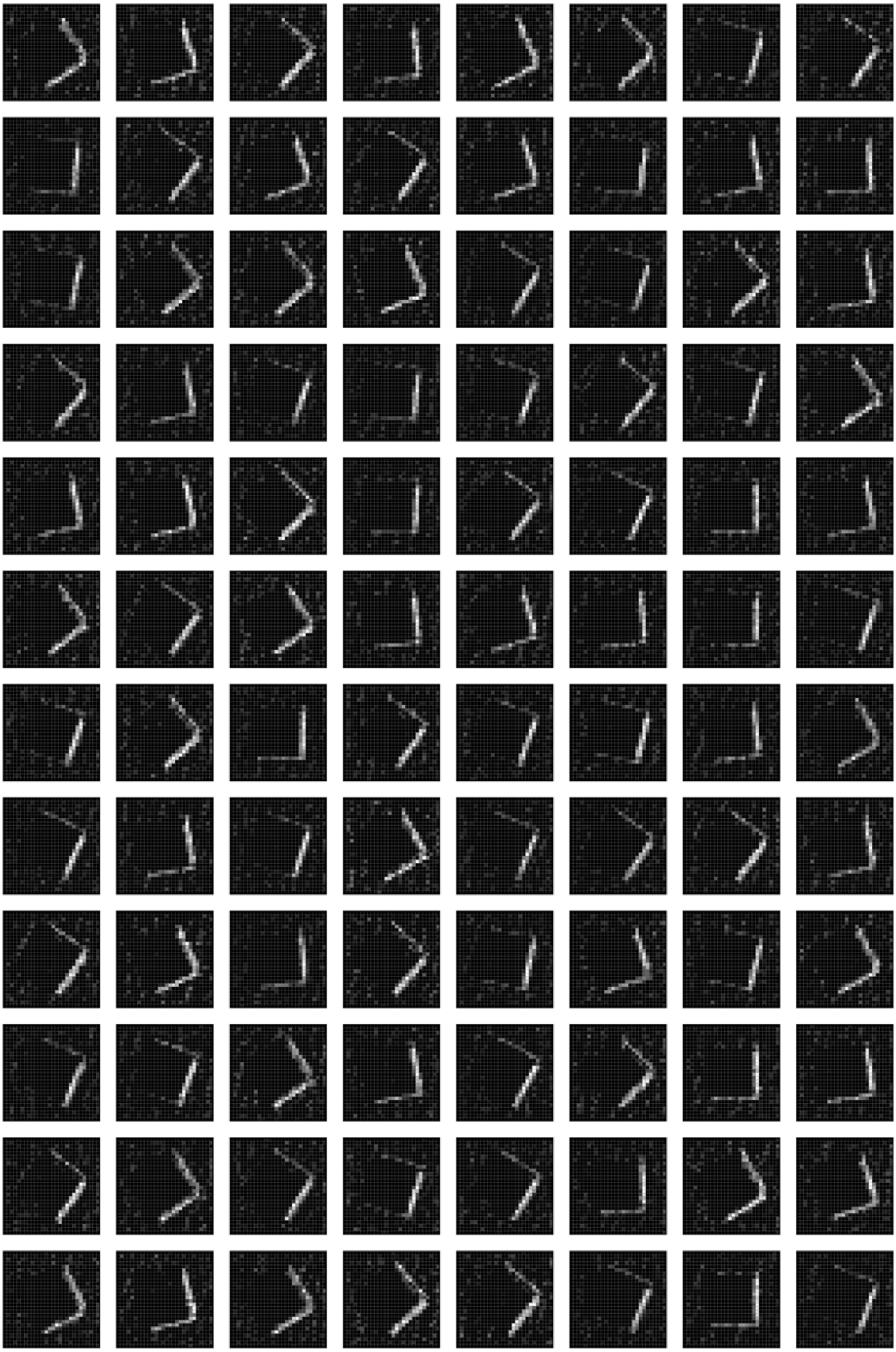

Figure 4.120. Feature maps of several rectangle samples with the figure in white on a black background from dataset 4.0 n20. Best viewed in electronic form, zoom in to distinguish the pixel grid.

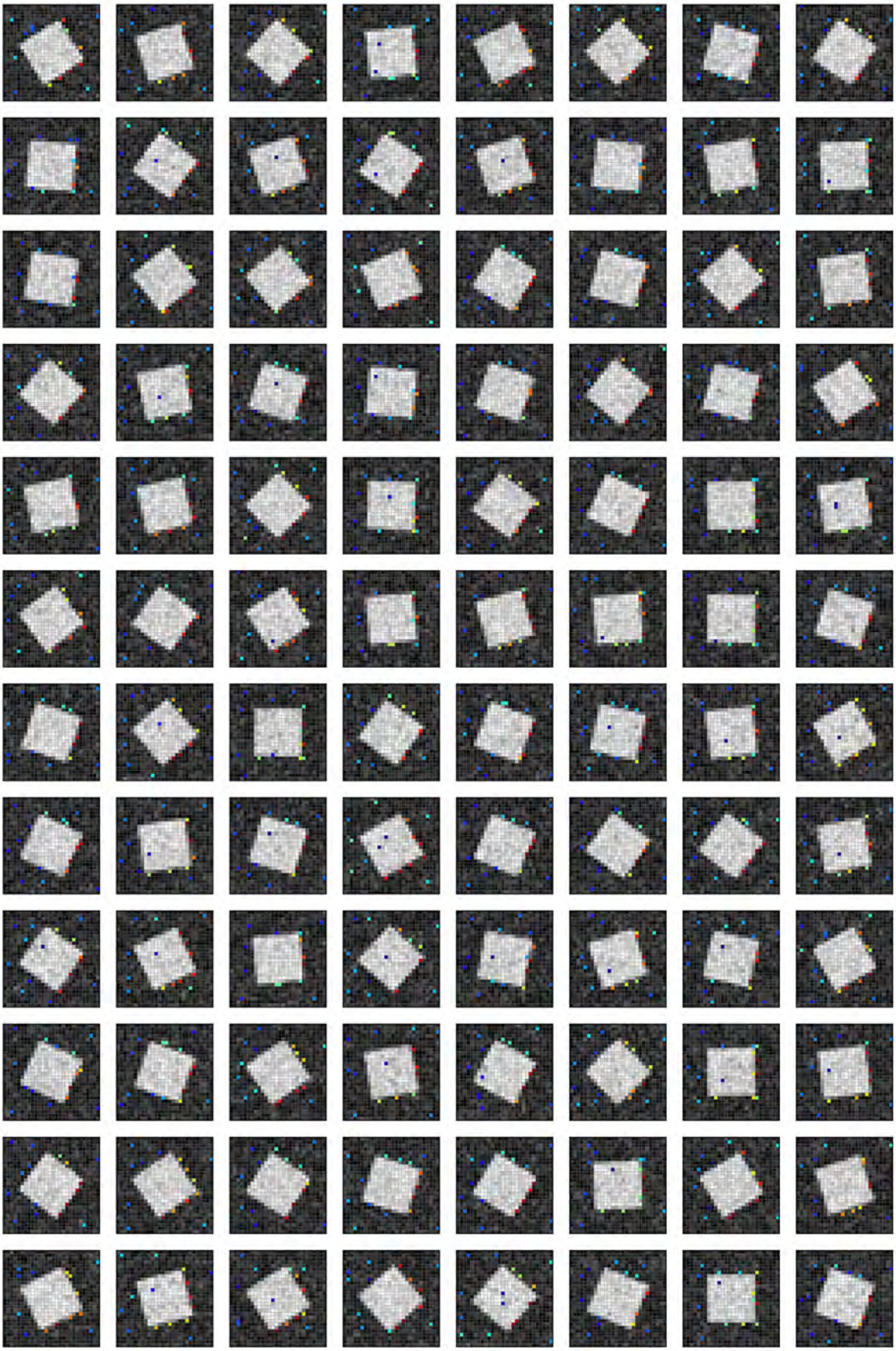

Figure 4.121. Original images of several rectangles from dataset 4.0 n20 with the figure in white on a black background and with maximum activation pixels in different colors according to each value (red representing the highest value and blue the lowest one). Best viewed in electronic form, zoom in to distinguish the pixel grid.

