ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

MÁSTER EN INGENIERÍA INDUSTRIAL

TRABAJO FIN DE MÁSTER

ADVANCED MODELS FOR COMPUTER VISION

Autor: Miguel Huertas Collado

Directores: Lucía Güitta López

Álvaro Jesús López López

Madrid, Julio 2020

AUTORIZACIÓN PARA LA DIGITALIZACIÓN, DEPÓSITO Y DIVULGACIÓN EN RED DE PROYECTOS FIN DE GRADO, FIN DE MÁSTER, TESINAS O MEMORIAS DE BACHILLERATO

1º. Declaración de la autoría y acreditación de la misma.

El autor D. Miguel Huertas Collado

DECLARA ser el titular de los derechos de propiedad intelectual de la obra: ADVANCED MODELS FOR COMPUTER VISION, que ésta es una obra original, y que ostenta la condición de autor en el sentido que otorga la Ley de Propiedad Intelectual.

2º. Objeto y fines de la cesión.

Con el fin de dar la máxima difusión a la obra citada a través del Repositorio institucional de la Universidad, el autor **CEDE** a la Universidad Pontificia Comillas, de forma gratuita y no exclusiva, por el máximo plazo legal y con ámbito universal, los derechos de digitalización, de archivo, de reproducción, de distribución y de comunicación pública, incluido el derecho de puesta a disposición electrónica, tal y como se describen en la Ley de Propiedad Intelectual. El derecho de transformación se cede a los únicos efectos de lo dispuesto en la letra a) del apartado siguiente.

3º. Condiciones de la cesión y acceso

Sin perjuicio de la titularidad de la obra, que sigue correspondiendo a su autor, la cesión de derechos contemplada en esta licencia habilita para:

a) Transformarla con el fin de adaptarla a cualquier tecnología que permita incorporarla a internet y hacerla accesible; incorporar metadatos para realizar el registro de la obra e incorporar "marcas de agua" o cualquier otro sistema de seguridad o de protección.
b) Reproducirla en un soporte digital para su incorporación a una base de datos electrónica, incluyendo el derecho de reproducir y almacenar la obra en servidores, a los efectos de garantizar su seguridad, conservación y preservar el formato.
c) Comunicarla, por defecto, a través de un archivo institucional abierto, accesible de modo libre y gratuito a través de internet.
d) Cualquier otra forma de acceso (restringido, embargado, cerrado) deberá solicitarse expresamente y obedecer a causas justificadas.
e) Asignar por defecto a estos trabajos una licencia Creative Commons.
f) Asignar por defecto a estos trabajos un HANDLE (URL *persistente)*.

4º. Derechos del autor.

El autor, en tanto que titular de una obra tiene derecho a:

a) Que la Universidad identifique claramente su nombre como autor de la misma
b) Comunicar y dar publicidad a la obra en la versión que ceda y en otras posteriores a través de cualquier medio.
c) Solicitar la retirada de la obra del repositorio por causa justificada.
d) Recibir notificación fehaciente de cualquier reclamación que puedan formular terceras personas en relación con la obra y, en particular, de reclamaciones relativas a los derechos de propiedad intelectual sobre ella.

5º. Deberes del autor.

- El autor se compromete a:

a) Garantizar que el compromiso que adquiere mediante el presente escrito no infringe ningún derecho de terceros, ya sean de propiedad industrial, intelectual o cualquier otro.
b) Garantizar que el contenido de las obras no atenta contra los derechos al honor, a la intimidad y a la imagen de terceros.

c) Asumir toda reclamación o responsabilidad, incluyendo las indemnizaciones por daños, que pudieran ejercitarse contra la Universidad por terceros que vieran infringidos sus derechos e intereses a causa de la cesión.
d) Asumir la responsabilidad en el caso de que las instituciones fueran condenadas por infracción de derechos derivada de las obras objeto de la cesión.

6º. Fines y funcionamiento del Repositorio Institucional.

La obra se pondrá a disposición de los usuarios para que hagan de ella un uso justo y respetuoso con los derechos del autor, según lo permitido por la legislación aplicable, y con fines de estudio, investigación, o cualquier otro fin lícito. Con dicha finalidad, la Universidad asume los siguientes deberes y se reserva las siguientes facultades:

- La Universidad informará a los usuarios del archivo sobre los usos permitidos, y no garantiza ni asume responsabilidad alguna por otras formas en que los usuarios hagan un uso posterior de las obras no conforme con la legislación vigente. El uso posterior, más allá de la copia privada, requerirá que se cite la fuente y se reconozca la autoría, que no se obtenga beneficio comercial, y que no se realicen obras derivadas.
- La Universidad no revisará el contenido de las obras, que en todo caso permanecerá bajo la responsabilidad exclusive del autor y no estará obligada a ejercitar acciones legales en nombre del autor en el supuesto de infracciones a derechos de propiedad intelectual derivados del depósito y archivo de las obras. El autor renuncia a cualquier reclamación frente a la Universidad por las formas no ajustadas a la legislación vigente en que los usuarios hagan uso de las obras.
- La Universidad adoptará las medidas necesarias para la preservación de la obra en un futuro.
- La Universidad se reserva la facultad de retirar la obra, previa notificación al autor, en supuestos suficientemente justificados, o en caso de reclamaciones de terceros.

Madrid, a 30 de Junio de 2020

ACEPTA

Fdo: Miguel Huertas Collado 

Motivos para solicitar el acceso restringido, cerrado o embargado del trabajo en el Repositorio Institucional:

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

MÁSTER EN INGENIERÍA INDUSTRIAL

TRABAJO FIN DE MÁSTER

ADVANCED MODELS FOR COMPUTER VISION

Autor: Miguel Huertas Collado

Directores: Lucía Güitta López

Álvaro Jesús López López

Madrid, Julio 2020

MODELOS AVANZADOS DE VISIÓN ARTIFICIAL

Autor: Huertas Collado, Miguel.

Director: Güita López, Lucía;

López López, Álvaro.

Entidad Colaboradora: ICAI - Universidad Pontificia Comillas.

RESUMEN DEL PROYECTO

1. Introducción

La visión artificial es el campo que estudia y analiza imágenes y vídeos del mundo real para que un ordenador pueda entenderlo, extrayendo y procesando dicha información. Hoy en día, los algoritmos de visón artificial incorporan redes neuronales convolucionales para procesar esta información. A menudo, estas redes son tratadas como cajas negras, limitando su uso en muchos casos, debido al desconocimiento de lo que ocurre en su interior, dificultando su tratamiento y la posibilidad de adaptarlas a distintos usos.

Este proyecto trata de profundizar sobre estos algoritmos, escogiendo los modelos más actuales y entrenándolos desde cero. Con ello, se pretende analizar cómo afectan cada uno de los parámetros que componen estos modelos, para su posterior integración en aplicaciones más complejas.

En lo referente a imágenes, la visión artificial aborda el problema de procesamiento y clasificación de imágenes de cuatro formas principalmente, que son recogidas visualmente en la siguiente ilustración:

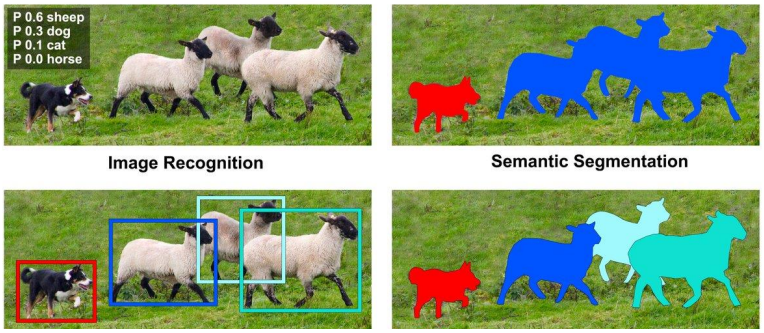


Figura 1: Tipos de clasificación de imágenes.

- **Clasificación de imagen**: identifica el objeto o la acción de la imagen.
- **Detección de objetos mediante cajas**: identifica y localiza los objetos de la imagen, situando una caja rectangular alrededor del objeto en cuestión.
- **Segmentación semántica**: clasifica cada píxel de la imagen según pertenezca o no a una determinada clase.
- **Segmentación de instancia**: clasifica cada píxel igual que en la segmentación semántica, diferenciando entre objetos pertenecientes a la misma clase.

En lo referente a vídeo, se ha abordado el problema de reconocimiento de acciones, por el cual no solo es suficiente reconocer los objetos presentes en el vídeo si no el contexto en el que ocurre la acción.

2. Metodología

El proyecto ha abordado la clasificación de imágenes y el reconocimiento de acciones en vídeo mediante redes neuronales convolucionales. Los modelos escogidos han sido los siguientes:

- **Faster-RCNN** [1]: Es un algoritmo de reconocimiento de imágenes mediante cajas. Primero se ha entrenado el modelo desde cero para 3 clases: personas, señales de tráfico y semáforos. Más tarde, las clases 'coche' y 'autobús' se han añadido mediante técnicas de *Transfer Learning*, comparando ambos planteamientos mediante distintos parámetros mencionados en el siguiente apartado. Finalmente, se validan los resultados con un modelo ya consagrado en el estado del arte como es YOLO (*You Only Look Once*).
- **U-Net** [2]: Analiza la clasificación de imágenes mediante segmentación semántica. Este modelo, menos exigente que el anterior, permite entrenar todo el modelo desde cero. Sin embargo, se ha decidido entrenar cada clase por separado para su posterior integración en el modelo completo, por temas de recursos disponibles.
- **SlowFast Networks** [3]: Ha sido el algoritmo escogido para realizar el reconocimiento de acciones en vídeos. Su fácil integración con modelos de reconocimiento de imágenes, como Faster-RCNN, permite su uso en aplicaciones combinadas de reconocimiento de objetos con cajas y vídeo.

A la hora de entrenar cada modelo se ha realizado una exploración previa del estado del arte, considerando códigos y trabajos ya realizados que puedan servir como punto de partida para el desarrollo del proyecto.

El entrenamiento de estos modelos requiere de GPUs (Graphics Processing Units) para reducir el tiempo de ejecución, siendo insostenible entrenando únicamente con CPU. Por ello, se ha utilizado Google Colaboratory, un servicio cloud adaptado para Machine Learning de Google, y una GPU propia.

3. Resultados

3.1 Faster-RCNN

Debido al gran número de parámetros del modelo (136 millones), se ha entrenado usando Google Colab para la mayoría de iteraciones, exceptuando la primera simulación, donde se ha usado la GPU NVIDIA GTX 2060.

Las condiciones iniciales han sido las siguientes:

- 1904 imágenes para entrenamiento, 479 para validación y 388 para test.
- Tamaño de la imagen de 150x150.
- Número de imágenes que se procesan en cada iteración igual a 1.
- Tasa de aprendizaje de 1x10-4

La siguiente figura muestra los pasos seguidos a la hora de entrenar el modelo, desde las condiciones iniciales hasta la configuración óptima:

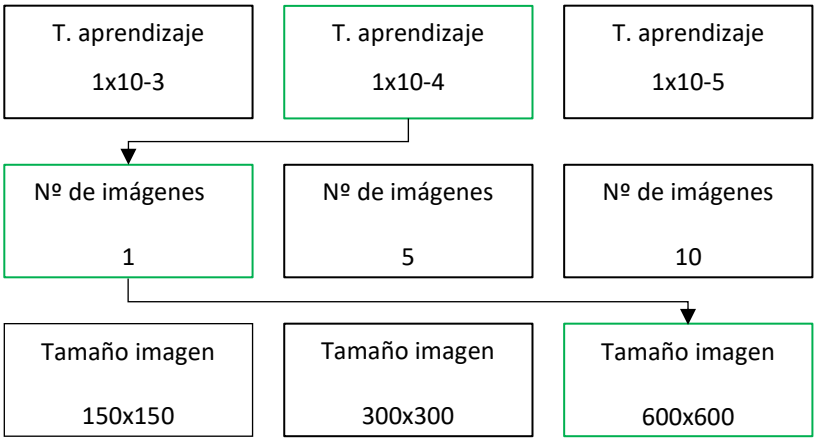


Figura 2: Estructura seguida para optimizar el modelo.

Por tanto, la tasa de aprendizaje final ha sido de 1x10-4, el número de imágenes igual a 1 y el tamaño de imagen de 600x600, que corresponde con los rectángulos en verde de la imagen de arriba. El modelo ha obtenido un valor de precisión de 0.35 para test, es decir, a la hora de analizar imágenes que el modelo nunca ha visto, el 35% de objetos los clasifica de manera correcta, localizándolos de la misma manera que un humano. La referencia [1] obtiene un valor de 0,42 para el mismo conjunto de imágenes, entrenando durante semanas, en comparación con las 12h entrenadas en este proyecto.

Una vez se entrena el modelo mediante *Transfer Learning*, tomando como partida la mejor configuración mostrada previamente, el modelo mejora la precisión hasta un 0.37. Las imágenes que se muestran a continuación son el resultado de este último modelo:

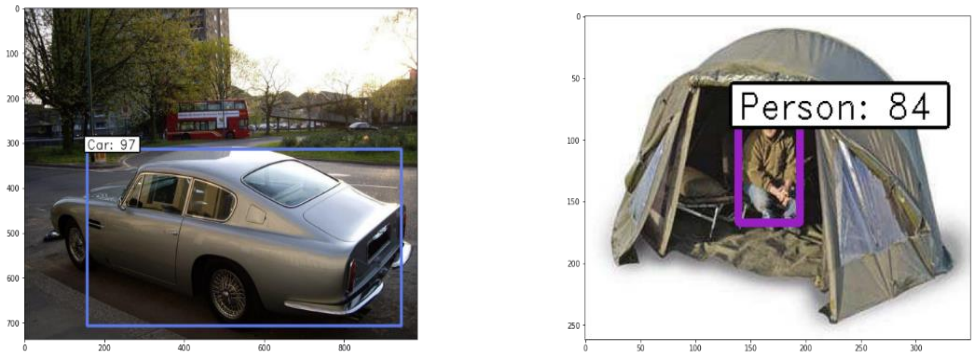


Figura 3: Ejemplos de funcionamiento del modelo para las clases de 'coche' y 'persona'.

3.2 U-Net

Tanto en U-Net como en el modelo siguiente, SlowFast, se ha utilizado la GPU propia para el entrenamiento de los modelos.

En este caso se ha entrenado un modelo para cada clase, sumando la salida de cada modelo para la composición final. Los parámetros de entrada han sido los siguientes:

- 4000 imágenes para entrenamiento, 800 para validación y 800 para test.
- Tamaño de la imagen de 128x128.
- Número de imágenes que se procesan en cada iteración igual a 2.
- Tasa de aprendizaje de 1x10-4.

