ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

OFFICIAL MASTER'S DEGREE IN BIG DATA

MASTER'S THESIS

DESIGN OF SMALL CNN VIA SYNTHETIC IMAGES AND FILTER EXTRACTION

Author: Irene España Novillo

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

MADRID
May 2025

This disertation was typeset with LaTeX and compiled in TeXstudio using the TeXLive 2019 distribution. The font families used are Bitstream Charter, Utopia, Bookman and Computer Modern. ChatGPT® was used to improve the writing. Unless otherwise noted, all figures were created by the author using Microsoft PowerPoint® and Python®.

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título "Design of Small CNN via synthetic images and filter extraction" en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2024/2025 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.

ESPAÑA NOVILLO IRENE - 06287687Q

Firmado digitalmente por ESPAÑA NOVILLO IRENE - 06287687Q
Fecha: 2025.05.06 12:32:49 +02'00'

Fdo.: Irene España Novillo Fecha:

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Eugenio Francisco Sánchez Úbeda

Firmado digitalmente por Eugenio Francisco Sánchez Úbeda
Fecha: 2025.05.06 10:06:57 +02'00'

Fdo.: Eugenio Francisco Sánchez Úbeda Fecha:

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Firmado digitalmente por BOAL MARTIN LARRAURI JAIME - 05304600H
Fecha: 2025.05.06 09:45:47 +02'00'

Fdo.: Jaime Boal Martín-Larrauri Fecha:

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

OFFICIAL MASTER'S DEGREE IN BIG DATA

MASTER'S THESIS

DESIGN OF SMALL CNN VIA SYNTHETIC IMAGES AND FILTER EXTRACTION

Author: Irene España Novillo

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

MADRID
May 2025

This disertation was typeset with LaTeX and compiled in TeXstudio using the TeXLive 2019 distribution. The font families used are Bitstream Charter, Utopia, Bookman and Computer Modern. ChatGPT® was used to improve the writing. Unless otherwise noted, all figures were created by the author using Microsoft PowerPoint® and Python®.

DISEÑO DE CNN COMPACTAS MEDIANTE IMÁGENES SINTÉTICAS Y EXTRACIÓN DE FILTROS

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

RESUMEN DEL PROYECTO

A pesar de su éxito en los últimos años, las Redes Neuronales Convolucionales (CNN) presentan importantes desafíos, entre ellos el elevado coste computacional, los largos tiempos de entrenamiento y una fuerte dependencia de grandes conjuntos de datos. Este proyecto presenta una metodología "bottom-up" para diseñar redes neuronales convolucionales eficientes utilizando conjuntos de imágenes sintéticas de figuras geométricas. Al aplicar transformaciones básicas —escalado, rotación y traslación— a un dataset sintéticamente generado, es posible extraer filtros especializados e identificar configuraciones clave de hiperparámetros. Este conocimiento permite una inicialización inteligente de los modelos, reduciendo tanto el coste computacional como el tiempo de entrenamiento, sin sacrificar precisión. El uso de técnicas de "transfer learning" aplicadas a filtros mejora el rendimiento y la estabilidad del modelo en escenarios más complejos. El enfoque propuesto es generalizable y especialmente adecuado para entornos con recursos limitados, contribuyendo al desarrollo de sistemas de visión artificial más interpretables, sostenibles y escalables.

Palabras clave: CNN, modelo mínimo, filtro, transfer learning, arquitecturas pequeñas.

1. Introducción

En los últimos años, las redes neuronales convolucionales (CNN) han transformado las aplicaciones de visión artificial en múltiples campos, desde diagnósticos médicos hasta control de calidad industrial. Sin embargo, su efectividad suele venir acompañada de altos costes computacionales, extensos tiempos de entrenamiento y la necesidad de grandes conjuntos de datos etiquetados. Además, su naturaleza opaca tipo "caja negra" representa un reto en dominios críticos donde la interpretabilidad es esencial. Para abordar dichas limitaciones, este trabajo propone una metodología novedosa y ascendente para diseñar arquitecturas de CNN compactas, eficientes e interpretables. Aprovechando conjuntos de datos sintéticos diseñados específicamente para tareas concretas, el enfoque busca construir modelos más transparentes y sostenibles sin comprometer el rendimiento.

2. Definición del proyecto

Este trabajo presenta el diseño e implementación de una metodología para desarrollar arquitecturas de CNN compactas capaces de alcanzar alta precisión con un número mínimo de parámetros. Utilizando un conjunto de datos sintéticos de figuras geométricas simples, se entrenan modelos de baja complejidad para extraer una base

de conocimiento compuesta de órdenes de magnitud de hiperparámetros, arquitecturas de modelos y filtros aprendidos. Este conocimiento permite la inicialización inteligente de nuevas CNN, mejorando la eficiencia desde el inicio. Además, se aplican técnicas de aprendizaje por transferencia para abordar nuevos problemas de clasificación, reduciendo considerablemente el tiempo de entrenamiento y el coste computacional en comparación con modelos inicializados aleatoriamente. Las tareas abordadas implican la identificación de figuras geométricas sometidas a escalado, rotación, traslación y todas las posibles combinaciones de estas transformaciones básicas en presencia de ruido.

3. Descripción de la metodología

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

- **Generación del conjunto de datos**. Se genera un conjunto sintético de imágenes de figuras geométricas pertenecientes a tres clases: elipses, rectángulos y triángulos. Se aplican transformaciones como escalado (Dataset 1.1), rotación (1.2), traslación (1.3), o combinaciones de estas (Datasets 4.1 a 7.1). También se añade ruido.
- **Búsqueda del modelo mínimo**. Se entrenan múltiples modelos con distintas combinaciones de hiperparámetros mediante una búsqueda en malla (“grid search”) identificando modelos de baja complejidad con alta precisión.
- **Extración de filtros óptimos**. A partir de los modelos mínimos seleccionados, se construye la base de conocimiento compuesta por filtros, arquitecturas de modelos y órdenes de magnitud de hiperparámetros óptimas.
- **Transfer learning de filtros**. Se reutilizan los filtros aprendidos en datasets con combinaciones de transformaciones (4.1 a 7.1) y se comparan con modelos entrenados desde cero.
- **Conclusión**. Se extraen las observaciones más relevantes de la comparación entre ambas metodologías.

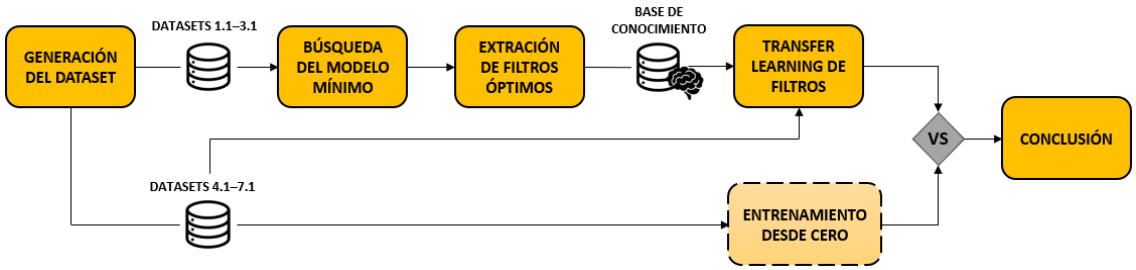


Figura 1. Diagrama de flujo de la metodología.

4. Resultados

- Los modelos mínimos seleccionados para los Datasets 1.1, 1.2 y 1.3 alcanzan un 99% de precisión en validación utilizando solo 21,000 parámetros, demostrando la efectividad de los filtros aprendidos. El hiperparámetro más determinante es el

tamaño del pooling, fijado en 3 en todos los casos. Cada filtro extraído está especializado en detectar una transformación geométrica concreta.

- Los modelos entrenados con "transfer learning" poseen mayor estabilidad que los entrenados desde cero, alcanzando niveles de precisión similares con diferentes semillas aleatorias.
- En cuanto a las combinaciones de dos transformaciones, la más sencilla de resolver es la de escalado más traslación, con la que el modelo alcanza un 93% de precisión utilizando solo 18,000 parámetros. La combinación de escalado y rotación presenta una dificultad intermedia, requiriendo 51,000 parámetros para lograr un 89% de precisión. Finalmente, la combinación de rotación y traslación es la más compleja, ya que el modelo necesita 83,000 parámetros para alcanzar apenas un 77% de precisión, evidenciando el mayor desafío que supone esta transformación compuesta.
- La combinación de las tres transformaciones —escalado, rotación y traslación— es el escenario más complejo. A pesar de tener solo 26,000 parámetros, la precisión desciende a aproximadamente un 68%.
- La rotación parece ser la transformación más difícil de detectar para las redes.
- El entrenamiento con "transfer learning" tarda, en promedio, 3 minutos y 34 segundos más que el entrenamiento desde cero, aunque esta diferencia es mínima en contextos de visión por computadora, donde los entrenamientos pueden durar horas o incluso días.

5. Conclusiones

Este proyecto presenta una novedosa metodología para el diseño de redes neuronales convolucionales ligeras, eficientes e interpretables enfocadas en tareas específicas con transformaciones geométricas. Comenzando con modelos mínimos entrenados en datasets sintéticos con transformaciones como escalado, rotación y traslación, se construye una base de conocimiento con hiperparámetros y filtros óptimos. Posteriormente, mediante aprendizaje por transferencia, se generalizan estos modelos para enfrentar escenarios más complejos, mejorando la estabilidad y el rendimiento al tiempo que se reducen los tiempos y costes de entrenamiento. A pesar de la mayor dificultad que supone combinar varias transformaciones, el método propuesto ofrece precisiones competitivas con arquitecturas mucho más pequeñas que las tradicionales.

DESIGN OF SMALL CNN VIA SYNTHETIC IMAGES AND FILTER EXTRACTION

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

ABSTRACT

Despite their success during the last years, CNNs present significant challenges, including high computational requirements, extensive training times and a dependency on large datasets. This thesis presents a bottom-up methodology for designing compact and efficient Convolutional Neural Networks (CNNs) using synthetic image datasets of geometric shapes. By applying basic transformations—scaling, rotation and translation—to synthetic-generated data, it becomes possible to extract specialized filters and identify key hyperparameter configurations. This knowledge enables intelligent model initialization, reducing both computational cost and training time while maintaining high accuracy. The use of filter-transfer learning further improves model performance and stability in more complex scenarios. The proposed approach is generalizable and well-suited for resource-constrained environments, contributing to the development of more interpretable, sustainable, and scalable computer vision systems.

Keywords: CNN, minimum model, filter, transfer learning, small architectures.

1. Introduction

In recent years, Convolutional Neural Networks (CNNs) have transformed computer vision applications across a wide range of fields, from medical diagnostics to industrial quality control. However, their effectiveness often comes at the cost of high computational demands, long training times and the need for large, labeled datasets. Moreover, their opaque "black-box" nature poses challenges in critical domains where model interpretability is essential. To address these limitations, this thesis proposes a novel bottom-up methodology for designing compact, efficient, and interpretable CNN architectures. By leveraging synthetically generated image datasets tailored to specific tasks, the approach aims to create more transparent and sustainable models without compromising performance.

2. Project definition

This work presents the design and implementation of a methodology for developing compact CNN architectures capable of achieving high accuracy with a minimal number of parameters. Using a synthetically generated dataset of simple geometric images, a series of low-complexity models are trained to extract a knowledge base consisting of the order of magnitude of the hyperparameters, architectural structures, and learned filters. This knowledge enables to intelligently initialize new CNNs, improving efficiency from the outset. Transfer learning techniques are then applied

to tackle new classification problems, significantly reducing training time and computational cost compared to randomly initialized models. The tasks addressed involve identifying geometric shapes subjected to scaling, rotation, translation and all possible combinations of these basic transformations in the presence of noise.

3. Methodology description

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

- **Dataset generation**. A synthetic dataset is generated. Both training and validation datasets contain images of geometric figures belonging to three different classes: ellipses, rectangles and triangles. A transformation is applied to the images: scaling (Dataset 1.1), rotation (Dataset 1.2) and translation (Dataset 1.3) or a combination of them (Datasets 4.1 to 7.1). Noise is added to the images as well.
- **Minimum model search**. Once the dataset has been built, a series of architectures are trained to address basic transformations (Dataset 1.1 to 1.3) varying the values of the hyperparameters following a grid-search method. Models that achieve the highest levels of accuracy with the minimum number of parameters and hence, the lower complexity, are selected.
- **Optimal filter extraction**. From the minimal models previously chosen the knowledge base of hyperparameters magnitude order, model architecture and filters to address basic transformations is obtained.
- **Filter-transfer learning**. With the filters obtained from the previous stage, transfer learning is applied to datasets containing combinations of transformations (Datasets 4.1 to 7.1). Additionally, the same architectures are trained from scratch, so both techniques can be compared.
- **Conclusion**. The most relevant aspects of the comparison between methodologies are drawn.

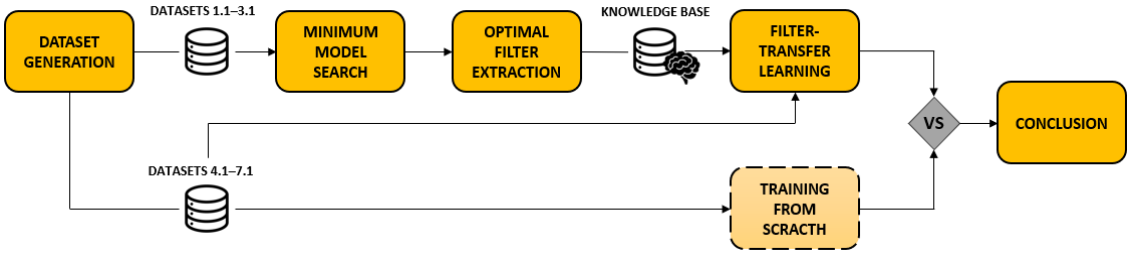


Figure 1. Workflow chart of the methodology.

4. Results

- The minimum models selected for Datasets 1.1, 1.2 and 1.3 achieves a validation accuracy of 99% with only 21,000 parameters, demonstrating the effectiveness of the learned filters and their suitability for forming a robust knowledge base. Among the hyperparameters, pooling size was the most influential, consistently set to 3 across all cases. Each of the three extracted filters shows specialization in detecting a specific geometric transformation (scaling, rotation, or translation).

- Models trained using transfer learning demonstrate greater stability than those trained from scratch, consistently achieving similar accuracy levels across different random seeds. This suggests improved generalization and reduced variance in performance.
- When analyzing combinations of two transformations, the pairing of scaling and translation is the simplest to resolve, with the model achieving 93% accuracy using just 18,000 parameters. The combination of scaling and rotation presents moderate complexity, requiring 51,000 parameters to reach 89% accuracy. The most challenging two-transformation scenario is rotation and translation, which demands 83,000 parameters to obtain a reduced accuracy of 77%
- The scenario involving all three transformations —scaling, rotation, and translation— proves to be the most difficult one to solve. Despite the model's relatively low parameter count (26,000), it achieves only around 68% accuracy, indicating the increased complexity introduced by the combined transformations.
- Rotation seems to be the most challenging transformation for networks to detect.
- On average, training using transfer learning takes 3 minutes and 34 seconds longer than training from scratch. However, this difference is acceptable in the broader context of computer vision tasks, where training durations often extend to several hours or even days.

5. Conclusions

This thesis presents a novel bottom-up methodology for designing lightweight, efficient and interpretable Convolutional Neural Networks (CNNs) tailored to specific tasks involving geometric image transformations. By starting with minimal models trained on synthetically generated datasets featuring transformations like scaling, rotation, and translation, the approach builds a knowledge base of optimal hyperparameters and filters. Transfer learning is then used to extend these models to more complex transformation scenarios, improving stability and performance while reducing training time and computational costs. Despite the increased complexity of combining multiple transformations, the proposed method offers competitive accuracy with significantly smaller architectures compared to traditional approaches.

A mis abuelas Riánsares y Sagrario, por cuidarme siempre.

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

Somos lo que hacemos repetidamente.
La excelencia no es un acto, es un hábito.
Aristóteles (384 a.C.–322 a.C.)

Agradecimientos

La realización de este trabajo final de máster no habría sido posible sin el apoyo, la guía y el impulso constante de mis directores, Eugenio y Jaime. Cuando os propuse de nuevo dirigirme un TFM no lo dudasteis ni un segundo. Gracias por vuestra confianza y por apostar por este proyecto con entusiasmo y compromiso.

A mis padres y a mi hermano, gracias por animarme a seguir formándome y brindarme la oportunidad de hacerlo en ICAI. No puedo estar más orgullosa de tener como referentes a personas tan generosas, tenaces y llenas de amor. A mis tíos, tías y primos, por su cariño incondicional, por mantenernos siempre unidos y por creer firmemente en mi capacidad para alcanzar mis metas.

A mis compañeros del departamento de GIRS, gracias por ser fuente de inspiración diaria. Vuestra dedicación y profesionalidad han sido ejemplo y motivación constante a lo largo de este trabajo. Bilbao y Madrid no serían lo mismo sin vosotros.

A mis amigas de toda la vida, gracias por creer en mí incluso cuando yo dudaba. Vuestra amistad es un regalo inmenso. Y a Pablo, este trabajo te debe mucho. Yo también.

A todas las personas que, aunque no estén mencionadas directamente, han aportado su granito de arena en este camino: ¡gracias de corazón!

Contents

List of Figures

List of Tables

1

Introduction

You only get so many firsts,
each one is a blessing.
Taylor Swift (1989–)

The first chapter outlines the rationale for this project, its primary objective and the tools employed during its development. Additionally, it offers the reader a clear overview of the dissertation's structure to facilitate easier navigation and understanding.

In recent years, Big Data has become a cornerstone of technological advancement, enabling innovation across different industry fields. Among its most impactful applications is the use of advanced techniques applied to computer vision tasks. Industrial computer vision systems are able to perform precise product inspections in real time on high-speed lines, correcting issues and improving product quality at the same time that operational costs are reduced.

The most representative technique to analyze images par excellence are Convolutional Neural Networks (CNNs), a specialized type of neural networks with a remarkable ability to detect and interpret complex visual patterns. These models have achieved groundbreaking results in fields such as medical imaging, where they assist in early disease detection, autonomous vehicles that rely on real-time object recognition, and even creative applications like generating art or enhancing image resolution.

However, despite their enormous potential, CNNs have certain limitations. One of the most critical challenges lies in their reliance on vast amounts of labeled data for training. Acquiring and annotating such datasets can be expensive and time-consuming, particularly in specialized fields like medicine or satellite imaging. Additionally, CNNs are computationally intensive, requiring powerful hardware like GPUs to train effectively.

Another major drawback is the time required for training. As the complexity of the network grows due to deeper architectures or larger datasets, training times can stretch from hours to weeks. Even after deployment, CNNs can struggle with scalability in real-world applications where new data is continuously introduced and retraining or fine-tuning is needed. Furthermore, these models lack interpretability; their "black-box" nature makes it difficult to understand or

explain why a particular prediction was made, posing challenges in high-stakes applications such as those in healthcare, pharmaceuticals, food or legal industries, where decisions can have a direct impact on human life. In order to address these limitations, the field continues to innovate with well-known techniques such as transfer learning and model compression.

There is growing interest in developing small, lightweight CNN models that are easier to interpret. Unlike large-scale CNNs, which require vast computational resources and extensive datasets, smaller models are designed to be lightweight and efficient, making them ideal for deployment on edge devices like smartphones, IoT devices and embedded systems. This trend reflects a shift towards resource-constrained environments where energy efficiency, faster inference times and reduced computational resource requirements are critical while maintaining high accuracy rates.

Moreover, the rise of edge computing has further amplified the need for compact CNN architectures. As more devices operate outside traditional data centers, the ability to process data locally without relying on cloud-based resources has become crucial. This not only reduces latency but also enhances privacy and security by minimizing data transmission. Lightweight CNNs enable these devices to handle complex tasks, such as image recognition or object detection, autonomously, paving the way for innovations in fields like smart cities, wearable technology and autonomous drones.

Another critical consideration is sustainability. The environmental impact of training and deploying large-scale machine learning models has come under scrutiny, as the energy consumption associated with these processes is significant. Lightweight CNNs align with the broader movement toward green AI [1], aiming to create models that deliver strong performance while minimizing their ecological footprint. This balance between efficiency and effectiveness represents a vital step in ensuring that the benefits of artificial intelligence can be achieved without aggravating environmental challenges.

The present dissertation introduces a methodology to design small CNN architectures tailored to specific problems, aiming to develop more explainable and sustainable models. By addressing the challenges of resource constraints, interpretability and sustainability, this work seeks to contribute to the evolution of computer vision systems that are not only powerful but also practical and ethically aligned with the needs of modern society.

1.1. Motivation

There has been a tendency over the last decade towards building increasingly large and complex CNNs, mainly driven by the pursuit of beating state-of-the-art performance in challenging tasks such as image recognition. These models, exemplified by architectures like ResNet or GoogleNet, contain deep layers and millions of parameters. As part of the the industrial change presented by the Industry 4.0, the demand for lightweight and more efficient CNNs is increasing. In order to meet this demand, the approach usually employed is a *top-down* one where, starting from these large architectures, their complexity is reduced with techniques like pruning. This results in networks that are still large and inefficient, where high rates of accuracy are achieved but also at a very high computational and time cost.

Building smaller architectures from a known and controlled point, following a *bottom-up* methodology, seems to be a valid and different approach to address the problem presented. This process enables to progressively add complexity to the model at succesive stages until the

desired accuracy is reached, resulting in a model with only the necessary parameters to perform the task addressed.

1.2. Objective

This thesis presents a methodology for designing CNN architectures of small size that are able to reach high levels of accuracy with fewer parameters. The solution is based on builiding the model that is capable of solving the classification problem with the minimum number of parameters. A synthetic image dataset of geometric shapes is generated specifically for this task, so that the study is carried on in a controlled environment, easing the interpretation of the CNN learning process and without dealing with an enourmous amount of images. The basic geometric transformations (scaling, rotation and traslation) are applied to the shapes, so a filter database can be obtained from the models trained, associating each filter to a specific geometric transformation.

The aim is, therefore, to obtain a knowledge base of the order of magnitude of the hyperparameters, model structure and filters; that enables to intelligently initialize a CNN architecture, reducing time and computational costs compared to an architecture randomly initialized.

In this way, when a new problem presents the same geometric transformations or a combination of them, a baseline is established for building the initial architecture and its complexity can be increased with the sucessive training until the desired level of accuracy is reached.

1.3. Resources

The research is primarily conducted using Python 3 [2], a programming language widely adopted for Machine Learning applications due to its extensive library support and user-friendly nature. Python facilitates rapid prototyping as well as seamless application deployment. For deep learning tasks, the Keras API is utilized with TensorFlow 2 as the backend [3], leveraging the `tf.keras` module.

The experiments are implemented on a computer equipped with GPUs, significantly accelerating the training process of Convolutional Neural Networks (CNNs). To gain deeper insights into the experimental results, tools like GIMP are used to analyze the network's behavior at the pixel level on input images.

1.4. Dissertation outline

The dissertation is structured into six chapters. This first chapter introduces the research problem and outlines the context of the project. Chapter 2 presents the literature review, covering key approaches related to the optimization of CNN models. Chapter 3 provides a detailed explanation of the synthetic dataset generation process, including its main characteristics. Chapter 4 describes the methodology used to extract a knowledge base consisting of hyperparameter magnitude orders, model architectures, and, most notably, optimal filters derived from minimal models. Chapter 5 builds upon this knowledge base, using the extracted filters to initialize new models that aim to improve both training efficiency and performance. Chapter 6 concludes the dissertation by summarizing the main findings and proposing future

directions for enhancing the presented approach. Additionally, Appendix A outlines the project's contributions to the Sustainable Development Goals (SDGs).

2

Literature Review

If I have seen further,
it is by standing on the shoulders of giants.
Isaac Newton (1643–1727)

Selecting the optimal CNN architecture for a given problem has raised attention among researchers in recent years. This chapter provides a comprehensive review of related work in this area, analyzing the various techniques employed and highlighting their differences from the approach proposed in this thesis.

This section offers an overview of the current state of the art regarding existing related work on CNN optimal architecture selection. Table 2.1 summarises the techniques used by each one of the articles referenced in this section and it can be observed which techniques are the most popular.

There are several techniques that have been applied in order to find the optimal CNN architecture when addressing classification problems. [4] presents a small CNN architecture trained with the CIFAR-10 dataset that reaches high levels of accuracy with fewer parameters. Two techniques are used: the growing approach and pruning. For the first one, the process starts with a minimal architecture and new layers are inserted as needed while evaluating the performance of the architecture. For the second one, a big complex network is proposed and pruning is performed to obtain a smaller and more efficient network. Pruning is also used by [5].

Another set of approaches to optimize CNN architectures rely on the Gradient-Based methodology, such as the ones used on [5] and [6]. [5] introduces a novel framework to efficiently search for optimal CNN architectures called Greedy and Progressive Architecture Search (GPAS) which uses a gradient-based bilevel optimization technique to search for optimal architectures in a continuous space.

Genetic algorithms are also found among the popular techniques to design optimal CNN architectures. [7] develops a new genetic algorithm to design CNN architectures automatically, without the need of expertise neither on CNNs nor on the problem domain. [8] uses GA as well to design CNN architectures automatically.

Table 2.1. Techniques used for CNN optimal architecture selection. *appr.* is the abbreviation for approach and *arch.* is the abbreviation for architecture.

REFERENCES	**TECHNIQUES**						
	Growing appr.	Pruning	Gradient-based	Genetic algorithms	*Well-known* arch.	*New* arch.	Small data
Springenberg et al, 2015 **[9]**						✓	
Keshari et al, 2018 **[10]**							✓
Ferreyra-Ramirez et al, 2019 **[11]**						✓	
Xu et al, 2019 **[12]**							✓
Liu et al, 2019 **[6]**			✓				
Sun et al, Apr. 2020 **[7]**				✓	✓		
Sun et al, Sep. 2020 **[8]**				✓			
Dhouibi et al, 2021 **[4]**	✓	✓					
Peng et al, 2021 **[5]**		✓	✓				
Foroughi et al, 2021 **[13]**							✓
Savinov et al, 2022 **[14]**					✓		
Ang et al, 2022 **[15]**						✓	
Berdos et al, 2022 **[16]**					✓		
Faris Al Hakim et al, 2023 **[17]**							✓
Mesárošová et al, 2024 **[18]**							✓
TOTAL	**1**	**2**	**2**	**2**	**3**	**3**	**5**

Another technique for designing optimal architectures focuses on taking a well-known CNN model as base architecture. The algorithm desing by [7] uses ResNet and DenseNet blocks as building elements for the CNN architecture and introduces a variable-length encoding scheme genetic algorithm to optimize CNN depth. Once the architectures are represented through ResNet Blocks (RBU), DenseNet Blocks (DBU) and pooling layers, genetic operators as crossover and mutation are applied and the fitness is evaluated based on accuracy on validation data after training. [14] uses the Bayesian optimization method to select the optimal model parameters to achieve maximum accuracy. Once the hyperparameters (kernels number, size and step for the pooling layers) are selected, they are applied to a well-known CNN architecture, based on the EGGNet model. [16] trains different well-known architectures in order to identify the most effective one for Speech Emotion Recognition (SER) problem. The 2D CNN model outperformed the others, using between 40 and 30 epochs per dataset evaluated.