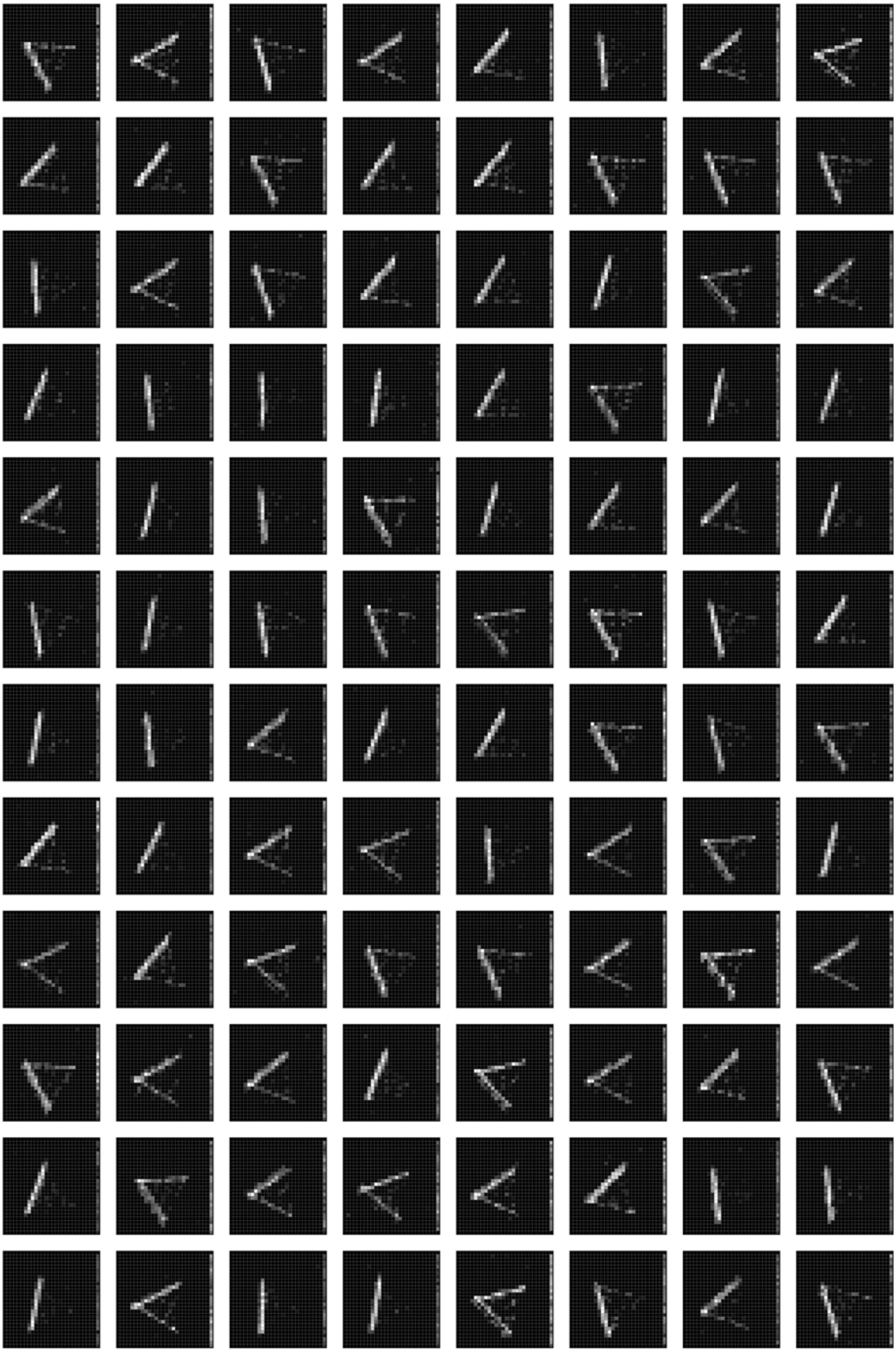

Figure 4.122. Feature maps of several triangle samples with the figure in black on a white background from dataset 4.0 n20. Best viewed in electronic form, zoom in to distinguish the pixel grid.

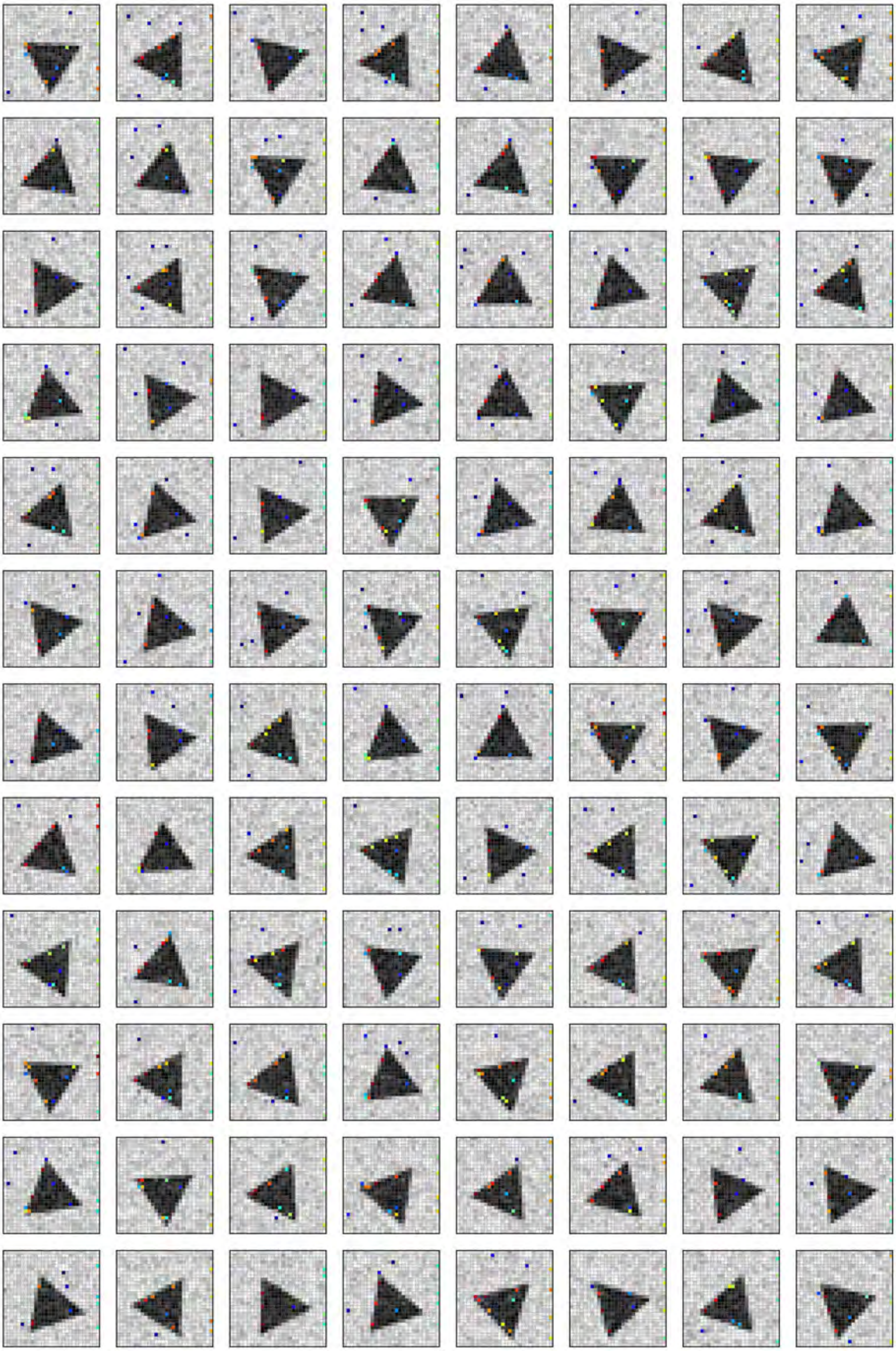

Figure 4.123. Original images of several triangle from dataset 4.0 n20 with the figure in black on a white background and with maximum activation pixels in different colours according to each value (red representing the highest value and blue the lowest one). Best viewed in electronic form, zoom in to distinguish the pixel grid.

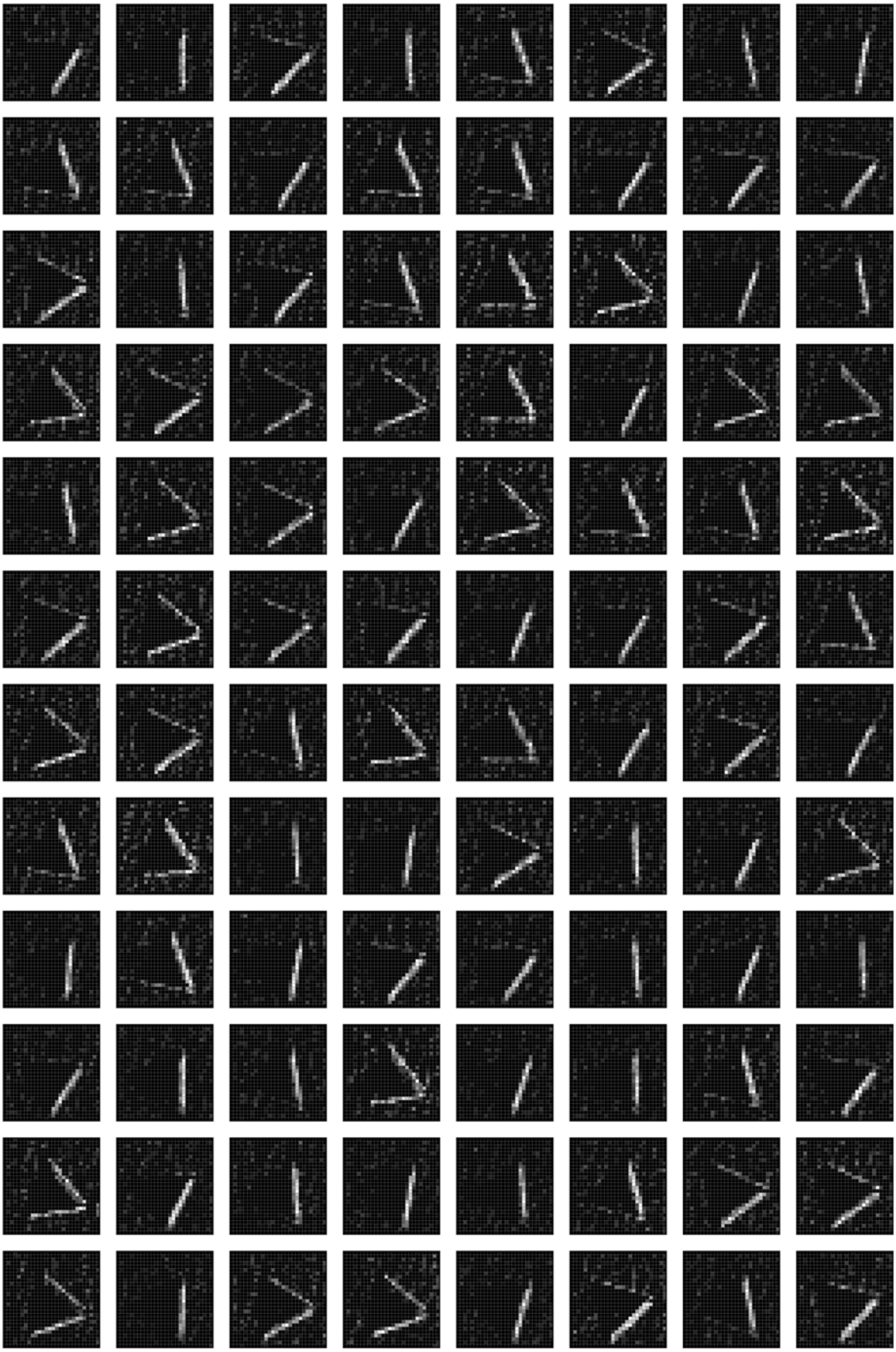

Figure 4.124. Feature maps of several triangle samples with the figure in white on a black background from dataset 4.0 n20. Best viewed in electronic form, zoom in to distinguish the pixel grid.