El proceso de entrenamiento seguido ha sido el siguiente:

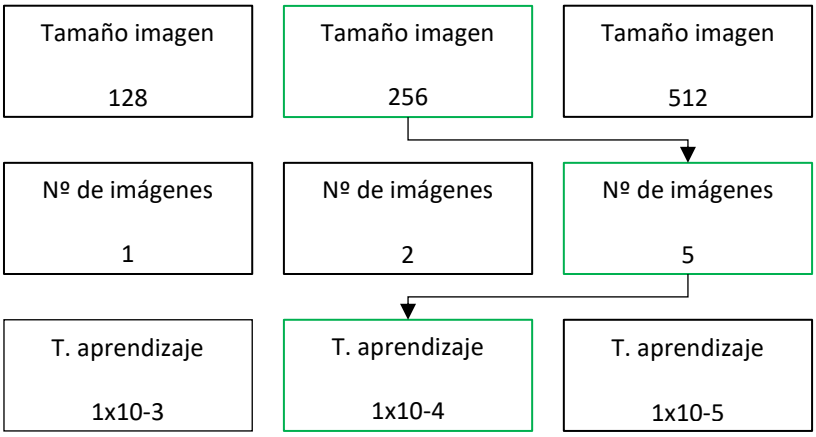


Figura 4: Estructura seguida para optimizar el modelo.

La precisión final del modelo, considerando todas las clases, es de 0.42. En este caso la precisión se realiza considerando cada píxel de la imagen, tanto si pertenece a un objecto como si no. Los resultados finales han sido los siguientes:

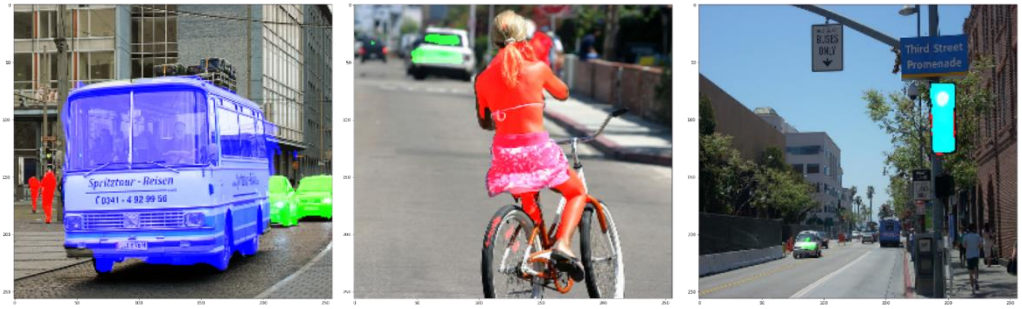


Figura 5: Ejemplo de funcionamiento del algoritmo U-Net.

Donde lo representado como azul corresponde a la clase 'autobús', lo rojo a la clase 'persona', verde para 'coche' y azul cian para 'semáforo'.

3.3 SlowFast Networks

Los vídeos escogidos tienen una duración media de 7 segundos, de los cuales se escogen 64 fotogramas que el modelo procesará como imágenes. Este modelo, al haber sido desarrollado por el equipo de Inteligencia Artificial de Facebook, está escrito en PyTorch en lugar de TensorFlow, que ha sido la librería escogida para los dos modelos anteriores.

Los parámetros de entrada para el entrenamiento del modelo han sido los siguientes:

- Optimizador: Stochastic Gradient Descent (SGD).

- Tasa de aprendizaje: 0.01
- Decaimiento de pesos: 0.00005
- Tamaño de lote: 8

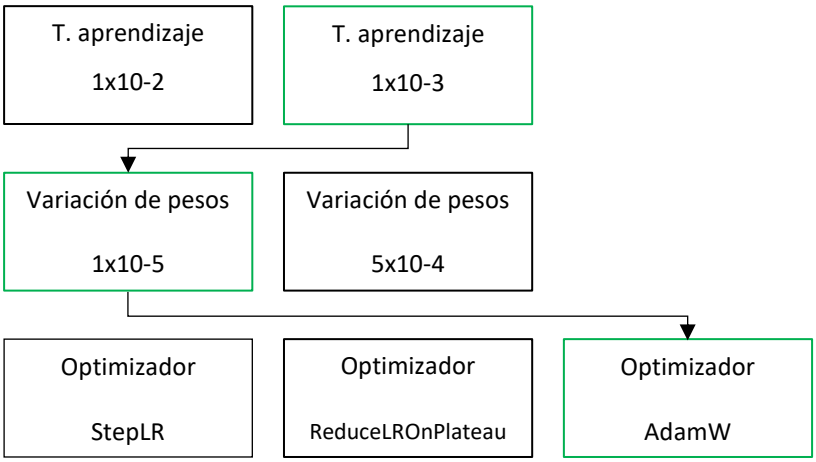


Figura 6: Estructura seguida para optimizar el modelo de SlowFast.

El optimizador fue escogido para replicar las condiciones del trabajo original. Sin embargo, a lo largo del proyecto se ha analizado el potencial de optimizadores más potentes según la aplicación. Por ello, se decidió analizar ReduceLROnPlateau y AdamW, siendo este último el optimizador que mejores resultados obtiene.

La precisión final del modelo ha sido del 55%, comparado con el 75% que obtiene el trabajo original. No obstante, en el proyecto son analizados los parámetros necesarios para minimizar esta diferencia.

En este caso, el modelo clasifica el contenido del vídeo, sin añadir cajas o colorear cada píxel de la foto, como en el caso anterior. Sin embargo, la posibilidad de añadir un modelo de clasificación de imágenes, como Faster-RCNN, es analizada durante el proyecto. Incluyéndolo, el modelo es capaz de analizar el contenido y señalarlo para determinados fotogramas, como muestra la siguiente imagen:

Figura 7: Ejemplo para la clase 'Aplicar maquillaje'

4. Conclusiones

En este proyecto se han analizado nuevos modelos de visión artificial que utilizan técnicas que marcarán el desarrollo de este campo los próximos años. Con ello, se ha conseguido comprender mejor el funcionamiento de estos algoritmos, así como un mejor entendimiento de las redes neuronales convolucionales, tan a menudo llamadas 'cajas negras' por su complejo funcionamiento.

Se han comparado los resultados obtenidos con lo referente al estado del arte, analizando las causas de las diferencias y comprendiendo su funcionamiento. La gran limitación a la hora de entrenar estos modelos ha sido la disponibilidad de recursos, en este caso de GPU. En el proyecto se incluye un análisis de las distintas opciones Cloud si no se dispone de GPU propia o se quiere complementar con una, ya que a menudo los recursos de los que se dispone en el estado del arte distan mucho de lo que dispone el usuario medio, influyendo gravemente en el resultado.

5. Referencias

[1] S. Ren, K. He, R. Girshick y J. Sun, Faster R-CNN: Towards Real-Time Object, 2016.

[2] O. Ronneberger, P. Fischer y T. Brox, U-Net: Convolutional Networks for Biomedical, 2015.

[3] C. Feichtenhofer, H. Fan, J. Malik y K. He, SlowFast Networks for Video Recognition, 2019.

ADVANCED MODELS FOR COMPUTER VISION

Author: Huertas Collado, Miguel.

Director: Güita López, Lucía;

López López, Álvaro.

Collaborating Entity: ICAI - Universidad Pontificia Comillas.

ABSTRACT

1. Introduction

Computer Vision (CV) is the field that studies and analyzes images and videos of the real world so that a computer can understand them, extracting and processing their information. Nowadays, CV algorithms incorporate convolutional neural networks to process this information. These networks are often named as black boxes, limiting their use in many cases, due to the lack of knowledge of what is happening inside themselves, making it difficult to treat them and the possibility of adapting them to different uses.

This project examines these algorithms, choosing actual models and training them 'from scratch'. By that, the model is analyzed in terms of its parameters, understanding its behavior for a potential integration in more complex applications.

Regarding images, Computer Vision addresses the problem of image processing classifying them in 4 different ways, which are collected visually in the following figure:

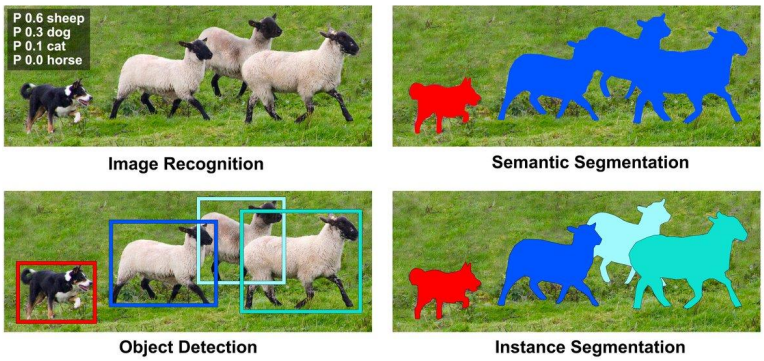


Figure 1: Image classification tasks.

- **Image classification**: identifies the object or the action of the picture.
- **Object detection:** each object is classified and localized by rectangular boxes around each object, known as bounding box.
- **Semantic segmentation**: each pixel of the image is labelled with a class.
- **Instance segmentation**: each pixel is classified in the same way as semantic segmentation, but differencing between objects from the same class.

In terms of video classification, the task has been action recognition, which not only objects are recognized, but also the context of the action needs to be analyzed as well.

2. Methodology

Convolutional neural networks have been used to address the image classification task in this project. The following algorithms have been trained:

- **Faster-RCNN** [1]: It is an algorithm for bounding box object detection. First, the model has been trained from scratch for 3 classes: 'people', 'traffic signs' and 'traffic lights'. Later, 'cars' and 'buses' classes were added by *Transfer Learning*, comparing both approaches using different parameters mentioned in next section. Finally, results have been validated with an already consolidated model in the state-of-the-art literature, such as YOLO (You Only Look Once).
- **U-Net** [2]: Analyze the classification of images using semantic segmentation. This model, less demanding than the previous one, allows training the entire model from scratch. However, it has been decided to train each class separately and integrate all together into the model, due to limited resources available.
- **SlowFast Networks** [3]: It has been the algorithm chosen to perform action recognition in videos. Its easy integration with image classification models, such as Faster-RCNN, allows it to be used combined with bounding box object detection and video applications.

When training every model, a prior exploration of the state of the art has been carried out, considering scripts and the work already done that can serve as a starting point for the development of the project.

Training these models requires GPUs (Graphics Processing Units) to reduce execution time, being unsustainable training them only with a CPU. Therefore, Google Colaboratory, a cloud service adapted for Machine Learning, and a personal GPU have been used.

3. Results

3.1 Faster-RCNN

Due to the large number of parameters in the model (136 millions), Google Colab has been used for training, except for the first simulation showed below (learning rate), where GPU NVIDIA GTX 2060 was used.

The following configuration has been chosen as the starting point:

- 1904 training images, 479 images for validation and 388 for test.
- Image size: 150x150.
- Batch size: 1.
- Learning rate: $1x10^{-4}$.

Starting from the base conditions, the next figure shows the path followed to optimize the model:

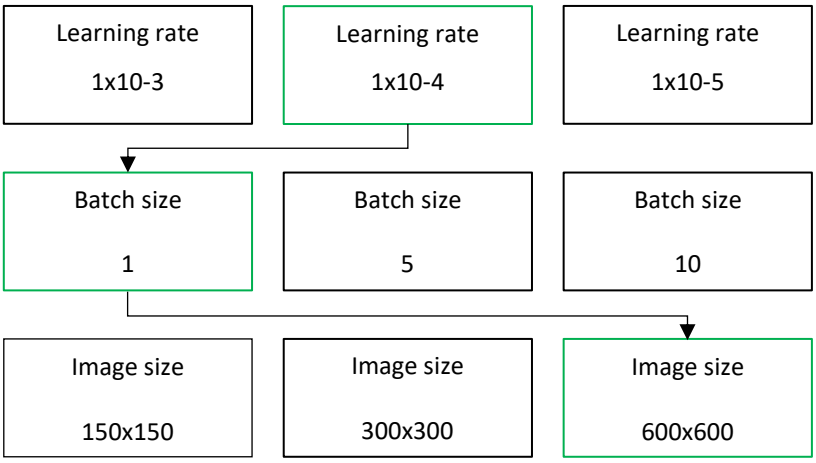


Figure 2: Structure followed during the optimization of the model.

Therefore, the final learning rate has been $1x10^{-4}$, the batch size equals to 1 and the image size is 600x600, which corresponds to the green rectangles showed before. The model has obtained a precision value of 0.35 for testing, what means that when analyzing images that the model has never seen, the 35% of objects are classified correctly, locating them in the

same way as a human would do. Reference [1] obtains a value of 0.42 for the same set of images, but training for weeks, compared to the 12h trained in this project.

Once the model is trained by means of *Transfer Learning*, considering as the best configuration previously shown as the starting point, the model improves have improved its precision up to 0.37. The images shown below are the result of this latest model:

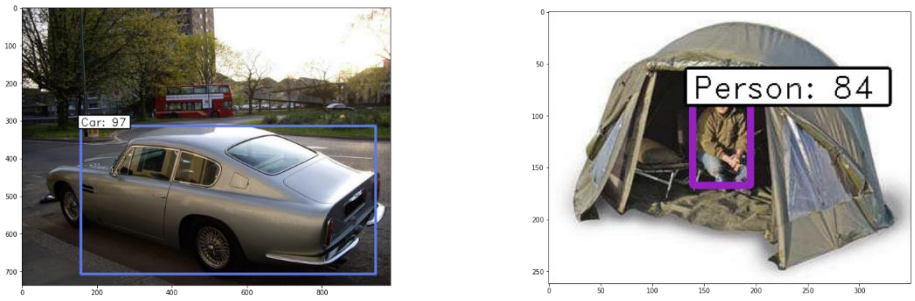


Figure 3: Samples for classes 'Car' and 'Person'.

3.2 U-Net

Both U-Net and the following model, SlowFast, the personnel GPU has been used when training the models.

In this case, the model has been trained for each class independently, adding the output of each model for the final integration. The input parameters have been the following:

- 4000 training images, 800 images for validation and 800 for test.
- Image size: 128x128.
- Batch size: 2.
- Learning rate: $1x10^{-4}$.

Next figure shows the procedure for optimizing the model:

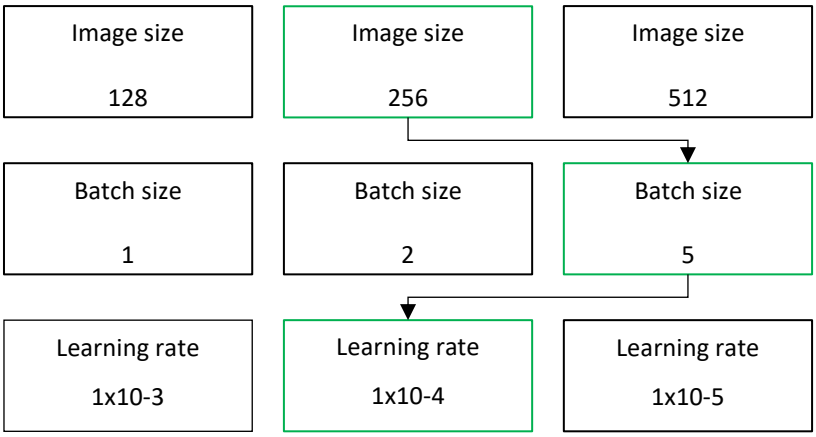


Figure 4: Structure followed during the optimization of the model

The final precision of the model, considering all classes, is 0.42. In this case, precision is performed considering each pixel in the image, whether it belongs to an object or not.