Instead of using well-known architectures, several references propose search methods to achieve optimal architectures when building from scratch. [15] proposes a new technique based on Teaching-Learning-Based Optimization (TLBO) to obtain an optimal CNN architecture design automatically, defining a specific encoding to represent the CNN network architecture, which will be the “learners". The optimal network architectures consist of a single fully connected layer, indicating that, in some cases, a one-layer architecture may produce better results than architectures with multiple fully connected layers. In addition, for certain image datasets it is not always necessary to instert a pooling layer between two convolutional layers. [11] builds a new CNN architecture called ACEnet with the aim of improving CNN performance while addressing overfitting. Its performance is compared against AlexNet architecture, increasing the accuracy by 5.11%. The key points of this architecture are: small kernels (reduces the number of training parameters), pooling with overlap (preserves spatial information), dropout layers (minimizes overfitting) and ReLU activation. [9] proposes a new CNN architecture called

“all-convolutional" that replaces max-pooling layers by convolutional layers with increased stride and fully connected layers by 1x1 convolutional layers combined with global averaging, enabling dimensionality reduction without performance loss. Small convolutional layers (e.g., 3x3) stacked in depth are sufficient to achieve high accuracy on small images. This network manages to either match or outperform state-of-the-art results on CIFAR-10, CIFAR-100 and ImageNet datasets.

The common ground to all the methodologies previously presented is their focus on finding solutions to classification problems with big datasets, (for example, CIFAR-10 dataset consists of 60000 colour images of size 32x32, 50000 of them are training images and 10000 are test images). This fact makes CNN resulting architectures complex by nature.

Over the last few years, several works have studied the possibility of applying CNN to small datasets in order to build robust models when there is not enough input data and also with the aim to gain control over the architectural-design of the CNN as well as the training process. The "Small Data" approach manages also to obtain optimized models that contain the minimal number of parameters.

A new architecture called SSF-CNN (Structure and Strenght filtered CNN), it is proposed in [10], which optimizes CNN filters specifically for small datasets. In order to do that, initializes filters using a dictionary-based learning algorithm to encode representative features and afterwards, learns a scalar parameter for each filter to adjust their influence, reducing the number of learnable parameters and mitigating overfitting.

In 2021, [13] introduce a novel loss function to improve the generalization capability of CNNs trained on small datasets based on cross-entropy loss. For a dataset containing 534 images, proposes a shallow CNN architecture with four convolutional layers, three max-pooling layers and two fully connected layers with a softmax for the classification part. This kind of architecture is also chosen by [17], which presents two architectures both with four convolutional plus pooling layers, one flatten layer and a dense output layer, over a dataset containing 765 images. Both of the references use data augmentation techniques, as well as [12], which introduces a semi-supervised data augmentation (SSDA) method to extend the dataset in a more efficient and controlled way.

Finally, [18] manages to solve a classification problem with a CNN that only contains 1700 learnable parameters (a much smaller architecture compared to the 391491 parameter architecture presented by [17]) composed by only two convolutional layers with hyperbolic tangent plus a max pooling layer and a fully connected layer with softmax function for the classification part.

3

Synthetic dataset generation

Without data,
you're just another person with an opinion.
William Edwards Deming (1900–1993)

This chapter provides a detailed description of the dataset synthetically generated to achieve the objective of this project. It includes both the process followed to create the images and an analysis of the resulting samples.

The basis to create any Machine Learning model is the data, as the algorithm is fed with it and learns from its patterns and features. The quality of the data is critical because noisy, inconsistent, or incomplete data can lead to inaccurate predictions and poor model performance. Clean data ensures that the algorithm focuses on meaningful information rather than irrelevant noise, which significantly impacts the accuracy and reliability of the model. Moreover, properly labeled and representative datasets are essential to avoid bias and generalize well to unseen scenarios. In essence, high-quality data is the foundation for building robust and effective machine learning models.

The complexity of the tasks performed by the model and, consequently, the complexity of the model itself, depends on the data characteristics mentioned earlier: quality, accuracy, completeness, etc. To address the problem presented in this thesis, a dataset has been synthetically generated, allowing its complexity—and therefore the learning process—to be controlled. The following sections describe the dataset generation process and present a series of sample images to help the reader visualize the data used to train the models.

3.1. Dataset generation

The dataset contains labelled images belonging to one out of three classes, depending on the geometric shape that is depicted in each one of them: ellipse, triangle or rectangle. Moreover, a geometric transformation (or a combination of them) is applied to the shapes: scaling, rotation or translation.

Images are generated using Python [2] as being one of the most-widely used language programming for image processing, mainly due to its extensive open-source libraries. The complete set of libraries used to generate the dataset is:

- **os**: Provides a way to interact with the operating system, including file and directory manipulation, environment variable access and execution of system commands [19].
- **random**: Used for generating random numbers, selecting random items from lists, and performing other randomization tasks such as shuffling [20].
- **math**: Offers a variety of mathematical functions like trigonometric, logarithmic and exponential functions, as well as constants like π and e [21].
- **operator**: Contains efficient functions for standard operations (e.g. addition, subtraction and comparison) that can be used to improve readability and performance in certain cases [22].
- **NumPy**: A powerful library for numerical computations, supporting multi-dimensional arrays, mathematical operations, linear algebra, Fourier transforms and more mathematical operations. It is widely used in scientific computing and machine learning [23].
- **cv2**: Part of the OpenCV library, it is used for computer vision tasks like image and video processing, feature detection or object recognition [24].
- **Matplotlib**: A popular library for data visualization, enabling the creation of static, interactive and animated plots, such as line graphs, bar charts, scatter plots and heatmaps[25].

3.1.1. Geometric shapes

Images that make up the dataset contain a single geometric shape, either an ellipse, a rectangle or a triangle. The process followed for building each type of figure is detailed below but there are certain features that are common to all figures, no matter of what kind.

First of all, the base size of all figures depends on a parameter called *proportion*, so it can be controlled independently of its geometric shape. This means that one of the dimensions of the figure will defined as the other one multiplied by the *proportion* parameter. It takes a value in the range (0,1]. In the case of the dataset generated for this project, its value has been set to 0.6.

In addition, all shapes are defined in terms of the circumference that circumscribes them, so the process for building them is also standarized. To build the three kinds of geometric shapes the basic parameters are: *center* (of the circumference that circumscribes the figure), *radius* (of the circumference that circumscribes the figure) and *proportion*.

Regarding the color, all images are in grayscale. For each class, half of the images contain a white shape over a black background and the other half, a black shape over a white background. The color in RGB scale is considered extra information that only adds complexity to the problem; hence it has not been considered.

3.1.1.1. Ellipses

According to the common points previously explained, Figure 3.1 shows how the ellipses are represented geometrically, where their mathematical expression will be defined as expressed in equations (3.1) and (3.2), being p the value assigned by the user to the *proportion*.

The function `mpatches.Ellipse` [26] from `matplotlib` library is used for creating the ellipses. This function takes as input the coordinates of the ellipse centre in format (x, y), the width (w); total length of horizontal axis or diameter as stated in (3.1), the height (h); total length of vertical axis as stated in (3.2) and the angle of rotation in degrees anti-clockwise. It is important to highlight that in the case of ellipses, the dimensions width and height are interchangeable to generate either vertical or horizontal ellipses.

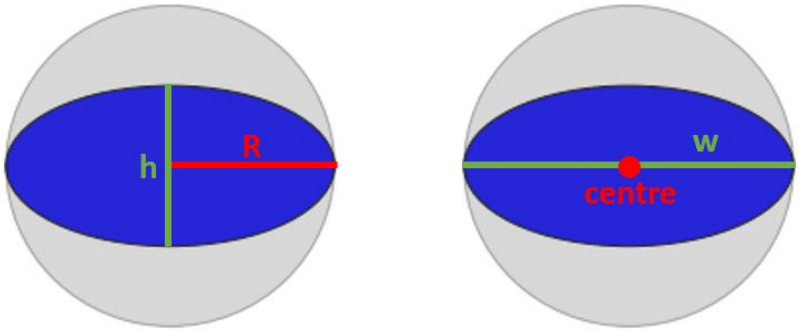


Figure 3.1. Ellipse geometrical definition.

$$w = 2 \times R \tag{3.1}$$

$$h = p \times w \tag{3.2}$$

3.1.1.2. Rectangles

Taking as input the same basic parameters: *center*, *radius* and *proportion*, Figure 3.2 depicts how the rectangles are represented geometrically based on the common parameters. The mathematical definition of height and width is expressed in equations (3.3) and (3.7) respectively. Equations from (3.4) to (3.6) show the application of the Pythagorean Theorem to represent the rectangle width based on the mentioned parameters.

The function `mpatches.Polygon` [27] from `matplotlib` library is used for creating the rectangles. With this function any polygon can be generated. It takes as input an array with the coordinates of the geometric shape vertexes in format (x, y). Having the mathematical representation for both width and height, the coordinates of each one of the four vertexes can be calculated as stated in equations from (3.8) to (3.11).

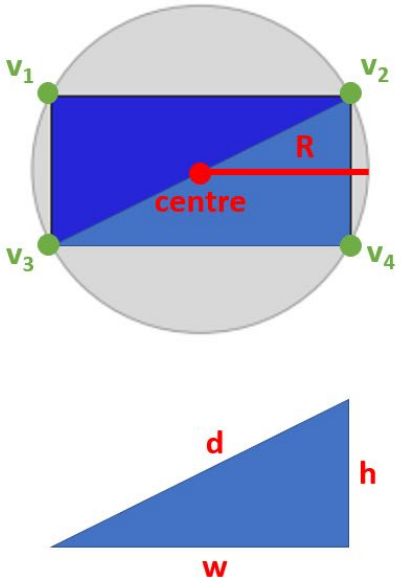


Figure 3.2. Rectangle geometrical definition based on the Pythagorean Theorem.

$$h = p \times w \tag{3.3}$$

$$d^2 = w^2 + h^2 \tag{3.4}$$

$$(2R)^2 = w^2 + (p \times w)^2 \tag{3.5}$$

$$(2R)^2 = (1 + p^2) \times w^2 \tag{3.6}$$

$$w = \frac{2R}{\sqrt{1 + p^2}} \tag{3.7}$$

$$v_1 = (centre[0] - \frac{w}{2}, centre[1] + \frac{h}{2}) \tag{3.8}$$

$$v_2 = (centre[0] + \frac{w}{2}, centre[1] + \frac{h}{2}) \tag{3.9}$$

$$v_3 = (centre[0] - \frac{w}{2}, centre[1] - \frac{h}{2}) \tag{3.10}$$

$$v_4 = (centre[0] + \frac{w}{2}, centre[1] - \frac{h}{2}) \tag{3.11}$$

3.1.1.3. Triangles

The process followed to build triangles is similar to the one stated before for both ellipses and rectagles. Figure 3.3 shows the geometrical definition of the triangles. The base of the triangle is represented in equation (3.12) as a function of the *proportion* and the height of the figure. Then, applying the Pythagorean Theorem, the height is represented as a function of the *radius* and *proportion* as equations from (3.13) to (3.18) show. Figure 3.3 shows the geometrical representation of triangles based on the common basic parameters.

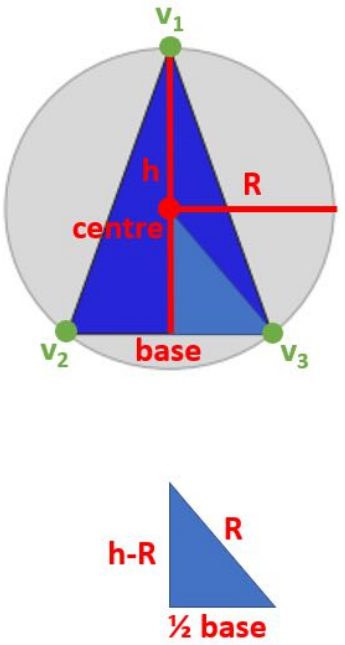


Figure 3.3. Triangle geometrical definition based on the Pythagorean Theorem.

$$base = p \times h \tag{3.12}$$

$$(h - R)^2 = R^2 - (\frac{1}{2}base)^2 \tag{3.13}$$

$$h^2 + R^2 - 2hR = R^2 - \frac{1}{4}base^2 \tag{3.14}$$

$$\frac{1}{4}base^2 = 2hR - h^2 \tag{3.15}$$

$$p^2h^2 = 8hR - 4h^2 \tag{3.16}$$

$$p^2h^2 + 4h^2 - 8hR = 0 \tag{3.17}$$

$$(4 + p^2)h^2 - 8hR = 0 \tag{3.18}$$

The solutions to the equation resulting in (3.18) can be either zero as stated in (3.19), which is not a valid solution, or the solution stated in (3.20), which represents the height of the triangle based on the radius of the circumference that circumscribes it and the *proportion* parameter.

$$\cancel{h = 0} \tag{3.19}$$

$$h = \frac{8R}{4 + p^2} \tag{3.20}$$

This way, the three vertexes of the triangle can be calculated as a function of the basic parameters: *center*, *radius* and *proportion* as it is stated in equations from (3.21) to (3.23). The function `mpatches.Polygon` [27] from `matplotlib` library is also used for generating the rectangles, so the coordinates of the vertexes are needed as input for the function.

$$v_1 = (centre[0], centre[1] + R) \tag{3.21}$$

$$v_2 = (centre[0] - \frac{base}{2}, centre[1] - [h - R]) \tag{3.22}$$

$$v_2 = (centre[0] + \frac{base}{2}, centre[1] - [h - R]) \tag{3.23}$$

3.1.2. Geometric transformations

A geometric transformation is an operation that moves or changes a shape in a geometric space while preserving certain properties. These transformations can be applied to points, lines and entire figures to alter their position, size or orientation. Once the three different kinds of geometric shapes have been generated, a single geometric transformation or a combination of them is applied to each one of the figures.

In the case of the dataset presented on this thesis, three different types of geometric transformations can be found: scaling, rotation and translation. As previously mentioned, the basic parameters to build any figure are *center*, *radius* and *proportion*. Additionaly, to apply each one of this transformations, some of these parameters are transformed and other ones are added as it is explained in the following sections.

It is important to highlight that, since the basic parameters have to change its value in order to perform the geometric transformations, a **seed** must be set before generating the figures. The seed enables to “freeze" the state of the random operation so the process can be repeated as many times as wanted obtaining the same result. Thus, each time the code is executed, the same figure is generated. It the seed is changed, the figures and transformations generated will change as well.

3.1.2.1. Scaling

Scaling is a transformation that changes the size of a geometric object. It is performed by multiplying the coordinates of the object by a scaling factor. If the scaling factor is greater than 1, the object enlarges; if it is between 0 and 1, the object shrinks.

In order to scale the figures, the basic parameter *radius* is replaced by two parameters: *maximum radius* and *minimum radius*. When both maximum and minimum radius have the same value, the shape does not change its size. However, if they are different, the size changes, resulting in a scaling transformation.

For the specific dataset generated in this thesis the scaling transformation takes 0.45 as *maximum radius* value and 0.20 as *minimum radius* value. The *center* coordinates, in this case are set to (0.5,0.5) so that the figures are placed on the center of the image. In the same way, the rotation angle is set to zero, so that the figure does not rotate.

3.1.2.2. Rotation

Rotation is a transformation that turns a geometric object around a fixed point, called the *center of rotation*. It is defined by an angle of rotation and a direction (clockwise or counterclockwise). The shape and size of the object remain unchanged, but its orientation changes.

In this case, the *center* is also set to (0.5,0.5) so the geometric shapes are centered and both the *maximum radius* and *minimum radius* take the same value, so that the size of the figures do not change. In addition, two more parameters are defined: *maximum rotation angle* and *minimum rotation angle*. When both parameters are equal to the same value, the figure does not rotate and when they have different values, the figure rotates around its center.

To apply this transformation in the present dataset, the *maximum rotation angle* is set to 360° and the *minimum rotation angle* is set to 0°. Therefore, the figures can perform a complete rotation over its center.

3.1.2.3. Translation

Translation is a transformation that moves every point of an object by the same distance in a specified direction. It does not alter the shape, size, or orientation of the object, only its position.

A new parameter called *delta* is defined to perform this transformation. Delta determines the distance that a figure can be moved from the center along both horizontal and vertical axis. In the case of this dataset the maximum value that delta can achieve is the center coordinate (in this case, 0.5) minus the *radius* minus 0.05 because there is a margin of this size so that figures do not exceed the margins of the image.

Once delta is calculated, using the random seed as well, it is applied to both coordinates of the figure center.

3.1.3. Noise

Adding noise to a dataset of synthetically generated images is crucial for training robust CNN models. Synthetic images often lack the natural imperfections found in real-world data, making models trained on them prone to poor generalization when exposed to real images. Introducing noise helps overcome this problem by simulating defects that can appear on images derivated from movement of sensor sensibility as well as lighting. This enhances the model's ability to handle diverse inputs, improving its resilience to distortions and preventing overfitting to overly clean synthetic patterns, making more adaptable CNN models in real-world applications.

For all the images presented in this dataset, two versions of each one of them is generated: with and without noise. A new fuction named `add_noise_ to_figures` generates Gaussian noise

and adds it to the images. The noise is generated with a normal distribution centered at 0 and with a standard deviation equal to the value of the parameter *noise level* which, in this case, is set to 100.

Gaussian noise is widely used in image processing and machine learning because it closely models real-world noise sources. In this case, the standard deviation is set to 100 (taking into account that luminance values go from 0 to 255 in 8 bits) as it has been considered as a break-even value, adding distortion to the image but not so much that the figures are unidentifiable.

3.1.4. Image reduction

Images are originally generated with a size of 224x224 since this is a standard size commonly used by popular architectures like ResNet [28] or VGG [29] because they were pre-trained on ImageNet, which uses this resolution [30]. Once the images are generated and noise has been added, their size is reduced to 28x28 pixels.

Images are reduced from 224x224 to 28x28 for two main reasons. First, this transformation better reflects real-world scenarios where images captured by cameras in industrial environments often have lower resolutions. By downsizing them, we simulate this use case while still preserving the essential details that were present when the images were originally created at 224x224. Second, generating realistic images at 28x28 is easier due to the smoothing effect of compression, which helps soften pixel values naturally making the images appear more coherent and visually plausible.

In order to reduce image size, the function `cv.resize` from OpenCV library which enables image resizing using bicubic interpolation `cv.INTER_CUBIC` which calculates new pixel values based on cubic interpolation of neighbour pixels. This generally produces smoother and more detailed images compared to other methods like linear interpolation (`cv.INTER_LINEAR`).

To sum up, the objective of image resizing is reducing the image size while preserving details.

3.2. Dataset analysis

As stated previously, each one of the images that make up the dataset contain a single geometric shape. The class with which the images are labelled corresponds to the geometric shape represented in it: *ellipse*, *rectangle* or *triangle*. Additionally, a geometric transformation is applied to these shapes. Due to this reason, the image dataset, in fact, is not only one but a group of datasets, each one of them containing all three kinds of shapes to which a particular transformation or a combination of them has been applied.

The following sections present an exhaustive analysis of the dataset synthetically generated, going into detail of how the dataset images have been organized as well as presenting a set of examples of images for all possible combination of figures, transformations and color combinations.

3.2.1. Dataset structure

The dataset presented in this thesis is a synthetic dataset of labeled images created to train small CNN models in a controlled environment. It was generated using the pipeline described in Section 3.1.

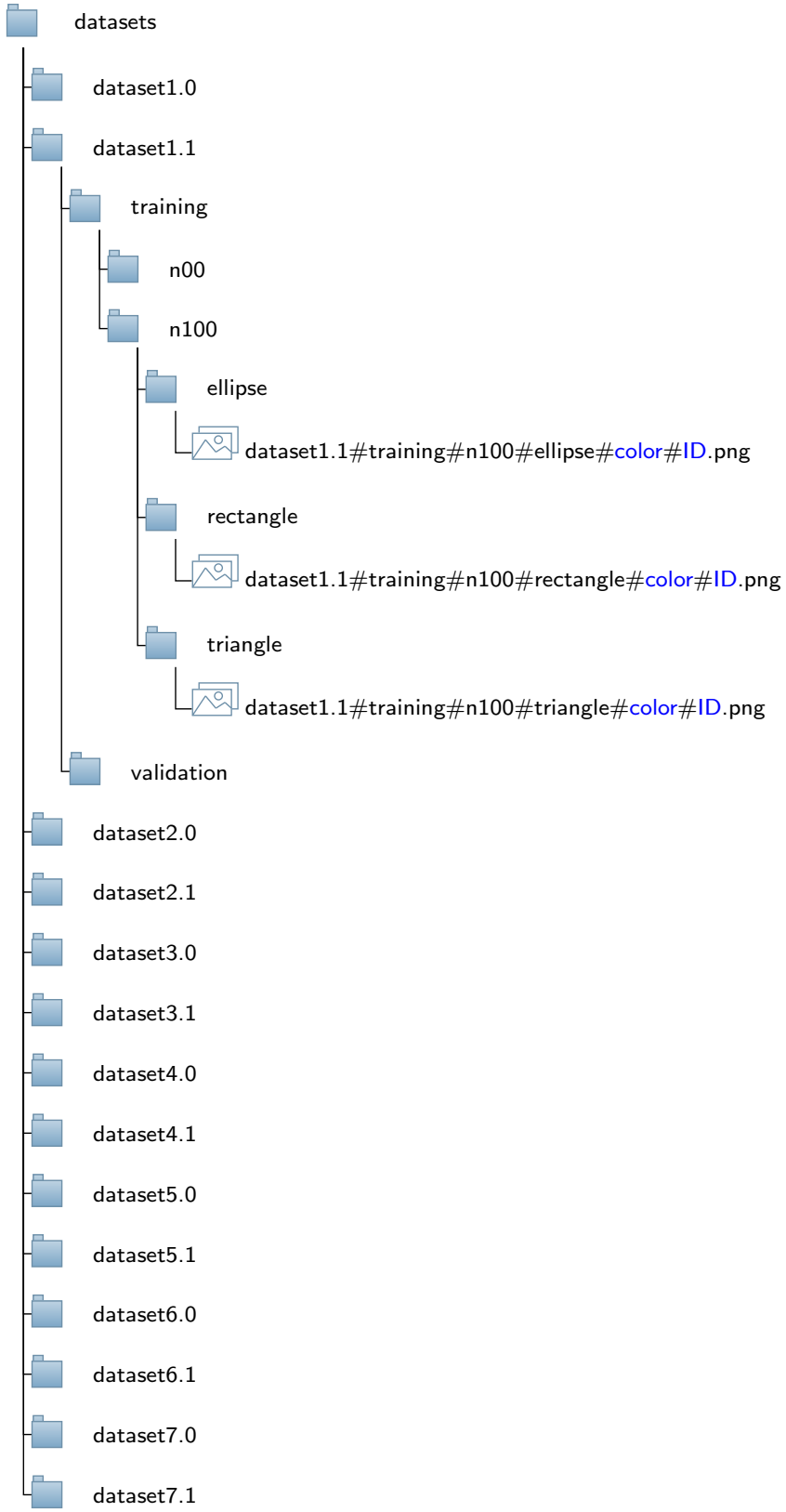


Figure 3.4. Dataset folder diagram. Parameters *color* and *ID* highlighted in blue since they change for each sample, *color* can take values “kw" or “wk" and *ID* is an integer.

The general dataset consists of 63k samples and is divided into seven datasets depending on the transformation applied to the images. Each one these datasets consist of 9k samples. Figure 3.4 shows the folder structure in which the dataset images are organized.

Each one of the datasets has its own folder and, since all have the same subfolder structure, Figure 3.4 only depicts the complete directory tree for dataset 1.1. Datasets 1.0, 2.0, 3.0, 4.0, 5.0, 6.0 and 7.0 are the ones containing images with the original size (224x224 pixels) whereas datasets 1.1, 2.1, 3.1, 4.1, 5.1, 6.1 and 7.1 contain the same samples but with the reduced size (28x28 pixels). Looking at the directory structure, it can be seen that first of all, samples are divided in into training set (with a total of 3k images) and validation set (consisting of 6k images. For both training and validation sets, same samples are generated with different levels of noise. In this way, folders *n00* (without noise) and *n100* (level 100 of noise) are found. Finally, the images are divided into separated directories according to the geometric figure they depict: ellipses (1k samples), rectangles (1k samples) and triangles (1k samples). For each class, half of the figures have the parameter *color* equal to “kw" which means that the image has a white figure over a black background or “wk" which means that the image has a black figure over a white backrground. The *ID* parameter is a sequential integer that identifies univocally the image inside its folder.

3.2.2. Dataset samples

This section presents a selection of images from each of the synthetically generated datasets, as previously described in Section 3.1 and organized according to the structure outlined in Section 3.2.1. This allows for an easy visual comparison across datasets. To provide a general overview, the following summary is included:

- **Dataset 1.0**: Contains images of size 224x224 with noise level 100 and scaling transformation applied. Sample images can be seen in Figures 3.5, 3.6 and 3.7.
- **Dataset 1.1**: Contains images resized to 28x28 with noise level 100 and scaling transformation applied. These images correspond to the same set used in Dataset 1.0, but at a reduced resolution. Sample images can be seen in Figures 3.8, 3.9, and 3.10.
- **Dataset 2.0**: Contains images of size 224x224 with noise level 100 and rotation transformation applied. Sample images can be seen in Figures 3.11, 3.12 and 3.13.
- **Dataset 2.1**: Contains images resized to 28x28 with noise level 100 and rotation transformation applied. These images correspond to the same set used in Dataset 2.0, but at a reduced resolution. Sample images can be seen in Figures 3.14, 3.15, and 3.16.
- **Dataset 3.0**: Contains images of size 224x224 with noise level 100 and translation transformation applied. Sample images can be seen in Figures 3.17, 3.18 and 3.19.
- **Dataset 3.1**: Contains images resized to 28x28 with noise level 100 and translation transformation applied. These images correspond to the same set used in Dataset 3.0, but at a reduced resolution. Sample images can be seen in Figures 3.20, 3.21, and 3.22.
- **Dataset 4.0**: Contains images of size 224x224 with noise level 100 and scaling plus rotation transformation applied. Sample images can be seen in Figures 3.23, 3.24 and 3.25.
- **Dataset 4.1**: Contains images resized to 28x28 with noise level 100 and scaling plus rotation transformation applied. These images correspond to the same set used in Dataset

4.0, but at a reduced resolution. Sample images can be seen in Figures 3.26, 3.27, and 3.28.