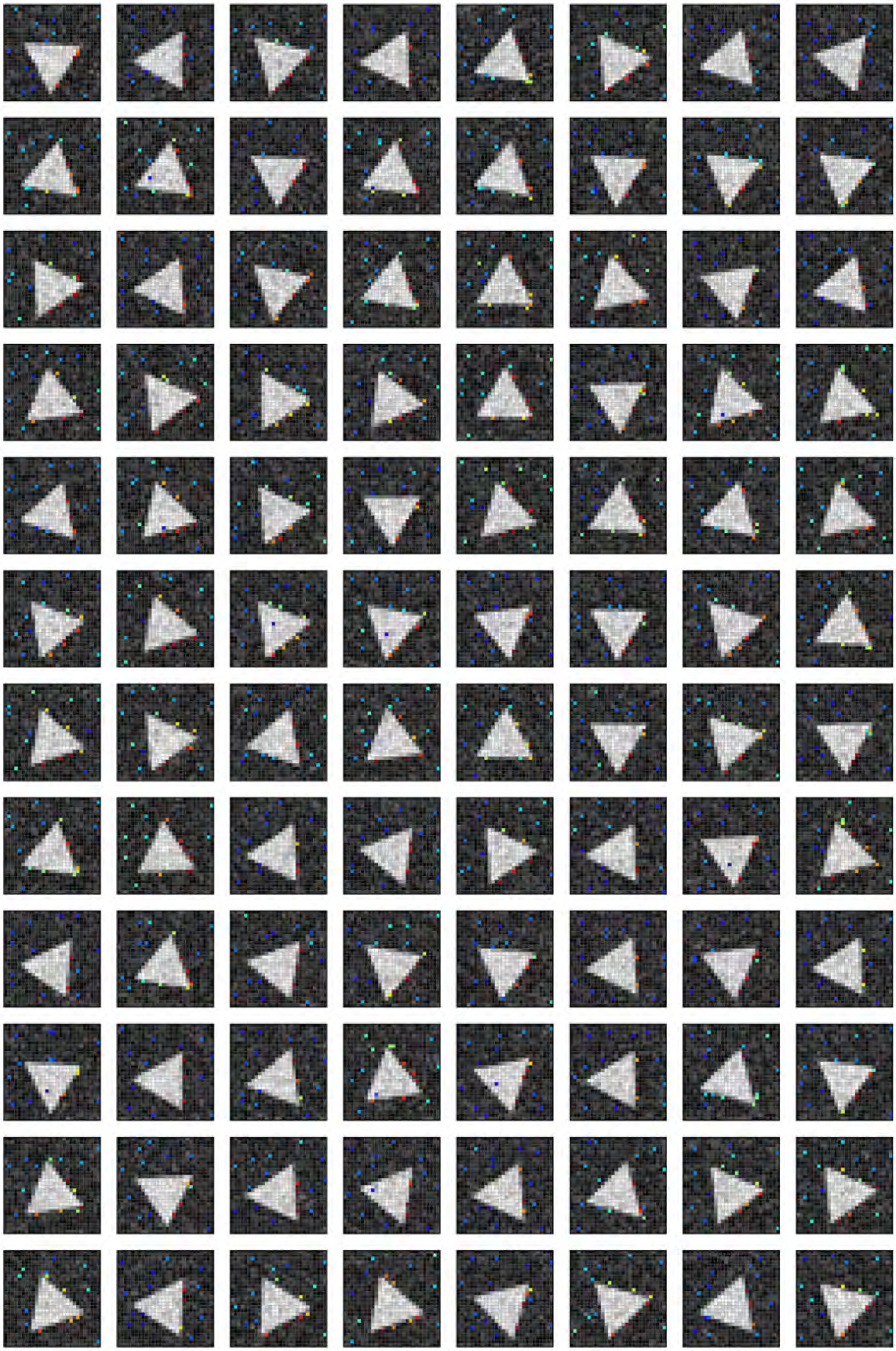

Figure 4.125. Original images of several triangles from dataset 4.0 n20 with the figure in white on a black background and with maximum activation pixels in different colours according to each value (red representing the highest value and blue the lowest one). Best viewed in electronic form, zoom in to distinguish the pixel grid.

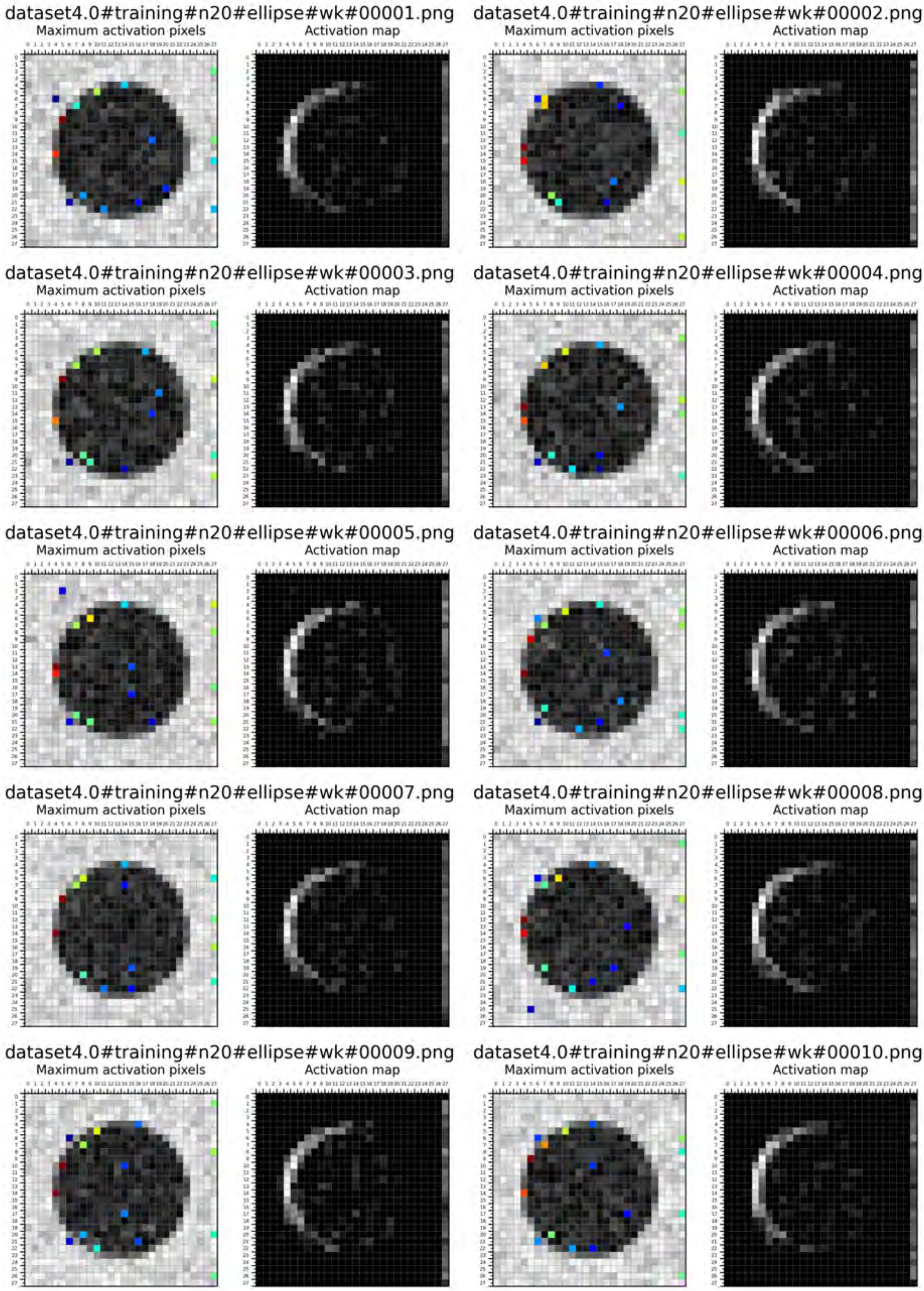

Figure 4.126. Both feature map and maximum activation pixel representations of several ellipses samples with the figure in black on a white background from dataset 4.0. n20 in a detailed form. Best viewed in electronic form, zoom in to distinguish the pixel grid.