Next images are the output of the overall model:

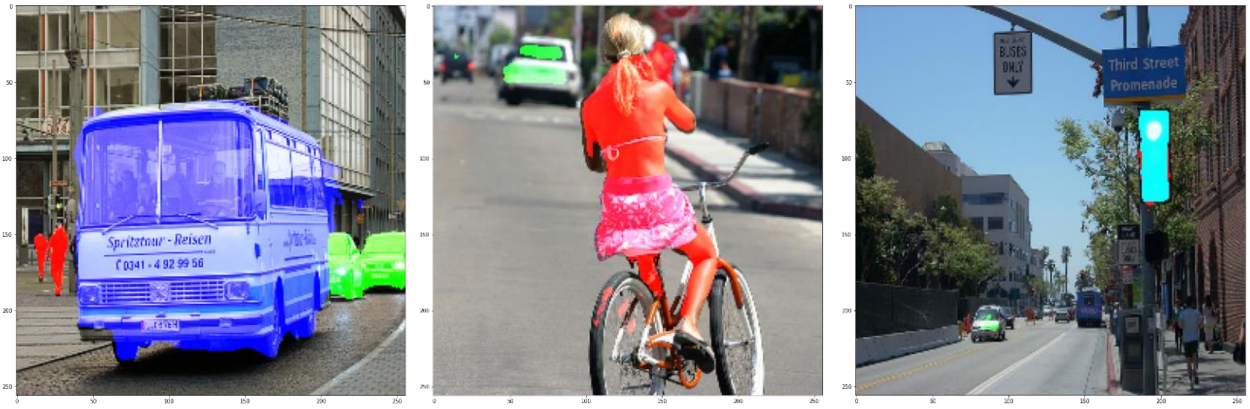


Figure 5: Samples of U-Net model for classes 'Car', 'Bus', 'Person' and 'traffic light'.

Where the blue pixels refer to class 'bus', the red ones are the class 'people', class 'car' is represented by green, and cyan refers to 'traffic light'.

3.3 SlowFast Networks

Each video of the dataset has an average duration of 7 seconds, and 64 frames are chosen from each clip that the model will process as images. The model, having been developed by Facebook's Artificial Intelligence team, is coded in PyTorch library instead of TensorFlow, which has been the library of the previous two models.

The base conditions have been the following:

- Optimizer: Stochastic Gradient Descent (SGD).
- Learning rate: 0.01
- Weight decay: 0.00005
- Batch size: 8

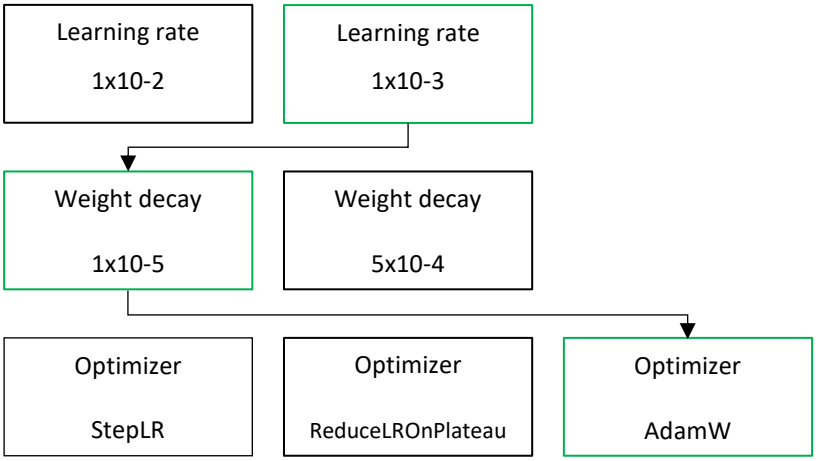


Figure 6: Structure followed during the optimization of the algorithm.

The optimizer was chosen to replicate the conditions from the paper. However, throughout the development of the project, more powerful optimizers have been analyzed depending on the application. For this reason, it was decided to analyze ReduceLROnPlateau and AdamW, being the latter the optimizer with the best results.

The final accuracy of the model has been 55%, compared to the 75% from the paper. Though, the parameters necessary to minimize this difference are analyzed in the project.

In this case, the model classifies the content of the video, without adding boxes or classifying each pixel in the picture, as in the previous case. However, the possibility of adding an image classification model, such as Faster-RCNN, is analyzed during the project.

Integrating both models, the algorithm is able to process the content of the video and localize the objects in determined frames, as the next image shows:

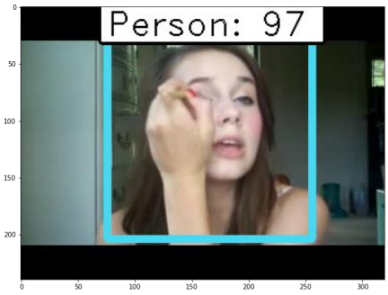


Figure 7: Sample for class 'Apply Eye Make Up'

4. Conclusions

In this project, innovative computer vision algorithms have been analyzed. These algorithms introduce new techniques that will mark the development of this field in the upcoming years. Thus, it has been possible to better understand the operation of these algorithms, as well as the convolution neural networks, so often called 'black boxes' due to their complex operation.

The results obtained have been compared with those in the state-of-the-art literature, analyzing the causes of the differences and understanding their operation. The great limitation when training has been the availability of resources, in this case of a GPU. The project includes an analysis of the different Cloud options if a personnel GPU is not possible, or to complement it, you do not have your own GPU or want to complement it with one, since very often the resources available in the state of the art are unaffordable for an average user, undermining the result.

5. References

[1] S. Ren, K. He, R. Girshick y J. Sun, Faster R-CNN: Towards Real-Time Object, 2016.

[2] O. Ronneberger, P. Fischer y T. Brox, U-Net: Convolutional Networks for Biomedical, 2015.

[3] C. Feichtenhofer, H. Fan, J. Malik y K. He, SlowFast Networks for Video Recognition, 2019.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

PART I. REPORT

1. INTRODUCTION

"More data was generated in the last two years than in the entire human history before that" and *"Each minute, 300 new hours of video show up on YouTube"* [4] are quotes from *HostingTribunal* that show the speed at which data is currently growing. However, in the same article, it is also said that the 99,5% of collected data is never analysed or used in any way. Additionally, computers capacity and efficiency are also improving very fast, allowing to solve complex problems and algorithms never imagined with the resources we had few years ago.

Hence, this is why artificial intelligence comes into play. By means of images and videos from the past, computers can learn from them and apply the knowledge in different scenarios. The field that interprets and understands the visual world is known as Computer Vision.

The integration of Computer Vision in the industry, added to brand-new concepts as the Internet of Things (IoT), has reshaped the business into what is called the Industry 4.0. Modern computer vision systems are able to work in changing environments, far from the structured and repetitive tasks from years ago. They can interact with people, respond to interactions in different situations and work independently.

With the huge amount of data collected in a manufacturing line, computer vision algorithms are able to do quality checks or track a part through a manufacturing line. These algorithms need to be trained for its specific application, to reduce the response time and increase the accuracy of the model. Therefore, although computer vision models are planned to work independently, they will always need a person that understands and implements the algorithms.

1.1 MOTIVATION

Deep Neural Networks have become very famous recently due to their achievements in various applications as mentioned before. However, there are concerns that sometimes make challenging to deeply understand them. That is why neural networks are often known as *'black-box*' models, compared to other models in artificial intelligence.

The number of layers, the number of neurons in each layer, or the activation functions are hyperparameters that cause these deep neural networks to be hard to understand. Convolutional neural networks can have thousands or millions of parameters and hyperparameters that must be calculated and optimized. What represent and how they affect the output is a challenge.

This project will analyse the main algorithms that use convolutional neural networks for object detection and explain how they are affected by their parameters.

Additionally, these algorithms train the computer to classify and identify objects, people or places. However, they are limited by the training dataset and by what they have already seen. Human intuition is not teachable, but the motivation of computer vision is to have a system that understand the image and its context as similar as a human would do.

1.2 OBJECTIVE OF THE PROJECT

The aim of this project is to review and understand the main algorithms of Deep Learning in the Computer Vision field, focusing on object detection and semantic segmentation for images and videos. To achieve it, the following objectives have been formulated:

- Analyse the performance of each algorithm based on its Average Precision (AP) comparing to the state-of-the-art literature.

- Improve the performance of a model compared to the previous case by means of Transfer Learning, adding a new class to the algorithm and validating its performance with an already consolidated model.
- Generalize a one-class model to perform multi-class classification.
- Recognize the context and situation in video datasets, obtaining an accuracy higher than 50% for multi-class classification.

1.3 METHODOLOGY

The project will be approached by means of convolutional neural networks focusing on object detection and semantic segmentation for image classification and action recognition in video.

For each algorithm, the way to proceed will be as follow:

1. Review the state-of-the-art, which includes neural networks, convolutional Neural Networks, sequence models and the papers describing the algorithms that will be studied.
2. Build, train and optimize one algorithm for each task: object detection, instance segmentation and action recognition.
3. Compare the results with the state-of-the-art literature.

Depending on the results achieved during the investigation, the models and concepts learned during the project will be applied to a real example which consists of a video application that recognize the people on it and measure the distance between them. This idea is based on the social distancing requirement applied to several countries due to the COVID-19 situation.

An indicative planning with every step of the project can be found in the next page.

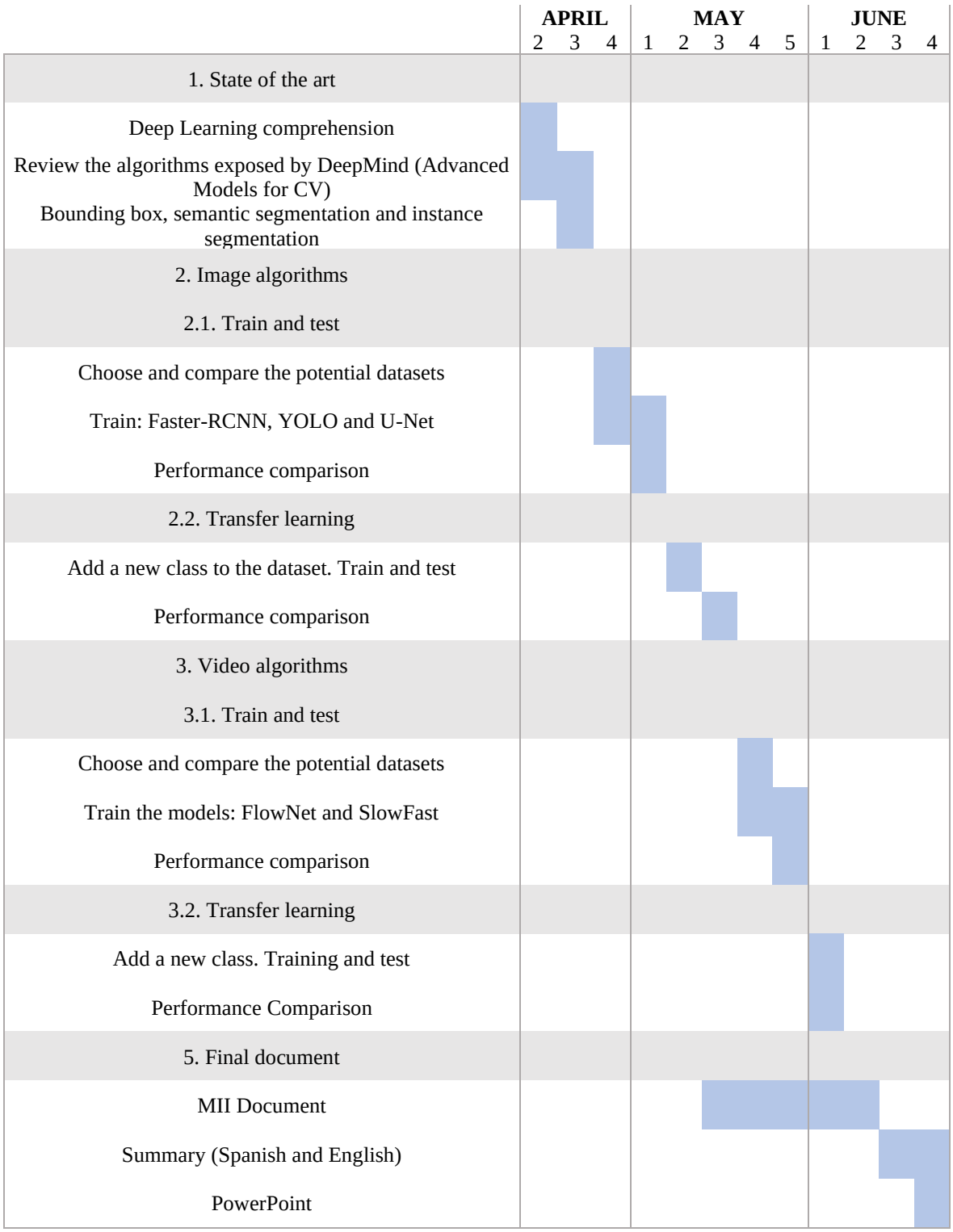


Figure 1-1: Planning followed during the project.

2. STATE OF THE ART

It seems that Computer Vision is very recent and it is related with supercomputers and complex algorithms. However, Computer Vision started 70 years ago, when the modern graphics processors were unimaginable.

The first work was published in 1959, and it had nothing to do with software engineering. The work, called "*Receptive fields of single neurons in the cat's striate cortex*" [5], described how the cat's visual neurons responded to visual experience. The authors realized that there were simple and complex neurons, but the visual process always starts with simple structures, such as lines and oriented edges.

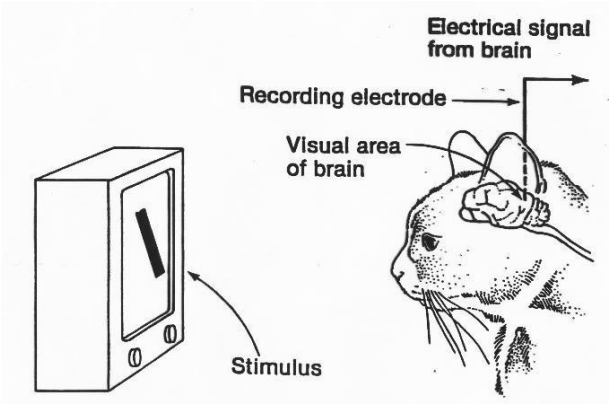


Figure 2-1: Hubel and Wiesel experiment with a cat responding to visual experiences.

Image source: https://goodpsychology.wordpress.com/2013/03/13/235/

The same year, the first digital image scanner was invented. Russell Kirsch transformed an image into a grid of numbers, so machines could understand. The first image to be scanned was Russell's son. It captured 30,976 pixels, to form a 176x176 array, from a 5x5 cm picture.

Later, in 1963, Lawrence Roberts published "*Machine perception of three-dimensional solids*" [6] and described the process of getting 3D info from 2D photographs. It was the first approach to computer-aided 3D systems.

In the 1960s, Seymour Papert and a small group of MIT students launched the *Summer Vision Project* and tried to perform an automatic application that detect objects from real-world images. They did not succeed, but it as the official birth of Computer Vision, according to many scientists.

In 1982, the Japanese Kunihiko Fukushima built an artificial network with simple and complex cells that could recognize patterns. *Necognitron*, as it was called, included several convolutional layers with weight vectors, the equivalent to filters nowadays. It was the first appearance of a neural network and it is considered the grandfather of today's convolutional neural networks.