- **Dataset 5.0**: Contains images of size 224x224 with noise level 100 and scaling plus translation transformation applied. Sample images can be seen in Figures 3.29, 3.30 and 3.31.
- **Dataset 5.1**: Contains images resized to 28x28 with noise level 100 and scaling plus translation transformation applied. These images correspond to the same set used in Dataset 5.0, but at a reduced resolution. Sample images can be seen in Figures 3.32, 3.33, and 3.34.
- **Dataset 6.0**: Contains images of size 224x224 with noise level 100 and rotation plus translation transformation applied. Sample images can be seen in Figures 3.35, 3.36 and 3.37.
- **Dataset 6.1**: Contains images resized to 28x28 with noise level 100 and rotation plus translation transformation applied. These images correspond to the same set used in Dataset 6.0, but at a reduced resolution. Sample images can be seen in Figures 3.38, 3.39, and 3.40.
- **Dataset 7.0**: Contains images of size 224x224 with noise level 100 and scaling plus rotation plus translation transformation applied. Sample images can be seen in Figures 3.41, 3.42 and 3.43.
- **Dataset 7.1**: Contains images resized to 28x28 with noise level 100 and scaling plus rotation plus translation transformation applied. These images correspond to the same set used in Dataset 7.0, but at a reduced resolution. Sample images can be seen in Figures 3.44, 3.45, and 3.46.

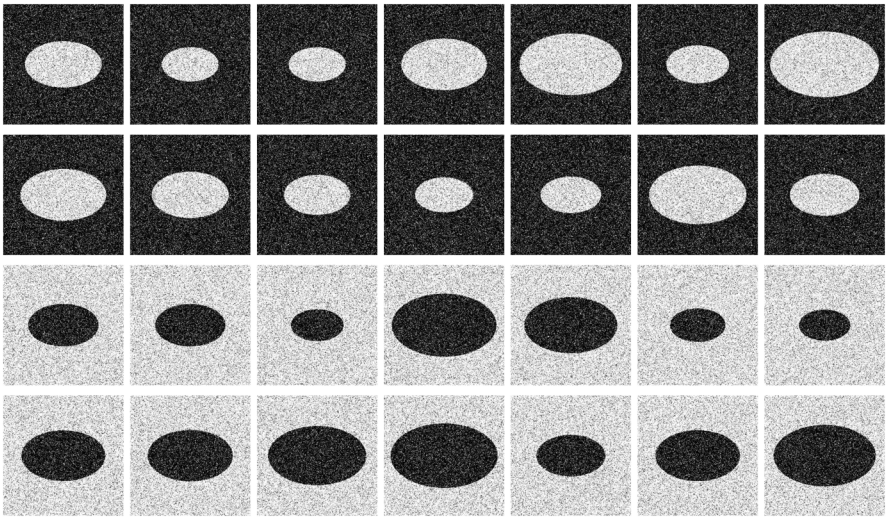


Figure 3.5. Several ellipse samples from dataset 1.0. Best viewed in electronic form.

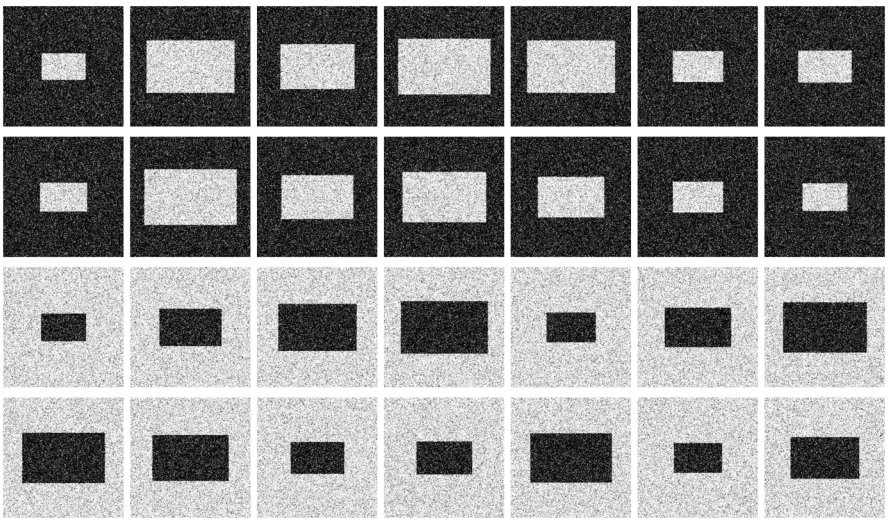


Figure 3.6. Several rectangle samples from dataset 1.0. Best viewed in electronic form.

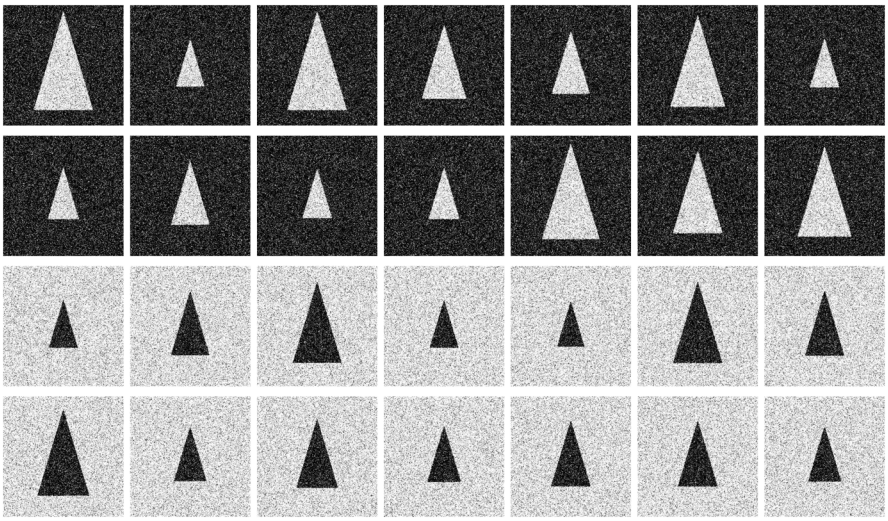


Figure 3.7. Several triangle samples from dataset 1.0. Best viewed in electronic form.

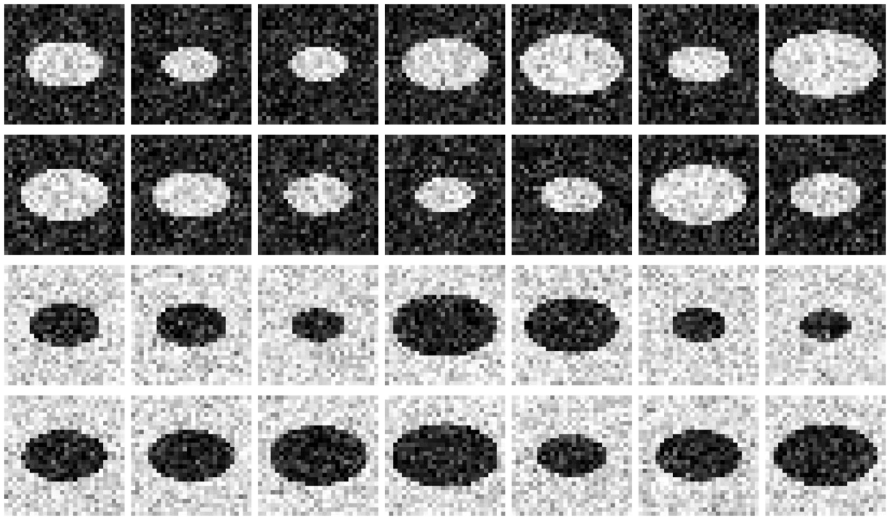


Figure 3.8. Several ellipse samples from dataset 1.1. Best viewed in electronic form.

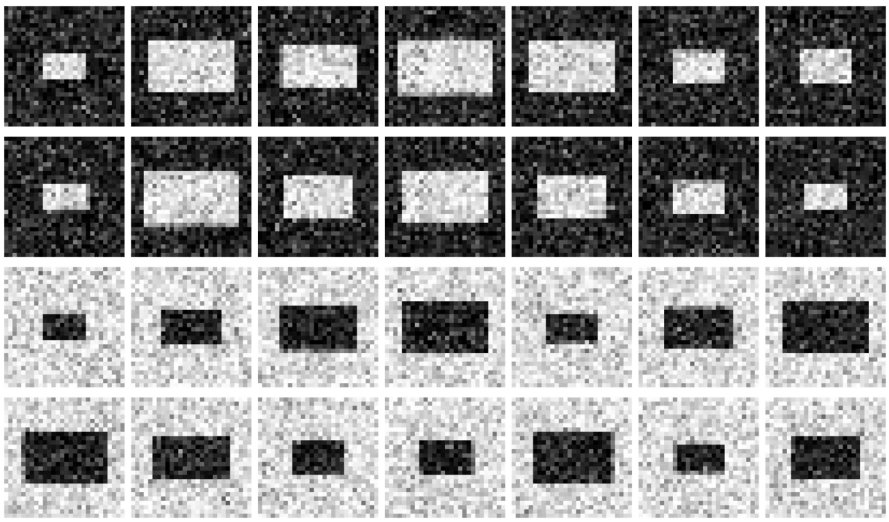


Figure 3.9. Several rectangle samples from dataset 1.1. Best viewed in electronic form.

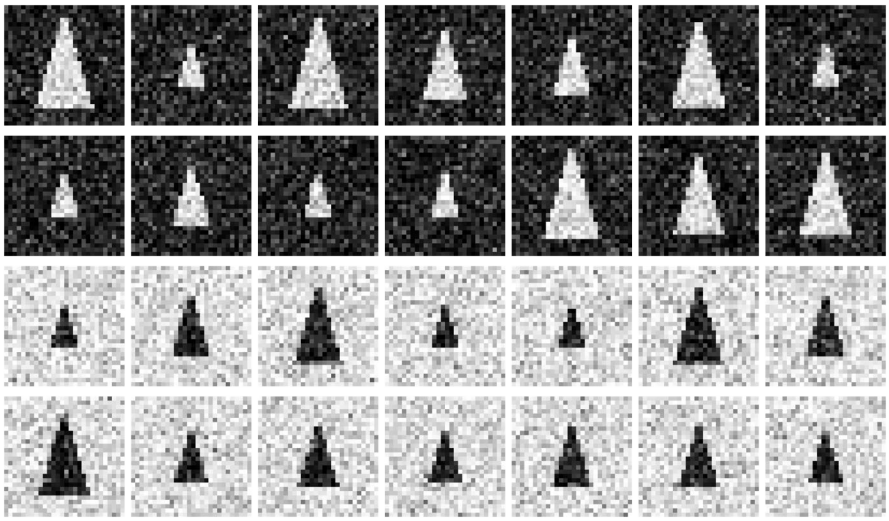


Figure 3.10. Several triangle samples from dataset 1.1. Best viewed in electronic form.

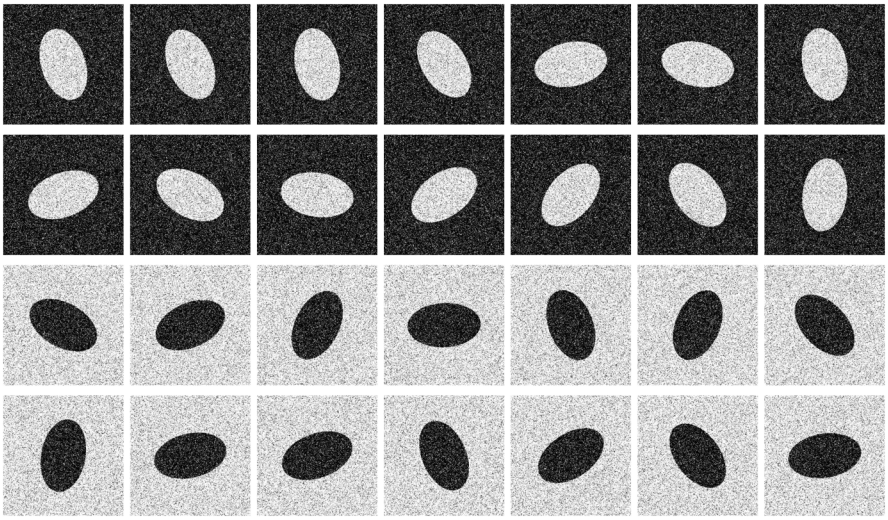


Figure 3.11. Several ellipse samples from dataset 2.0. Best viewed in electronic form.

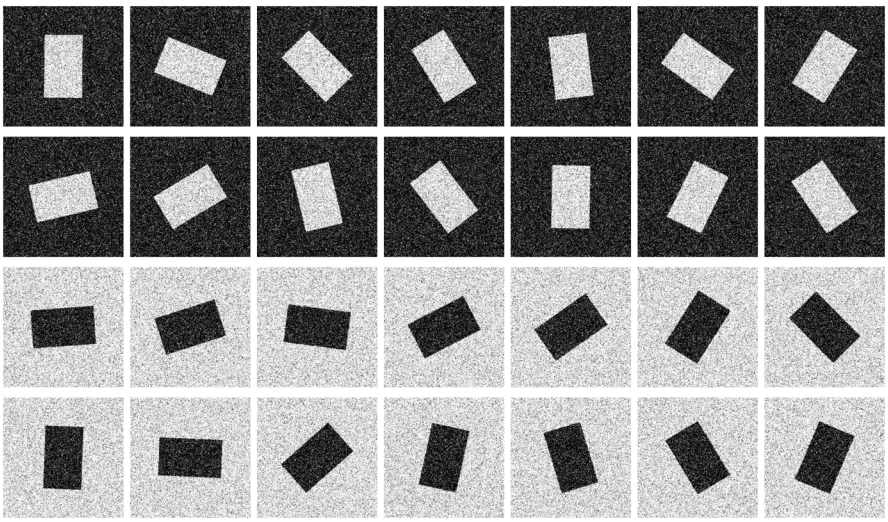


Figure 3.12. Several rectangle samples from dataset 2.0. Best viewed in electronic form.

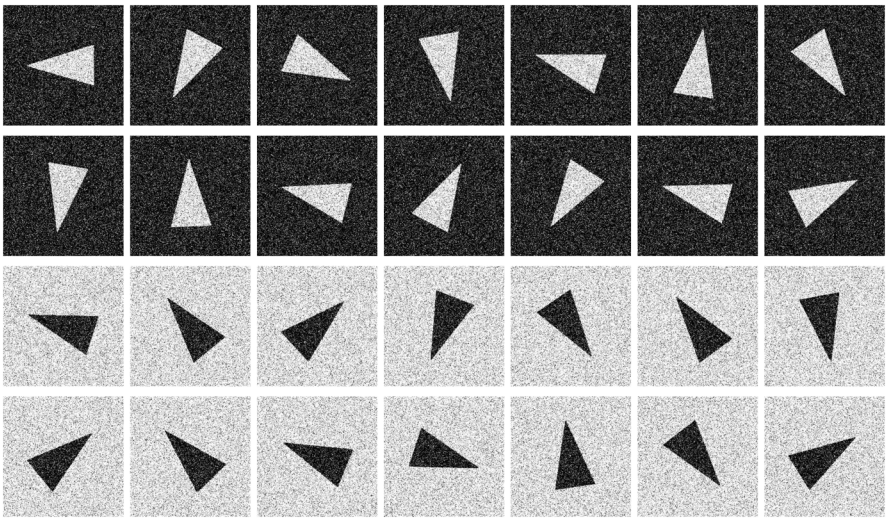


Figure 3.13. Several triangle samples from dataset 2.0. Best viewed in electronic form.

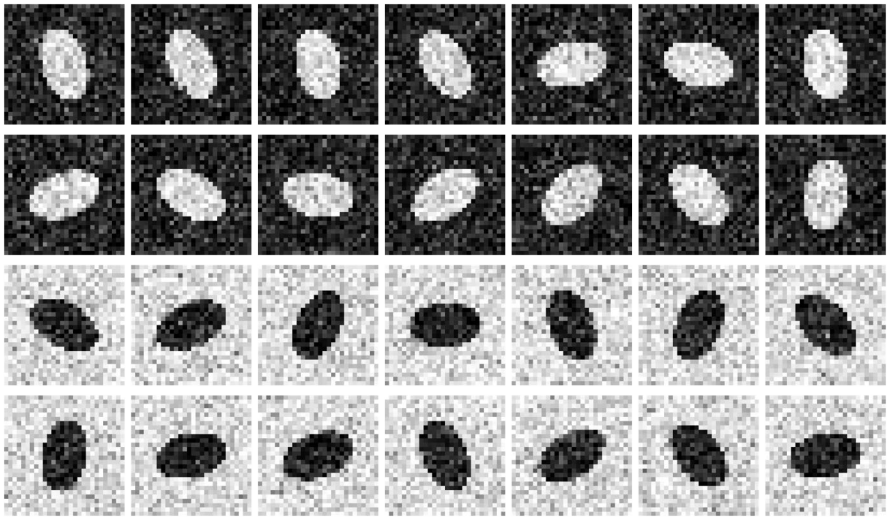


Figure 3.14. Several ellipse samples from dataset 2.1. Best viewed in electronic form.

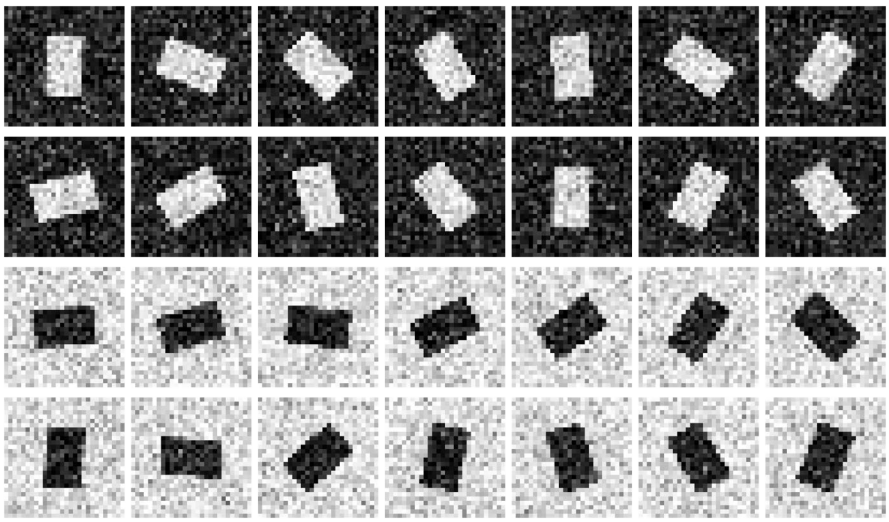


Figure 3.15. Several rectangle samples from dataset 2.1. Best viewed in electronic form.

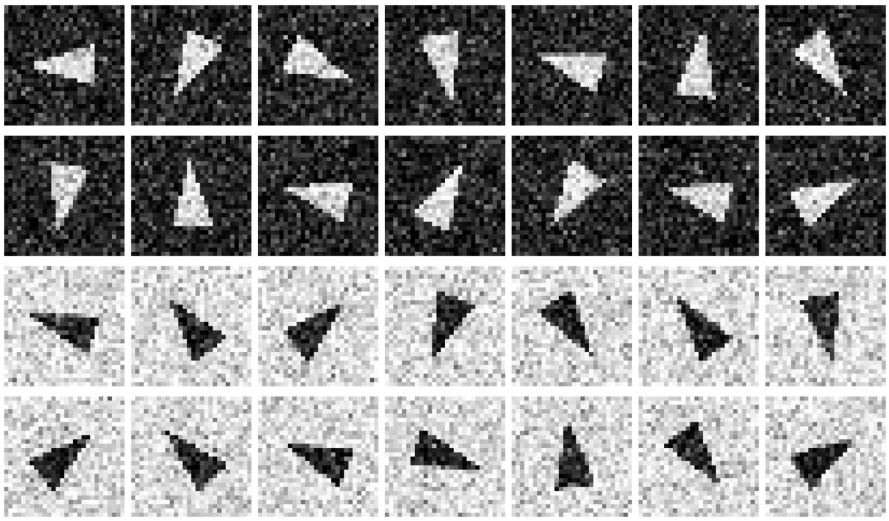


Figure 3.16. Several triangle samples from dataset 2.1. Best viewed in electronic form.

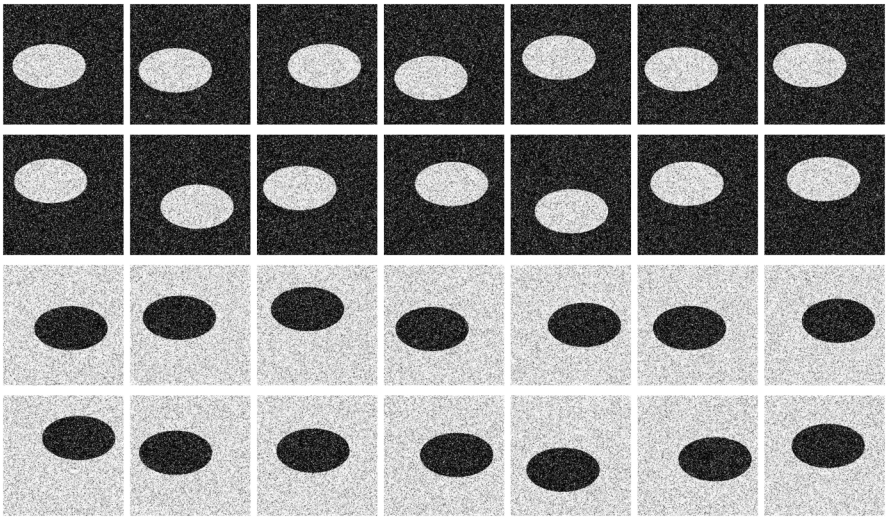


Figure 3.17. Several ellipse samples from dataset 3.0. Best viewed in electronic form.

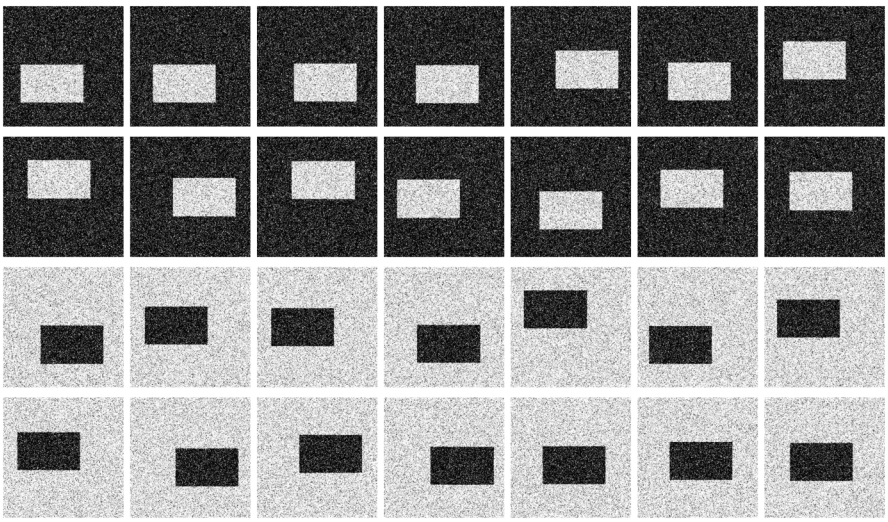


Figure 3.18. Several rectangle samples from dataset 3.0. Best viewed in electronic form.

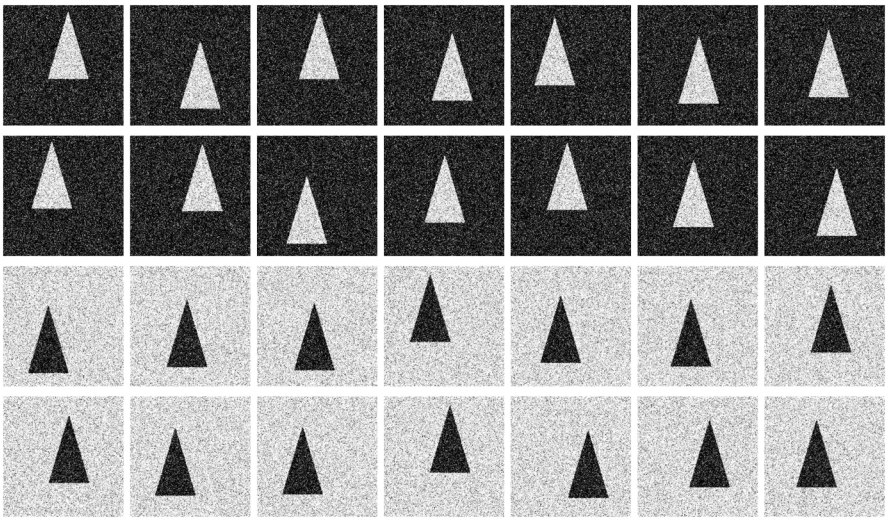


Figure 3.19. Several triangle samples from dataset 3.0. Best viewed in electronic form.

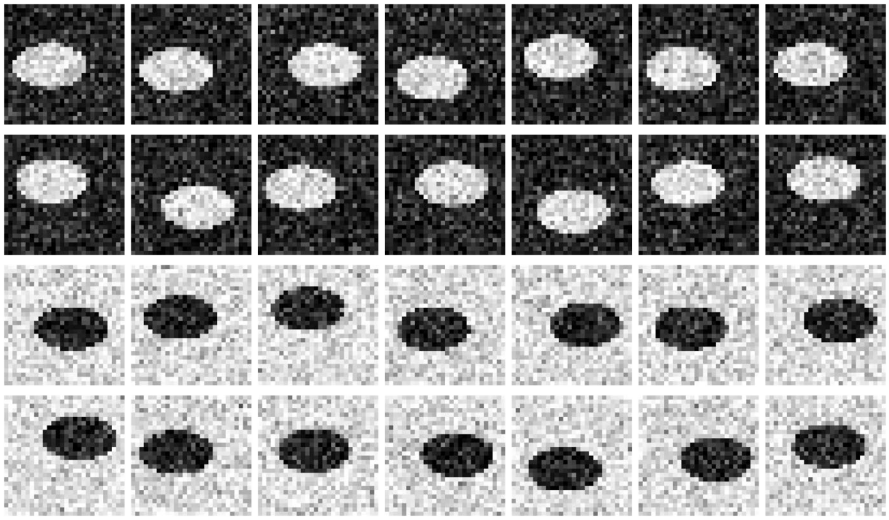


Figure 3.20. Several ellipse samples from dataset 3.1. Best viewed in electronic form.