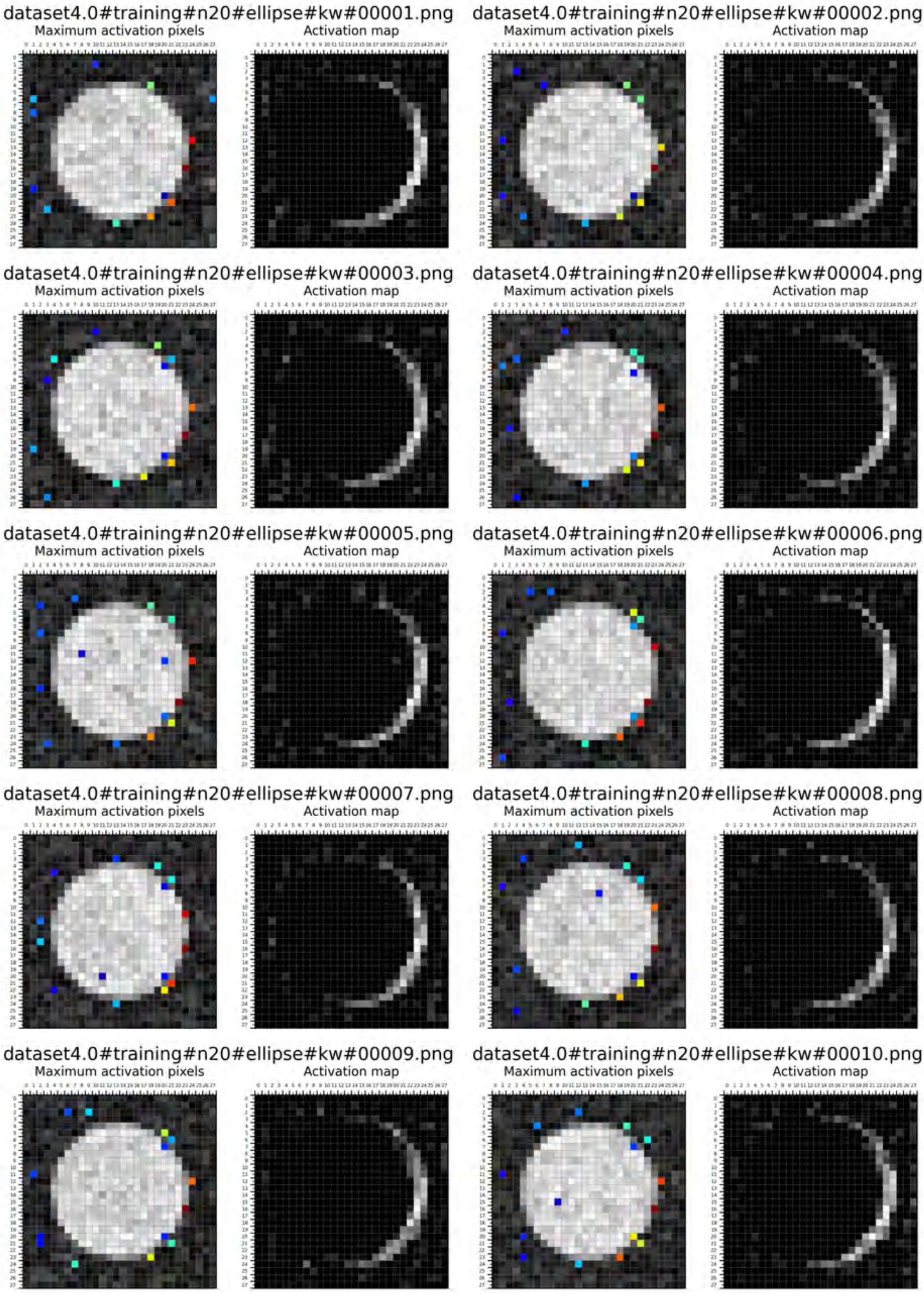

Figure 4.127. Both feature map and maximum activation pixel representations of several ellipses samples with the figure in white on a black background from dataset 4.0. n20 in a detailed form. Best viewed in electronic form, zoom in to distinguish the pixel grid.

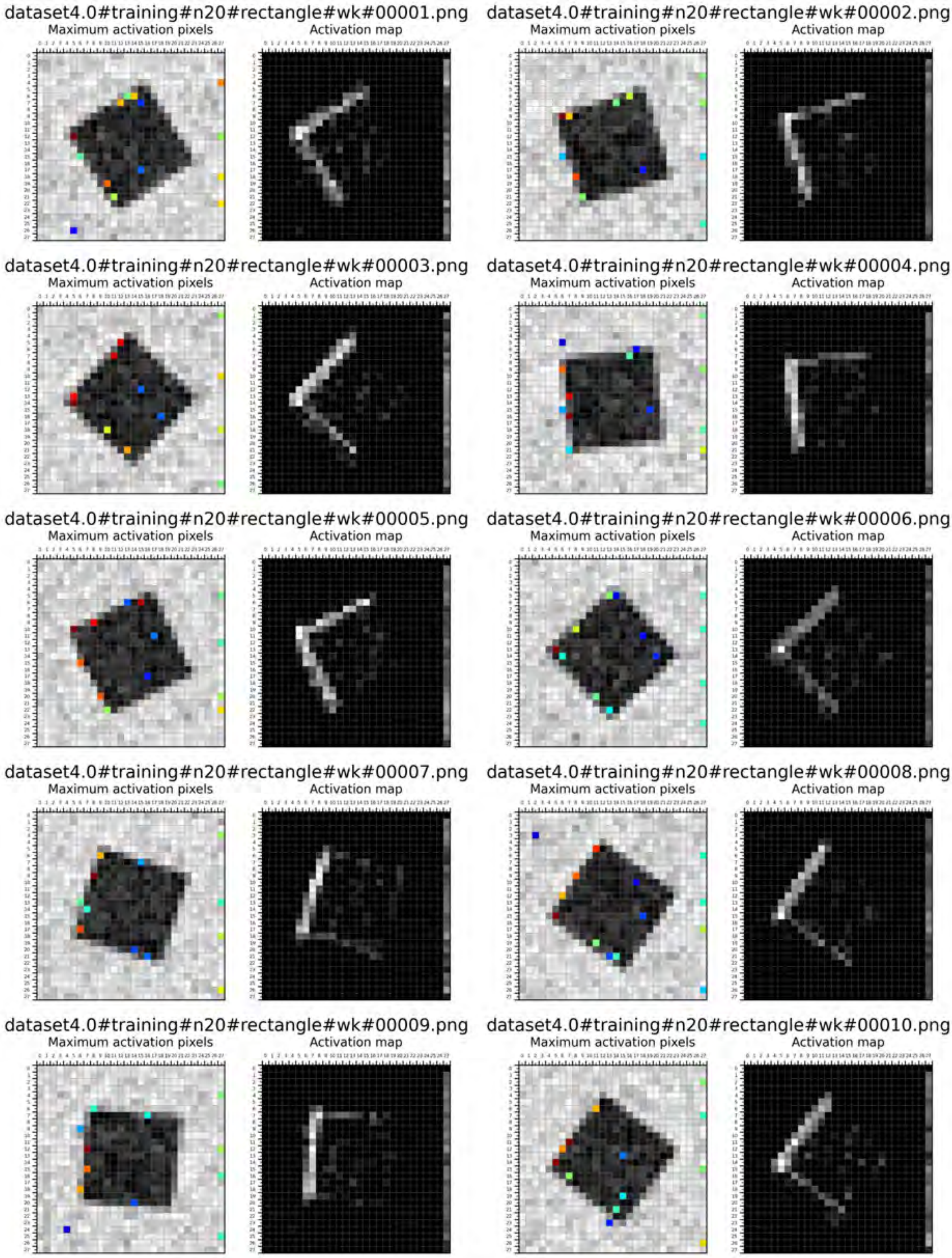

Figure 4.128. Both feature map and maximum activation pixel representations of several rectangles samples with the figure in black on a white background from dataset 4.0. n20 in a detailed form. Best viewed in electronic form, zoom in to distinguish the pixel grid.