Seven years later, in 1989, Yann LeCun applied the backprop learning technique to Fukushima's network, and a few years later, LeCun released *LeNet5*, the first convolutional neural network with some of the features that these networks use today. His work resulted in MNIST dataset, one of the most famous benchmark datasets in machine learning which contains handwritten digits.

The first face detection model was developed by Paul Viola and Michael Jones in 2001. It worked in real-time and learned which features could help localize faces. The detector is still used, being a binary classifier made of several weak classifier and trained using Adaboost [7]. The way it learns, with edge detection and filters is widely used:

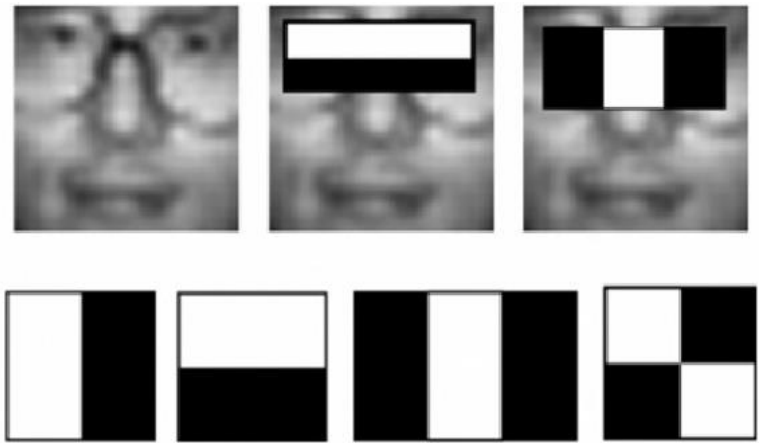


Figure 2-2: Visual representation of simple edge detectors.

Image source: https://www.researchgate.net/figure/Haar-features-used-for-Viola-Jones-face-detection-

The computer vision community kept advancing, and the need of a benchmark dataset and common metrics for everyone was latent. For that reason, in 2006, Pascal VOC provided a standardized dataset for object classification and ran an annual competition, from 2006 to 2012, were object class recognition algorithms were evaluated.

In 2009, Pedro Felzenszwalb and his colleagues developed the *Deformable Part Model*, that decompose objects into a collection of parts. It showed great performance for object detection tasks with bounding boxes, becoming very popular at that time.

ImageNet started in 2010 the Visual Recognition Competition adding more than a million images with one thousand object classes. In this contest, a team from the University of Toronto trained *AlexNet* and changed the competition in 2012 by reducing the error rate of the competition from 26% to 16,4%. *AlexNet* was the first modern convolutional neural network and since then, the winners of ImageNet competition have always been algorithms that use convolutional networks.

As it was said, the first convolutional neural network was introduced by Fukushima in 1982, but the popularity of these networks came with *AlexNet* in 2012 driven mainly for three reasons:

- Computers are vastly more powerful and fast nowadays that in the 1980s.
- Graphics Processing Units (GPU) have made a huge impact in image processing.
- The large number of images, annotations and high-dimensional datasets available compared to previous years.

Deep Learning as a Computer Vision approach began with Krizhevsky [8] in 2012. Since then, Deep Learning has dominated the image classification approaches due to its outstanding better performance compared to traditional methods.

However, traditional methods are the backbone of these algorithms, and their fundamentals help to understand the modern approaches of Deep Learning. There are mainly three traditional algorithms:

- Scale Invariant Feature Transform (SIFT) [9].
- Speeded Up Robust Features (SURF) [10].
- Features from Accelerated Segment Test (FAST) [11].

SIFT and SURF are feature descriptors, where feature refers to descriptive and informative patches in every image. The challenge of these algorithms was deciding which features are important and train by trial and error the features that best describe the objects in the image. Translating the object to predict features such as edges, color or shapes was doing manually to interpret the input image as a matrix of numbers.

Deep Learning introduced the concept of end-to-end training, which adjust automatically the features knowing the input and the output of the network. Therefore, the workflow of Computer Vision engineers has changed from learning how to extract hand-crafted features to learning how to iterate through deep learning architectures. Next image shows the change of paradigm in computer vision.

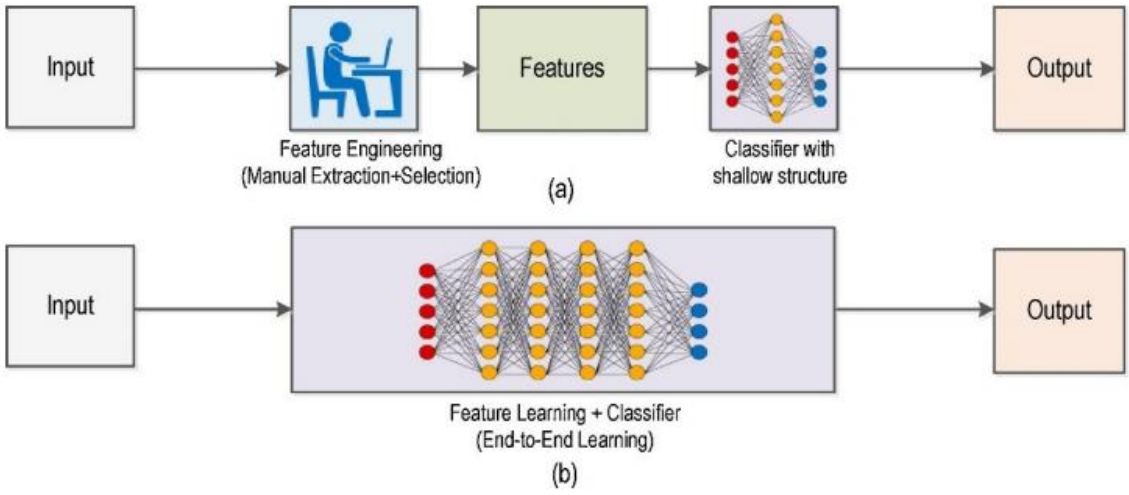


Figure 2-3: Traditional Computer Vision approach (a) vs Deep Learning approach (b)

Image source: Ref. [64]

The challenge now in computer vision is to understand not only the objects in the picture, but the context, allowing the computer to see the picture and interpret it as a human would do.

3. IMAGE CLASSIFICATION WITH CNN

Convolutional Neural Networks (CNN) are the state-of-the-art solution for Computer Vision nowadays. They differ from the Fully Connected Neural Networks (FCNN), which are the regular networks in artificial intelligence, in terms of how their layers are connected. The term *convolution* refers to the mathematical operation where two signals are combined. Due to this new type of NN, computer vision algorithms reduce the number of parameters drastically, being able to train models with thousands and millions of images and videos that would be unapproachable otherwise.

Image classification is the process of taking an image and identify the object in it. Early detectors were based on handcrafted features and inefficient algorithms. However, the rise of CNNs has produced significant results, introducing new algorithms and classification approaches. Depending on the task, Computer Vision divides the object recognition task into:

- **Image classification**: identifies objects in the image.
- **Object detection**: each object is classified and localized. Models create a rectangular box around each object known as bounding box. The main algorithms in object detection are: Faster R-CNN, RetinaNet, YOLO (You Only Look Once) and SSD (Single Shot MultiBox Detector).
- **Semantic segmentation**: instead of classifying the object by putting a box around it, with semantic segmentation each pixel is labelled with a class. Mask-RCNN and U-Net are two examples of semantic segmentation algorithms.
- **Instance segmentation**: in addition to semantic segmentation, these algorithms recognize and label each instance of every class.

Next image shows an example of the four tasks, where the sheep, the road and the grass are labelled depending on the task.

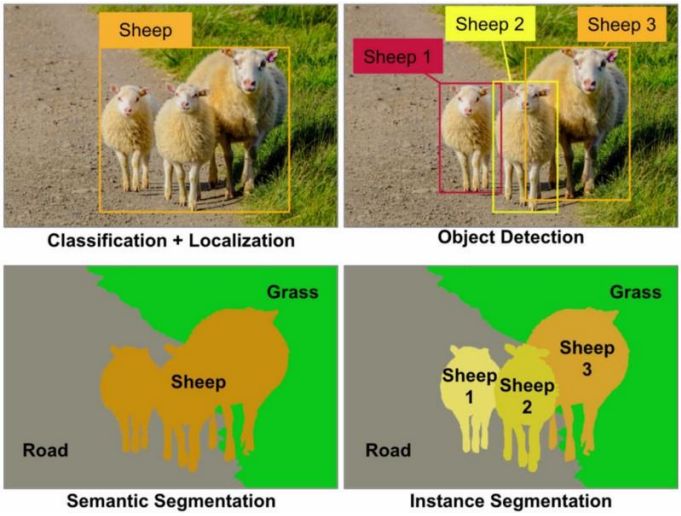


Figure 3-1: Comparison between the different image classification tasks

Image source. https://mc.ai/detection-and-segmentation/

Prior to a deeper description of each task, a general overview of conv nets and how they work is listed hereunder. Each algorithm shares this architecture and the same process, adding some features to approach better the different tasks.

Finally, at the end of the section, the main applications of convolutional networks and computer vision are listed.

3.1 ARCHITECTURE

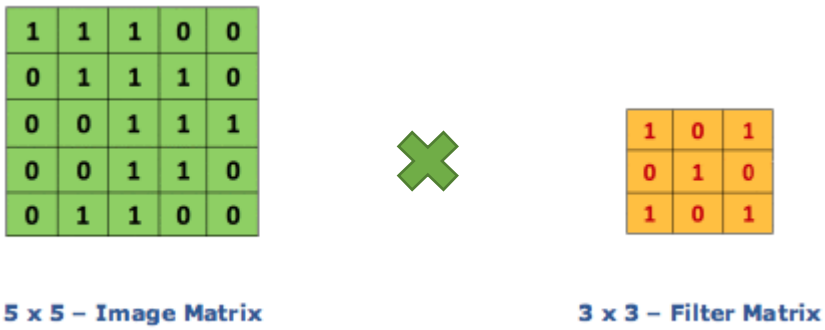
In this section, the main concepts of convolutional neural networks are explained. It is important to mark that the parameters derived from these features, such as the weights of the convolutional layers, are determined by the model itself, and not by the user. These parameters are calculated after training the model, while the parameters determined by the user are referred as hyperparameters, explained in Chapter 5.

3.1.1 CONVOLUTIONAL LAYER

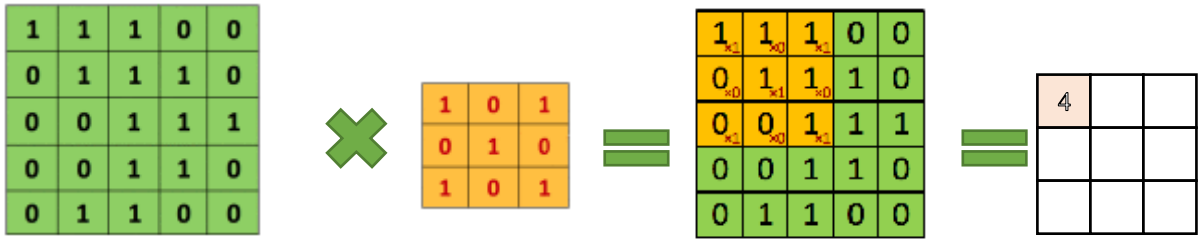
Convolutional neural networks see the input image as a matrix of pixels based on the image size, that will be denoted as: $HxWxD$, corresponding to the Height, Width and Dimension

of the image. Typically, D will be 3, referring to the RGB channels: Red, Green and Blue. For instance, in binary classification where images are black and white (grey scale), dimensions will be HxWx1.

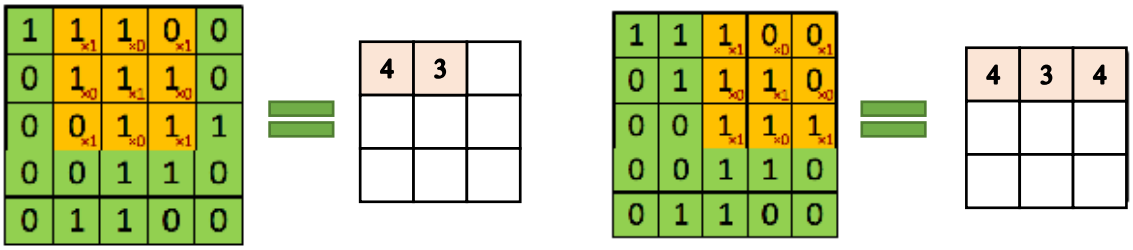
Then, the first convolution layer is applied. To preserve the relationship between pixels, a filter or "kernel" is applied to the input data. Kernels are also matrixes that are multiplied to the input data resulting in the output shown below called "Feature Map". To better understand this operation, next figures illustrate an example:

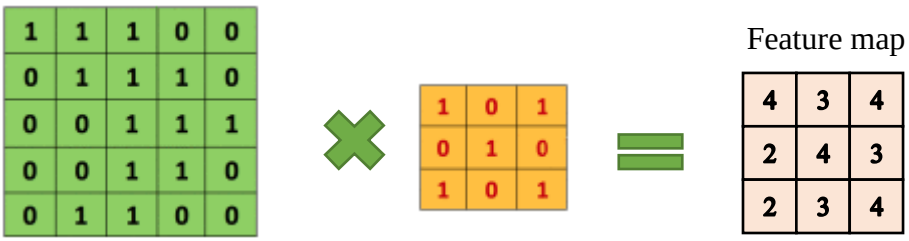


The 5x5 matrix is the input image (5x5x1) and the 3x3 matrix is the kernel (3x3x1). Then, the convolution operation between the image and the filter is applied:

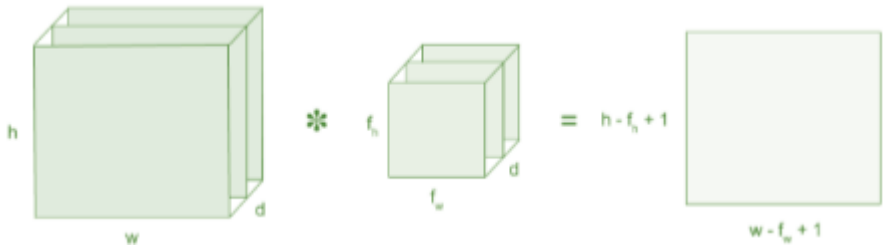


$$1x1 + 1x0 + 1x1 + 0x0 + 1x1 + 1x0 + 0x1 + 0x0 + 1x1 = 4$$





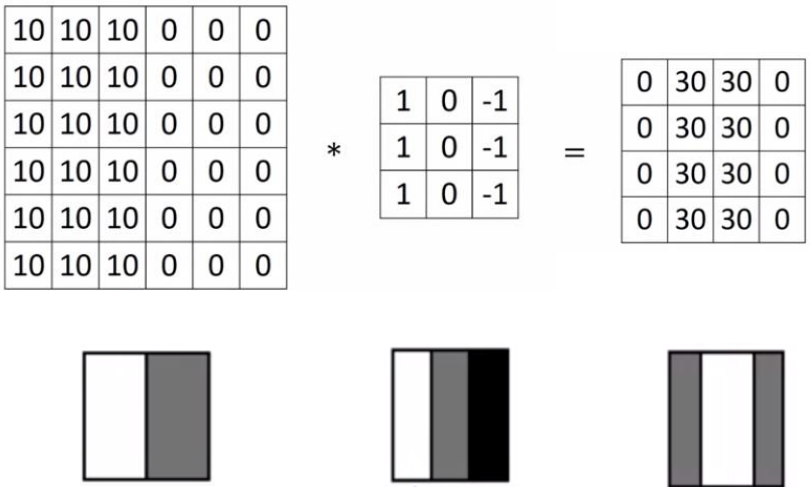
The height and the width of the kernel can be any value between one and the size of the input. In fact, it is one of the hyperparameters to tune when training a model. The only restriction is the dimension (D), that must match the dimension of the input image:



where h and w are the height and the width of the input; d the dimension of the input and the kernel; f_h and f_w the height and the width of the kernel.