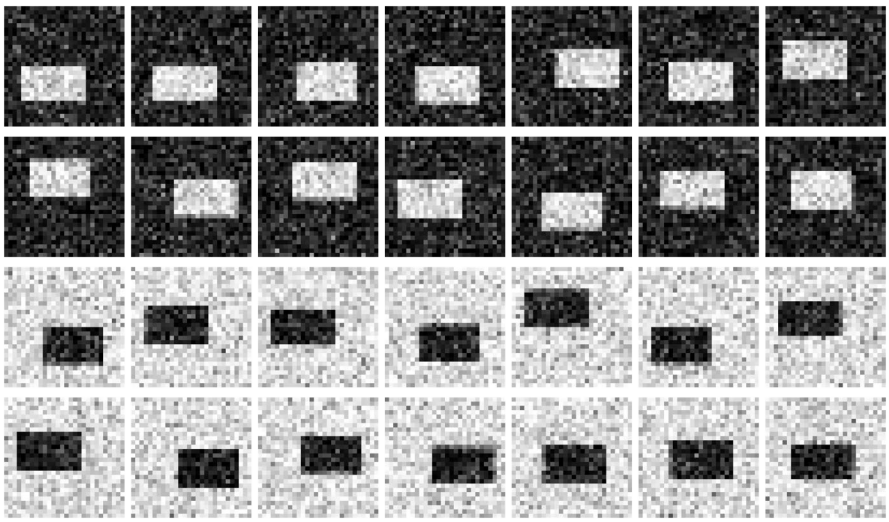


Figure 3.21. Several rectangle samples from dataset 3.1. Best viewed in electronic form.

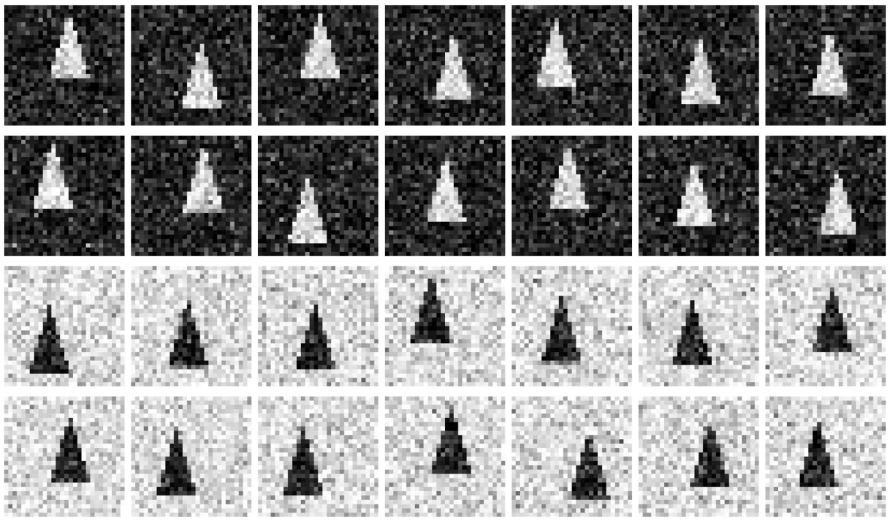


Figure 3.22. Several triangle samples from dataset 3.1. Best viewed in electronic form.

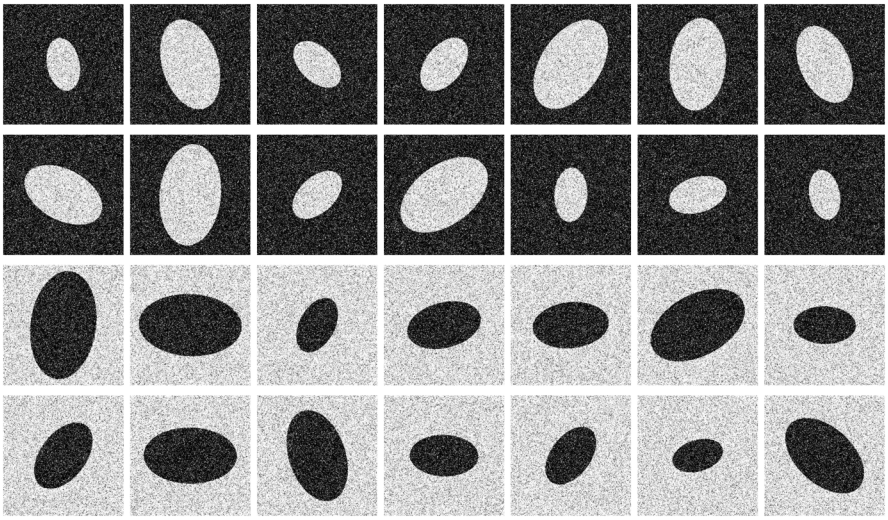


Figure 3.23. Several ellipse samples from dataset 4.0. Best viewed in electronic form.

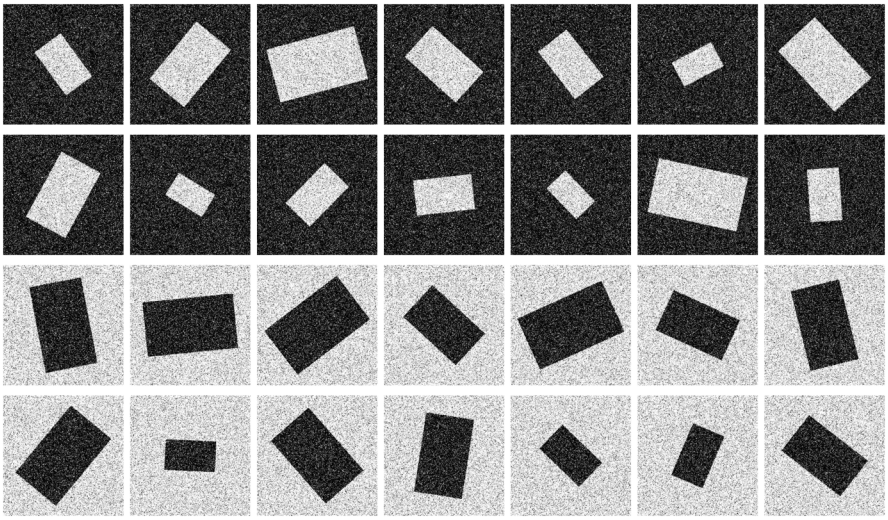


Figure 3.24. Several rectangle samples from dataset 4.0. Best viewed in electronic form.

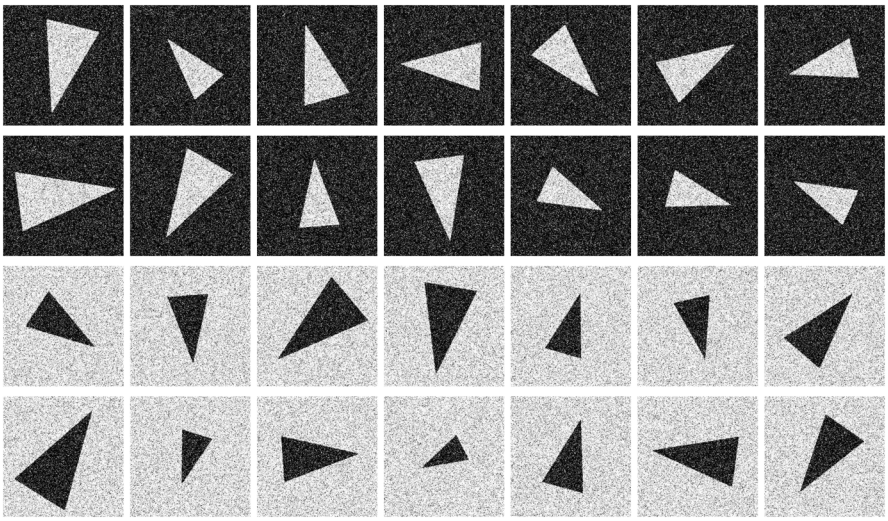


Figure 3.25. Several triangle samples from dataset 4.0. Best viewed in electronic form.

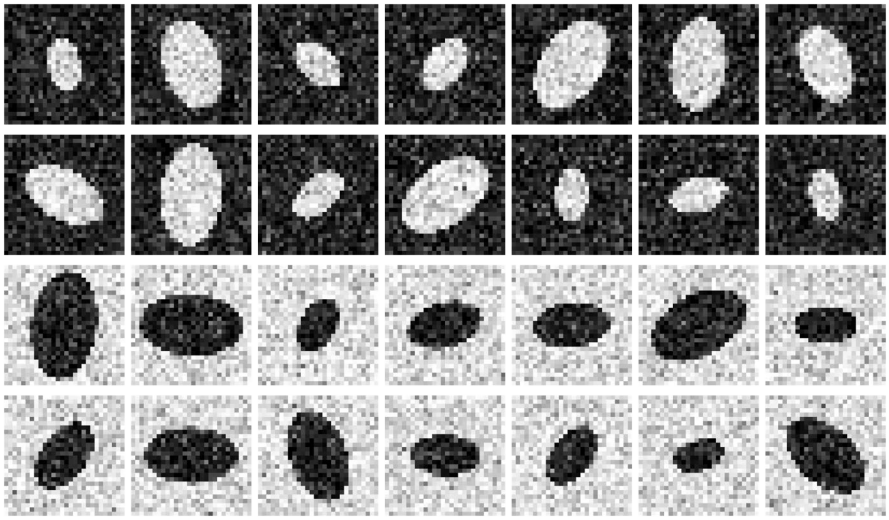


Figure 3.26. Several ellipse samples from dataset 4.1. Best viewed in electronic form.

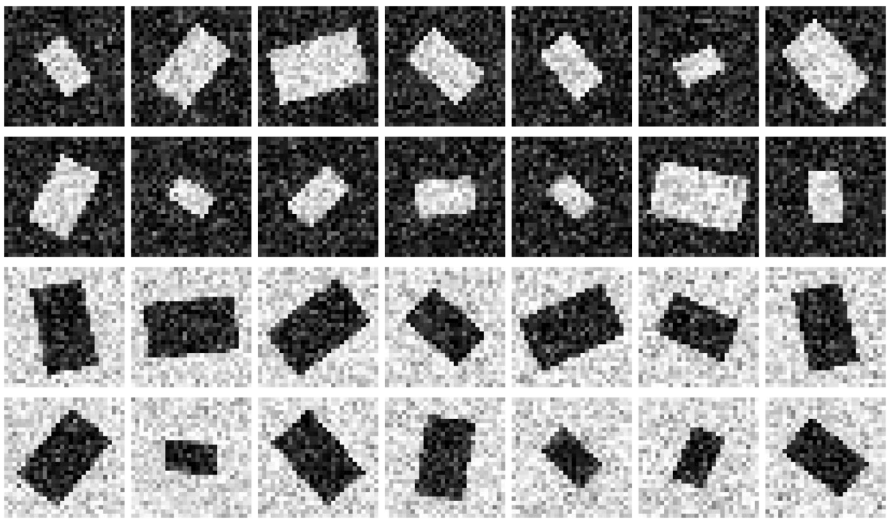


Figure 3.27. Several rectangle samples from dataset 4.1. Best viewed in electronic form.

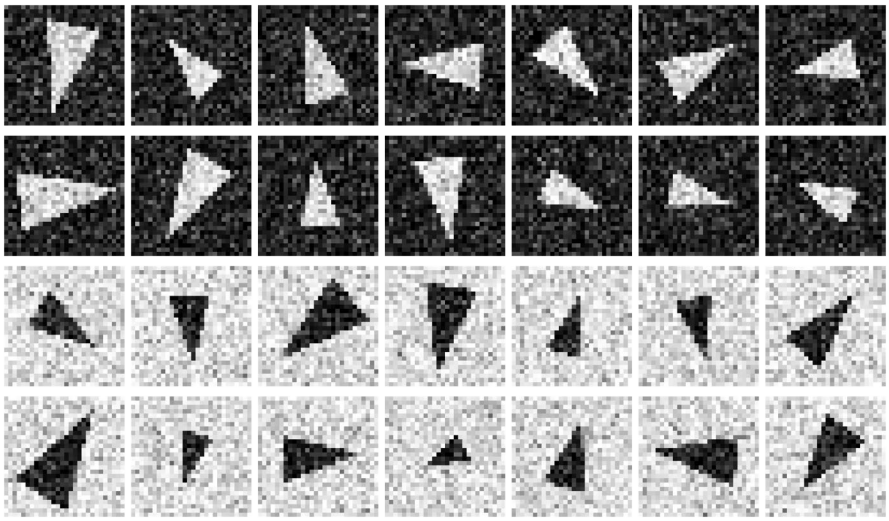


Figure 3.28. Several triangle samples from dataset 4.1. Best viewed in electronic form.

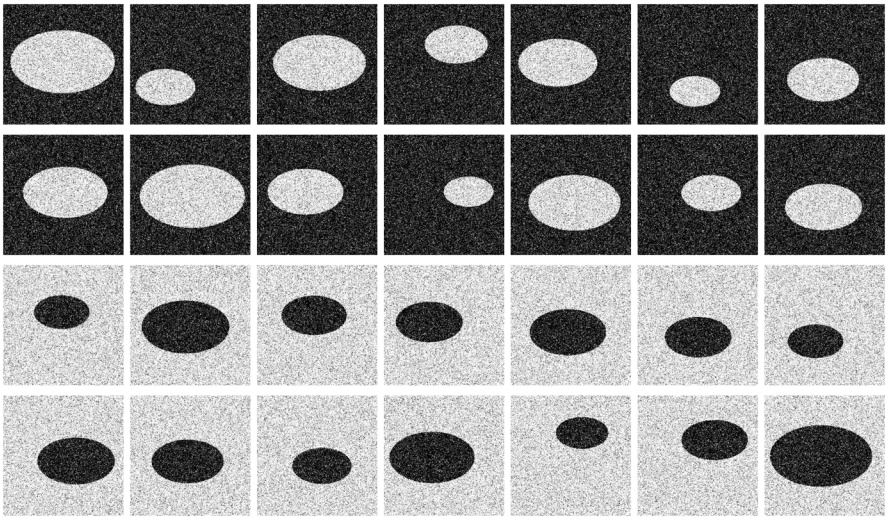


Figure 3.29. Several ellipse samples from dataset 5.0. Best viewed in electronic form.

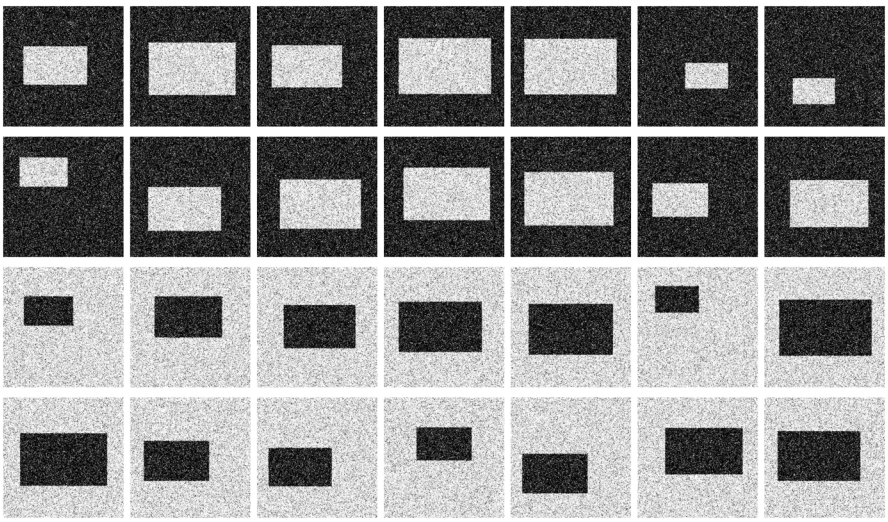


Figure 3.30. Several rectangle samples from dataset 5.0. Best viewed in electronic form.

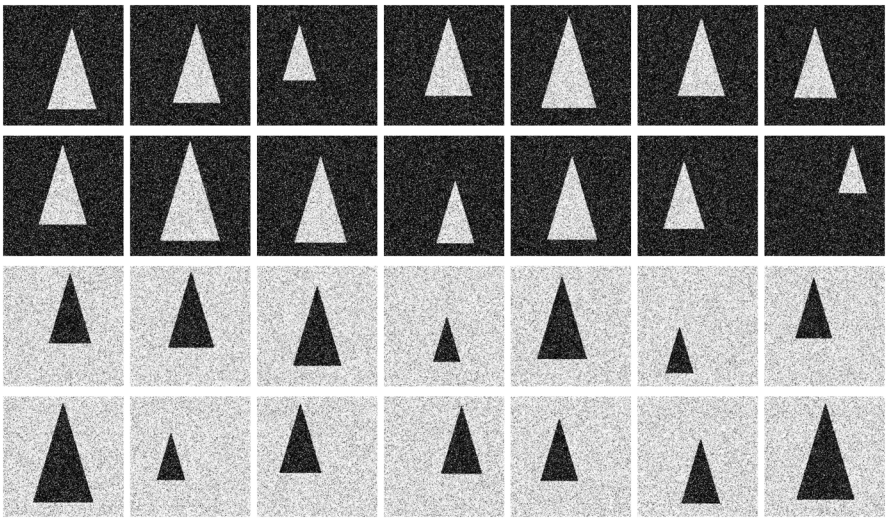


Figure 3.31. Several triangle samples from dataset 5.0. Best viewed in electronic form.

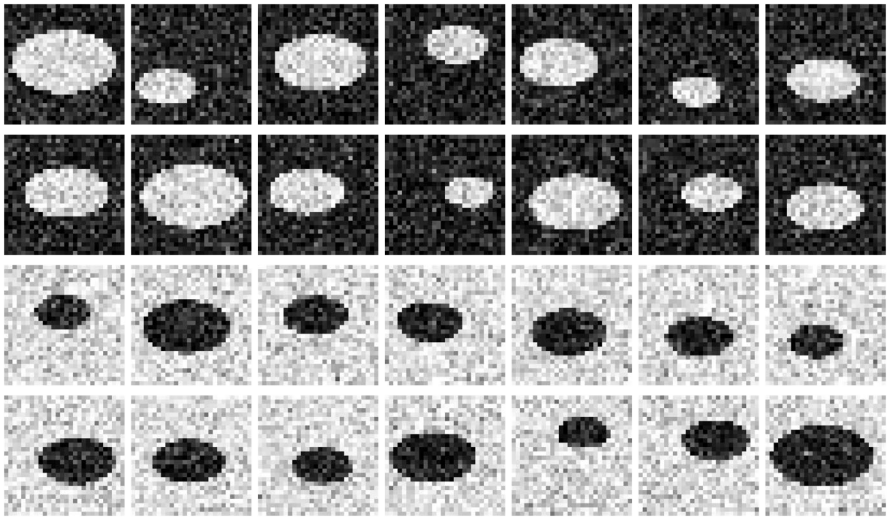


Figure 3.32. Several ellipse samples from dataset 5.1. Best viewed in electronic form.

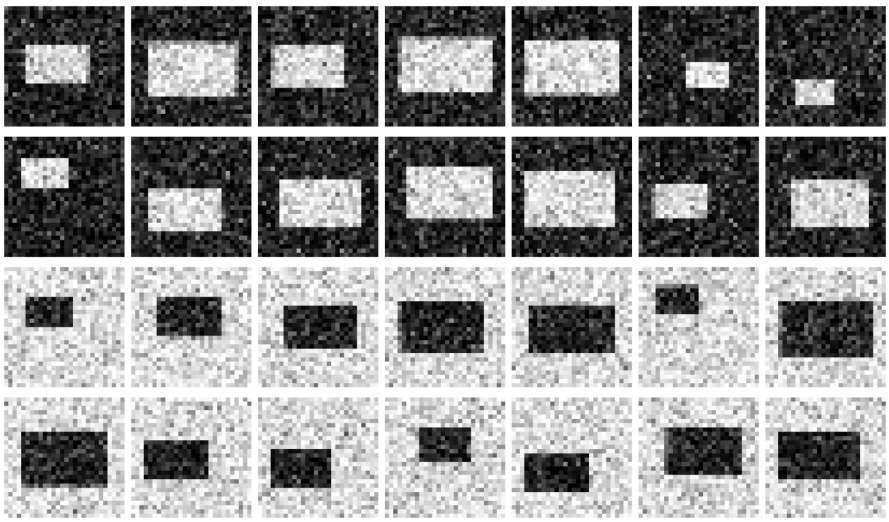


Figure 3.33. Several rectangle samples from dataset 5.1. Best viewed in electronic form.

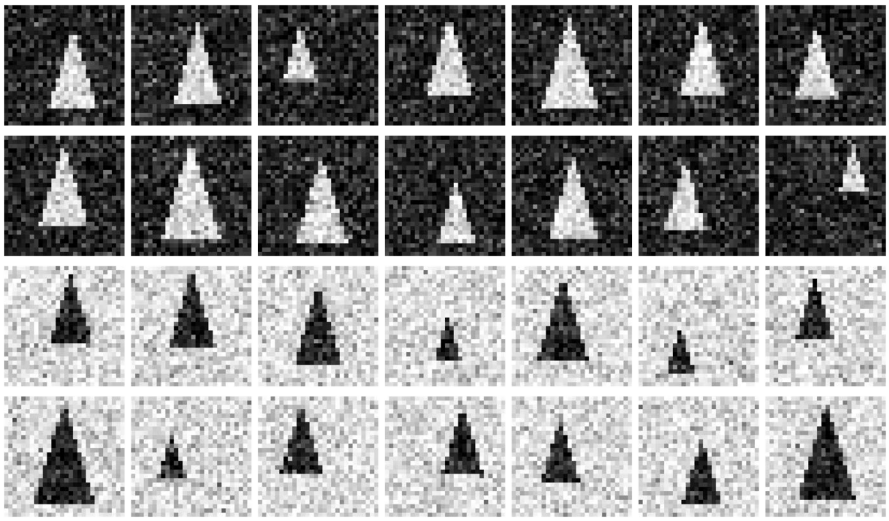


Figure 3.34. Several triangle samples from dataset 5.1. Best viewed in electronic form.

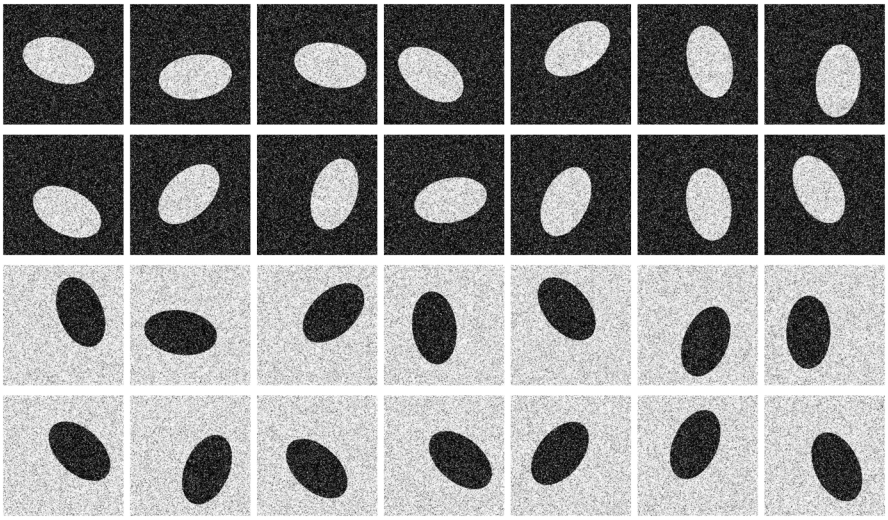


Figure 3.35. Several ellipse samples from dataset 6.0. Best viewed in electronic form.

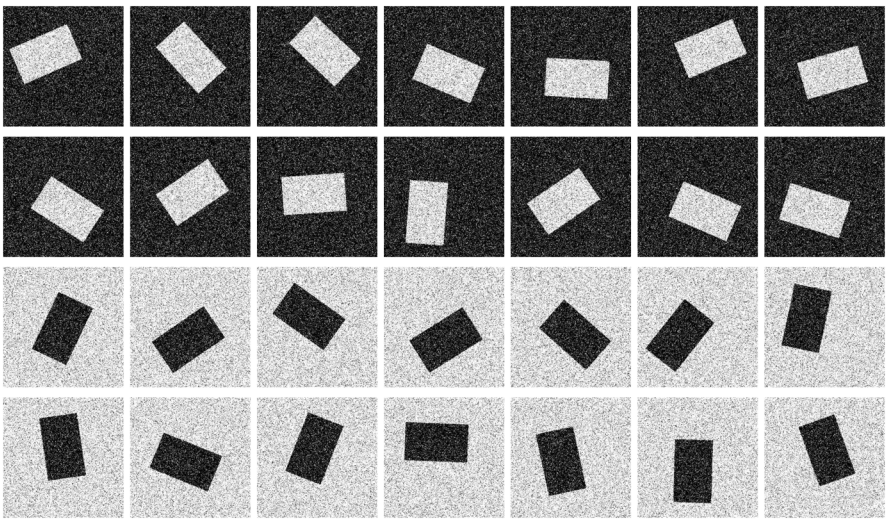


Figure 3.36. Several rectangle samples from dataset 6.0. Best viewed in electronic form.

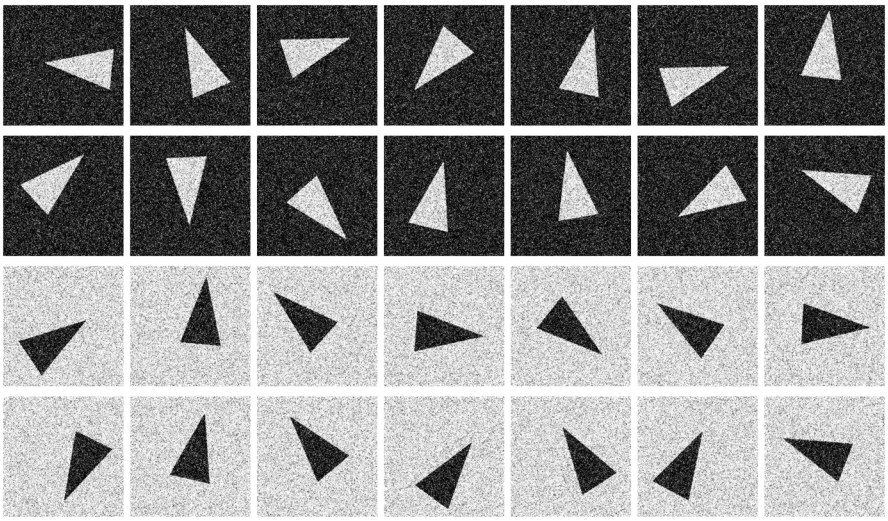


Figure 3.37. Several triangle samples from dataset 6.0. Best viewed in electronic form.