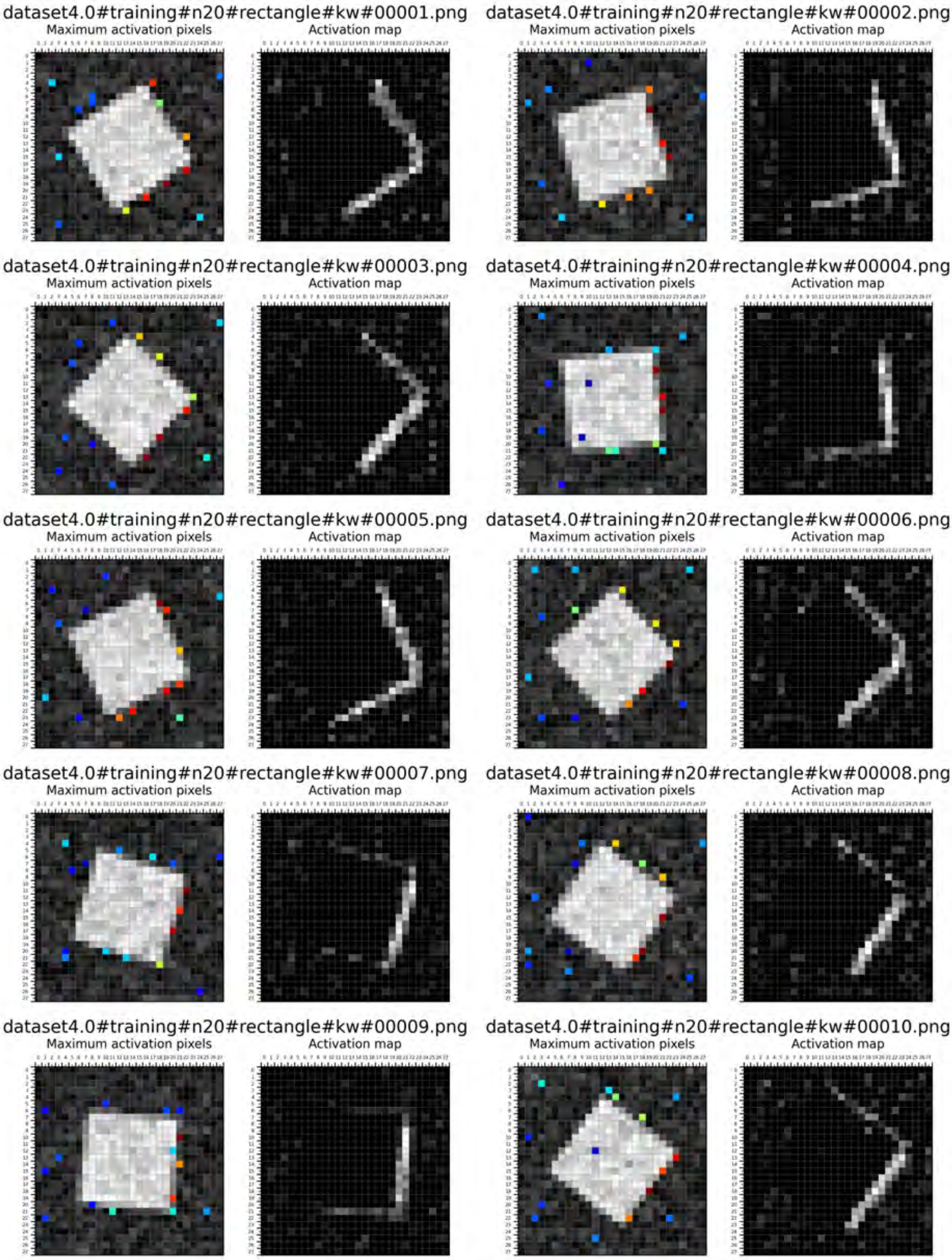

Figure 4.129. Both feature map and maximum activation pixel representations of several rectangles samples with the figure in white on a black background from dataset 4.0. n20 in a detailed form. Best viewed in electronic form, zoom in to distinguish the pixel grid.

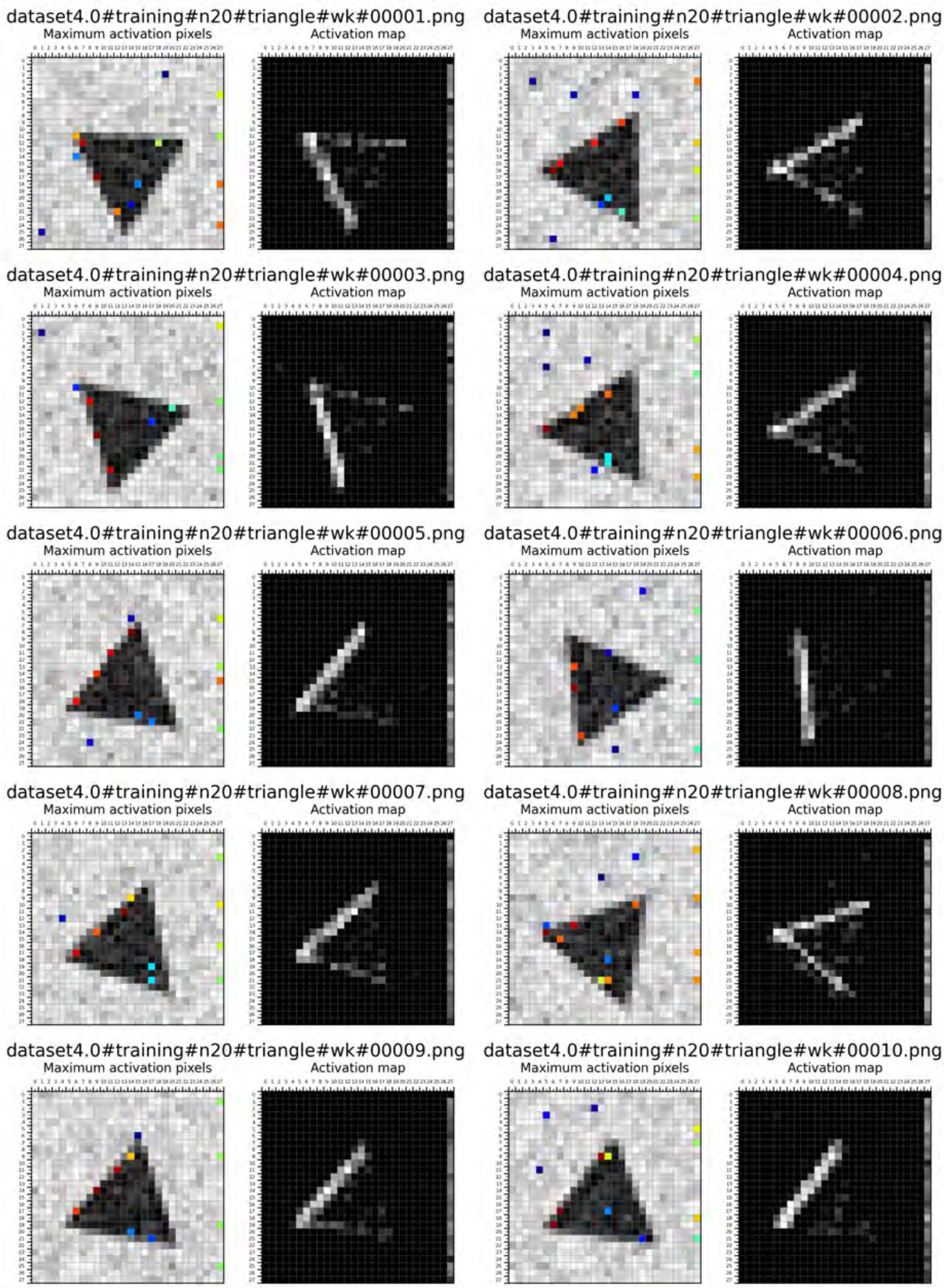

Figure 4.130. Both feature map and maximum activation pixel representations of several triangles samples with the figure in black on a white background from dataset 4.0. n20 in a detailed form. Best viewed in electronic form, zoom in to distinguish the pixel grid.

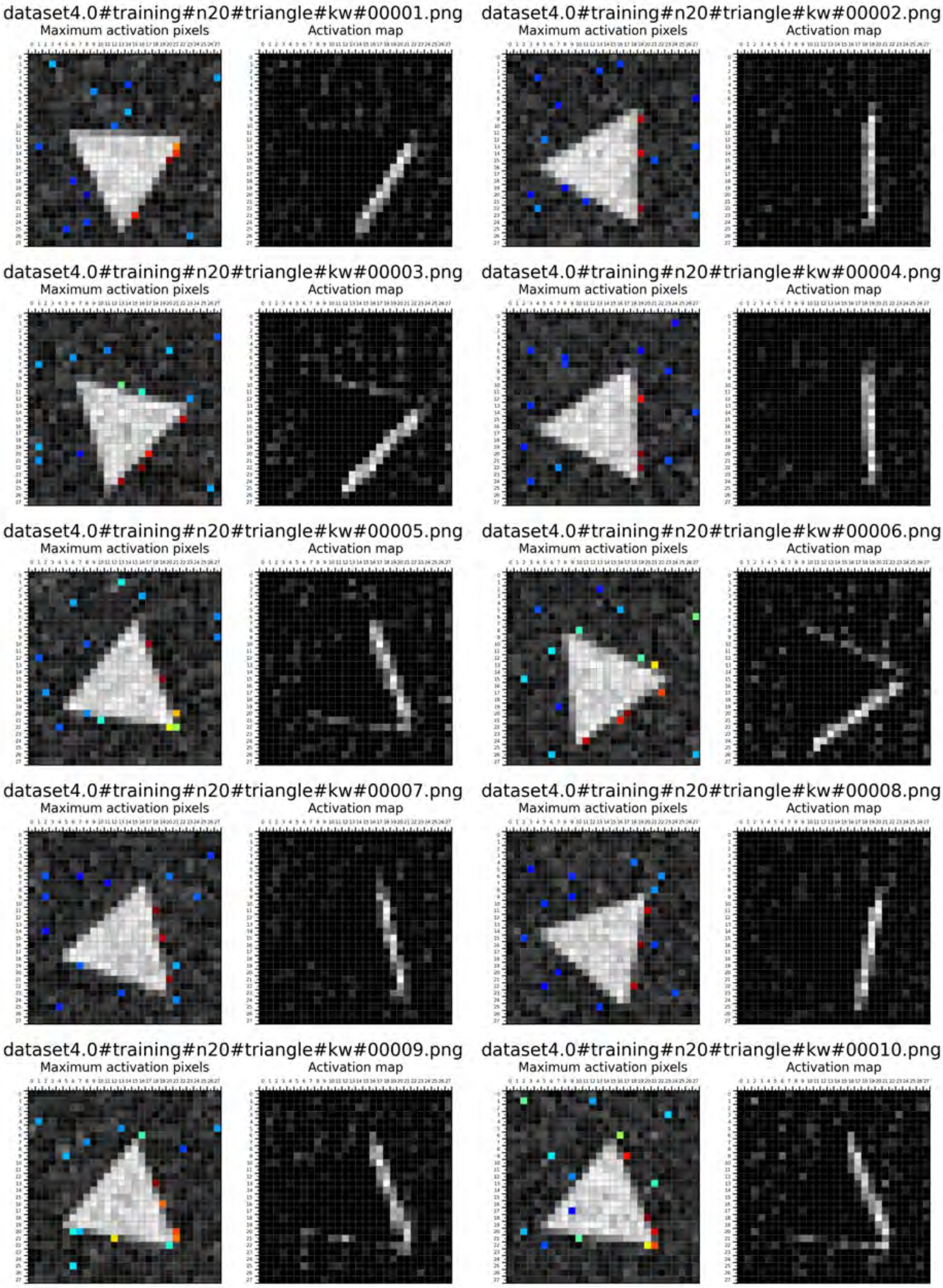

Figure 4.131. Both feature map and maximum activation pixel representations of several triangles samples with the figure in white on a black background from dataset 4.0. n20 in a detailed form. Best viewed in electronic form, zoom in to distinguish the pixel grid.