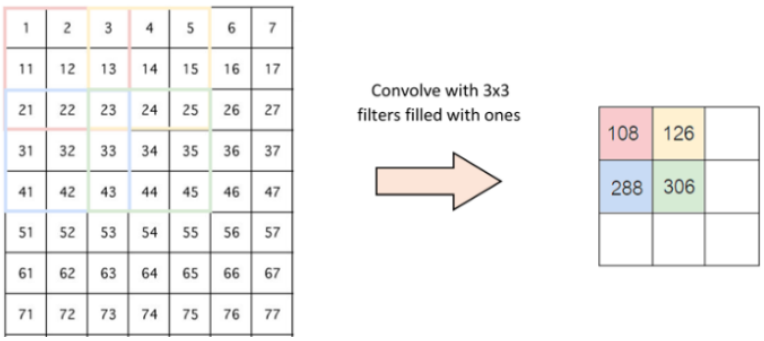
Image: $h\ x\ w\ x\ d$; Kernel: $f_h\ x\ f_w x\ d$; Output: $(h - f_h + 1)\ x\ (w - f_w + 1)\ x\ 1$

Depending on the values of the kernel, convolution can perform different detections, such as vertical and horizontal edge detections. The next example is a kernel used to detect the input vertical line of the input image:



3.1.2 STRIDE

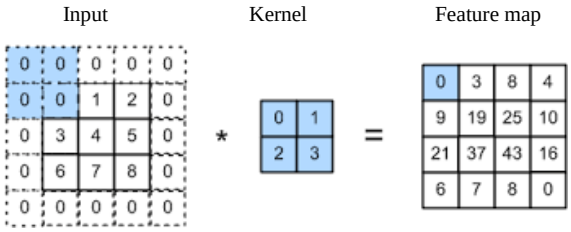
Stride is defined to reduce the height, the width and the dimension of the previous layer, using the same kernel. It is defined as "*the number of pixels shifts over the input matrix*" [12]. A stride of 1 will move the kernel 1 pixel at a time, as the example in the previous section showed. Hence, a stride of 2 would move the kernels 2 pixels at a time:



3.1.3 PADDING

However, some important features may be in the edge of the picture. As the convolution is defined above, it is more complex to capture the information of the edges, as the kernel is multiplied more often in the middle of the matrix than in the edges. To avoid losing that

information, padding is defined. Padding adds as many zeros as it is necessary to each row and column of the matrix edges.



By convention, there are two padding values that are known as:

- **Valid**: pad equals to zero or "no-padding".
- **Same**: Pad so that output size is the same as the input size. Considering n the size of the input matrix (HxW), f the size of the kernel and p the padding value:

$$n + 2 * p - f + 1 = n \tag{1}$$

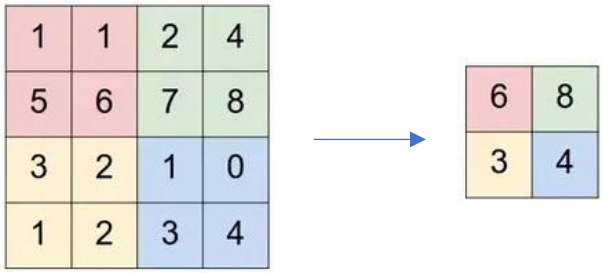
$$p = \frac{f - 1}{2} \tag{2}$$

3.1.4 POOLING LAYERS

Convolutional networks often use pooling layers to reduce the size of the matrix, to speed the computation and to make some of the features that detects more robust. There are three main type of pooling layers:

- **Max** Pooling: it takes the highest number of the feature map.
- **Average** Pooling: it applies the average of the feature map
- **Sum** Pooling: sums all elements in the feature map.

Considering a max pooling, with a 2x2 kernel and a stride of 2:



In this case, taking the largest value of the sub-matrix could help to notice when certain features are present in the input.

3.1.5 ACTIVATION FUNCTIONS

Activation functions [13] are usually at the end of the convolutional neural layer. They determine the output of the model and classify the output depending on the class, having a significant effect on the network's ability to converge. For this reason, the election of the activation function is critical, considering each situation and the possibilities each one offers.

Next figure illustrates the role of activation functions, also known as transfer functions, in a neuron:

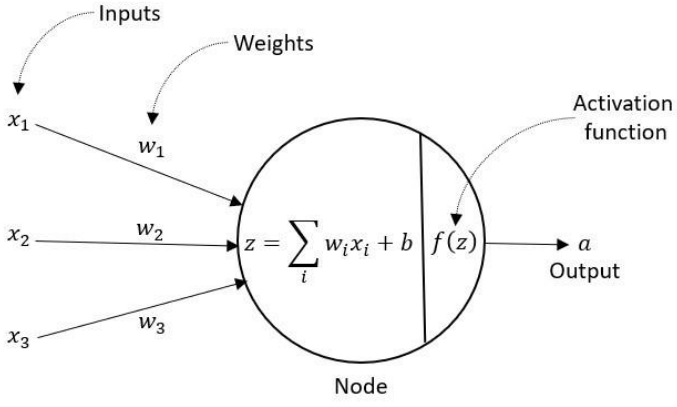
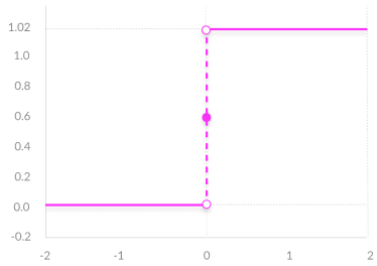


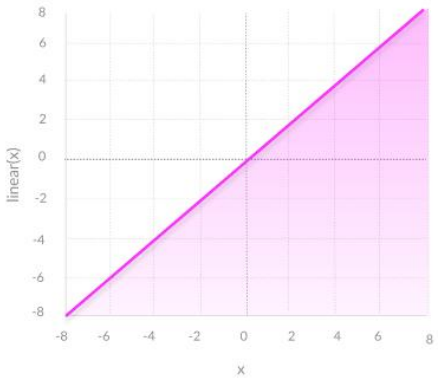
Figure 3-3: Visual representation of a neuron

The image source of every figure showed in this section below is from reference [13]. There are three types of activation function: step-functions, linear and non-linear functions:

1. **Step function**: if the input value is above certain threshold, it sends the same signal to the next layer. The main drawback of this function is that only works with binary classification problem.



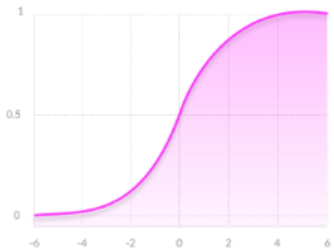
2. **Linear functions**: it takes the inputs and computes a value proportional to the input. It addresses the binary classification problem of step functions but presents two limitations:
    a. The derivative of the function is constant, so that backpropagation (see section 3.1.8 and reference [14]) cannot be performed.
    b. If all layers use linear function, every layer is a linear combination of the rest, and the neural network is simply a linear regression model.



3. **Non-linear functions**: in computer vision, almost every activation function is non-linear, which allows the model to learn very complex data. They allow multi-class classification problems, backpropagation and deep neural networks. The main non-linear functions are:

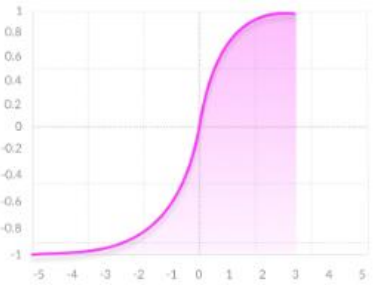
a. **Sigmoid** or Logistic: it computes the output between 0 and 1, therefore, it is very useful for multi-class tasks, where the probability of belonging to a class is computed. However, for very high or very low values of the input, there is no change in the prediction, resulting in the network to get stuck during training (vanishing gradient problem).

$$\phi(z) = \frac{1}{1 + e^{-z}} \tag{3}$$



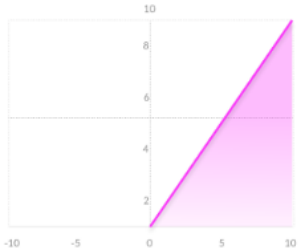
b. **Hyperbolic tangent** (tanh): it addressed the task of model strongly negative and positive values, although it presents the same vanishing gradient problem. It is mainly used in two-classes classification problems.

$$\sigma(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{4}$$



c. **Rectified Linear Unit** (ReLU): it allows the network to converge faster than previous activation functions. However, when the input signal values are close to zero or negative, the gradient of the function becomes null and backpropagation cannot be performed.

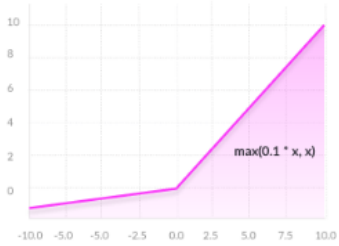
$$ReLU(z) = \begin{cases} z & z > 0 \\ 0 & otherwise \end{cases} \quad (5)$$



d. **Leaky ReLU**: It allows the backpropagation when the input is negative. Nonetheless, predictions are not very consistent for very negative input values.

$$Leaky\ ReLU(z) = \begin{cases} z & z > 0 \\ a * z & otherwise \end{cases} \quad (6)$$

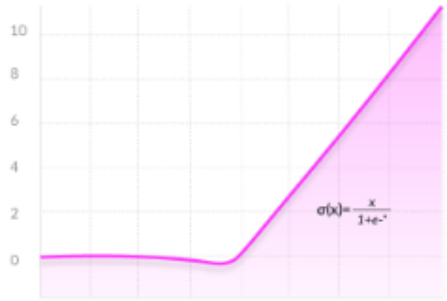
e. **SoftMax**: It is widely used for multi-class algorithms in the last layer of the model, as it normalizes the output between 0 and 1 and divides it by their sum, representing the probability of the input being in a specific class.



$$f_i(\vec{a}) = \frac{e^{a_i}}{\sum_k e^{a_k}} \quad (7)$$

f. **Swish**: recently discovered by Google [15] , it is very similar to ReLU but performs better with a similar level of computational efficiency. It is a non-monotonic function, unbounded above and bounded below. Experiments shows that Swish work better than ReLU for deeper models, as it solves the ReLU problem when derivative is 0. When that happens, Swish function never update that parameter and assigns the parameter back to itself.

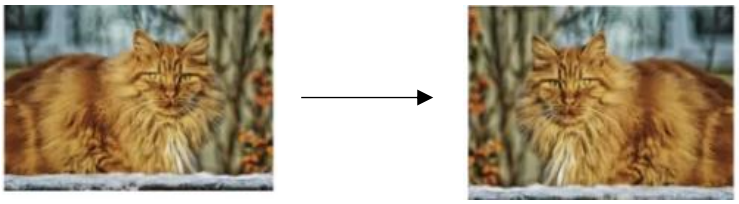
$$\sigma(x) = \frac{x}{1 + e^{-x}} \tag{8}$$



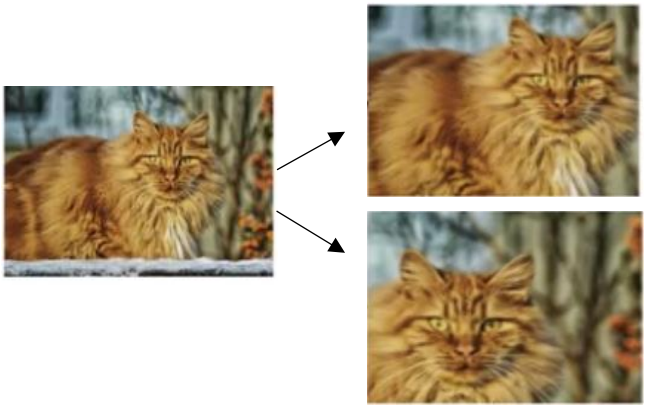
3.1.6 DATA AUGMENTATION

Data augmentation is applied when more data is needed and to improve the performance of the model. Almost every model improves if more data is fed to the model., however, that is not always possible. For that reason, there are some techniques to increase the size of the dataset:

- **Mirroring** on the horizontal or the vertical axis.



- **Random Cropping**



- **Rotate certain angle**.

- **Colour shifting**.
- **Local warping**.

3.1.7 LOSS FUNCTIONS

The function used to evaluate the performance of the model, for instance, a set of weights and bias, is called the objective function. This function is maximized or minimized using the backpropagation of error algorithm (see 3.1.8). Typically for neural networks, the function is minimized and it is referred as cost function, loss function or error function. It plays an important role during training, as it says whether the model is improving or decreasing its performance. For that reason, knowing which loss function to use becomes crucial:

1. **Cross-Entropy**: Also known as log-loss, it "measures the performance of a classification model whose output is a probability value between 0 and 1" [16]. As the probability diverges from the actual label, cross-entropy loss increases. Therefore, the perfect model would have a 0 log-loss value. As it can be seen in the graph below, as probability decreases, cross-entropy increases rapidly, penalizing predictions that are confident and wrong.

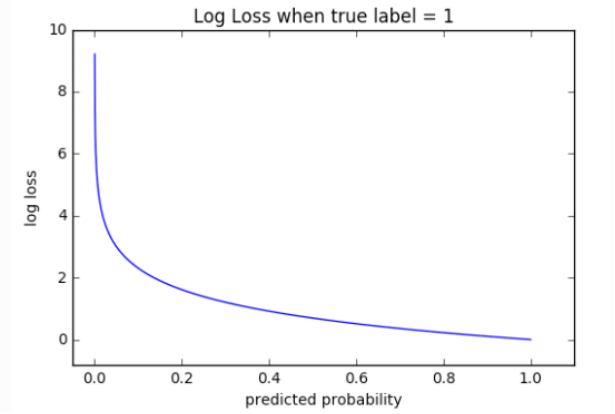


Figure 3-4: Cross entropy values between 0 and 1.

$$Cross\ entropy = -\sum_{c=1}^{M} y_{o,c} \log\left(p_{o,c}\right) \tag{9}$$

where M is the number of classes, y is a binary indicator whether class label c is the correct classification for observation o and p is the predicted probability.

2. **Triplet loss** [17]: it operates on triplets, which basically are three examples from the dataset:
   a. x_i^a – the anchor example.
   b. x_i^p – the positive example, an image that has the same identity as the anchor
   c. x_i^n – the negative example, an image from a different entity than the anchor.

   Then, the loss function is described using a Euclidean distance, so that the positive example is closer to the anchor than the negative:

$$\mathcal{L}(x_i^a, x_i^p, x_i^n) = \max\left(\sum_i^N \left[\left\|f(x_i^a) - f(x_i^p)\right\|^2 - \|f(x_i^a) - f(x_i^n)\|^2 + \alpha, 0\right]\right) \quad (10)$$

3. **Huber Loss**: it represents the absolute error that becomes quadratic when error is small.

$$L_\delta = \begin{cases} \frac{1}{2}(y - \hat{y})^2 & if\ |(y - \hat{y}| < \delta \\ \delta\left((y - \hat{y}) - \frac{1}{2}\delta\right) & otherwise \end{cases} \quad (11)$$

where δ is a threshold that can be modified while training.