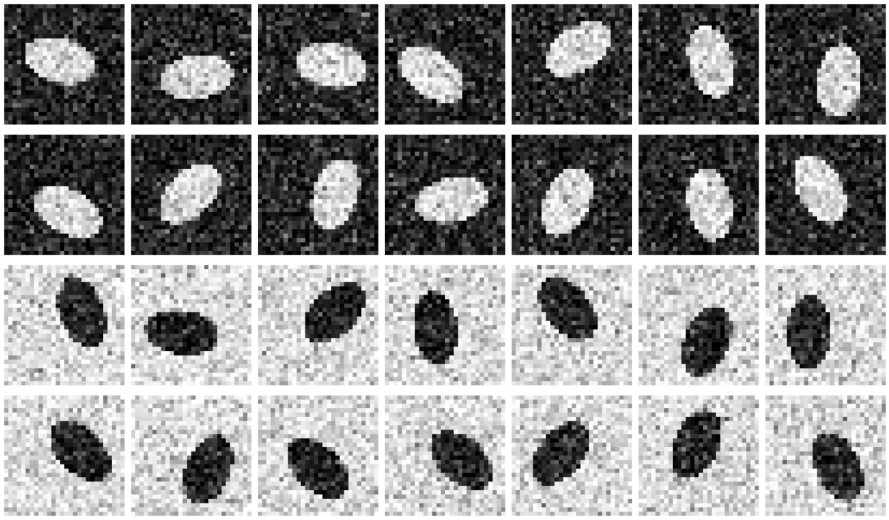


Figure 3.38. Several ellipse samples from dataset 6.1. Best viewed in electronic form.

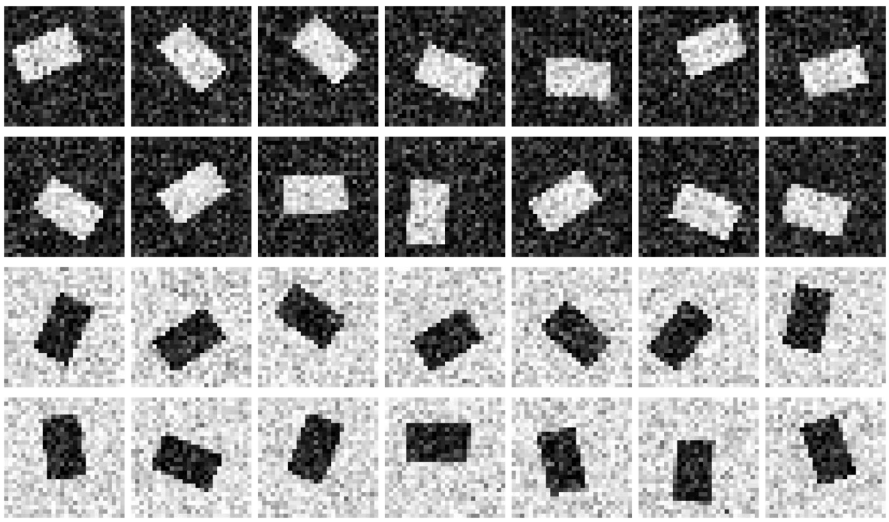


Figure 3.39. Several rectangle samples from dataset 6.1. Best viewed in electronic form.

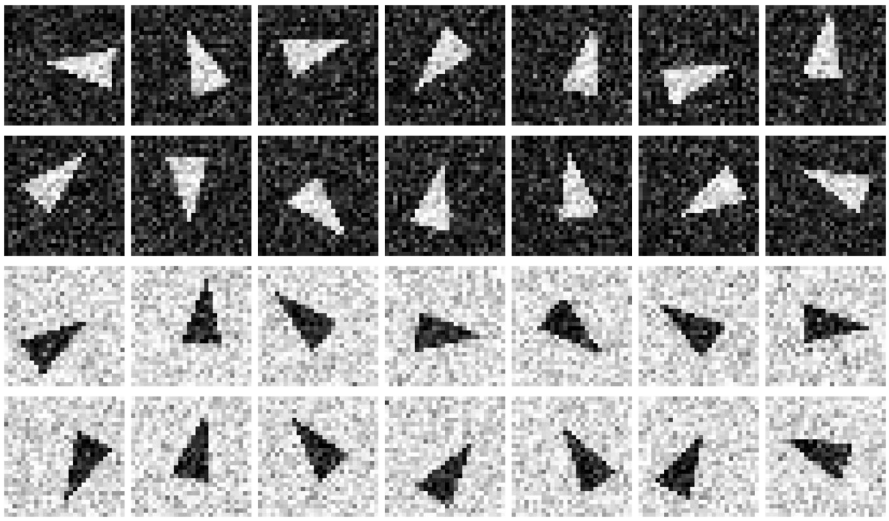


Figure 3.40. Several triangle samples from dataset 6.1. Best viewed in electronic form.

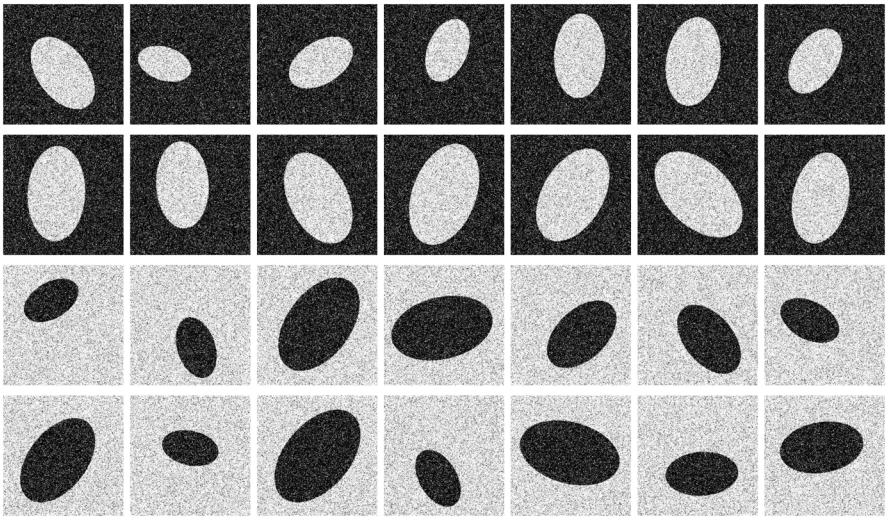


Figure 3.41. Several ellipse samples from dataset 7.0. Best viewed in electronic form.

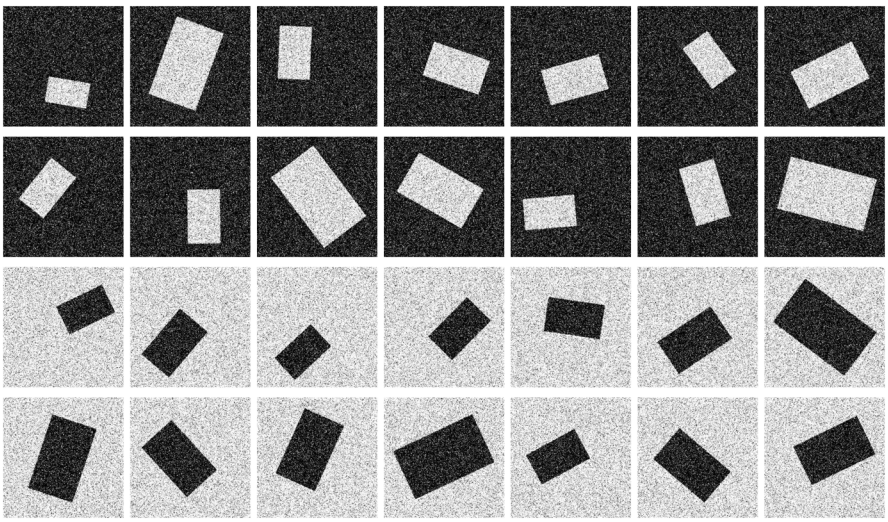


Figure 3.42. Several rectangle samples from dataset 7.0. Best viewed in electronic form.

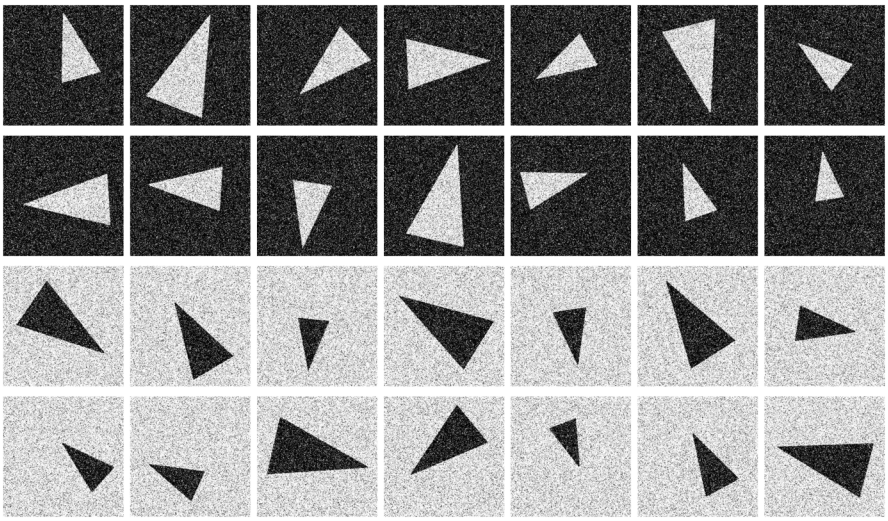


Figure 3.43. Several triangle samples from dataset 7.0. Best viewed in electronic form.

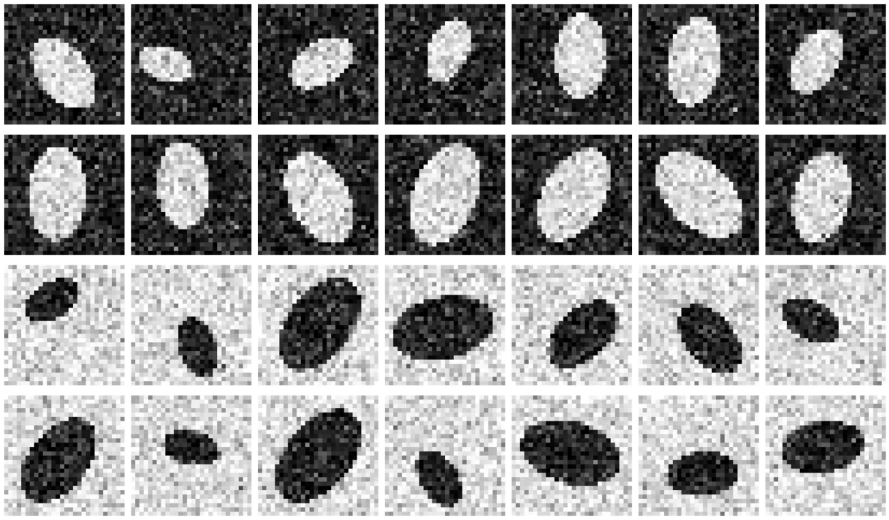


Figure 3.44. Several ellipse samples from dataset 7.1. Best viewed in electronic form.

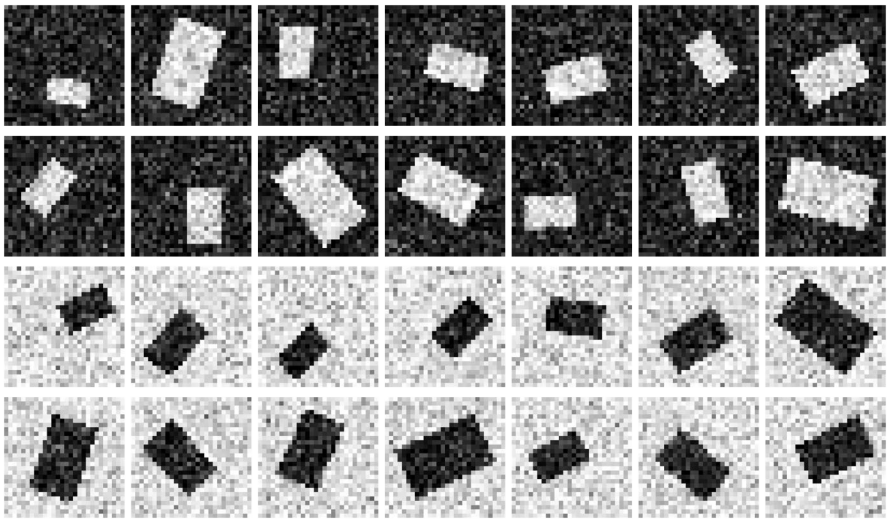


Figure 3.45. Several rectangle samples from dataset 7.1. Best viewed in electronic form.

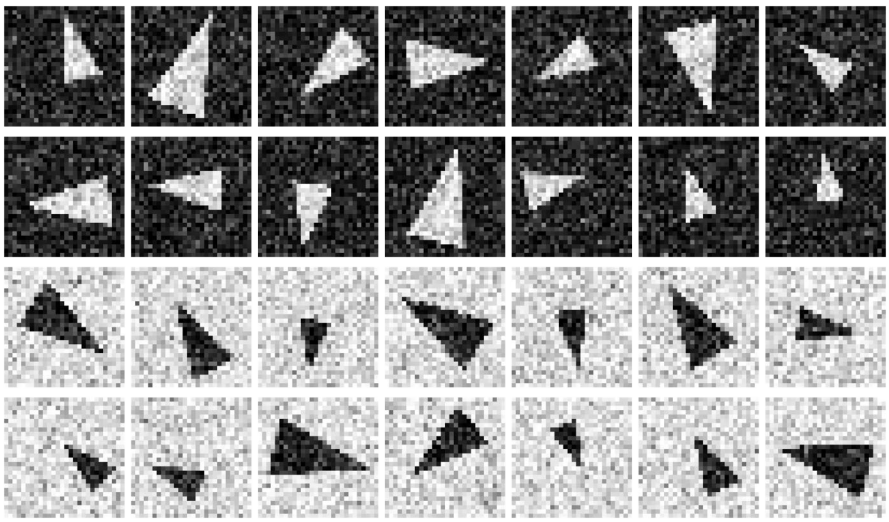


Figure 3.46. Several triangle samples from dataset 7.1. Best viewed in electronic form.

4

Optimal filter selection

In the end, we are our choices.
Jeff Bezos (1964–)

This chapter introduces a methodology to determine the minimum CNN architecture required to solve a problem with the fewest parameters. It aims to build a knowledge base for intelligently initializing new models, focused on extracting the filters learnt by the models. The methodology is applied to three datasets and key insights are summarized to support later chapters.

To address the first part of the thesis objective, “to obtain a knowledge base of the order of magnitude of the hyperparameters, model structure and filter that enables to intelligently initialize a CNN architecture", a dedicated methodology has been developed. The goal of this methodology is to identify the *minimum model architecture*, understood as the model capable of solving the given problem using the fewest possible parameters.

The chapter is organized into five sections. Section 4.1 outlines the methodology used to derive the minimum model architectures and, consequently, the knowledge base that supports smart initialization and training of new models from scratch. Sections 4.2, 4.3 and 4.4 detail the application of this methodology to Datasets 1.1, 2.1, and 3.1, respectively. Finally, Section 4.5 summarizes the key findings of this chapter, which are foundational for the work presented in the subsequent chapters.

4.1. Proposed methodology

The methodology is based on training a set of CNN architectures using the grid technique, where the value of several hyperparameters is changed over different trainings, so that the combinations give as a result different models that explore a wide space of possible solutions to a specific problem. All the models trained have the same architecture, that is composed by the following layers by order:

1. One **input layer** of size 28x28, the same as the dataset images.

2. One **convolutional layer** to extract features from the image by applying filters or kernels.

3. One **pooling layer** performing the operation of Max Pooling, extracting the maximum value for patches of the feature map resulting from the previous layer.

4. One **flatten layer** to convert the resultant bidimensional arrays into a single continuous linear vector.

5. One **dense layer** or a fully-connected layer of neurons to perform the classification operation taking as input the features extracted from the image.

6. One final **dense layer** as output layer of size 3, since this is the number of possible classes that can result from the classification problem.

The convolutional part of the CNN consists of three layers as well as the classification part. The hyperparameters selected for the grid and their possible values are shown below:

- **Seed**: it is used to initialized the random number generator with an specific value, ensuring reproducibility across experiments. Several seeds are used since they initialization plays a key role when finding the optimal architecture. When exploring the space in search of a solution to the classification problem, depending on the initial point from which the CNN starts its search, it will find a better solution to the problem or not. If for different seeds the model is able to find a valid solution, it will mean that it has been properly adjusted. The seeds used are: 4, 10, 550 and 1234.

- **Number of filters**, also referred to as *nfilters_1st*: the total amount of filters used in the first (and only) convolutional layer. Each filter focuses con identifying a certain feature. This parameter is set to 1 for all the models trained in this chapter.

- **Kernel size**, also referred to as *ksize_1st*: size of the filters used in the first convolutional layer. In this case, the kernel size is set to 3, which means that filters have a size of 3x3. There are several reasons behind this choice. First of all, smaller filters enable local feature detection focusing on local patterns like edges or corners (which, in this case, are the main features to distinguish one figure from another). In addition, small filters mean fewer parameters, which helps avoid overfitting and speeds up training while reducing complexity.

- **Pooling size**, also referred to as *poolsize_1st*: size of the window that slides across the image to perform the pooling operation. The bigger the window, the lesser number of resulting features. For the grid presented in thistThesis, this parameter takes the values: 6 (16 features), 5 (25 features) , 4 (49 features), 3 (81 features), 2 (196 features).

- **Number of neurons in the hidden layer**, also referred to as *hlneurons*: number of neurons used for the classification part of the network. Note that the classifier needs to be complex enough to derive classes from all the features extracted. This way, the accuracy of the model depends entirely on the performance of the filters learnt. The parameter can take the values 250 or 500.

- **Number of epochs**, also referred to as *nepochs*: this parameter is not strictly fixed for the training but derived from the training process itself. Hence, a sufficiently high number of epochs, 5000, is set but the *Early Stopping* is enabled. This means that the training is stop when a monitored metric (in this case, the validation loss) has stopped improving (that is,

it has stopped decreasing) so, in the end, the number of epochs varies from one model to another, depending on how well the training has been.

Once all different models resulting from the grid have been trained, another set of parameters is used to compare them and chose which ones achieve the best results with the less complex architecture.

- **Number of parameters**, also referred to as *number_params*: total amount of parameters the architecture contains. The number of parameters depends on the combination of hyperparameters selected in the grid. There is a direct correlation between the number of parameters and the training time. The more parameters, the longer it will take for the model to complete the training, hence, the complexity increases.
- **Training accuracy**: the percentage of correct predictions the CNN makes on the data it was trained on. It tells how well the model is learning to recognize patterns in the training set.
- **Validation accuracy**: the percentage of correct predictions the CNN makes on unseen data, called the validation set, which is not used during training. It tells how well the model is generalizing to new, unseen data. It is a better indicator of real-world performance than training accuracy.

This methodology can be extrapolated to any dataset but, in this case, it is applied exclusively to Datasets 1.1, 2.1 and 3.1, which correspond to the fundamental geometric transformations: scaling, rotation and translation. The objective is to develop filters specialized in recognizing the type of geometric figure under each transformation. These three transformations were selected because they are considered classical and foundational, commonly encountered in industrial applications involving object localization and identification. This way, the combination of filters obtained for each one of the transformations can address more complex problems, thereby broadening the scope of the analysis. It is important to highlight that the models are only trained with the datasets that contain noise, in order to avoid possible biases and to simulate more real use cases. The noise level chosen is 100, as it is considered to be high enough to avoid bias, but low enough not to pose a problem for the dataset being analysed.

4.2. Dataset 1.1: Scaling

The first problem addressed is the scaling. Table 4.1 shows all the models trained for Dataset 1.1 using the methodology presented in Section 4.1. In general, good results for both training and validation sets are obtained, some of the models even reach the 100% of success in training and 99% in validation, which could be an indicator of the capability of the architecture to find a solution to the problem.

Ordering models by validation accuracy, training accuracy and number of parameters, enables to look for a minimum complexity with the maximum amount of correct answers. In this way, the model chosen is *20250202T191455_dataset1.1n100_1L_nf1_1_ks1_3_ps1_3_s_550* which uses one filter of size 3x3 for the convolutional layer, a pooling size of 3x3, a classifier with 250 neurons in the hidden layer and the seed 550. This model is able to reach a validation accuracy level of 0.9915 while reaching a training accuracy level of 0.99967 with 21k parameters approximately. The reason behind choosing this model is the trade-off between accuracy and complexity. There are another three models that are able to reach higher levels of precision but for that they use the double amount of parameters (nearly 43k).

Table 4.1. Model hyperparameters and results obtained after applying the proposed methodology for obtaining the minimum model to Dataset 1.1 with noise level 100. Ordered by validation accuracy from higher to lower. *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 "pooling size of the first layer", *hlneurons* stands for "number of neurons in the hidden layer for the classification part of the model" *number_param* stands for "number of parameters of the model", *tr_acc* stands for "training accuracy" and *val_acc* stands for "validation accuracy", both values are represented with five decimals. Model names follow the structure: <model timestamp>_dataset1.1n100_1L_nf1_<nfilters_1st>_ks1_<ksize_1st>_ps1_<poolsize_1st>_s_<seed>. The model selected as the minimum model is highlighted in gray.

Model timestamp	seed	nepochs	nfilters_1st	ksize_1st	poolsize_1st	hlneurons	number_params	training_acc	val_acc ↓
20250202T200620	550	94	1	3	3	500	42513	0.99967	0.99550
20250202T195322	10	112	1	3	3	500	42513	1.00000	0.99467
20250202T193835	1234	128	1	3	3	500	42513	1.00000	0.99234
20250202T191455	550	120	1	3	3	250	21263	0.99967	0.99150
20250202T185948	10	130	1	3	3	250	21263	0.99834	0.99134
20250202T184010	1234	170	1	3	3	250	21263	1.00000	0.99084
20250202T175511	1234	113	1	3	4	500	26513	1.00000	0.98900
20250202T170248	1234	136	1	3	4	250	13263	099967	0.98717
20250202T181754	550	99	1	3	4	500	26513	0.99867	0.98500
20250202T173053	550	106	1	3	4	250	13263	0.99767	0.98250
20250202T180818	10	82	1	3	4	500	26513	0.99000	0.98050
20250202T192854	4	83	1	3	3	500	42513	1.00000	0.98034
20250202T152620	10	165	1	3	5	250	7263	0.99234	0.97884
20250202T162654	10	116	1	3	5	500	14513	0.99534	0.97800
20250202T150728	1234	163	1	3	5	250	7263	0.99300	0.97767
20250202T154526	550	179	1	3	5	250	7263	0.99034	0.97717
20250202T182924	4	92	1	3	3	250	21263	1.00000	0.97634
20250202T171832	10	106	1	3	4	250	13263	0.98567	0.97450
20250202T164025	550	71	1	3	5	500	14513	0.98367	0.97157
20250202T174311	4	103	1	3	4	500	26513	1.00000	0.96984
20250202T161548	1234	95	1	3	5	500	14513	0.99034	0.96967
20250202T144520	550	69	1	3	6	500	10013	0.98634	0.96684
20250202T164846	4	121	1	3	4	250	13263	1.00000	0.96500
20250202T135421	550	98	1	3	6	250	5013	0.97934	0.96317
20250202T160606	4	83	1	3	5	500	14513	0.99734	0.96167
20250202T145327	4	121	1	3	5	250	7263	0.99667	0.96000
20250202T132847	1234	118	1	3	6	250	5013	0.97734	0.95467
20250202T142120	1234	89	1	3	6	500	10013	0.97934	0.95317
20250202T143144	10	117	1	3	6	500	10013	0.96667	0.95117
20250202T130855	4	171	1	3	6	250	5013	0.98734	0.94967
20250202T134228	10	102	1	3	6	250	5013	0.96200	0.94967
20250202T140548	4	134	1	3	6	500	10013	0.99034	0.94550

Focusing the analysis on the images that are wrongly classified. Tables 4.2 and 4.3 show both training and validation confusion matrix. For the training, it is important to highlight that it only makes a mistake when trying to classify a rectangle because it is interpreted as an ellipse. Looking at Figure 4.1 it can be seen that the image does not have nothing special, the figure

can be perfectly distinguished from an ellipse so it is not clear the reason behind this mistake. It could be that in the dataset there are not enough examples of figures with this size but this would require a more exhaustive analysis.

For the validation images, it misclassifies some ellipses that are interpreted as rectangles and some rectangles that are interpreted as ellipses. The triangles are distinguished perfectly. In both cases, it finds problems to classify those images in which the figures have a smaller size. Figure 4.2 shows an example of the images wrongly classified.

Table 4.2. Confusion matrix for the training set in Dataset 1.1.

KNOWN/PREDICTED	ellipse	rectangle	triangle
ellipse	1000	0	0
rectangle	1	999	0
triangle	0	0	1000

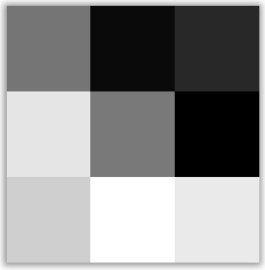
Table 4.3. Confusion matrix for the validation set in Dataset 1.1.

KNOWN/PREDICTED	ellipse	rectangle	triangle
ellipse	1964	36	0
rectangle	15	1985	0
triangle	0	0	2000

Figure 4.1. Training set images wrongly classified by the chosen model for Dataset 1.1 (interpreted as ellipse).

Figure 4.2. Validation set images wrongly classified by the chosen model for Dataset 1.1 (interpreted as ellipses or rectangles).

To conclude this section, it is important to review the filter learnt by the network, as it will form part of the knowledge base extracted for the scaling problem and will be used in the next chapter to analyse more complex problems. Figure 4.3 shows the filter luminance representation as well as the exact luminance values. It seems that is detecting diagonal borders from the upper left corner to the bottom right corner, since these are the values that multiply positively in the convolution operation but from this representation it is difficult to deduce with precision what exact features of the image it is extracting.



-0.007	-0.402	-0.286
0.408	0.011	-0.441
0.325	0.505	0.427

20250202T191455_dataset1.1n100_1L_nf1_1_ks1_3_ps1_3_s_550

Figure 4.3. Filter learnt by the minimum model selected for Dataset 1.1, addressing the scaling problem. Luminance representation on the left and the exact luminance values on the left.

4.3. Dataset 2.1: Rotation

The second problem addressed is the rotation. Table 4.4 shows all the models trained for Dataset 2.1 using the methodology presented in Section 4.1. As it happened with Dataset 1.1, good results for both training and validation sets are obtained with this architectures. Some of the more complex models reach the 100% of success in training and 99% in validation.