5 Conclusions and future work

Love of learning is the most necessary passion...
in it it lies our happiness.

Émilie du Châtelet (1706–1749)

The last chapter summarizes the most relevant aspects of this thesis. In addition, the conclusions drawn from the different experiments carried out are set forth. Finally, possible future research lines as well as potential improvements of the existing methodology are discussed.

5.1. Summary and conclusions

This thesis presents a new methodology that contributes to the field of Explainable Artificial Intelligence (XAI), which focuses on making results given by AI algorithms understandable for humans, since the process these techniques follow to obtain their solutions are usually so complex that the model designers are not able to explain why it has arrived at a specific decision. In particular, this thesis is focused on making the CNN learning process easier to interpret but with a different approach from the existing ones. That is, training a very simple model with a dataset generated synthetically, so that the complexity of the problem can be carefully controlled. The methodology consists of three main phases: dataset generation, minimum model search and interpretation of results. From each one of them several conclusions can be drawn.

Several datasets have been built synthetically but all of them have the same structure: grayscale images of centred geometric shapes belonging to three different classes; ellipses, rectangles and triangles. Over this datasets, some transformations have been applied such as scaling, rotation and noise, giving as a result different datasets with which each problem can be analysed independently. This is the first difference with the most commonly used approaches, each dataset of images is made in such a way that the model learns to solve a particular problem; which at the same time eases the understanding of that learning process.

The key point of the methodology is the search for the minimum model, understood as the one that achieves the maximum accuracy with the lowest level of complexity. The reason behind this is that simple models are more easily explained, since they have fewer parameters. This minimum model is obtained in the first place thanks to the synthetic generation of datasets and also to a careful process of grid search, which trains a wide set of architectures trying different values for the main hyperparameters until the minimum model is obtained. It has been observed that it is possible to perform image classification achieving high levels of accuracy by training an architecture comprised of a single convolutional layer with one filter. Contrary to what the general knowledge about CNN states, for a simple problem there is no need for a big and complex network.

However, it is important to take into account the requirement of performing many adjustments to the model. In particular, an architecture should not be discarded unless it has been trained with different initialization seeds and with the sufficient amount of epochs. Otherwise, the model is either stuck in a local minimum or unable to finish its training and converge to the point of maximum accuracy. The simplest architectures should not be dismissed just because they seem to be little complex, but rather a fine-tuning of the number of epochs and the initialization point should be carried out. It has been confirmed that the grid search method involves high amounts of resources like time and computing power to obtain valid results but it is a very controlled technique which enables to analyse step by step the models that are being trained. In addition, it is easy to detect mistakes when building groups of architectures.

All minimum models selected are able to recognise images with that basic architecture. In addition, for scaling and rotation problems without noise, the model manages to perform the classification with just one feature, whereas for the scaling plus noise problem the model has four features and for the rotation plus noise problem the model needs sixteen features. Important insights are derived from these architectures. First of all, it does not make sense to train datasets of this kind without noise, since bias is inevitably introduced. The level of noise used should be another hyperparameter whose value has to be obtained during the fine-tuning of the CNN model. On the other hand, the rotation problem seems to be more difficult for the network than the scaling one, since the model needs a high amount of features to solve it.

Regarding the interpretation of results, CNN models are focused on very specific details of the image, with a level of detail that is impossible for a human being to think of. The CNN model is able to find an optimal features space where the image samples are separable by its class and, as it has been previously stated, these features are not the ones that a human may think the network is extracting (for example, number of sides for each geometric figure), they are not evident. For the model the features are just different luminance values, and the combination of them enables the CNN to perform the classification. In addition the feature extraction power of the CNN can be combined with other ML classification algorithms, in a similar way BoW algorithm makes use of an independent human-designed feature extraction technique and a classification algorithm.

For these reasons, it is confirmed again that the dataset needs to be carefully built, otherwise is very easy to introduce bias. Due to the kind of very specific features that the network is learning, for datasets without noise in which the network is learning a unique feature, changing a single pixel can affect the whole decision the network is making resulting in classifying an ellipse as a triangle, for example.

As pointed out on the state of the art review, on the first convolutional layer the network learns filters focused on extracting very simple features, as in this case: borders. With just one filter the CNN is able to obtain the same kind of border through different images and activate with distinct values of luminance for each class, which makes the classifier at the end of the network able to separate images later on. Combination of colours have a great impact on the behaviour of the filter, detecting opposite values depending on whether the image has a black figure on a white background or a white figure on a black background, as the gradient of the border will change.

On the other hand, a new methodology to analyse model features is presented on this thesis. It is about using ML techniques such as decision trees, PCA or K-means algorithms to study the feature space given by the CNN. This is possible to perform due to the low amount of features that the minimum model extracts, which reaffirms the usefulness of the new methodology. In some cases, a series of special datasets with specific features have been built in order to train these ML algorithms and compare its solutions with the ones given by the network, so a correspondence could be established and even determining if one of the image datasets is biased.

5.2. Future work

The methodology that has been presented in this thesis consists of several phases. The second of them is searching for the minimum model and it is performed by making use of the grid search. Several architectures are trained changing the values of the hyperparameters. Once all the models have finished its training, the model selected is one with a high level of accuracy in both training and validation but also with a low level of complexity. Although the grid search technique increases complexity in a progressive way, it may be the case that between two sequential architectures in complexity, there is a high rise in the number of parameters contained in the CNN but not all those parameters are essential for the network to perform the classification. An intelligent way of removing the parameters that are unnecessary without losing overall accuracy is pruning. Pruning enables to obtain models which are smaller in size, faster in their runtime and more efficient both in memory and power.

An improvement for the methodology presented is to add a new phase to the process in which, after having executed the grid search algorithm and selected the minimum model, a pruning technique is applied to the CNN. It is necessary to study which kind of pruning is the most accurate to this problem: weight pruning or unit/neuron pruning, but one way or another is one of the most commonly used methodologies to reduce the complexity of the network without increasing its error rate, as it has been previously mentioned in Section 3.1 of the Literature Review. On the other hand, another improvement related to the methodology is to put more emphasis on the filter analysis. This can be done by attempting to decompose the filters obtained by the CNN in linear combinations of classic ones and applying them to an architecture without training, to see if it is able to perform the classification. This will enable to obtain a catalogue of standard filters that can be applied depending on the kind of problem. Moreover, these filters could perhaps be used to initialize the network and reduce training times.

An idea to further extend the series of experiments conducted in this thesis, will be to analyse the datasets that have already been generated with the levels of noise that are different from n20. For groups of experiments 3 and 4 it could be analysed both scaling and rotation problems when the noise introduced in the images is lower, with levels n01, n02, n05 and n10. It would be interesting to determine to what extent reducing the level of noise decreases the number of parameters required to accurately classify the dataset.

Another interesting new approach is to generate synthetic datasets with images that show more elaborated geometric figures, so the experiments can follow the same common thread, but with examples more complex to classify. For example, each one of the three classes can be replaced by a set of emojis that have the same geometry: face emojis could substitute ellipses, rectangles can be replaced by flags and triangles for hearts. It can be analysed if the CNN models extracted are able to perform the classification correctly when the images have similar geometric shapes but they are not exactly the same. And if the complexity of the problem is similar to the experiments carried out in this thesis or not, will the filters learned be similar?

Finally, the analysis of the samples that are wrongly classified by the network has little detail. A future line of work is to go in depth in the study of the mistakes, for example, analysing them with the ML techniques that have been used in the other analyses. Apart from PCA, the mistakes could be plotted on the K-means visualizations to check in what clusters these samples are included. Knowing the reason behind the mistakes of the CNN models can help gain insight into the behaviour of the model and, in the end, its learning process.

A

Alignment with the Sustainable Development Goals

How wonderful it is that
nobody need wait a single moment
before starting to improve the world.
Anne Frank (1929–1945)

This appendix includes a reflection on the alignment of the project with the Sustainable Develoment Goals (SDG). The goals to which the project is most closely related are named and its contribution to each one of them is explained in detail.

Adopted by all United Nations Member States in 2015, the Sustainable Development Goals (SDG) establish a common blueprint to achieve in 2030 a better and more sustainable planet, facing global problems like poverty, inequality, climate change or peace [106]. There are 17 goals, each one of them focused on a different aspect of the society that needs to be transformed worldwide.

In order to reach these goals, projects that are developed in each SDG supporting country need to show how their objectives are aligned with them. In this case, the thesis supports mainly two SDG and one more in an indirect way being the former the numbers 9 and 12, and the latter, number 10.

Goal 9, *Build resilient infrastructure, promote sustainable industrialization and foster innovation*, groups industries, innovation and infrastructure all together in order to create new economy forces able to generate employment and income sources making a responsible use of the resources [106]. It is undeniable that AI is playing and important role in the modernization of industry or, as it is commonly referred, the Industry 4.0. In order to implement ML algorithms in both the supply chain and the final product, the first step is to understand how these "black boxes" are working, so they can be fully controlled without risks as well as optimized. With an optimal automation of the basic tasks, human labour can reinvent themselves and focus on more rewarding activities that really provide added value to the enterprise. In addition, the thesis is also aligned with the European Commission's strategy on Artificial Intelligence, which aims at addressing socio-economic, legal and ethical effects of this kind of algorithms [107], enabling to gain control over CNN learning process and therefore, avoiding negative impacts on society.

Goal 12 is to *Ensure sustainable consumption and production patterns* and fight backs against current lifestyles based on linear economies (buy-use-dispose). It is widely known that the technology industry is one of the most polluting ones, not only because it creates up to 3.5% of global emissions [108] but also because 50 million tonnes of e-waste are produced each year [109]. Understanding deeply how the algorithms are learning is decisive to train them in a more efficient way. This involves making less use of computational resources and therefore, reducing the power consumption made by tech industry.