3.1.8 BACKPROPAGATION

Proper training of model is one of the most important and critical aspects. Backpropagation is the technique of tuning the weights (3.1.1) of each convolutional layer, based on the error rate (or lost in 3.1.7) obtained in the previous iteration. By tuning the weights, the model learns to predict the output, and that is characteristic from Supervised Learning.

Taking the layers, filters and activation functions defined previously, the model output a probability of an object being from a determined class. As the weights are randomly initialized first, this output will not match the real object. However, the model now can go back and compute the difference between the output and the real object and knows how every layer and every activation function contributed through this process.

Gradient Descent is the most used optimization function for backpropagation. It calculates the partial derivative of each convolution backwards from the output and adjust the weights, so that it minimizes the difference between the output and the real object in the next iteration. The purpose of partial derivative is to find out which node is responsible for the highest loss value in each layer and penalizes it by decreasing its weight.

For a similar example as the one in section 3.1.1, here the blue matrix (X_{ij}) represents the input, and the orange matrix (F_{ij}) the filter:

$$\begin{bmatrix} O_{11} & O_{12} \\ O_{21} & O_{22} \end{bmatrix} = \text{Convolution}\left(\begin{bmatrix} X_{11} & X_{12} & X_{13} \\ X_{21} & X_{22} & X_{23} \\ X_{31} & X_{32} & X_{33} \end{bmatrix}, \begin{bmatrix} F_{11} & F_{12} \\ F_{21} & F_{22} \end{bmatrix} \right)$$

$$\begin{aligned} O_{11} &= F_{11}X_{11} + F_{12}X_{12} + F_{21}X_{21} + F_{22}X_{22} \\ O_{12} &= F_{11}X_{12} + F_{12}X_{13} + F_{21}X_{22} + F_{22}X_{23} \\ O_{21} &= F_{11}X_{21} + F_{12}X_{22} + F_{21}X_{31} + F_{22}X_{32} \\ O_{22} &= F_{11}X_{22} + F_{12}X_{23} + F_{21}X_{32} + F_{22}X_{33} \end{aligned} \qquad (12)$$

The error, E, is defined as follows:

$$\begin{aligned}
\frac{\partial E}{\partial F_{11}} &= \frac{\partial E}{\partial O_{11}}\frac{\partial O_{11}}{\partial F_{11}} + \frac{\partial E}{\partial O_{12}}\frac{\partial O_{12}}{\partial F_{11}} + \frac{\partial E}{\partial O_{21}}\frac{\partial O_{21}}{\partial F_{11}} + \frac{\partial E}{\partial O_{22}}\frac{\partial O_{22}}{\partial F_{11}} \\
\frac{\partial E}{\partial F_{12}} &= \frac{\partial E}{\partial O_{11}}\frac{\partial O_{11}}{\partial F_{12}} + \frac{\partial E}{\partial O_{12}}\frac{\partial O_{12}}{\partial F_{12}} + \frac{\partial E}{\partial O_{21}}\frac{\partial O_{21}}{\partial F_{12}} + \frac{\partial E}{\partial O_{22}}\frac{\partial O_{22}}{\partial F_{12}} \\
\frac{\partial E}{\partial F_{21}} &= \frac{\partial E}{\partial O_{11}}\frac{\partial O_{11}}{\partial F_{21}} + \frac{\partial E}{\partial O_{12}}\frac{\partial O_{12}}{\partial F_{21}} + \frac{\partial E}{\partial O_{21}}\frac{\partial O_{21}}{\partial F_{21}} + \frac{\partial E}{\partial O_{22}}\frac{\partial O_{22}}{\partial F_{21}} \\
\frac{\partial E}{\partial F_{22}} &= \frac{\partial E}{\partial O_{11}}\frac{\partial O_{11}}{\partial F_{22}} + \frac{\partial E}{\partial O_{12}}\frac{\partial O_{12}}{\partial F_{22}} + \frac{\partial E}{\partial O_{21}}\frac{\partial O_{21}}{\partial F_{22}} + \frac{\partial E}{\partial O_{22}}\frac{\partial O_{22}}{\partial F_{22}}
\end{aligned} \tag{13}$$

That is simplified to:

$$\begin{aligned}
\frac{\partial E}{\partial F_{11}} &= \frac{\partial E}{\partial O_{11}}X_{11} + \frac{\partial E}{\partial O_{12}}X_{12} + \frac{\partial E}{\partial O_{21}}X_{21} + \frac{\partial E}{\partial O_{22}}X_{22} \\
\frac{\partial E}{\partial F_{12}} &= \frac{\partial E}{\partial O_{11}}X_{12} + \frac{\partial E}{\partial O_{12}}X_{13} + \frac{\partial E}{\partial O_{21}}X_{22} + \frac{\partial E}{\partial O_{22}}X_{23} \\
\frac{\partial E}{\partial F_{21}} &= \frac{\partial E}{\partial O_{11}}X_{21} + \frac{\partial E}{\partial O_{12}}X_{22} + \frac{\partial E}{\partial O_{21}}X_{31} + \frac{\partial E}{\partial O_{22}}X_{32} \\
\frac{\partial E}{\partial F_{22}} &= \frac{\partial E}{\partial O_{11}}X_{22} + \frac{\partial E}{\partial O_{12}}X_{23} + \frac{\partial E}{\partial O_{21}}X_{32} + \frac{\partial E}{\partial O_{22}}X_{33}
\end{aligned} \tag{14}$$

Once the error and the filters are updated, the next iteration takes places considering the new weights. For further information on this topic, reference [18] includes the calculations of the gradient descent for an easy example.

3.2 OBJECT DETECTION

Object detection is an essential step for many computer vision applications, such as autonomous driving, visual surveillance and facial recognition [19]. The architecture defined for object detection is usually used as the basis to extract objects' spatial and classification information. Depending on the application, more modules are added to this architecture. For

this reason, big effort and resources has been put into the development of the object detection algorithms.

Before object detection came into play, the deep learning community was focused in image classification. For this task, the object of the image was usually centred and dominating the image frame. However, images usually contain more than one object, and as result image classification was limited.

To address the problem of pictures with more than one object, a two-stage approach is done. First, a region proposal and then, image classification on the proposed regions. These regions are smaller than the original image and allow the model to detect multiple objects with different sizes. Furthermore, same algorithms of image classification can be used after region proposal.

Two-stage models usually yields higher accuracy. However, these models require huge computation and are limited for real-time detection. To avoid that, thanks to the increment of computing power available, one-stage detectors have arisen significantly.

One-stage models divide the image into grids, "*where each grid is responsible for the object whose centre falls into that grid*" [20]. Due to its simpler architecture, one-stage models usually require less computation resource. On the other hand, accuracy is compromised, especially for small objects, caused by extreme foreground-background class imbalance during training.

To solve these drawbacks, algorithms described below have developed different strategies depending whether they are one-stage or two-stage detectors. But first, two useful concepts for object detection are defined to better understand the algorithms.

1. Non-max Suppression

When the image is divided into cells, one object can be in multiple bounding boxes. This usually happens because the grid is small enough to detect very small objects. When

detecting a car, for example, multiple grid cells detect the car with a given probability. To avoid that the model thinks that there are three cars, the non-max suppression technique discards all boxes which intersection is above certain threshold, and the box with the highest probability remains in the image.

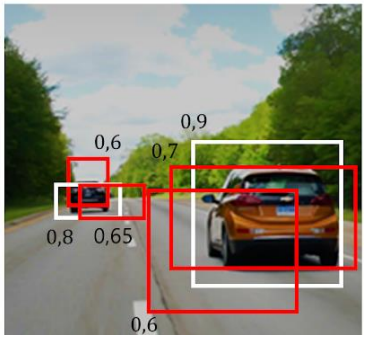


Figure 3-5: Example of multiple boxes around each object, before applying non-max suppression

Image source: http://datahacker.rs/deep-learning-non-max-suppression/

In this image, white boxes are the ones with the highest probability. Therefore, for the car on the right, boxes with a probability of 0,7 and 0,6 would be deleted because they are overlapping the white box with the 0,9 probability and they are lower.

2. Anchor Boxes

Anchor boxes are a set of predefined bounding boxes to detect the objects in the image. These boxes are chosen based on object sizes (scale and aspect ratio) in the training dataset, with a certain height and width to better detect the specific class of the object. Then, predictions are used to refine each anchor box depending on the object size. Next example shows the initial step where two anchor boxes are defined: the red box is for the class

Figure 3-6: Example of bounding box processing

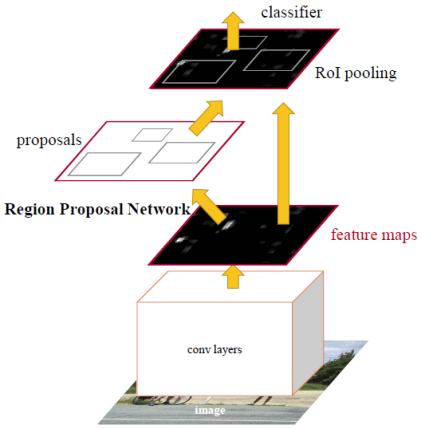
Image source:: https://es.mathworks.com/help/vision/ug/anchor-boxes-for-object-detection.html

"airplane" and the green one for the class "sailboat". After training, bounding boxes are placed on the objects, and finally, non-max suppression is applied to remove the redundant boxes.

3.2.1 FASTER-RCNN

Faster-RCNN is a two-stage detector that shares the same convolutional neural network to accomplish both tasks: region proposal generation and object detection. Its name comes after RCNN [21] and Fast-RCNN [22] where region proposal is generated by selective search [23] and the CNN network is used to classify the object class, in contrast to Faster-RCNN, that uses the same convolutional net for both tasks.

1. **Region Proposal Network (RPN)**:
    a. The input image goes through the convolutional network to obtain the feature maps. (see 3.1.1). The CNN can be VGG-16 (see Appendix 3), ResNet50 (see Appendix 4), ResNet100, etc.
    b. A sliding window is applied for each location over the feature map.
    c. k anchor boxes ($k = 9$ in the paper) with 3 scales (128, 256 and 512) and 3 aspect ratios (1:1, 1:2, 2:1) are used for each location to generate region proposals.



Source: [1]

    d. Then, a classification (cls) layer outputs 2k scores whether there is an object or not for every k boxes.
    e. At the same time, a regression (reg) layer outputs 4k scores for the coordinates of k boxes: box centre coordinates (x,y), width and height.

f. Finally, each feature map has a size of WxH and there are WHk anchors in total.

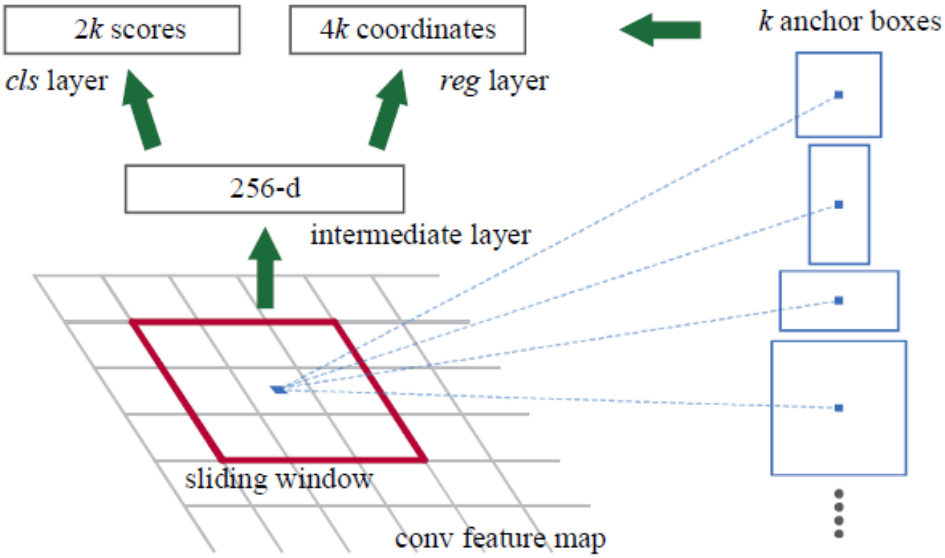


Image source: ref. [25]

The loss function used is:

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i,p_i^*) + \lambda\frac{1}{N_{reg}}\sum_i p_i^* L_{reg}(t_i,t_i^*). \quad (15)$$

where the first term is the classification loss whether there is an object or not, and the second term is the regression loss of bounding boxes when there is an object.

RPN is used to check the locations that contain objects, and pass the bounding boxes to the detection network. As these regions are highly overlapped, non-max suppression (SEE 2.2.1) is applied to reduce the number of proposals from 6000 to 300, depending on the image.

2. **Region of interest pooling (RoI Pooling)**:

   After RPN, the network has all object proposal made by the RPN with no class assigned. The traditional approach would be taking each one of the proposals and pass it through the base network. However, considering that there are 6000 proposals this technique would take too long.

   To avoid that, region of interest pooling extracts fixed-sized feature maps for each proposal, reused from the existing convolutional feature map. Next figure shows an example where the conv net is processing the class "bicycle":

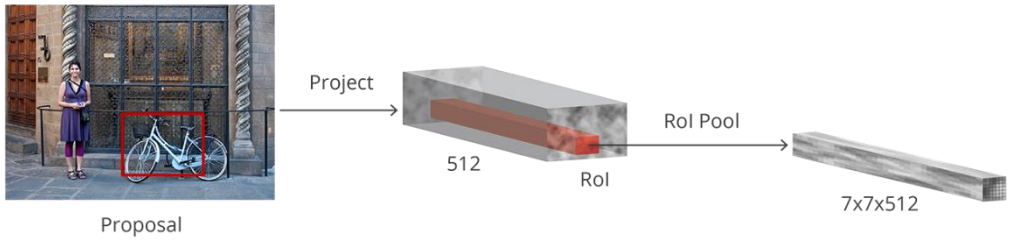


Figure 3-7: RoI pooling example.

Image source. https://tryolabs.com/blog/2018/01/18/faster-r-cnn-down-the-rabbit-hole-of-modern-object-detection/

   RoI pooling splits the input feature map into feature maps with the same size, and then apply Max-Pooling (see 3.1.4) on every region. Hence, the output of RoI is always the same. The main advantage of this method is that different sized regions would require different CNN feature maps, reducing the efficiency of the structure. In contrast, with fixed-size inputs and outputs, the same architecture is used for the regressor and the classifier.

   A deeper explanation about Region of Interest pooling can be found in reference [24].

3. **Region-based Convolutional Neural Network (R-CNN)**:

   After extracting features for each proposal with RoI Pooling, these features are used for classification with R-CNN, the last step of the algorithm. Therefore, R-CNN has two goals:

   - Classify proposals (including background to remove bad proposals).
   - Better adjust bounding box according to the class.