Nevertheless, there are some models that are wrongly adjusted, since very different results are obtained from using the same architecture but different seeds. As an example, for models with pooling size equal to six, the initialization point has a clear impact on the performance of the model. In addition, it seems that the more complex the problem is, the more difficult is to find the correct hyperparameters that really decrease the mistakes.

In this case, the rotation problem seems to be more difficult than the scaling problem, due to the fact that the models, in general, are not able to reach levels of accuracy as high as in the scaling problem. Moreover, the rotation models have a higher overfitting than the scaling ones, another indicator that the task is harder to solve for the same architecture.

To identify the most efficient model, candidates were compared based on validation accuracy, training accuracy and the total number of parameters, aiming for minimal complexity with high predictive performance. *20250203T021541_dataset2.1n100_1L_nf1_1_ks1_3_ps1_3_s_1234* is the chosen model, which features a single 3×3 convolutional filter, a 3×3 pooling layer and a classifier with 250 hidden neurons, initialized with seed 550. It achieves a validation accuracy of 0.9915 and a training accuracy of 0.99967, while maintaining a relatively low parameter count of approximately 21k. This model represents a favorable balance between accuracy and computational efficiency. Although three alternative models attain slightly higher precision, they do so at the cost of nearly doubling the number of parameters (approximately 43k), reducing their overall efficiency.

Focusing the analysis on the images that are wrongly classified. Tables 4.5 and 4.6 show both training and validation confusion matrix. For the training, it is important to highlight that it only makes a mistake when trying to classify an ellipse because it is interpreted as a rectangle. Looking at Figure 4.4 it can be seen that the image does not have nothing special, it is just an ellipse rotated at an angle of 45°, the figure can be perfectly distinguished from a rectangle, so it is not clear the reason behind this mistake. It could be that in the dataset there are not enough examples of figures rotated at this angle but this would require a more exhaustive analysis of the dataset generated.

Table 4.4. Model hyperparameters and results obtained after applying the proposed methodology for obtaining the minimum model to Dataset 2.1 with noise level 100. Ordered by validation accuracy from higher to lower. *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 "pooling size of the first layer", *hlneurons* stands for "number of neurons in the hidden layer for the classification part of the model" *number_param* stands for "number of parameters of the model", *tr_acc* stands for "training accuracy" and *val_acc* stands for "validation accuracy", both values are represented with five decimals. Model names follow the structure: <model timestamp>_dataset2.1n100_1L_nf1_<nfilters_1st>_ks1_<ksize_1st>_ps1_<poolsize_1st>_s_<seed>. The model selected as the minimum model is highlighted in gray.

Model timestamp	seed	nepochs	nfilters_1st	ksize_1st	poolsize_1st	hlneurons	number_params	training_acc	val_acc ↓
20250203T033925	550	95	1	3	3	500	42513	0.99833	0.99083
20250203T031841	1234	83	1	3	3	500	42513	1.00000	0.99050
20250203T021541	1234	149	1	3	3	250	21263	0.99967	0.98683
20250203T032822	10	95	1	3	3	500	42513	0.99600	0.98567
20250203T002645	1234	142	1	3	4	250	13263	0.99633	0.98433
20250203T012939	1234	87	1	3	4	500	26513	0.99633	0.98250
20250203T023254	10	116	1	3	3	250	21263	0.99133	0.98233
20250203T004308	10	150	1	3	4	250	13263	0.99033	0.97600
20250203T024621	550	154	1	3	3	250	21263	0.99667	0.97567
20250203T013948	10	73	1	3	4	500	26513	0.99067	0.97567
20250203T014818	550	103	1	3	4	500	26513	0.99367	0.97400
20250203T010027	550	161	1	3	4	250	13263	0.99300	0.97067
20250202T222944	1234	193	1	3	5	250	7263	0.96867	0.95300
20250202T234813	10	82	1	3	5	500	14513	0.96167	0.94033
20250202T225153	10	126	1	3	5	250	7263	0.96000	0.93283
20250203T030408	4	126	1	3	3	500	42513	1.00000	0.92833
20250202T235746	550	115	1	3	5	500	14513	0.97167	0.92300
20250202T233441	1234	117	1	3	5	500	14513	0.95433	0.92067
20250202T230628	550	161	1	3	5	250	7263	0.95000	0.91467
20250203T020015	4	135	1	3	3	250	21263	1.00000	0.90000
20250202T215115	10	76	1	3	6	500	10013	0.93533	0.89933
20250202T205520	10	116	1	3	6	250	5013	0.92833	0.89767
20250202T232502	4	83	1	3	5	500	14513	0.98833	0.88350
20250202T220008	550	95	1	3	6	500	10013	0.92667	0.87633
20250202T210850	550	164	1	3	6	250	5013	0.91967	0.87150
20250202T221110	4	161	1	3	5	250	7263	0.99367	0.86983
20250202T212750	4	118	1	3	6	500	10013	0.96767	0.86700
20250202T203654	1234	159	1	3	6	250	5013	0.92933	0.86383
20250202T214126	1234	84	1	3	6	500	10013	0.91533	0.86267
20250203T011858	4	92	1	3	4	500	26513	1.00000	0.85750
20250202T201719	4	169	1	3	6	250	5013	0.95100	0.84717
20250203T001104	4	136	1	3	4	250	13263	0.99933	0.82000

For the validation images, it misclassifies some rectangles that are interpreted as ellipses and in four cases, as triangles. It does not seem to exist a correlation between the mistakes and the angle of rotation. Figure 4.5 shows an example of the images wrongly classified for rectangles and ellipses. In addition, it also finds it difficult to classify some ellipses correctly, they mistakes

it for rectangles. Figure 4.6 shows some examples of these mistakes, it seems as the majority of samples mistaken for triangles are rectangles placed in horizontal position.

Table 4.5. Confusion matrix for the training set in Dataset 2.1.

KNOWN/PREDICTED	ellipse	rectangle	triangle
ellipse	999	1	0
rectangle	0	1000	0
triangle	0	0	1000

Table 4.6. Confusion matrix for the validation set in Dataset 2.1.

KNOWN/PREDICTED	ellipse	rectangle	triangle
ellipse	1966	34	0
rectangle	41	1955	4
triangle	0	0	2000

Figure 4.4. Training set images wrongly classified by the chosen model for Dataset 2.1 (interpreted as a rectangle).

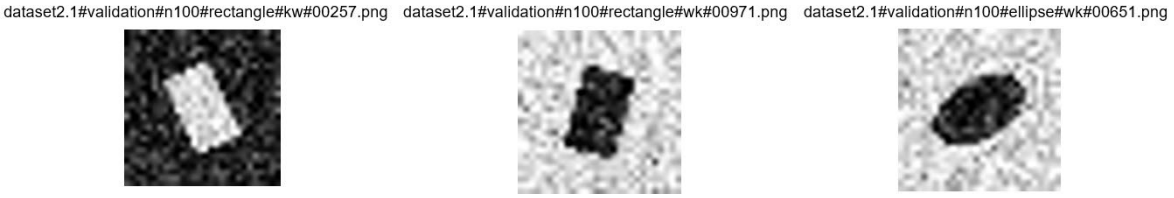


Figure 4.5. Validation set images wrongly classified by the chosen model for Dataset 2.1 (interpreted as ellipses or rectangles).

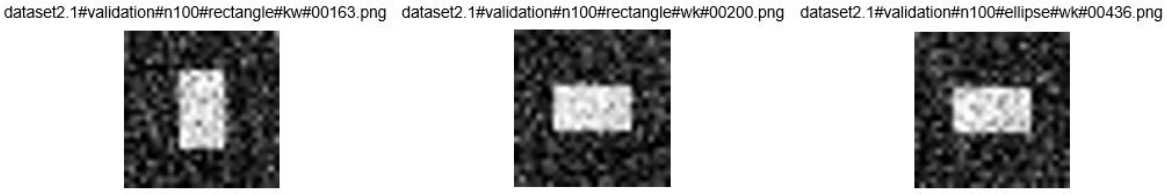
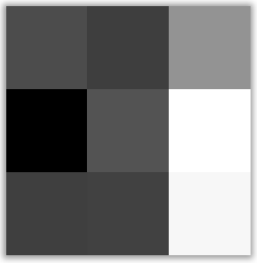


Figure 4.6. Validation set images wrongly classified by the chosen model for Dataset 2.1 (interpreted as triangles).

In order to complete this section, it is important to review the filter learnt by the network, as it will form part of the knowledge base extracted for the scaling problem and will be used

in the next chapter to analyse more complex problems. Figure 4.7 shows the filter luminance representation as well as the exact luminance values. It seems that is detecting vertical borders, since these are the values that multiply positively in the convolution operation but from this representation it is difficult to deduce with precision what exact features of the image it is extracting.



-0.021	-0.063	-0.186
-0.247	-0.002	0.505
-0.060	-0.051	0.479

20250203T021541_dataset2.1n100_1L_nf1_1_ks1_3_ps1_3_s_1234

Figure 4.7. Filter learnt by the minimum model selected for Dataset 2.1, addressing the rotation problem. Luminance representation on the left and the exact luminance values on the left.

4.4. Dataset 3.1: Translation

The third problem addressed is the translation. Table 4.7 shows all the models trained for Dataset 3.1 using the methodology presented in Section 4.1. As it happened with Datasets 1.1 and 2.1, good results for both training and validation sets are obtained with this architectures. Some of the more complex models manage to reach the 100% of success in training and 99% in validation.

As it happened for Dataset 2.1, there are some models that are not correctly adjusted, since very different results are obtained from using the same architecture but different seeds. As an example, for models with pooling size equal to six, the initialization point has a clear impact on the performance of the model.

In this case, the translation problem seems to be more difficult than the scaling problem but easier than the rotation problem, due to the fact that the models, in general, are able to reach levels of accuracy higher than for the rotation problem but lower that the scaling problem. The translation models have a lower overfitting than the rotation ones.

Models are ranked based on validation accuracy, training accuracy and number of parameters to identify the lowest-complexity model that maintains high performance. According to these criteria, the selected model is *20250203T103824_dataset3.1n100_1L_nf1_1_ks1_3_ps1_3_s_10*. This model utilizes a single 3×3 convolutional filter, a 3×3 pooling size and a classifier with 250 neurons in the hidden layer, with seed 10. It achieves a validation accuracy of 0.99033 and a training accuracy of 0.95857, using approximately 21k parameters. The selection is based on its favorable trade-off between accuracy and complexity. While three other models achieve slightly higher accuracy, they require nearly twice the number of parameters (around 43k), making them less efficient.

Focusing the analysis on the images that are wrongly classified. Tables 4.8 and 4.9 show both training and validation confusion matrix. For the training, it is important to highlight that the triangles are differentiated perfectly whereas it finds some difficulties in distinguishing ellipses from rectangles and vice-versa. Looking at Figure 4.8 it can be seen that the images

Table 4.7. Model hyperparameters and results obtained after applying the proposed methodology for obtaining the minimum model to Dataset 3.1 with noise level 100. Ordered by validation accuracy from higher to lower. *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 "pooling size of the first layer", *hlneurons* stands for "number of neurons in the hidden layer for the classification part of the model" *number_param* stands for "number of parameters of the model", *tr_acc* stands for "training accuracy" and *val_acc* stands for "validation accuracy", both values are represented with five decimals. Model names follow the structure: <model timestamp>_dataset3.1n100_1L_nf1_<nfilters_1st>_ks1_<ksize_1st>_ps1_<poolsize_1st>_s_<seed>. The model selected as the minimum model is highlighted in gray.

Model timestamp	seed	nepochs	nfilters_1st	ksize_1st	poolsize_1st	hlneurons	number_params	training_acc	val_acc ↓
20250203T115750	550	121	1	3	3	500	42513	0.99933	0.97067
20250203T114719	10	90	1	3	3	500	42513	0.99567	0.96700
20250203T113101	1234	141	1	3	3	500	42513	1.00000	0.96383
20250203T103824	10	173	1	3	3	250	21263	0.99033	0.95850
20250203T095150	550	94	1	3	4	500	26513	0.99333	0.95833
20250203T093451	10	147	1	3	4	500	26513	0.99700	0.95483
20250203T073853	550	142	1	3	5	500	14513	0.99133	0.95417
20250203T105823	550	172	1	3	3	250	21263	0.99333	0.95267
20250203T101940	1234	162	1	3	3	250	21263	0.99967	0.95100
20250203T091947	1234	130	1	3	4	500	26513	0.99967	0.95033
20250203T111814	4	110	1	3	3	500	42513	1.00000	0.94150
20250203T063651	550	201	1	3	5	250	7263	0.99133	0.93750
20250203T052805	550	131	1	3	6	500	10013	0.98033	0.93567
20250203T083030	10	181	1	3	4	250	13263	0.98567	0.93433
20250203T081124	1234	166	1	3	4	250	13263	0.99367	0.93333
20250203T085114	550	148	1	3	4	250	13263	0.99033	0.93117
20250203T100248	4	146	1	3	3	250	21263	1.00000	0.92683
20250203T065957	4	116	1	3	5	500	14513	0.99967	0.92500
20250203T071322	1234	103	1	3	5	500	14513	0.97800	0.92350
20250203T090821	4	98	1	3	4	500	26513	1.00000	0.92200
20250203T043710	550	163	1	3	6	250	5013	0.96500	0.92067
20250203T055945	1234	162	1	3	5	250	7263	0.96867	0.91633
20250203T072519	10	117	1	3	5	500	14513	0.96433	0.91350
20250203T075516	4	140	1	3	4	250	13263	1.00000	0.91233
20250203T054314	4	143	1	3	5	250	7263	0.99633	0.90267
20250203T050509	1234	78	1	3	6	500	10013	0.94967	0.90250
20250203T040447	1234	118	1	3	6	250	5013	0.94767	0.89833
20250203T061825	10	160	1	3	5	250	7263	0.95400	0.89267
20250203T035028	4	123	1	3	6	250	5013	0.96033	0.89050
20250203T045557	4	79	1	3	6	500	10013	0.96400	0.88800
20250203T051417	10	119	1	3	6	500	10013	0.94967	0.87217
20250203T041825	10	163	1	3	6	250	5013	0.92700	0.85600

do not have nothing special, they are just ellipse translated from either sides of the image, the figure can be perfectly distinguished from a rectangle, so it is not clear the reason behind this mistake. It could be that in the dataset there are not enough examples of figures in this position of the image, but this would require a more exhaustive analysis of the dataset generated.

In the validation images, the model struggle most with distinguishing ellipses from rectangles and, to a lesser extent, rectangles from ellipses. There are also images where both ellipses and

rectangles are misclassified as triangles. A pattern appears to emerge among the misclassified images, as they are predominantly located in the bottom-left corner. Further analysis of the dataset could help confirm whether this spatial bias is significant. Figures 4.9 shows some example of images wrongly classified in the validation dataset.

Table 4.8. Confusion matrix for the training set in Dataset 3.1.

KNOWN/PREDICTED	ellipse	rectangle	triangle
ellipse	980	20	0
rectangle	9	991	0
triangle	0	0	1000

Table 4.9. Confusion matrix for the validation set in Dataset 3.1.

KNOWN/PREDICTED	ellipse	rectangle	triangle
ellipse	1848	151	1
rectangle	96	1903	1
triangle	0	0	2000

Figure 4.8. Training set images wrongly classified by the chosen model for Dataset 3.1 (interpreted as a rectangle).

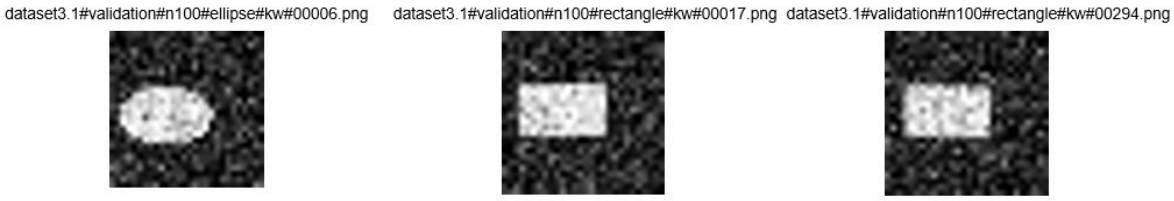


Figure 4.9. Validation set images wrongly classified by the chosen model for Dataset 3.1 (interpreted as ellipses or rectangles).

To finish this section, it is essential to examine the filter learned by the network, as it contributes to the knowledge base for the scaling problem and will be referenced in the next chapter when analysing more complex scenarios. Figure 4.10 presents both the luminance representation of the filter and its exact luminance values. The filter appears to detect corners in the upper-right region of the images, as all values are positive except for the first two pixels along the lower horizontal axis. However, based on this representation alone, it is challenging to precisely determine the specific features being extracted from the images.

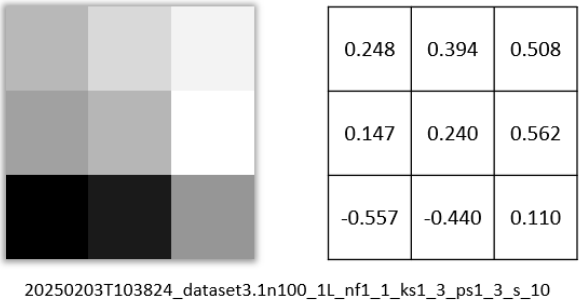


Figure 4.10. Filter learnt by the minimum model selected for Dataset 3.1, addressing the translation problem. Luminance representation on the left and the exact luminance values on the left.

4.5. Conclusion

Applying the proposed methodology to each dataset containing basic geometric transformations (scaling, rotation, and translation) has led to several general conclusions:

- The selected minimal models share the same hyperparameter values. Specifically, the most decisive hyperparameter in model selection was the pooling size, which is set to 3 in all three cases.
- In terms of computational resources, all models contain approximately 21,000 parameters. This is considered relatively low for CNN architectures, which typically contain millions of parameters.
- Regarding training time, the models completed their training in 120, 149, and 171 epochs for scaling, rotation and translation problems respectively. These durations are reasonable, especially considering that the input dataset consists of 3,000 images.
- All models achieved 99% validation accuracy, indicating that the learned filters are effective and suitable for forming the knowledge base used in the next chapter. Each of the three extracted filters appears to specialize in detecting a specific type of basic geometric transformation.
- Based on validation and training accuracy, the complexity of transformations could be ranked from simplest to most complex as follows: scaling, translation, and rotation — with rotation being the most challenging, as the models found it more difficult to learn.
- The most frequently confused shapes are ellipses and rectangles. This is expected, given their visual similarity. Triangles, on the other hand, have more distinct shapes and are less often misclassified.

Considering all of the above, the first part of the thesis objective — “to obtain a knowledge base of hyperparameter magnitudes, model structure, and learned filters that enables intelligent initialization of a CNN architecture” — has been successfully achieved. The work can now progress to the next chapters. Additionally, the proposed methodology is generalizable and can be applied to any dataset to extract a more targeted knowledge base suited to a specific use case.

5

Filter-transfer learning

Knowledge has to be improved,
challenged and increased constantly,
or it vanishes.
Peter Drucker (1909–2005)

This chapter introduces a methodology for smart CNN initialization using a previously built knowledge base. By applying transfer learning with pre-selected filters and leveraging known hyperparameter and model structure patterns, the approach aims to reduce training cost and time. It is tested across four datasets and compared against models trained from scratch. Results highlight the benefits of guided initialization in both performance and efficiency.

To address the second part of the thesis objective, “intelligently initialize a CNN architecture, reducing time and computational costs compared to an architecture randomly initialized", a dedicated methodology has been developed. This approach tackles composite problems that combine those explored in Chapter 4, leveraging transfer learning with the filters extracted from that chapter. Additionally, it incorporates the knowledge base of hyperparameter magnitude orders and model structures to guide the design of new architectures. Once trained, these new models are compared to models trained from scratch, evaluating performance in terms of both accuracy and efficiency.

This chapter is organized into six sections. Section 5.1 describes the methodology for smart initialization, based primarily on the filters extracted in Chapter 4. Sections 5.2, 5.3, 5.4 and 5.5 present the application of this method to Datasets 4.1 through 7.1, respectively. These sections also include a comparative analysis between the proposed approach and traditional training from scratch. Finally, Section 5.6 summarizes the key insights of this chapter.

5.1. Proposed methodology

The methodology presented in this chapter is based on training a set of CNN architectures using a filter-based transfer learning approach. Unlike traditional transfer learning —which typically

involves using a pre-trained model as a base, freezing its initial layers and adding new layers on top for fine-tuning— this method extracts the filters learned by the initial models, combines them and integrates them into new architectures. The convolutional layers containing these transferred filters are then frozen and only the classification layers are trained, enabling more efficient learning while preserving the valuable representations captured in earlier stages.

All the new models trained using the filter-based transfer learning have the same architecture, that is composed by the following layers by order:

1. One **input layer** of size 28x28, the same as the dataset images.
2. One **convolutional layer** to extract features from the image by applying filters or kernels.
3. One **pooling layer** performing the operation of Max Pooling, extracting the maximum value for patches of the feature map resulting from the previous layer.
4. One **flatten layer** to convert the resultant bidimensional arrays into a single continuous linear vector.
5. One **dense layer** or a fully-connected layer of neurons to perform the classification operation taking as input the features extracted from the image.
6. One final **dense layer** as output layer of size 3, since this is the number of possible classes that can result from the classification problem.

A grid of hyperparameters is also constructed for addressing the problems presented in this chapter. The hpyerparameters selected for the grid and their possible values are shown below:

- **Seed**: it is used to initialized the random number generator with an specific value, ensuring reproducibility across experiments. In addition, several seeds are used since they initialization plays a key role when finding the optimal architecture. When exploring the space in search of a solution to the classification problem, depending on the initial point from which the CNN starts its search, it will find a better solution to the problem or not. If for different seeds the model is able to find a valid solution, it will mean that it has been properly adjusted. The seeds used are: 4, 1234, 10 and 550.
- **Number of filters**, also referred to as *nfilters_1st*: this refers to the total number of filters used in the first (and only) convolutional layer. Each filter is responsible for identifying specific features within the input data. For the models trained on Datasets 4.1, 5.1 and 6.1, this parameter is set to 2. For the models trained on Dataset 7.1, it is set to 3.
- **Kernel size**, also referred to as *ksize_1st*: size of the filters used in the first convolutional layer. In this case, the kernel size is set to 3 since the filters extracted have the dimension 3x3.
- **Pooling size**, also referred to as *poolsize_1st*: size of the window that slides across the image to perform the pooling operation. The bigger the window, the lesser number of resulting features. For the grid presented in this Thesis, this parameter takes the values: 6 (16 features), 5 (25 features) , 4 (49 features), 3 (81 features), 2 (196 features).
- **Number of neurons in the hidden layer**, also referred to as *hlneurons*: number of neurons used for the classification part of the network. Note that the classifier needs to be complex enough to derive classes from all the features extracted. In this way, the accuracy

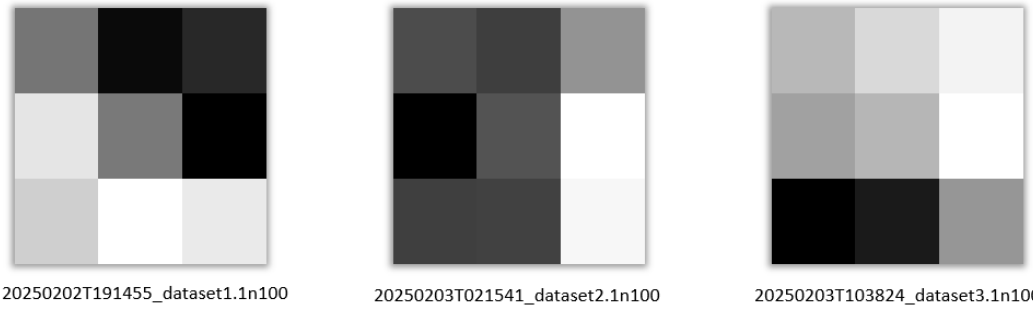
of the model depends entirely on the performance of the filters learnt. In this case, the parameter can take the values 250 or 500.

- **Number of epochs**, also referred to as *nepochs*: this parameter is not strictly fixed for the training but derived from the training itself. This means that a sufficiently high number of epochs, 5000, is set but the *Early Stopping* is enabled. This means that the training is stop when a monitored metric (in this case, the validation loss) has stopped improving (that is, it has stopped decreasing) so, in the end, the number of epochs varies from one model to another, depending on how well the training has been.

The convolutional component of the CNN architecture consists of three layers, as does the classification part. While the overall architecture mirrors the one introduced in Chapter 4, there are two key differences.

First, during training, the convolutional layers are frozen to preserve the filters transferred from the knowledge base, ensuring they remain unchanged. Figure 5.1 shows representation of the filters extracted from each model together with the basic transformation they detect.

Second, the values of certain hyperparameters —particularly the number of filters in the first convolutional layer— have been adjusted. In this chapter, the first convolutional layer contains either two or three filters, compared to just one in the previous chapter. This change reflects the increased complexity of the current problems, which involve combinations of geometric transformations. Specifically, Datasets 4.1 through 7.1 correspond to combinations of scaling plus rotation, scaling plus translation, rotation plus translation and all three transformations together, respectively. By using multiple filters, each targeting a specific transformation, the combined model is better equipped to handle more complex scenarios.



20250202T191455_dataset1.1n100_1L_nf1_1_ks1_3_ps1_3_s_550

-0.007	-0.402	-0.286
0.408	0.011	-0.441
0.325	0.505	0.427

SCALING

20250203T021541_dataset2.1n100_1L_nf1_1_ks1_3_ps1_3_s_1234

-0.021	-0.063	-0.186
-0.247	-0.002	0.505
-0.060	-0.051	0.479

ROTATION

20250203T103824_dataset3.1n100_1L_nf1_1_ks1_3_ps1_3_s_10

0.248	0.394	0.508
0.147	0.240	0.562
-0.557	-0.440	0.110

TRANSLATION

Figure 5.1. All three filters learnt by the minimum models selected for Datasets 1.1, 2.1 and 3.1, addressing the scaling, rotation and translation problems. Luminance representation on the top and the exact luminance values on the bottom.

Once all different models resulting from the grid have been trained, another set of parameters is used to compare them and chose which ones achieve the best results with the less complex architecture.

- **Number of parameters**, also referred to as *number_params*: this refers to the total number of parameters in the architecture, which varies depending on the specific combination of hyperparameters selected during the grid search. There is a direct relationship between the number of parameters and the training time: as the number of parameters increases, so does the training duration, indicating greater model complexity. A central objective of this thesis is to identify models that achieve optimal performance while maintaining a minimal number of parameters.