Finally, the present thesis pushes indirectly goal number 10: *Reduce inequality within and among countries*, which aims to achieve equal opportunities among all communities. Giving its ability to imitate human behaviour and help enterprises in the decision-making process, AI algorithms are increasingly used in numerous processes that can have a great impact on people's lives. That is another of the main reasons why it is crucial to know the how the algorithm is learning and be aware of the biases that can appear. For example, if a neural network is used to help in the selection process made by an enterprise to hire new employees, there is a risk that certain features of the candidates such as gender, race or religious orientation, are being prioritised inadvertently, which can derive into social injustices.

Bibliography

[1] *Python*. [Online]. Available: https://www.python.org/ (visited on 06/25/2020).

[2] *TensorFlow Guide*. [Online]. Available: https://www.tensorflow.org/guide (visited on 06/25/2020).

[3] *What is Machine Learning? A definition*. [Online]. Available: https://expertsystem.com/machine-learning-definition/ (visited on 09/07/2020).

[4] D. Soni, *Spiking Neural Networks, the Next Generation of Machine Learning*. [Online]. Available: https://towardsdatascience.com/spiking-neural-networks-the-next-generation-of-machine-learning-84e167f4eb2b (visited on 07/09/2020).

[5] F. Shaikh, *Deep Learning vs. Machine Learning – the essential differences you need to know!* [Online]. Available: https://www.analyticsvidhya.com/blog/2017/04/comparison-between-deep-learning-machine-learning/ (visited on 07/09/2020).

[6] M. Z. Alom, T. M. Taha, C. Yakopcic, S. Westberg, M. Hasan, B. C. V. Esesn, A. A. S. Awwal, Asari, and V. K., "The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches", 2018. [Online]. Available: https://arxiv.org/ftp/arxiv/papers/1803/1803.01164.pdf.

[7] *Scale-invariant feature transform*. [Online]. Available: https://en.wikipedia.org/wiki/Scale-invariant_feature_transform (visited on 02/02/2021).

[8] *MathWorks. Deep Learning. Tres cosas que es necesario saber.* [Online]. Available: https://es.mathworks.com/discovery/deep-learning.html (visited on 07/09/2020).

[9] K. Fukushima, "Neocognitron: A hierarchical neural network capable of visual pattern recognition", *Neural Networks*, vol. 1, no. 2, pp. 119–130, 1988.

[10] D. H. Hubel and T. N. Wiesel, "Receptive field of single neurones in the cat's striate cortex", 1959.

[11] *Visual Cortex Cell Recording*. [Online]. Available: https://www.youtube.com/watch?time_continue=3&v=Cw5PKV9Rj3o&feature=emb_logo (visited on 07/11/2020).

[12] *Neural Network*. [Online]. Available: https://deepai.org/machine-learning-glossary-and-terms/neural-network (visited on 07/11/2020).

[13] *MC.AI. How to Implement a Neural Network with only +1 and -1?* [Online]. Available: https://mc.ai/how-to-implement-a-neural-network-with-only-1-and-1/ (visited on 07/11/2020).

[14] *Artificial Neuron*. [Online]. Available: https://www.oreilly.com/library/view/hands-on-artificial-intelligence/9781788836067/d1a64b7c-df5c-4f7f-8f24-f4429b4c7908.xhtml (visited on 07/11/2020).

[15] L. García, *Machine Learning y Deep Learning con MATLAB*, 2015.

[16] *WTF is a Tensor?!?* [Online]. Available: https://www.kdnuggets.com/2018/05/wtf-tensor.html (visited on 07/13/2020).

[17] *Introduction to Tensors*. [Online]. Available: https://www.tensorflow.org/guide/tensor (visited on 07/13/2020).

[18] *Convolutional Neural Networks (CNNs / ConvNets) for Visual Recognition*. [Online]. Available: https://becominghuman.ai/convolutional-neural-networks-cnns-convnets-for-visual-recognition-cae879a70f1a (visited on 02/03/2020).

[19] *Deep Learning Book Series · 2.1 Scalars Vectors Matrices and Tensors*. [Online]. Available: https://hadrienj.github.io/posts/Deep-Learning-Book-Series-2.1-Scalars-Vectors-Matrices-and-Tensors/ (visited on 07/13/2020).

[20] *What do you mean by Tensor and Explain about Tensor Datatype and Ranks?* [Online]. Available: https://www.i2tutorials.com/what-do-you-mean-by-tensor-and-explain-about-tensor-datatype-and-ranks/ (visited on 07/21/2020).

[21] *CS231n Convolutional Neural Networks for Visual Recognition*. [Online]. Available: https://cs231n.github.io/convolutional-networks/ (visited on 07/27/2020).

[22] *Local connection and weight sharing of convolutional neural networks*. [Online]. Available: https://www.programmersought.com/article/795075236/ (visited on 02/03/2021).

[23] E. S. Úbeda, *Deep Learning. Machine learning II. Introduction Convolutional Neural Networks*, 2020.

[24] *Quora. What exactly is meant by shared weights in convolutional neural network?* [Online]. Available: https://www.quora.com/What-exactly-is-meant-by-shared-weights-in-convolutional-neural-network (visited on 07/13/2020).

[25] *CNN Convolutional Layer in Depth*. [Online]. Available: https://becominghuman.ai/convolutional-layer-ab60395b8b27 (visited on 02/03/2021).

[26] *Gradient Descent Algorithm and Its Variants*. [Online]. Available: https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3 (visited on 07/14/2020).

[27] *Effect of batch size on training dynamics*. [Online]. Available: https://medium.com/mini-distill/effect-of-batch-size-on-training-dynamics-21c14f7a716e (visited on 02/03/2020).

[28] *Difference between a Batch and an Epoch in a Neural Network*. [Online]. Available: https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/ (visited on 07/21/2020).

[29] *Dropout Regularization in Deep Learning Models With Keras*. [Online]. Available: https://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/ (visited on 07/21/2020).

[30] *Dropout in (Deep) Machine Learning*. [Online]. Available: https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5 (visited on 07/21/2020).

[31] *MC.AI. What is Convolutional Neural Network (CNN) ? — with Keras*. [Online]. Available: https://mc.ai/what-is-convolutional-neural-network-cnn-âĂŁ-âĂŁwith-keras/ (visited on 07/21/2020).

[32] *How to Convert a Picture to Numbers*. [Online]. Available: https://www.kdnuggets.com/2020/01/convert-picture-numbers.html (visited on 07/27/2020).

[33] *Two-dimensional Arrays*. [Online]. Available: https://processing.org/tutorials/2darray/ (visited on 07/27/2020).

[34] *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*. [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 (visited on 07/27/2020).

[35] *A Comprehensive Introduction to Different Types of Convolutions in Deep Learning*. [Online]. Available: https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215 (visited on 11/16/2020).

[36] *Applied Deep Learning - Part 4 Convolutional Neural Networks*. [Online]. Available: https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2 (visited on 08/01/2020).

[37] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks", *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8689 LNCS, no. PART 1, pp. 818–833, 2014. arXiv: 1311.2901.

[38] C. Ware, *Visual thinking for design*. Morgan Kaufmann. [Online]. Available: https://www.sciencedirect.com/book/9780123708960/visual-thinking#book-description.

[39] *Human visual pathway*. [Online]. Available: https://commons.wikimedia.org/wiki/File:Human_visual_pathway.svg (visited on 08/01/2020).

[40] *FigShare. Ventral visual stream*. [Online]. Available: https://figshare.com/articles/Ventral_visual_stream/106794/2 (visited on 08/01/2020).

[41] *GitHub. Convolution arithmetic*. [Online]. Available: https://github.com/vdumoulin/conv_arithmetic (visited on 08/02/2020).

[42] *A Beginner's Guide To Understanding Convolutional Neural Networks Part 2*. [Online]. Available: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/ (visited on 08/02/2020).

[43] *Complete Guide of Activation Functions*. [Online]. Available: https://towardsdatascience.com/complete-guide-of-activation-functions-34076e95d044 (visited on 08/06/2020).

[44] *An Intuitive Explanation of Convolutional Neural Networks*. [Online]. Available: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/comment-page-2/ (visited on 08/06/2020).

[45] *A Gentle Introduction to the Rectified Linear Unit (ReLU)*. [Online]. Available: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/ (visited on 08/06/2020).

[46] *Wikipedia: Vanishing gradient problem*. [Online]. Available: https://en.wikipedia.org/wiki/Vanishing_gradient_problem (visited on 08/06/2020).

[47] *CNN. Introduction to Pooling Layer*. [Online]. Available: https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/ (visited on 08/08/2020).

[48] *Convolutional Neural Networks (CNN): Step 3 - Flattening*. [Online]. Available: https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-3-flattening (visited on 08/02/2020).

[49] *Interactive Plot 2D version*. [Online]. Available: https://www.cs.ryerson.ca/$\sim$aharley/vis/conv/flat.html (visited on 08/23/2020).

[50] *Interactive Plot 3D version*. [Online]. Available: https://www.cs.ryerson.ca/$\sim$aharley/vis/conv/ (visited on 08/23/2020).

[51] A. W. Harley, "An Interactive Node-Link Visualization of Convolutional Neural Networks", *ISVC*, pp. 867–877, 2015.

[52] *The MNIST database of handwritten digits*. [Online]. Available: http://yann.lecun.com/exdb/mnist/ (visited on 08/23/2020).

[53] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-Based Learning Applied to Document Recognition", *proc. OF THE IEEE*, 1998. [Online]. Available: http://ieeexplore.ieee.org/document/726791/#full-text-section.

[54] *Illustrated: 10 CNN Architectures*. [Online]. Available: https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d (visited on 08/23/2020).