To do so, the paper [25] uses two fully-connected layers of size 4096 and ReLU activation function are applied to the feature map after Max-Pooling:

- The first fully-connected layer with $N+1$ units, where N is the number of classes plus the background.
- The second fully-connected layer with $4*N$, for $\Delta_{center_x}, \Delta_{center_y}, \Delta_{width}, \Delta_{height}$ for N classes.

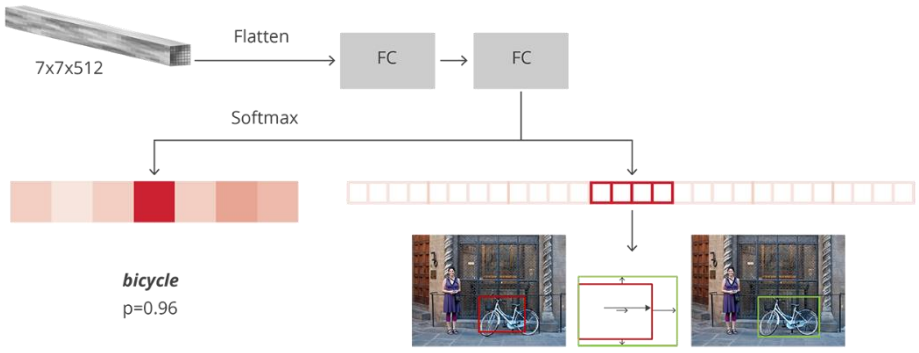


Figure 3-8: Continuation of the RoI pooling example for the class 'bicycle'

To assign each object to a class, a softmax activation function is applied which computes the probability of each class.

3.2.2 YOU ONLY LOOK ONCE (YOLO)

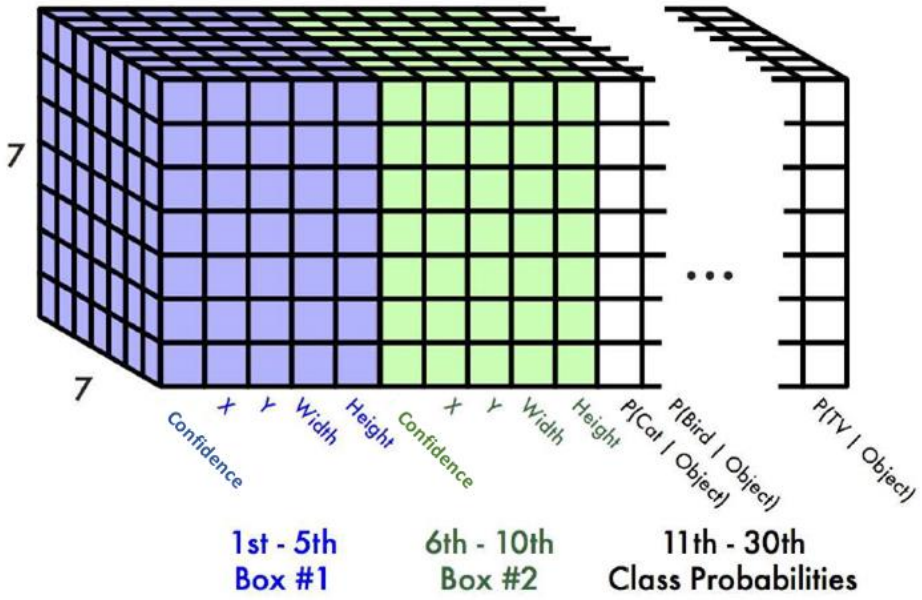
As its name says, this network only looks the image once to detect multiple objects. Thanks to this, the detection speed achieved is one of the highest in the state-of-the-art algorithms for object detection. In contrast to Faster-RCNN, YOLO is a one-stage detector, that covers both region proposal and classification at the same time.

1. **Unified Detection**:

   Input image is divided into an SxS grid (in the YOLO9000 paper [26], $S = 7$). Then, each grid cell predicts B bounding boxes ($B = 2$), computing the confidence score for each of those boxes. Therefore, each box has 5 labels:

- The (x, y) coordinates of the center of the box with respect to the grid cell in which is contained.
- The width w and the height h of the box.
- The confidence with respect to the Intersection over Union (see 3.2.4.7) between the box and the ground truth box (if any).

Then, the output of the network should include the probability for each class. In next example, the number of classes is 20, $S = 7$ and two boxes:



Therefore, the output size is: $7 * 7 * (5 + 5 + 20) = 1470$

2. **Architecture of the network**:

YOLO network consists of 26 convolutional layers, including two fully-connected layers at the end, with size of 4096, similar to the Faster-RCNN algorithm. As it can be seen in the next figure, 1x1 convolutional layers are alternated through the architecture to reduce the feature maps, and reduce the number of parameters with it. GoogleLeNet [27] uses this approach to develop very deep convolutional networks.

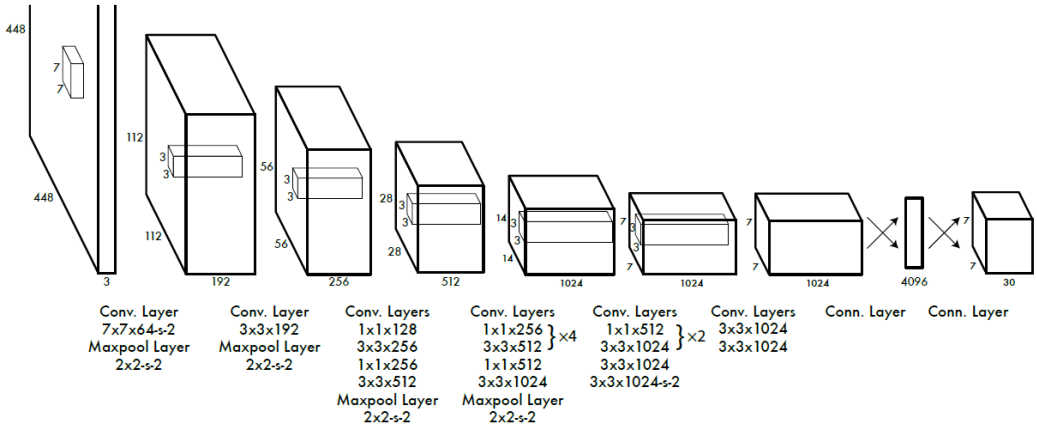


Figure 3-9: YOLO architecture. Source: ref. [58]

3. **Loss Function**:

There are five terms in the equation shown above:

- The (x, y) coordinates of the box are parametrized for a particular grid cell location so they are bounded between 0 and 1. The term 1_{ij}^{obj} indicates whether there is an object or not, so that the sum of square error is only computed when there is one.

  In the first two terms, λ_{coord} is used to increase the loss of the bounding boxes when there are objects. A typical value is $\lambda_{noobj} = 5$.
- The width and the height of the box are normalized by the image width and height. As in the previous case, the sum of square error is only computed when there is an object. Square root is used because small deviations in large boxes are less important than in small boxes, and square root addresses this problem.
- The third and the fourth term are related with the confidence, expressed as the Intersection over Union between the predicted box and the ground truth box. As there are many grid cells in every image that do not contain any object, the loss from confidence predictions for these boxes is decreased by

using $\lambda_{noobj} = 0{,}5$. This is done to avoid that the grid cells that do not contain any object do not push the confidence score of these cells towards zero.

- The last term represents the sum of square errors of each class probability when there are objects.

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2 \qquad (16)$$

4. **Activation functions**:

   Every layer uses leaky ReLU activation function, excepts for the final layer, that uses softmax to compute the class probability for each prediction.

   Additionally, non-max suppression and data augmentation are used when training this network.

3.2.3 RETINANET

As it was said at the beginning of the chapter, the main drawback with one-stage detectors is the accuracy in comparison with two-stage detectors, like Faster-RCNN. In RetinaNet, the introduction of a new concept, the focal loss, tries to deal with this reduction in accuracy by focusing in "hard" samples, instead of "easy" samples. The term "easy" samples refer to those samples where the background is the predominant class, which are mainly dismissed in the first stage of Faster-RCNN, while one-stage detectors do not have that possibility.

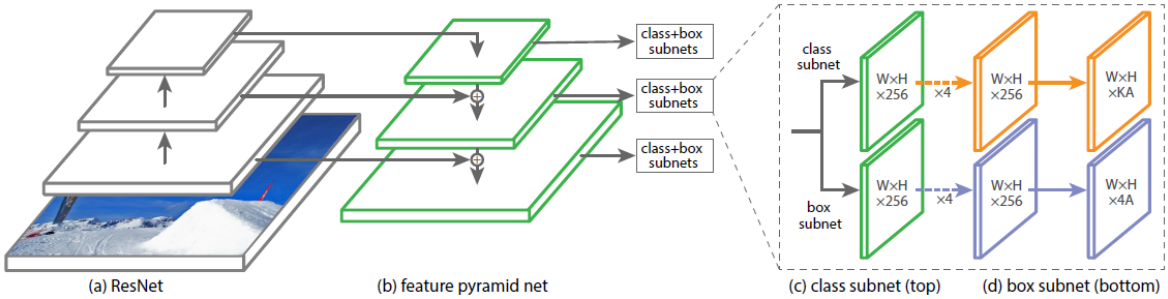


Figure 3-10: Feature Pyramid Network, bounding box regression and classification schemes. Image source: ref. [28]

In the figure above, it can be seen that RetinaNet is composed of a ResNet model (a), a Feature Pyramid network (b), and two task-specific networks for bounding box regression and classification (c and d).

1. ResNet - Focal Loss:

   Many one-stage detectors, like Single Shot Detector (SSD), struggles when there is a dominating class imbalance between the class to predict and the background. The traditional approach is to reduce the number of boxes that the model process in every iteration. However, RetinaNet uses Focal Loss to deal with class imbalance and takes around 100k boxes, in contrast with YOLO that uses 1k and the 2k boxes of Faster-RCNN.

   Starting from the cross entropy (CE) loss for binary classification:

$$CE(p,y) = \begin{cases} -\log(p) & if\ y = 1 \\ -\log(1-p) & otherwise \end{cases} \tag{17}$$

   where y specifies the ground-truth class and p is the probability for the class y.

   Then, CE is extended to multi-class case, and p_t is defined as:

$$p_t = \begin{cases} p & if\ y = 1 \\ 1-p & otherwise, \end{cases} \tag{18}$$

   and $CE(p,y) = CE(p_t) = -\log(p_t)$

But still, class imbalance is not solved yet. To do so, the loss function is reshaped to down-weight easy examples and focus on hard negatives, with a modulating factor $-(1-p_t)^{\gamma}$. The value of γ is a hyperparameter to be tuned, test from 0 to 5 in the experiment. The focal loss is defined as:

$$FL(p_t) = -\alpha_t(1-p_t)^{\gamma}\log(p_t) \tag{19}$$

The term α_t is added to deal with class imbalance, which vary its value depending on the class. Moreover, a prior value of $p = 0{,}01$ is set for the initial value during training, so the model estimation p for examples containing rare classes are low.

2. Feature Pyramid Network [28]:
    - **Bottom-Up Pathway**: It has one pyramid level for each stage, where the output of the previous layer feeds the next layer of the convolutional network. Moreover, these outputs are used for enriching the top-down pathway by means of lateral connection.
    - **Top-Down Pathway**: as it goes higher, *feature maps are semantically stronger* [29]. Lateral connections are made from the bottom-up pathway to the top-down by means of $1x1$ convolutions to reduce the channel dimension. Finally, element-wise addition is used to merge both pathways.
    - **Prediction**: a $3x3$ convolution generates the final feature map from the element-wised map. The feature dimension of every map is fixed at 256, as every layer of the pyramid uses shared classifiers and regressors parameters.
3. Classification and bounding box subnetworks:

    Both are Fully Connected Networks that applies four $3x3$ convolutional layers followed by ReLU activations functions, $3x3$ convolutional layers with $classes\ x\ anchors$ filters and softmax activation function in the case of

classification, and $4 \, x \, anchors \; linear \; outputs \; per \; spatial \; location$ [29] in the case of bounding box regression.

RetinaNet uses **hard negative mining** [30] to improve the accuracy of the classifier. Hard negative mining consists on taking the wrong classified bounding boxes, and feeding the model with those examples as negative examples (bounding boxes where the object to classify is not present).

3.2.4 EVALUATING OBJECT DETECTION MODELS

Measuring and evaluating the performance of the model is essential to distinguish the next steps while training any model. Using more than one indicator is a good practice to really test the performance of the model in difference situations.

This section introduces the main indicators used by the Deep Learning community, which are:

- Classification Accuracy
- Confusion Matrix
- Logarithmic Loss
- Area Under de Curve
- Mean Absolute Error (MAE)
- Mean Squared Error (MSE)
- Mean Average Precision (mAP)

3.2.4.1 Classification Accuracy

It is the most extended indicator as it represents the common understanding of accuracy, where the correct predictions are divided by the total number of predictions:

$$Accuracy = \frac{Number \; of \; correct \; predictions}{Total \; number \; of \; predictions \; made}$$

It is widely used during training and validation. However, it could fail if the number of samples of each class in a multi-class algorithm is not the same. For instance, if the model predicts the output always belonging to class A, and class A represents the 80% of samples in the dataset, the model will obtain an 80% accuracy, although the model has not really learned anything.

3.2.4.2 Confusion Matrix

As the name suggests, confusion matrix is a matrix that describes the output of the model. It is very useful with binary classification problems, where the inputs are A and B:

- **True Positive**: the prediction and the actual output is the same, A
- **True Negative**: both prediction and output are B.
- **False Positive**: the prediction is A and the output is B
- **False Negative**: the prediction is B and the output is A.

Then, the accuracy of the matrix is calculated as follows:

$$Accuracy = \frac{True\ Positives + False\ Negatives}{Total\ number\ of\ samples}$$

3.2.4.3 Logarithmic Loss

Log Loss penalize the false classifications, especially in multi-class classification problems. It is defined as follows:

$$Log\ Loss = \frac{-1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij} * \log(p_{ij}) \tag{20}$$

where N indicates the number of samples, M is the number of classes, y_{ij} whether sample I belongs to class j, and p_{ij} its probability.

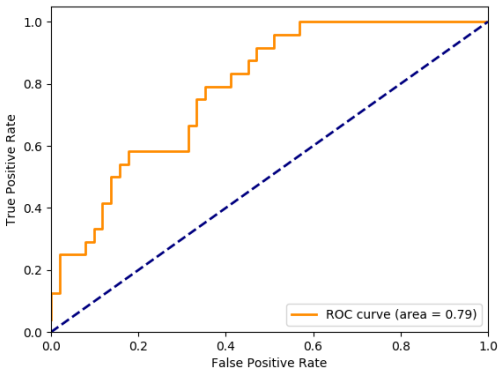
A log loss value near 0 indicates high accuracy, while greater values indicates lower accuracy.

3.2.4.4 Area Under the Curve (AUC)

Very related with confusion matrix, the AUC outputs the probability of the classifier choosing a positive example rather than a negative one. To do so, two new terms are defined:

- **True Positive Rate – Sensitivity**: it is the ratio between True Positive samples and all positive data samples, which are the False Negative plus the True Positives samples.
- **False Positive Rate** – **Specificity**: represents the samples mistakenly considered positives (False Positives) divided by the negative samples (False Positive and True Negative)

Then, both False and True Positive Rates are computed at different values from 0 to 1 and plotted:



The AUC indicates the area in a range between 0 and 1. The greater the value, the better the performance.