- **Training accuracy**: the percentage of correct predictions the CNN makes on the data it was trained on. It tells how well the model is learning to recognize patterns in the training set.

- **Validation accuracy**: the percentage of correct predictions the CNN makes on unseen data, called the validation set, which is not used during training. It tells how well the model is generalizing to new, unseen data. It is a better indicator of real-world performance than training accuracy.

As it happened in Chapter 4 the models are trained exclusively on the datasets containing noise. This decision is made to minimize potential biases and to better reflect real-world scenarios. A noise level of 100 is selected, as it is deemed sufficiently high to prevent bias while remaining low enough to avoid compromising the integrity of the dataset under analysis.

In conclusion, it is important to note that, in the following sections, models trained using the proposed transfer learning methodology are evaluated alongside models trained from scratch using the same hyperparameter grid. This allows for a fair comparison of performance metrics, particularly in terms of accuracy and training time. Regarding training times on a GPU model *NVIDA GeForce GTX1070*.

5.2. Dataset 4.1: Scaling plus rotation

The first problem addressed involves the combination of scaling and rotation transformations. Table 5.1 presents all models trained on Dataset 4.1 using the transfer learning approach described in Section 5.1. In contrast, Table 5.2 displays the same set of models trained from scratch. The primary objective is to compare the performance of models developed using both methodologies.

Transfer learning proves to be a more effective approach in this context. While training a model from scratch can occasionally yield better results, doing so typically demands a significantly larger dataset, as well as greater computational time and resources. In this particular problem, the number of parameters to estimate is disproportionately high relative to the available data, making transfer learning a more practical and reliable option. This scenario closely mirrors typical use cases of transfer learning, where a large pre-trained network is adapted to a more specific task with limited data.

Table 5.1. Model hyperparameters and results obtained after applying transfer learning for filters to train models for dataset 4.1 with noise level 100. Ordered by timestamp from earlier to later. *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 "pooling size of the first layer", *hlneurons* stands for "number of neurons in the hidden layer for the classification part of the model" *number_param* stands for "number of parameters of the model", *tr_acc* stands for "training accuracy" and *val_acc* stands for "validation accuracy", both values are represented with five decimals. Model names follow the structure: <model timestamp>_dataset4.1n100_1L_nf1_<nfilters_1st>_ks1_<ksize_1st>_ps1_<poolsize_1st>_s_<seed>. Groups of models with the same pooling size are separated from each other with double horizontal bars. The model selected as the best model is the one in bold.

Model timestamp ↓	seed	nepochs	nfilters_1st	ksize_1st	poolsize_1st	hlneurons	number_params	training_acc	val_acc
20250309T175950	4	151	2	3	6	250	9021	0.91767	0.82550
20250309T181736	1234	110	2	3	6	250	9021	0.88833	0.80033
20250309T183036	10	134	2	3	6	250	9021	0.91700	0.82233
20250309T184624	550	141	2	3	6	250	9021	0.90800	0.80300
20250309T190257	4	154	2	3	6	500	18021	0.96767	0.87383
20250309T192044	1234	119	2	3	6	500	18021	0.94867	0.85233
20250309T193434	10	157	2	3	6	500	18021	0.96533	0.86367
20250309T195244	550	146	2	3	6	500	18021	0.95533	0.86417
20250309T200937	4	195	2	3	5	250	13521	0.96467	0.85517
20250309T203207	1234	203	2	3	5	250	13521	0.96533	0.84900
20250309T205530	10	193	2	3	5	250	13521	0.95800	0.84933
20250309T211744	550	163	2	3	5	250	13521	0.95500	0.84367
20250309T213637	4	163	2	3	5	500	27021	0.98267	0.88533
20250309T215520	1234	128	2	3	5	500	27021	0.98133	0.89000
20250309T221010	10	119	2	3	5	500	27021	0.97367	0.88283
20250309T222357	550	109	2	3	5	500	27021	0.97300	0.87617
20250309T223638	4	146	2	3	4	250	25521	0.96367	0.86200
20250309T225335	1234	159	2	3	4	250	25521	0.95400	0.85367
20250309T231159	10	129	2	3	4	250	25521	0.94600	0.84867
20250309T232659	550	159	2	3	4	250	25521	0.95167	0.85650
20250309T234521	**4**	**141**	**2**	**3**	**4**	**500**	**51021**	**0.97967**	**0.89767**
20250310T000145	1234	103	2	3	4	500	51021	0.96033	0.87250
20250310T001346	10	113	2	3	4	500	51021	0.97133	0.87583
20250310T002654	550	133	2	3	4	500	51021	0.97533	0.89017
20250310T004220	4	119	2	3	3	250	41521	0.94900	0.84600
20250310T005610	1234	123	2	3	3	250	41521	0.95633	0.84967
20250310T011030	10	118	2	3	3	250	41521	0.95433	0.85267
20250310T012413	550	118	2	3	3	250	41521	0.94933	0.85150
20250310T013757	4	109	2	3	3	500	83021	0.96867	0.88433
20250310T015039	1234	98	2	3	3	500	83021	0.96333	0.88183
20250310T020205	10	99	2	3	3	500	83021	0.96333	0.88467
20250310T021339	550	125	2	3	3	500	83021	0.96767	0.87283

Table 5.2. Model hyperparameters and results obtained after training models from scratch for dataset 4.1 with noise level 100. *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 "pooling size of the first layer", *hlneurons* stands for "number of neurons in the hidden layer for the classification part of the model" *number_param* stands for "number of parameters of the model", *tr_acc* stands for "training accuracy" and *val_acc* stands for "validation accuracy", both values are represented with five decimals. Model names follow the structure: <model timestamp>_dataset4.1n100_1L_nf1_<nfilters_1st>_ks1_<ksize_1st>_ps1_<poolsize_1st>_s_<seed>. Groups of models with the same pooling size are separated from each other with double horizontal bars. The model selected as the best model is the one in bold.

Model timestamp ↓	**seed**	**nepochs**	**nfilters_1st**	**ksize_1st**	**poolsize_1st**	**hlneurons**	**number_params**	**training_acc**	**val_acc**
20250209T121225	4	149	2	3	6	250	9023	0.98267	0.78400
20250209T122941	1234	190	2	3	6	250	9023	0.94133	0.84583
20250209T125136	10	192	2	3	6	250	9023	0.98433	0.84833
20250209T131347	550	140	2	3	6	250	9023	0.86633	0.77550
20250209T133000	4	111	2	3	6	500	18023	0.99967	0.81100
20250209T134253	1234	135	2	3	6	500	18023	0.97000	0.87967
20250209T135831	10	113	2	3	6	500	18023	0.98967	0.86767
20250209T141141	550	149	2	3	6	500	18023	0.92233	0.81867
20250209T142857	4	136	2	3	5	250	13523	0.99700	0.76533
20250209T144445	1234	203	2	3	5	250	13523	0.97133	0.88050
20250209T150814	10	209	2	3	5	250	13523	0.99733	0.86150
20250209T153225	550	185	2	3	5	250	13523	0.93967	0.76083
20250209T155351	4	108	2	3	5	500	27023	1.00000	0.81983
20250209T160627	1234	127	2	3	5	500	27023	0.97833	0.90617
20250209T162114	10	102	2	3	5	500	27023	0.99500	0.87600
20250209T163306	550	144	2	3	5	500	27023	0.97233	0.81467
20250209T164947	4	102	2	3	4	250	25523	0.99967	0.73517
20250209T170139	1234	162	2	3	4	250	25523	0.97733	0.87267
20250209T172022	10	132	2	3	4	250	25523	0.99267	0.82933
20250209T173538	550	149	2	3	4	250	25523	0.93533	0.77283
20250209T175253	4	82	2	3	4	500	51023	1.00000	0.77733
20250209T180228	**1234**	**114**	**2**	**3**	**4**	**500**	**51023**	**0.97367**	**0.89533**
20250209T181543	10	117	2	3	4	500	51023	1.00000	0.88033
20250209T182919	550	118	2	3	4	500	51023	0.98567	0.80100
20250209T184300	4	88	2	3	3	250	41523	1.00000	0.74683
20250209T185317	1234	110	2	3	3	250	41523	0.96633	0.85967
20250209T190604	10	143	2	3	3	250	41523	0.99967	0.83633
20250209T192237	550	125	2	3	3	250	41523	0.98633	0.80333
20250209T193707	4	80	2	3	3	500	83023	1.00000	0.78883
20250209T194629	1234	91	2	3	3	500	83023	0.98433	0.90700
20250209T195709	10	116	2	3	3	500	83023	1.00000	0.87050
20250209T201038	550	104	2	3	3	500	83023	0.99500	0.82850

Among the models trained, the best-performing model is the one with 51k parameters. With a pool size of 4, the resulting feature map is 7×7 pixels (i.e., 49 features). The effectiveness of this configuration stems from two main factors. First, since 4 is a divisor of 28 (the image size), the pooling operation produces feature maps where all values represent meaningful regions

of the image, eliminating the need for padding. Second, the pooling size is small enough to capture detailed local features, yet not so small that it sacrifices the network's ability to learn more general, global characteristics, such as the position of the object within the image.

In the 83k-parameter model, transfer learning consistently produces models with similar accuracy across different random seeds, indicating greater stability during training, obtaining a standard deviation of the accuracy of 0.012 in the case of transfer learning compared to 0.058 in the case of training from scratch. In contrast, training from scratch shows more pronounced overfitting and higher variability depending on the seed used. A similar trend is observed in the 51k-parameter model, where transfer learning also demonstrates better generalization capabilities.

Furthermore, models with less complex classifiers (250 neurons, resulting in 25k and 41k parameters) perform worse than the version with 500 neurons. This highlights the importance of having sufficient classifier capacity to fully exploit the rich feature representations provided by the convolutional layers.

To conclude this section, there is not a great difference in the number of epochs to complete the training with transfer learning (141 epochs) in comparison with the one trained from scratch (114 epochs). In terms of time (executing on *NVIDA GeForce GTX1070*), the first one takes 16 minutes and 32 seconds to train whereas the second one takes 13 minutes and 15 seconds. This fact seems a bit counterintuitive but it could be due to the fact that transfer learning often uses lower learning rates to avoid destroying pretrained weights, which can lead to slower convergence. From-scratch models can often be trained more aggressively and, in this case, the seed selected for training from scratch seems to converge quicker to the solution. All in all, the difference in training time is not remarkably high.

5.3. Dataset 5.1: Scaling plus translation

As it happened in the previous case, generally transfer learning proves to be the superior approach for this problem. While it is possible to achieve good results from scratch, doing so requires significantly more data, time and computational resources—conditions that are not met in our scenario. The number of parameters to estimate is simply too high relative to the available sample size. This is a common challenge when adapting a large pre-trained network to a specific task for which limited data is available.

The best-performing model under transfer learning had just 18,000 parameters, contrary to the 83,000 parameters in the best model trained from scratch. This smaller model benefits greatly from a pooling size of 6, which produces a 5x5 feature map; that is, 25 features. The improved performance can be attributed to the larger pooling window capturing more global features, an essential quality when addressing translation problems, where identifying the location of an object within an image is a prerequisite to analyzing its geometric form.

Moreover, the 18k-parameter model consistently achieves similar accuracy across different random seeds, obtaining a standard deviation of the accuracy of 0.006 in the case of transfer learning compared to 0.019 in the case of training from scratch. This fact highlights the stability of the training process when using transfer learning. In contrast, training from scratch not only led to higher levels of overfitting but also greater variability in performance depending on the initialization. Simpler classifier architectures —specifically those using 250 neurons and comprising 9k and 13k parameters— performed noticeably worse than the more complex

500-neuron version. This clearly indicates that it is necessary a certain level of model complexity in the classifier part of the architecture to effectively exploit the extracted features.

Table 5.3. Model hyperparameters and results obtained after applying transfer learning for filters to train models for dataset 5.1 with noise level 100. *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 "pooling size of the first layer", *hlneurons* stands for "number of neurons in the hidden layer for the classification part of the model" *number_param* stands for "number of parameters of the model", *tr_acc* stands for "training accuracy" and *val_acc* stands for "validation accuracy", both values are represented with five decimals. Model names follow the structure: <model timestamp>_dataset5.1n100_1L_nf1_<nfilters_1st>_ks1_<ksize_1st>_ps1_<poolsize_1st>_s_<seed>.Groups of models with the same pooling size are separated from each other with double horizontal bars. The model selected as the best model is the one in bold.

Model timestamp ↓	**seed**	**nepochs**	**nfilters_1st**	**ksize_1st**	**poolsize_1st**	**hlneurons**	**number_params**	**training_acc**	**val_acc**
20250329T105123	4	175	2	3	6	250	9021	0.97300	0.90450
20250329T111154	1234	200	2	3	6	250	9021	0.98100	0.90167
20250329T113511	10	194	2	3	6	250	9021	0.97533	0.90717
20250329T115745	550	128	2	3	6	250	9021	0.95067	0.88700
20250329T121244	4	135	2	3	6	500	18021	0.98833	0.91850
20250329T122836	**1234**	**152**	**2**	**3**	**6**	**500**	**18021**	**0.99033**	**0.93117**
20250329T124624	10	137	2	3	6	500	18021	0.98400	0.92117
20250329T130223	550	149	2	3	6	500	18021	0.98967	0.92167
20250329T131950	4	162	2	3	5	250	13521	0.98600	0.90883
20250329T133844	1234	144	2	3	5	250	13521	0.97000	0.89933
20250329T135537	10	89	2	3	5	250	13521	0.95367	0.88350
20250329T140610	550	136	2	3	5	250	13521	0.97300	0.89500
20250329T142210	4	95	2	3	5	500	27021	0.98733	0.91650
20250329T143326	1234	143	2	3	5	500	27021	0.99133	0.92133
20250329T145010	10	82	2	3	5	500	27021	0.98200	0.90733
20250329T145953	550	118	2	3	5	500	27021	0.99300	0.92267
20250329T151348	4	116	2	3	4	250	25521	0.95100	0.87767
20250329T152726	1234	119	2	3	4	250	25521	0.95967	0.88667
20250329T154126	10	114	2	3	4	250	25521	0.94633	0.88517
20250329T155449	550	92	2	3	4	250	25521	0.93867	0.86817
20250329T160541	4	75	2	3	4	500	51021	0.95867	0.89300
20250329T161434	1234	65	2	3	4	500	51021	0.95267	0.89333
20250329T162219	10	87	2	3	4	500	51021	0.96400	0.89433
20250329T163236	550	78	2	3	4	500	51021	0.96067	0.89750
20250329T164149	4	98	2	3	3	250	41521	0.95033	0.87433
20250329T165321	1234	131	2	3	3	250	41521	0.95767	0.87950
20250329T170843	10	106	2	3	3	250	41521	0.94400	0.87733
20250329T172113	550	99	2	3	3	250	41521	0.95067	0.88333
20250329T173253	4	66	2	3	3	500	83021	0.94867	0.88400
20250329T174044	1234	57	2	3	3	500	83021	0.95533	0.89133
20250329T180132	10	67	2	3	3	500	83021	0.95233	0.88283
20250329T180931	550	98	2	3	3	500	83021	0.96633	0.89867

Table 5.4. Model hyperparameters and results obtained after training models from scratch for dataset 5.1 with noise level 100. *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 "pooling size of the first layer", *hlneurons* stands for "number of neurons in the hidden layer for the classification part of the model" *number_param* stands for "number of parameters of the model", *tr_acc* stands for "training accuracy" and *val_acc* stands for "validation accuracy", both values are represented with five decimals. Model names follow the structure: <model timestamp>_dataset5.1n100_1L_nf1_<nfilters_1st>_ks1_<ksize_1st>_ps1_<poolsize_1st>_s_<seed>.Groups of models with the same pooling size are separated from each other with double horizontal bars. The model selected as the best model is the one in bold.

Model timestamp ↓	**seed**	**nepochs**	**nfilters_1st**	**ksize_1st**	**poolsize_1st**	**hlneurons**	**number_params**	**training_acc**	**val_acc**
20250209T221541	4	129	2	3	6	250	9023	0.99900	0.89567
20250209T223036	1234	177	2	3	6	250	9023	0.95333	0.87450
20250209T225058	10	136	2	3	6	250	9023	0.99100	0.89800
20250209T230639	550	162	2	3	6	250	9023	0.97067	0.88117
20250209T232520	4	95	2	3	6	500	18023	1.00000	0.90283
20250209T233624	1234	136	2	3	6	500	18023	0.98033	0.91300
20250209T235206	10	108	2	3	6	500	18023	0.99800	0.91400
20250210T000436	550	129	2	3	6	500	18023	0.99100	0.89750
20250210T001932	4	108	2	3	5	250	13523	0.99967	0.89633
20250210T003203	1234	147	2	3	5	250	13523	0.97000	0.88133
20250210T004901	10	148	2	3	5	250	13523	0.99900	0.90317
20250210T010605	550	158	2	3	5	250	13523	0.98467	0.87367
20250210T012416	4	72	2	3	5	500	27023	1.00000	0.89967
20250210T013241	1234	129	2	3	5	500	27023	0.98733	0.90250
20250210T014734	10	72	2	3	5	500	27023	0.99767	0.90150
20250210T015600	550	116	2	3	5	500	27023	0.99700	0.89400
20250210T020925	4	93	2	3	4	250	25523	1.00000	0.87433
20250210T022003	1234	111	2	3	4	250	25523	0.98300	0.87800
20250210T023255	10	147	2	3	4	250	25523	1.00000	0.90267
20250210T024951	550	148	2	3	4	250	25523	0.99200	0.86550
20250210T030655	4	76	2	3	4	500	51023	1.00000	0.89450
20250210T031548	1234	91	2	3	4	500	51023	0.99100	0.90267
20250210T032624	10	95	2	3	4	500	51023	1.00000	0.91583
20250210T033727	550	97	2	3	4	500	51023	0.99867	0.89033
20250210T034844	4	70	2	3	3	250	41523	0.99967	0.86717
20250210T035657	1234	101	2	3	3	250	41523	0.97067	0.87550
20250210T040842	10	141	2	3	3	250	41523	1.00000	0.90500
20250210T042459	550	97	2	3	3	250	41523	0.99067	0.85850
20250210T043616	4	56	2	3	3	500	83023	1.00000	0.87950
20250210T044253	1234	78	2	3	3	500	83023	0.98100	0.89450
20250210T045201	**10**	**97**	**2**	**3**	**3**	**500**	**83023**	**1.00000**	**0.92167**
20250210T050318	550	99	2	3	3	500	83023	1.00000	0.88567

Ultimately, there is a clear difference in the number of epochs required to complete training: the transfer learning model needed 152 epochs, while the model trained from scratch required only 95. In terms of training time (executing on *NVIDA GeForce GTX1070*), the transfer learning model took 17 minutes and 48 seconds, compared to 11 minutes and 17 seconds for the

from-scratch model. Although the difference in training time is not particularly large, it can be attributed to the fact that transfer learning often uses lower learning rates to protect the pretrained weights, which slows down convergence. In contrast, models trained from scratch typically allow for more aggressive optimization, and in this case, the selected random seed likely contributed to the model reaching a solution more quickly.

5.4. Dataset 6.1: Rotation plus translation

Transfer learning proves to be the better approach. As in previous cases, better performance can sometimes be achieved when training from scratch, but only if there is a significantly larger dataset available, along with sufficient time and computational resources. That is not the case here — there are too many parameters to estimate with a relatively small sample size. This is a common challenge when applying transfer learning from a large pretrained network to a specific task where data is limited.

In this case, the best-performing model has 83k parameters. Interestingly, the top model using transfer learning share the same number of parameters as the one trained from scratch. With a pooling size of 3, the resulting feature map is 9x9 pixels, giving 81 features in total. This configuration leads to better results for two main reasons. First, 3 is not a divisor of 28 (the original image size), so the output is a 9x9 matrix without applying padding. As a result, the last row and column of pixels are not included. However, this doesn't significantly impact performance because the generated figures are positioned with a margin between the image border and the enclosing circle. Therefore, the features extracted after pooling are still highly representative of the image. Second, a pooling size of 3 is small enough to capture local features with high detail about the shape of the figure, while still being large enough to retain some global context, such as the figure's position in the image. This pooling size is also very close to 4, which yielded good results in the scaling-plus-rotation scenario (Dataset 4.1).

When using the 83k-parameter model with transfer learning, the different seeds tested produce very similar accuracy scores, indicating greater stability during the training process. In contrast, training from scratch showed greater overfitting and higher variability in the final results depending on the random seed, obtaining a standard deviation of the accuracy of 0.015 in the case of transfer learning compared to 0.087 in the case of training from scratch.

Overall, in the transfer learning approach, models with the same number of parameters tend to produce more consistent results across different seeds — that is, they generalize better than those trained from scratch. This is clearly observed in models with 41k parameters.

Models with less complex classifiers (25k and 41k parameters) perform worse than the 500-neuron version, providing clear evidence that higher model complexity for the classification part of the CNN is necessary to fully exploit the features extracted.

Ultimately, there is a clear difference in the number of epochs required to complete training: the transfer learning model needed 93 epochs, while the model trained from scratch required only 69. In terms of training time (executing on *NVIDA GeForce GTX1070*), the transfer learning model took 10 minutes and 53 seconds, compared to 8 minutes and 6 seconds for the from-scratch model. Although the difference in training time is not particularly large, it can be attributed to the fact that transfer learning often uses lower learning rates to protect the pretrained weights, which slows down convergence. In contrast, models trained from scratch typically allow for more aggressive optimization, and in this case, the selected random seed likely contributed to the model reaching a solution more quickly.

Table 5.5. Model hyperparameters and results obtained after applying transfer learning for filters to train models for dataset 6.1 with noise level 100. *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 "pooling size of the first layer", textithlneurons stands for "number of neurons in the hidden layer for the classification part of the model" *number_param* stands for "number of parameters of the model", *tr_acc* stands for "training accuracy" and *val_acc* stands for "validation accuracy", both values are represented with five decimals. Model names follow the structure: <model timestamp>_dataset6.1n100_1L_nf1_<nfilters_1st>_ks1_<ksize_1st>_ps1_<poolsize_1st>_s_<seed>. Groups of models with the same pooling size are separated from each other with double horizontal bars. The model selected as the best model is the one in bold.

Model timestamp ↓	**seed**	**nepochs**	**nfilters_1st**	**ksize_1st**	**poolsize_1st**	**hlneurons**	**number_params**	**training_acc**	**val_acc**
20250310T231624	4	170	2	3	6	250	9021	0.82233	0.66617
20250310T233605	1234	133	2	3	6	250	9021	0.80533	0.65400
20250310T235129	10	171	2	3	6	250	9021	0.83367	0.67533
20250311T001114	550	182	2	3	6	250	9021	0.82600	0.66617
20250311T003223	4	105	2	3	6	500	18021	0.86600	0.69867
20250311T004439	1234	165	2	3	6	500	18021	0.8900	0.70367
20250311T010346	10	119	2	3	6	500	18021	0.84733	0.69700
20250311T011738	550	114	2	3	6	500	18021	0.87633	0.69817
20250311T013059	4	107	2	3	5	250	13521	0.84867	0.68467
20250311T014328	1234	133	2	3	5	250	13521	0.87433	0.71267
20250311T015854	10	153	2	3	5	250	13521	0.87100	0.70683
20250311T021641	550	172	2	3	5	250	13521	0.88833	0.72250
20250311T023643	4	107	2	3	5	500	27021	0.92133	0.73683
20250311T024920	1234	121	2	3	5	500	27021	0.93067	0.74533
20250311T030326	10	134	2	3	5	500	27021	0.93333	0.76100
20250311T031900	550	116	2	3	5	500	27021	0.92867	0.75167
20250311T033231	4	145	2	3	4	250	25521	0.88233	0.70683
20250311T034920	1234	137	2	3	4	250	25521	0.87433	0.71250
20250311T040513	10	123	2	3	4	250	25521	0.87767	0.71083
20250311T041931	550	116	2	3	4	250	25521	0.85200	0.69733
20250311T043302	4	104	2	3	4	500	51021	0.91033	0.76633
20250311T044511	1234	100	2	3	4	500	51021	0.90700	0.75167
20250311T045657	10	79	2	3	4	500	51021	0.89733	0.73817
20250311T050617	550	65	2	3	4	500	51021	0.88733	0.73467
20250311T051359	4	98	2	3	3	250	41521	0.85900	0.72500
20250311T052527	1234	99	2	3	3	250	41521	0.83133	0.6895
20250311T053702	10	122	2	3	3	250	41521	0.88300	0.72217
20250311T055114	550	128	2	3	3	250	41521	0.88533	0.72967
20250311T060608	**4**	**93**	**2**	**3**	**3**	**500**	**83021**	**0.91767**	**0.77117**
20250311T061701	1234	91	2	3	3	500	83021	0.89667	0.76033
20250311T062740	10	64	2	3	3	500	83021	0.88233	0.73600
20250311T063515	550	83	2	3	3	500	83021	0.90033	0.75300

Table 5.6. Model hyperparameters and results obtained after training models from scratch for dataset 6.1 with noise level 100. *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 "pooling size of the first layer", *hlneurons* stands for "number of neurons in the hidden layer for the classification part of the model" *number_param* stands for "number of parameters of the model", *tr_acc* stands for "training accuracy" and *val_acc* stands for "validation accuracy", both values are represented with five decimals. Model names follow the structure: <model timestamp>_dataset6.1n100_1L_nf1_<nfilters_1st>_ks1_<ksize_1st>_ps1_<poolsize_1st>_s_<seed>. Groups of models with the same pooling size are separated from each other with double horizontal bars. The model selected as the best model is the one in bold.