[55] G. E Krizhevsky, A., Sutskever, I., and Hinton, "ImageNet classification with deep convolutional neural networks", *NIPS*, pp. 1106–1114, 2012.

[56] *ZFNet/DeconvNet: Summary and Implementation*. [Online]. Available: https://hackmd.io/@bouteille/ByaTE80BI (visited on 08/30/2020).

[57] M. Lin, Q. Chen, and S. Yan, "Network in network", *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*, pp. 1–10, 2014. arXiv: arXiv:1312.4400v3.

[58] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition", *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pp. 1–14, 2015. arXiv: arXiv:1409.1556v6.

[59] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions", *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 07-12-June, pp. 1–9, 2015. arXiv: 1409.4842.

[60] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision", *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2016-Decem, pp. 2818–2826, 2016. arXiv: 1512.00567.

[61] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-ResNet and the impact of residual connections on learning", *31st AAAI Conference on Artificial Intelligence, AAAI 2017*, pp. 4278–4284, 2017. arXiv: 1602.07261.

[62] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition", *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2016-Decem, pp. 770–778, 2016. arXiv: 1512.03385.

[63] *Why ResNets Are A Major Breakthrough In Image Processing*. [Online]. Available: https://analyticsindiamag.com/why-resnets-are-a-major-breakthrough-in-image-processing/ (visited on 09/01/2020).

[64] *Intuitive Explanation of Skip Connections in Deep Learning*. [Online]. Available: https://theaisummer.com/skip-connections/ (visited on 09/01/2020).

[65] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks", *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, vol. 2017-Janua, pp. 5987–5995, 2017. arXiv: 1611.05431.

[66] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks", *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, vol. 2017-Janua, pp. 2261–2269, 2017. arXiv: 1608.06993.

[67] *ImageNet*. [Online]. Available: http://www.image-net.org/about-overview (visited on 09/02/2020).

[68] *Wikipedia: ImageNet*. [Online]. Available: https://en.wikipedia.org/wiki/ImageNet (visited on 09/02/2020).

[69] *Accessing Text Corpora and Lexical Resources*. [Online]. Available: https://www.nltk.org/book/ch02.html (visited on 09/02/2020).

[70] J. Deng, "Large Scale Visual Recognition (Dissertation)", 2012.

[71] *WordNet*. [Online]. Available: https://wordnet.princeton.edu/ (visited on 09/02/2020).

[72] *Wikipedia: WordNet*. [Online]. Available: https://en.wikipedia.org/wiki/WordNet (visited on 09/02/2020).

[73] *ImageNet Large Scale Visual Recognition Challenge (ILSVRC)*. [Online]. Available: http://www.image-net.org/challenges/LSVRC/ (visited on 09/09/2020).

[74] *A Gentle Introduction to the ImageNet Challenge (ILSVRC)*. [Online]. Available: https://machinelearningmastery.com/introduction-to-the-imagenet-large-scale-visual-recognition-challenge-ilsvrc/ (visited on 09/02/2020).

[75] *Pretrained Deep Neural Networks*. [Online]. Available: https://es.mathworks.com/help/deeplearning/ug/pretrained-convolutional-neural-networks.html (visited on 09/02/2020).

[76] *An Overview of Model Compression Techniques for Deep Learning in Space*. [Online]. Available: https://medium.com/gsi-technology/an-overview-of-model-compression-techniques-for-deep-learning-in-space-3fd8d4ce84e5 (visited on 12/27/2020).

[77] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks", *Advances in Neural Information Processing Systems*, vol. 2015-Janua, pp. 1135–1143, 2015. arXiv: 1506.02626.

[78] *Pruning Neural Networks*. [Online]. Available: https://towardsdatascience.com/pruning-neural-networks-1bb3ab5791f9 (visited on 12/27/2020).

[79] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks", *arXiv*, pp. 1–42, 2018. arXiv: 1803.03635.

[80] T. Dettmers and L. Zettlemoyer, "Sparse networks from scratch: Faster training without losing performance", *arXiv*, no. 1, pp. 1–14, 2019. arXiv: 1907.04840.

[81] X. Ma, S. Lin, S. Ye, Z. He, L. Zhang, G. Yuan, S. H. Tan, Z. Li, D. Fan, X. Qian, X. Lin, K. Ma, and Y. Wang, "Non-Structured DNN Weight Pruning: Is It Beneficial in Any Platform ?", vol. 14, no. 8, pp. 1–15, 2015. arXiv: arXiv:1907.02124v2.

[82] K. Zhou, "Learning to Prune Filters in Convolutional Neural Networks", arXiv: arXiv:1801.07365v1.

[83] M. Jaderberg and A. Vedaldi, "Speeding up Convolutional Neural Networks with Low Rank Expansions", 2014.

[84] R. Rigamonti and E. P. F, "Learning Separable Filters", pp. 2754–2761, 2013.

[85] "How to Visualize Filters and Feature Maps in Convolutional Neural Networks", [Online]. Available: https://machinelearningmastery.com/how-to-visualize-filters-and-feature-maps-in-convolutional-neural-networks/.

[86] *Image Kernels*. [Online]. Available: https://setosa.io/ev/image-kernels/ (visited on 12/29/2020).

[87] Q. Zhang, X. Wang, Y. N. Wu, H. Zhou, and S.-C. Zhu, "Interpretable CNNs for Object Classification", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2020. arXiv: 1901.02413.

[88] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Object detectors emerge in deep scene CNNs", *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 2015. arXiv: 1412.6856.

[89] M. Peters, L. Kempen, M. Nauta, and C. Seifert, "Visualising the training process of convolutional neural networks for non-experts", *CEUR Workshop Proceedings*, vol. 2491, 2019.

[90] *CNN Heat Maps: Class Activation Mapping (CAM)*. [Online]. Available: https://glassboxmedicine.com/2019/06/11/cnn-heat-maps-class-activation-mapping-cam/ (visited on 10/03/2020).

[91] J. R. Zech, M. A. Badgeley, M. Liu, A. B. Costa, J. J. Titano, and E. K. Oermann, "Confounding variables can degrade generalization performance of radiological deep learning models", pp. 1–15, 2018. arXiv: 1807.00431. [Online]. Available: http://arxiv.org/abs/1807.00431%0Ahttp://dx.doi.org/10.1371/journal.pmed.1002683.

[92] *Wikipedia: Heat map*. [Online]. Available: https://en.wikipedia.org/wiki/Heat_map (visited on 09/30/2020).

[93] *Wah lau, so hot! — Singapore's changing weather in 6 heat maps*. [Online]. Available: https://medium.com/@chinhonchua/wah-lau-so-hot-singapore-s-changing-weather-in-6-heat-maps-6d65bd360038 (visited on 09/30/2020).

[94] *ProVim. Heat maps record user mouse movements*. [Online]. Available: http://provim.net/heat-maps (visited on 09/30/2020).

[95] *Github: Grad-CAM implementation in Pytorch*. [Online]. Available: https://github.com/jacobgil/pytorch-grad-cam (visited on 10/03/2020).

[96] A. T. Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, "Learning Deep Features for Discriminative Localization", *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

[97] *Grad-CAM: Visual Explanations from Deep Networks*. [Online]. Available: https://glassboxmedicine.com/2020/05/29/grad-cam-visual-explanations-from-deep-networks/ (visited on 01/03/2021).

[98] *CNN Heat Maps: Saliency/Backpropagation*. [Online]. Available: https://glassboxmedicine.com/2019/06/21/cnn-heat-maps-saliency-backpropagation/ (visited on 01/03/2021).

[99] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps", *2nd International Conference on Learning Representations, ICLR 2014 - Workshop Track Proceedings*, pp. 1–8, 2014. arXiv: 1312.6034.

[100] W. Xu, J. Wang, Y. Wang, G. Xu, D. Lin, W. Dai, and Y. Wu, "Where is the Model Looking At? - Concentrate and Explain the Network Attention", *IEEE Journal on Selected Topics in Signal Processing*, vol. 14, no. 3, pp. 506–516, 2020. arXiv: 2009.13862.

[101] Q. shi Zhang and S. chun Zhu, "Visual interpretability for deep learning: a survey", *Frontiers of Information Technology and Electronic Engineering*, vol. 19, no. 1, pp. 27–39, 2018. arXiv: 1802.00614.

[102] *GitHub: Convolutional Neural Network Visualizations*. [Online]. Available: https://github.com/utkuozbulak/pytorch-cnn-visualizations (visited on 01/03/2021).

[103] A. Barredo Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. Garcia, S. Gil-Lopez, D. Molina, R. Benjamins, R. Chatila, and F. Herrera, "Explainable Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI", *Information Fusion*, vol. 58, pp. 82–115, 2020. arXiv: arXiv:1910.10045v2.

[104] *Keras: Conv2D layer*. [Online]. Available: https://keras.io/api/layers/convolution_layers/convolution2d/ (visited on 01/04/2021).

[105] *Keras: MaxPooling2D layer*. [Online]. Available: https://keras.io/api/layers/pooling_layers/max_pooling2d/ (visited on 01/04/2021).

[106] *SDG: Take Action for the Sustainable Development Goals*. [Online]. Available: https://www.un.org/sustainabledevelopment/sustainable-development-goals/ (visited on 11/09/2020).

[107] *European Commission: Artificial Intelligence*. [Online]. Available: https://ec.europa.eu/digital-single-market/en/artificial-intelligence.

[108] *Top 7 Most Polluting Industries*. [Online]. Available: https://www.theecoexperts.co.uk/blog/top-7-most-polluting-industries#link-technology (visited on 11/10/2020).

[109] *World Economic Forum: The world's e-waste is a huge problem. It's also a golden opportunity*. [Online]. Available: https://www.weforum.org/agenda/2019/01/how-a-circular-approach-can-turn-e-waste-into-a-golden-opportunity/ (visited on 11/10/2020).