Model timestamp ↓	**seed**	**nepochs**	**nfilters_1st**	**ksize_1st**	**poolsize_1st**	**hlneurons**	**number_params**	**training_acc**	**val_acc**
20250210T065701	4	107	2	3	6	250	9023	0.94567	0.64450
20250210T070923	1234	111	2	3	6	250	9023	0.79300	0.67017
20250210T072213	10	142	2	3	6	250	9023	0.92100	0.65317
20250210T073834	550	157	2	3	6	250	9023	0.80467	0.64700
20250210T075639	4	84	2	3	6	500	18023	0.98867	0.67950
20250210T080628	1234	71	2	3	6	500	18023	0.83267	0.69600
20250210T081446	10	100	2	3	6	500	18023	0.95333	0.68383
20250210T082621	550	106	2	3	6	500	18023	0.84667	0.67667
20250210T083838	4	108	2	3	5	250	13523	0.98667	0.65450
20250210T085107	1234	130	2	3	5	250	13523	0.91433	0.73833
20250210T090606	10	141	2	3	5	250	13523	0.97233	0.71400
20250210T092221	550	141	2	3	5	250	13523	0.88333	0.67600
20250210T093835	4	82	2	3	5	500	27023	1.00000	0.68850
20250210T094809	1234	92	2	3	5	500	27023	0.93667	0.75617
20250210T095850	10	100	2	3	5	500	27023	0.99267	0.72417
20250210T101026	550	95	2	3	5	500	27023	0.90667	0.70283
20250210T102129	4	64	2	3	4	250	25523	0.98667	0.56650
20250210T102859	1234	116	2	3	4	250	25523	0.92667	0.74933
20250210T104226	10	108	2	3	4	250	25523	0.9660	0.62833
20250210T105456	550	118	2	3	4	250	25523	0.92667	0.70200
20250210T110834	4	64	2	3	4	500	51023	1.00000	0.62600
20250210T111604	1234	94	2	3	4	500	51023	0.96900	0.77767
20250210T112659	10	89	2	3	4	500	51023	0.99933	0.70183
20250210T113721	550	83	2	3	4	500	51023	0.94667	0.72917
20250210T114701	4	52	2	3	3	250	41523	0.99633	0.54650
20250210T115309	1234	95	2	3	3	250	41523	0.93800	0.74950
20250210T120412	10	101	2	3	3	250	41523	0.99000	0.63617
20250210T121555	550	95	2	3	3	250	41523	0.96467	0.71750
20250210T122656	4	43	2	3	3	500	83023	1.00000	0.58733
20250210T123204	**1234**	**69**	**2**	**3**	**3**	**500**	**83023**	**0.94833**	**0.78083**
20250210T124010	10	68	2	3	3	500	83023	0.99967	0.64583
20250210T124755	550	73	2	3	3	500	83023	0.99200	0.73383

5.5. Dataset 7.1: Scaling, rotation and translation

Neither of the two model types achieves high levels of accuracy in this scenario. Nevertheless, transfer learning performs better overall. As it happened with the previous cases, while it is theoretically possible to achieve better results from scratch, doing so would require a significantly larger dataset, along with the necessary time and computational resources — which is not the case here. With such a small sample size relative to the number of parameters, it becomes difficult to estimate the model properly. This is a common issue when applying transfer learning from a large pretrained network to a specific task where data is limited.

The best-performing model in this case has 26k parameters. With a pooling size of 6, the resulting feature map is 5x5 pixels — that is, 25 features. One possible explanation for this result is that a larger pooling size leads to more global features, which aligns well with the challenge of handling translation. In such tasks, it is important for the model to first determine the location of the figure within the image before analyzing its geometric shape. However, given the model's relatively low precision, its logic may be questionable. This suggests the model likely requires more complexity to solve the problem effectively.

With the 26k-parameter model, transfer learning yields very consistent accuracy across different seeds, indicating greater stability during training. In contrast, the from-scratch approach shows more overfitting and greater variability depending on the random seed used, obtaining a standard deviation of the accuracy of 0.007 in the case of transfer learning compared to 0.017 in the case of training from scratch.

Models with less complex classifiers (such as the 25k and 41k versions) perform worse than those with 500 neurons, clearly demonstrating that higher model complexity is necessary to fully leverage the extracted features.

To end this section, there is a slightly difference in the number of epochs required to complete training: the transfer learning model needed 125 epochs, while the model trained from scratch required 108. In terms of training time (executing on *NVIDA GeForce GTX1070*), the transfer learning model took 14 minutes and 39 seconds, compared to 12 minutes and 49 seconds for the from-scratch model. Although the difference in training time is not particularly large, it can be attributed to the fact that transfer learning often uses lower learning rates to protect the pretrained weights, which slows down convergence. In contrast, models trained from scratch typically allow for more aggressive optimization, and in this case, the selected random seed likely contributed to the model reaching a solution more quickly.

Table 5.7. Model hyperparameters and results obtained after applying transfer learning for filters to train models for dataset 7.1 with noise level 100. *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 "pooling size of the first layer", *hlneurons* stands for "number of neurons in the hidden layer for the classification part of the model" *number_param* stands for "number of parameters of the model", *tr_acc* stands for "training accuracy" and *val_acc* stands for "validation accuracy", both values are represented with five decimals. Model names follow the structure: <model timestamp>_dataset6.1n100_1L_nf1_<nfilters_1st>_ks1_<ksize_1st>_ps1_<poolsize_1st>_s_<seed>. Groups of models with the same pooling size are separated from each other with double horizontal bars. The model selected as the best model is the one in bold.

Model timestamp ↓	**seed**	**nepochs**	**nfilters_1st**	**ksize_1st**	**poolsize_1st**	**hlneurons**	**number_params**	**training_acc**	**val_acc**
20250329T182104	4	152	3	3	6	250	13030	0.83700	0.62650
20250329T183851	1234	130	3	3	6	250	13030	0.81367	0.61417
20250329T185406	10	85	3	3	6	250	13030	0.76300	0.59767
20250329T190410	550	116	3	3	6	250	13030	0.78967	0.60783
20250329T191747	4	109	3	3	6	500	26030	0.86367	0.66133
20250329T193037	1234	102	3	3	6	500	26030	0.88333	0.66450
20250329T194237	10	95	3	3	6	500	26030	0.86067	0.66483
20250329T195350	**550**	**125**	**3**	**3**	**6**	**500**	**26030**	**0.89933**	**0.67817**
20250329T200829	4	84	3	3	5	250	19780	0.76300	0.60083
20250329T201825	1234	100	3	3	5	250	19780	0.79033	0.61967
20250329T203009	10	67	3	3	5	250	19780	0.73167	0.57083
20250329T203806	550	116	3	3	5	250	19780	0.80867	0.60383
20250329T205144	4	116	3	3	5	500	39530	0.90333	0.67750
20250329T210519	1234	80	3	3	5	500	39530	0.85667	0.63733
20250329T211445	10	100	3	3	5	500	39530	0.88100	0.66883
20250329T212631	550	75	3	3	5	500	39530	0.83933	0.63383
20250329T213524	4	82	3	3	4	250	37780	0.72933	0.59167
20250329T214506	1234	88	3	3	4	250	37780	0.73167	0.59983
20250329T215531	10	51	3	3	4	250	37780	0.67300	0.54750
20250329T220138	550	86	3	3	4	250	37780	0.74167	0.5820
20250329T221149	4	66	3	3	4	500	75530	0.77567	0.59817
20250329T221941	1234	53	3	3	4	500	75530	0.76333	0.60400
20250329T222603	10	70	3	3	4	500	75530	0.78033	0.61067
20250329T223423	550	55	3	3	4	500	75530	0.75533	0.59133
20250329T224059	4	75	3	3	3	250	61780	0.72300	0.56167
20250329T224954	1234	46	3	3	3	250	61780	0.66200	0.53633
20250329T225523	10	74	3	3	3	250	61780	0.71400	0.57083
20250329T230403	550	51	3	3	3	250	61780	0.65900	0.55083
20250329T231006	4	61	3	3	3	500	123530	0.76433	0.59017
20250329T231721	1234	77	3	3	3	500	123530	0.77500	0.60917
20250329T232614	10	34	3	3	3	500	123530	0.67533	0.55817
20250329T233016	550	79	3	3	3	500	123530	0.7700	0.59650

Table 5.8. Model hyperparameters and results obtained after training models from scratch for dataset 7.1 with noise level 100. *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 "pooling size of the first layer", textithlneurons stands for "number of neurons in the hidden layer for the classification part of the model" *number_param* stands for "number of parameters of the model", *tr_acc* stands for "training accuracy" and *val_acc* stands for "validation accuracy", both values are represented with five decimals. Model names follow the structure: <model timestamp>_dataset7.1n100_1L_nf1_<nfilters_1st>_ks1_<ksize_1st>_ps1_<poolsize_1st>_s_<seed>. Groups of models with the same pooling size are separated from each other with double horizontal bars. The model selected as the best model is the one in bold.

Model timestamp ↓	**seed**	**nepochs**	**nfilters_1st**	**ksize_1st**	**poolsize_1st**	**hlneurons**	**number_params**	**training_acc**	**val_acc**
20250330T082542	4	121	3	3	6	250	13033	0.86900	0.60500
20250330T083956	1234	154	3	3	6	250	13033	0.79700	0.59383
20250330T085756	10	101	3	3	6	250	13033	0.83033	0.60400
20250330T090951	550	109	3	3	6	250	13033	0.77133	0.59217
20250330T092240	4	108	3	3	6	500	26033	0.97567	0.65750
20250330T093526	1234	99	3	3	6	500	26033	0.81100	0.61850
20250330T094708	10	85	3	3	6	500	26033	0.94400	0.64383
20250330T095712	550	102	3	3	6	500	26033	0.86400	0.636500
20250330T100910	4	109	3	3	5	250	19783	0.88667	0.62400
20250330T102201	1234	119	3	3	5	250	19783	0.80067	0.59733
20250330T103600	10	113	3	3	5	250	19783	0.89467	0.61217
20250330T104917	550	139	3	3	5	250	19783	0.85067	0.62417
20250330T110536	4	95	3	3	5	500	39533	0.98167	0.65667
20250330T111652	1234	91	3	3	5	500	39533	0.88467	0.63183
20250330T112745	10	85	3	3	5	500	39533	0.97300	0.65533
20250330T113754	**550**	**108**	**3**	**3**	**5**	**500**	**39533**	**0.91100**	**0.67367**
20250330T115043	4	70	3	3	4	250	37783	0.90833	0.57700
20250330T115908	1234	91	3	3	4	250	37783	0.78033	0.56717
20250330T120956	10	94	3	3	4	250	37783	0.94100	0.61150
20250330T122106	550	91	3	3	4	250	37783	0.79633	0.58833
20250330T123152	4	68	3	3	4	500	75533	0.98233	0.61900
20250330T124002	1234	90	3	3	4	500	75533	0.86633	0.62117
20250330T125045	10	72	3	3	4	500	75533	0.98667	0.65533
20250330T125920	550	87	3	3	4	500	75533	0.86167	0.61750
20250330T130941	4	66	3	3	3	250	61783	0.93400	0.57800
20250330T131738	1234	102	3	3	3	250	61783	0.82267	0.59200
20250330T132947	10	63	3	3	3	250	61783	0.94100	0.59717
20250330T133722	550	56	3	3	3	250	61783	0.76933	0.55617
20250330T134409	4	52	3	3	3	500	123533	0.98900	0.61900
20250330T135027	1234	78	3	3	3	500	123533	0.83567	0.62133
20250330T135952	10	65	3	3	3	500	123533	0.99467	0.64967
20250330T140745	550	68	3	3	3	500	123533	0.87500	0.62433

5.6. Conclusion

By applying the methodology described in the first section of this chapter to datasets with combinations of transformations, the following conclusions were drawn:

- The filter-transfer learning technique enables the resolution of more complex problems with high accuracy. This goal is achieved this by combining filters specialized in detecting basic transformations, reducing computational costs while keeping training times within reasonable limits.
- Models trained using transfer learning exhibit greater stability compared to those trained from scratch, as they tend to achieve similar accuracy levels across different random seeds.
- When using filter-based transfer learning, it is essential to use a CNN architecture with a sufficiently complex classifier. This ensures that the network can fully leverage the information extracted by the filters.
- Regarding the level of complexity for combinations of two transformations, *scaling plus translation* is the easiest problem to solve. The network requires fewer parameters (18k) to achieve a performance of 93% in this setting. The combination of transformations *scaling plus rotation* represents a problem of intermediate difficulty, as the network requires a moderate number of parameters (51k) to get a 89% of accuracy. Lastly, *rotation and translation* turns out to be the most difficult scenario to solve, as the network requires more parameters (83k) to achive a 77% of accuracy.
- In general, rotation appears to be the most challenging transformation for networks to detect. Models tend to require a higher number of parameters to correctly identify the geometric shapes when rotation is involved.
- The combination of all three transformations (scaling, rotation and translation) represents by far the most challenging problem among all scenarios. Even with a parameter count not so high (26k) the resulting accuracy is significantly lower, of 68% approximately.
- Models with transfer learning take as average 14 minutes and 56 seconds to complete training whereas models trained from scratch take an average of 11 minutes and 22 seconds. Therefore, the average time difference between the two training approaches (transfer learning vs. from scratch) is 3 minutes and 34 seconds, with transfer learning taking longer. However, this is not significant in the context of image recognition models, which can often take hours or even days to train.
- The training time difference is largely due to the fact that transfer learning typically uses lower learning rates to preserve pretrained weights, which slows convergence. In contrast, models trained from scratch can use more aggressive optimization strategies, and in this case, the random seed used may have helped the model converge faster. Nevertheless, the time trade-off is acceptable given the improvements in both accuracy and model stability that transfer learning provides.

Taking all these factors into account, the second objective of this thesis — "intelligently initializing a CNN architecture to reduce time and computational costs compared to random initialization" — has been successfully achieved. Furthermore, the proposed methodology is generalizable, and the filter-transfer learning approach can be applied to any dataset involving images with basic transformations.

6

Conclusion and future work

Learning never exhausts the mind.
Leonardo da Vinci (1452–1519)

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.

6.1. Summary and conclusion

Convolutional Neural Networks (CNNs) have become a central tool in computer vision, enabling real-time and high-precision tasks across industries such as manufacturing, healthcare, and autonomous systems. Traditional CNNs, while powerful, suffer from drawbacks including high computational demands, long training times, low interpretability and dependency on large labeled datasets. These issues hinder their scalability and practical deployment, particularly in resource-constrained environments like the industrial ones.

Due to this fact, there is a clear trend toward developing smaller, more efficient CNN architectures. These lightweight models are especially suited for edge computing and embedded devices, where power, speed and explainability are critical factors. Additionally, reducing the ecological footprint of AI systems has become increasingly important. Lightweight CNNs not only improve deployment efficiency but also align with sustainable AI practices by minimizing energy consumption.

This thesis introduces a bottom-up approach to CNN architecture design, starting from minimal complexity and adding parameters only as needed. This contrasts with traditional top-down methods that trim large models post hoc, often inefficiently. Furthermore, the use of synthetic datasets with basic geometric transformations (scaling, rotation and translation) applied to classical geometric figures as ellipses, rectangles and triangles, is the key to successfully constructing a knowledge base of optimal hyperparameters, architecture structures and transformation-specific filters.

The models designed to address a single specific transformation achieved up to 99% accuracy with only 21k parameters approximately, validating the effectiveness of the proposed method for simpler transformations. In relation to these models, rotation is consistently the most difficult transformation for CNNs to learn, requiring more parameters and training time. Ellipses and rectangles are the most commonly confused shapes due to their visual similarity whereas triangles are normally distinguished easily.

The use of filter-transfer learning allows for tackling more complex transformation combinations with improved stability and accuracy compared to training from scratch. It also helps generalize the methodology across different problems. The knowledge base built previously from the training of minimal models proves to be effective for smart initializating CNN architectures. It is important to highglight the impact of having a classifier complex enough on the CNN architecture, so all the information contained in the features is properly extracted.

For both methodologies, the use of different seeds ensures the robustness of the model architecture, as it allows for the evaluation of the model's stability and consistency across multiple training runs. This approach helps verify that the observed performance is not the result of a favorable initialization but rather a reflection of the architecture's true capability. Consistent results across various seeds indicate that the model generalizes well and is less sensitive to random variations during training.

Regarding the complexity level of transformation combinations:

- Scaling plus Translation is the easiest to solve. A network with just 18k parameters can reach 93% accuracy in this setting.
- Scaling plus Rotation presents an intermediate level of difficulty, requiring 51k parameters to reach 89% accuracy.
- Rotation plus Translation is the most difficult of the two-transformation combinations, requiring 83k parameters to achieve only 77% accuracy.

The combination of all three transformations—scaling, rotation, and translation—represents the most challenging scenario. Despite the model having a moderate parameter count of 26k, the accuracy drops considerably, reaching only about 68%.

Although transfer learning slightly increases training time, the benefits in accuracy and stability make it a worthwhile trade-off, especially in high-performance image recognition tasks.

All in all, both the optimal filter extraction from minimal models and filter-transfer learning strategies are generalizable and can be applied to new datasets and real-world problems involving geometric image transformations to obtain small CNN models, increasing accuracy and stability while reducing time and operational costs.

6.2. Future work

A more comprehensive analysis of the dataset could be conducted to further understand model behavior and performance. Since the dataset is generated synthetically, all underlying parameters and features that define the images are available by design. This offers a valuable opportunity to create a secondary dataset containing metadata derived from the original one.

Such metadata could include attributes like the number of white and black pixels, rotation angles, scaling and translation deltas and other geometric descriptors. Basic representations such as luminance histograms can be plotted for images as well.

This enriched metadata dataset can be instrumental in identifying potential biases in the training data. By extracting and analyzing features such as scaling factors, rotation angles, translation vectors and figure positioning, it becomes possible to explore patterns in model errors. For instance, one could investigate whether misclassifications are correlated with certain transformations or figure positions, such as images where the object is located near the edges of the frame. This kind of exploratory analysis could help uncover systematic error patterns or repetitive conditions that challenge the model's generalization.

Such insights would allow for a deeper examination of whether the model is biased toward —or against— specific types of inputs, potentially due to imbalanced representation in the training data. If such biases are found, targeted data augmentation could be implemented to improve model fairness and performance.

Moreover, it is important to note that the models trained on Dataset 7.1, which involves the combination of all three geometric transformations (scaling, rotation and translation), did not achieve high accuracy levels. This suggests that, in this complex scenario, the classification task becomes non-linear and exceeds the capacity of the current architecture. To address this challenge, new strategies could be explored. For example, one promising approach involves freezing the first convolutional layer —composed of filters specialized in detecting basic transformations— and introducing an additional convolutional layer after the pooling stage. This new layer could then be trained from scratch to better capture higher-order features necessary for accurate classification in more complex scenarios.

Finally, it would be valuable to apply the proposed filter-transfer learning methodology, along with the knowledge base of optimal hyperparameter scales and architectural structures, to real-world image datasets. A relevant example could be a dataset of emoji images, commonly used in daily messaging, which share similar geometric properties and could be applied in tasks like sentiment analysis. Additionally, the methodology could be adapted for industrial use cases involving object or part recognition in manufacturing processes—environments where speed, reliability and interpretability are essential.

A

Alignment with the Sustainable Development Goals

The greatest threat to our planet
is the belief that someone else will save it.
Robert Swan (1956–)

This appendix provides a reflection on how the project aligns with the Sustainable Development Goals (SDGs). It identifies the specific goals most closely connected to the project and offers a detailed explanation of its contribution to each of them.

The Sustainable Development Goals (SDGs) provide a shared framework to create a more sustainable and equitable world by 2030, addressing critical global challenges such as poverty, inequality, climate change, and peace [31]. Adopted by all United Nations Member States in 2015, the SDGs consist of 17 distinct goals, each targeting a key area of society that requires global transformation.

To achieve these goals, projects developed within countries that support the SDGs must demonstrate how their objectives align with them. In this context, this thesis primarily supports two SDGs directly and one additional goal indirectly. The direct focus is on Goals 9 and 12, while Goal 10 is supported in a more indirect manner.

Goal 9, *Build resilient infrastructure, promote sustainable industrialization and foster innovation*, encompasses industries, innovation, and infrastructure to drive economic growth by creating employment opportunities and new income sources through the responsible use of resources [31]. Artificial Intelligence (AI) is undeniably central to modernizing industries, commonly referred to as Industry 4.0. Effectively implementing Machine Learning (ML) algorithms in both supply chains and end products requires a comprehensive understanding of these "black box" systems to control and optimize them safely. By automating basic tasks optimally, human labor can be redirected to more meaningful, value-added activities within companies.

Goal 12, *Ensure sustainable consumption and production patterns*, aims to challenge current consumer behaviors rooted in linear economies (buy-use-dispose). The technology industry is

known to be a major contributor to pollution, accounting for up to 3.5% of global emissions [32] and generating 50 million tonnes of electronic waste annually [33]. Gaining a deeper understanding of how algorithms learn is critical for improving their training efficiency, which can reduce the tech industry's energy consumption and minimize its environmental footprint.

Finally, this thesis indirectly contributes to Goal 10: *Reduce inequality within and among countries*, which seeks to ensure equal opportunities across all communities. With its capacity to emulate human behavior and assist businesses in decision-making, AI is increasingly being integrated into numerous processes that significantly impact people's lives. This underscores the importance of understanding how algorithms learn and being vigilant about potential biases. For example, if a neural network is used in hiring processes within a company, there is a risk that characteristics such as gender, race, or religious orientation may inadvertently influence outcomes, potentially leading to social injustices.

Bibliography

[1] *CIDAI: Green AI*. [Online]. Available: https://cidai.eu/en/green-ai/ (visited on 01/04/2025).

[2] *Python*. [Online]. Available: https://www.python.org/ (visited on 05/05/2025).

[3] *TensorFlow Guide*. [Online]. Available: https://www.tensorflow.org/guide (visited on 05/05/2025).

[4] M. Dhouibi, A. K. B. Salem, and S. B. Saoud, "Optimization of CNN model for image classification", *3rd IEEE International Conference on Design and Test of Integrated Micro and Nano-Systems, DTS 2021*, pp. 1–6, 2021.

[5] C. Peng, Y. Li, L. Jiao, and R. Shang, "Efficient Convolutional Neural Architecture Search for Remote Sensing Image Scene Classification", *IEEE Transactions on Geoscience and Remote Sensing*, vol. 59, no. 7, pp. 6092–6105, 2021.

[6] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search", *7th International Conference on Learning Representations, ICLR 2019*, pp. 1–13, 2019. arXiv: 1806.09055.

[7] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Completely Automated CNN Architecture Design Based on Blocks", *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 4, pp. 1242–1254, 2020.

[8] Y. Sun, B. Xue, M. Zhang, G. G. Yen, and J. Lv, "Automatically Designing CNN Architectures Using the Genetic Algorithm for Image Classification", *IEEE Transactions on Cybernetics*, vol. 50, no. 9, pp. 3840–3854, 2020. arXiv: 1808.03818.

[9] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: The all convolutional net", *3rd International Conference on Learning Representations, ICLR 2015 - Workshop Track Proceedings*, pp. 1–14, 2015. arXiv: 1412.6806.

[10] R. Keshari, M. Vatsa, R. Singh, and A. Noore, "Learning Structure and Strength of CNN Filters for Small Sample Size Training", *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 9349–9358, 2018. arXiv: 1803.11405.

[11] A. Ferreyra-Ramirez, C. Aviles-Cruz, E. Rodriguez-Martinez, J. Villegas-Cortez, and A. Zuñiga-Lopez, "An Improved Convolutional Neural Network Architecture for Image Classification", *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11524 LNCS, pp. 89–101, 2019.

[12] X. Xu, H. Zheng, Z. Guo, X. Wu, and Z. Zheng, "SDD-CNN: Small data-driven convolution neural networks for subtle roller defect inspection", *Applied Sciences (Switzerland)*, vol. 9, no. 7, 2019.

[13] F. Foroughi, Z. Chen, and J. Wang, "A CNN-based system for mobile robot navigation in indoor environments via visual localization with a small dataset", *World Electric Vehicle Journal*, vol. 12, no. 3, 2021.

[14] V. Savinov, V. Sapunov, N. Shusharina, S. Botman, and G. Kamyshov, "Research and selection of the optimal neural network architecture and parameters for depression classification using harmonized datasets", *Proceedings - 4th International Conference "Neurotechnologies and Neurointerfaces", CNN 2022*, pp. 132–135, 2022.

[15] K. M. Ang, E. S. M. El-Kenawy, A. A. Abdelhamid, A. Ibrahim, A. H. Alharbi, D. S. Khafaga, S. S. Tiang, and W. H. Lim, "Optimal Design of Convolutional Neural Network Architectures Using Teaching–Learning-Based Optimization for Image Classification", *Symmetry*, vol. 14, no. 11, 2022.

[16] P. J. B. Berdos, J. O. Saligumba, K. P. Deveza, and J. E. Estrada, "Discovering the Optimal Setup for Speech Emotion Recognition Model Incorporating Different CNN Architectures", *2022 IEEE 14th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment, and Management, HNICEM 2022*, pp. 1–5, 2022.

[17] M. Faris Al Hakim, B. Prasetiyo, Jumanto, and M. A. Muslim, "CNN Model for the Small Data in Vehicle Miniature Classification", *2023 1st International Conference on Advanced Engineering and Technologies, ICONNIC 2023 - Proceeding*, pp. 368–372, 2023.

[18] M. Mesárošová, O. Mihálik, and M. Jirgl, "CNN Architecture for Posture Classification on Small Data", *IFAC-PapersOnLine*, vol. 58, no. 9, pp. 299–304, 2024.

[19] *Python documentation: os — Miscellaneous operating system interfaces*. [Online]. Available: https://docs.python.org/3/library/os.html (visited on 01/05/2025).

[20] *Python documentation: random — Generate pseudo-random numbers*. [Online]. Available: https://docs.python.org/3/library/random.html# (visited on 01/05/2025).

[21] *Python documentation: math — Mathematical functions*. [Online]. Available: https://docs.python.org/3/library/math.html (visited on 01/05/2025).

[22] *Python documentation: operator — Standard operators as functions*. [Online]. Available: https://docs.python.org/3/library/operator.html (visited on 01/05/2025).

[23] "Numpy Documentation", [Online]. Available: https://numpy.org/doc/.

[24] *OpenCV*. [Online]. Available: https://docs.opencv.org/4.x/ (visited on 01/05/2025).

[25] *Matplotlib*. [Online]. Available: https://matplotlib.org/ (visited on 01/05/2025).

[26] *Matplotlib: matplotlib.patches.Ellipse*. [Online]. Available: https://matplotlib.org/stable/api/_as_gen/matplotlib.patches.Ellipse.html (visited on 01/06/2025).

[27] *Matplotlib: matplotlib.patches.Polygon*. [Online]. Available: https://matplotlib.org/stable/api/_as_gen/matplotlib.patches.Polygon.html (visited on 02/02/2025).

[28] *Detailed Guide to Understand and Implement ResNets*. [Online]. Available: https://cv-tricks.com/keras/understand-implement-resnets/ (visited on 03/04/2025).

[29] *Keras: VGG16 and VGG19*. [Online]. Available: https://keras.io/api/applications/vgg/ (visited on 03/04/2025).

[30] *ImageNet: VGGNet, ResNet, Inception, and Xception with Keras*. [Online]. Available: https://pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/ (visited on 03/04/2025).

[31] *SDG: Take Action for the Sustainable Development Goals*. [Online]. Available: https://www.un.org/sustainabledevelopment/sustainable-development-goals/ (visited on 05/05/2025).

[32] *Top 7 Most Polluting Industries*. [Online]. Available: https://www.theecoexperts.co.uk/blog/top-7-most-polluting-industries#link-technology (visited on 05/05/2025).

[33] *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 05/05/2025).