3.2.4.5 Mean Absolute Error (MAE)

It represents how far the predictions are from the output. However, it is not possible to know whether the prediction is under or over the data. The formula to calculate the mean absolute error is:

$$Mean\ Absolute\ Error = \frac{1}{N}\sum_{j=1}^{N}|y_j - \hat{y}_j| \tag{21}$$

3.2.4.6 Mean Squared Error (MSE)

MSE is very similar to MAE except that it takes the average of the square of the difference. Hence, the effect of larger errors is more pronounced than smaller errors. MSE is computed as follows:

$$Mean\ Squared\ Error = \frac{1}{N}\sum_{j=1}^{N}\left(y_j - \hat{y}_j\right)^2 \tag{22}$$

3.2.4.7 Intersection over Union (IoU)

IoU measures the overlap between the prediction and the ground truth. In the case of object detection with bounding boxes. It is measure as follows:

$$IoU = \frac{Area\ of\ overlap}{Area\ of\ Union}$$

In some cases, an IoU threshold value is defined to classify whether the prediction is a TP (True Positive) or a FP (False Positive).

3.2.4.8 Mean Average Precision (mAP)

Average Precision indicator has become very popular, thanks to the COCO (Microsoft Common Objects in Context) challenge, which uses this indicator to measure each algorithm performance during test.

mAP is a combination of Intersection Over Union, precision and recall, which are defined as follow:

- **Precision** measures the percentage of the predictions that are correct.

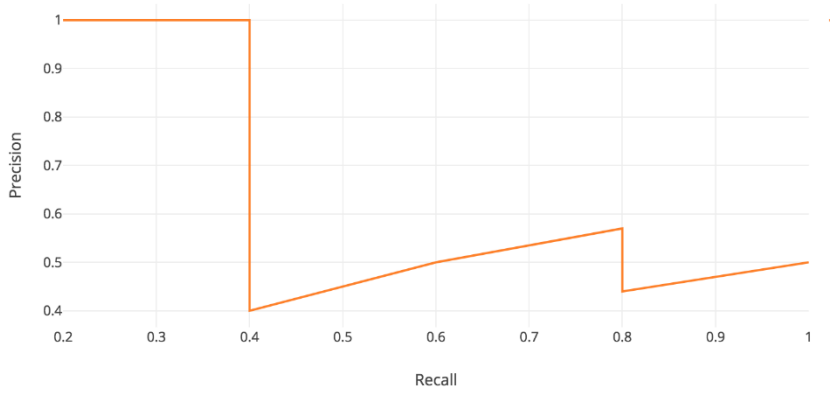
$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

- **Recall** measures how good all the positives predictions are found.

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

Given the following example, where the second column ("Correct?") indicates True if the IoU ≥ 0,5 and the precision and recall are plotted:

Rank	Correct?	Precision	Recall
1	True	1.0	0.2
2	True	1.0	0.4
3	False	0.67	0.4
4	False	0.5	0.4
5	False	0.4	0.4
6	True	0.5	0.6
7	True	0.57	0.8



The Average Precision is defined as the area under the precision-recall curve and will always be between 0 and 1.

For the COCO competition, the AP is the average over IoU threshold from 0,50 to 0,95 with a step size of 0,05. Then, the AP is averaged over all categories, which is traditionally known as the mean Average Precision.

3.3 SEMANTIC SEGMENTATION

Semantic segmentation, also known as image segmentation, classifies every pixel of an image into a class. Every object of the same class will have the same color. In contrast to object detection where the model rounds the object with a box, semantic segmentation colors the pixel depending on the class. Next figure shows an example:

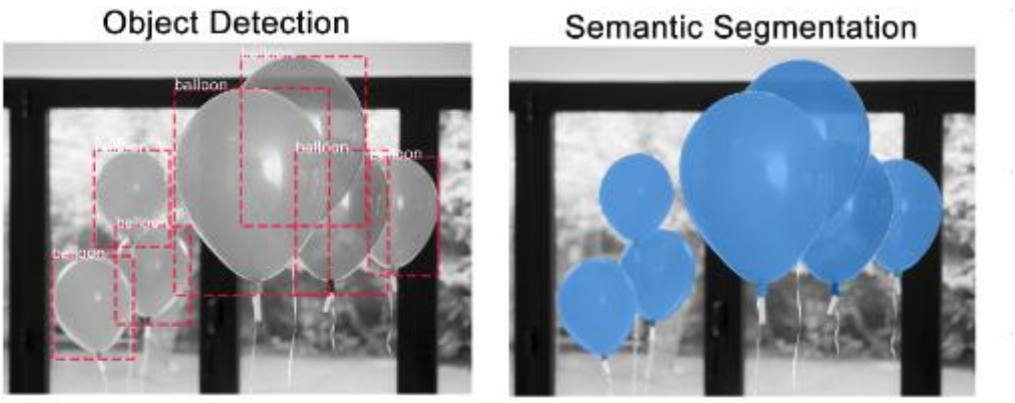


Figure 3-11: Object detection and semantic segmentation samples

Source: https://www.researchgate.net/figure/Difference-between-object-detection-semantic-segmentation-and-instance-segmentation_fig1_338570196

3.3.1 MASK-RCNN

Mask-RCNN extends Faster R-CNN [31]. Additionally to the two outputs of Faster-RCNN, which are the class label and the bounding-box, Mask-RCNN adds a branch to predict pixel segmentation masks on each Region of Interests.

As it was seen in Faster R-CNN (Chapter 3.2.1), Region Proposal Network generates multiple Region of Interests where the classifier and the regressor process the object class and the bounding boxes. But, before the two fully connected layers, the warped feature maps

are fed into the mask generator, where two convolutional layers generates the mask for each class.

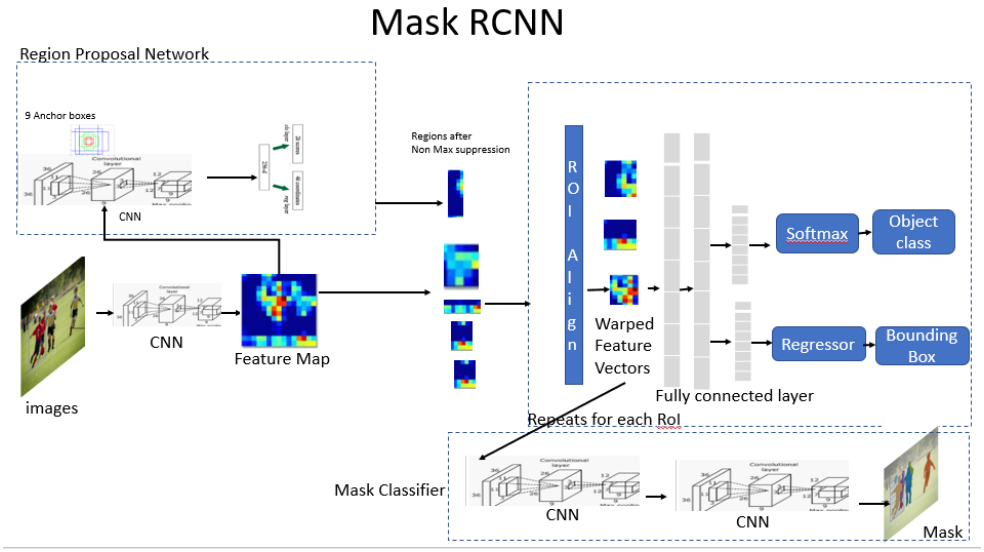
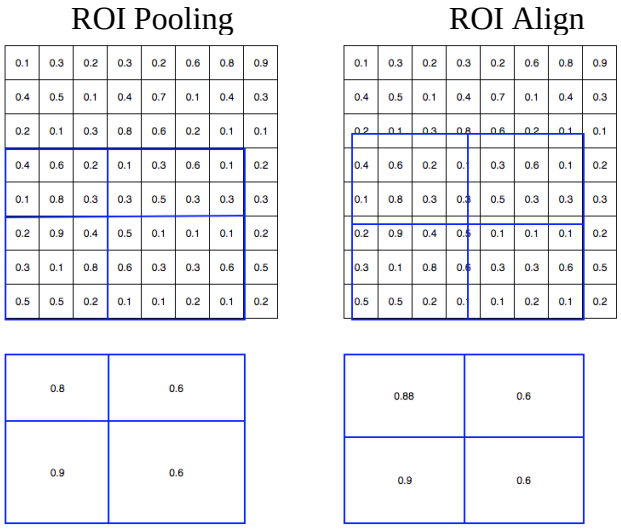


Figure 3-12: Mask-RCNN representation. Source: ref. [31]

The figure above shows that ROI Pooling module in Faster-RCNN is replaced with ROI Align. The problem with ROI Pooling is that certain regions of the feature map are slightly misaligned from the original image. This is the case when the boundaries of the feature map are out of the input size. To avoid this, the authors of the paper [31] developed ROI align, that make every feature map has the same size. Furthermore, it applies interpolation to compute the value more accurate:



The value of 0.88 of the bottom-right cell is an interpolation between the 0.8 and the 0.9 values of the matrix above. It can be seen that this value differs from the 0.8 obtained by ROI pooling. Test results showed that ROI align improved the accuracy of the model.

Finally, the total loss of the model is the sum of the bounding box regression, the box classifier and the mask classifier. Moreover, non-max suppression is applied to remain only with the top Intersection over Union masks.

3.3.2 U-NET

U-Net [32] consists on a Fully Convolutional Network whose inputs and outputs have the same size. Its architecture can be seen below. It is divided into contraction and expansion path. Some literature names the region in between both paths as the 'Bottleneck' of the model.

1. Architecture.
    - **Contraction path**: two 3x3 convolutions followed by a 2x2 maximum pooling. As it goes more advanced features are extracted and the size of the feature maps is reduced.
    - **Expansion path**: two 3x3 convolutions and 3x3 up-sampling convolutions (inverse of max. pooling) to resize the image to its original size. By doing that, advanced features are obtained at the expense of localization information. To avoid that, lateral connections between feature maps from the contraction and the expansion paths are done, which are the grey arrows in the image below. The last 1x1 convolution is from the original paper [32] where only 2 classes where present and the feature map dimension had to be resized from 64 to 2.

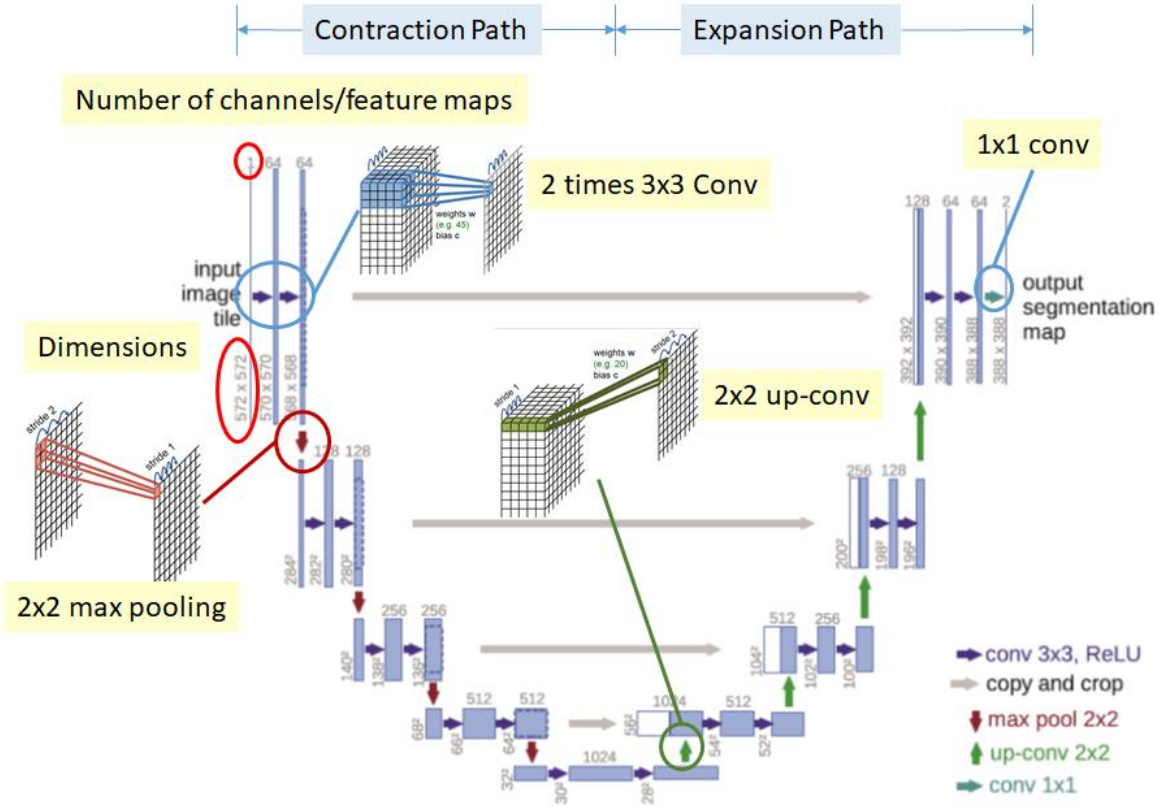


Figure 3-14: U-Net architecture. Source: ref. [32]

2. Overlap-tile strategy.

   Padding is equal to zero in each convolution. Therefore, the size of the output is smaller than the input. To solve that, overlap tile strategy is used to predict each part of the image dividing the input image in smaller parts. Next figure shows an example where the blue area is used to predict the yellow one. When the yellow box is close to the boundary, mirroring is applied to extrapolate the image.

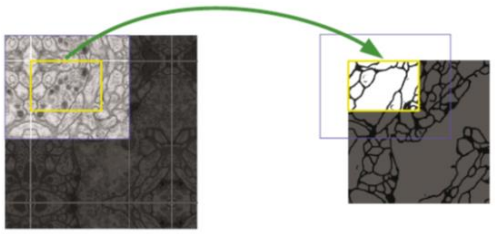
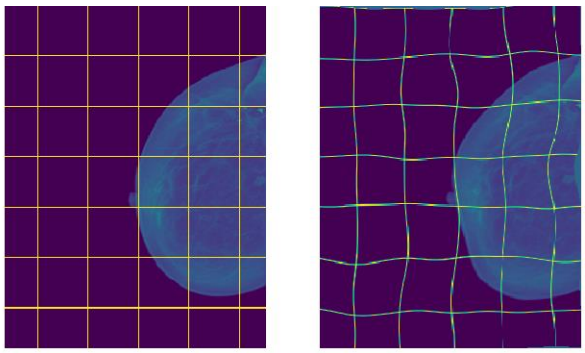


Figure 3-15: Example of overlap-tile strategy.

Image source: https://towardsdatascience.com/review-u-net-biomedical-image-segmentation-

3. Biomedical implementation.

   The first paper that introduced a U-Net architecture was *O. Ronnengerger* [32], that applied this architecture for *segmentation of neuronal structures in electron microscopic stacks*. As the dataset was limited, data augmentation was implemented to improve the performance of the model. Specifically, they implemented elastic deformation that distort the grid of the image. Next figure shows an example of elastic deformation:



(a) Original (b) Deformed

Figure 3-16: example of elastic data augmentation.

Moreover, the relevance to detect the boundaries of neuronal structures and compensate the different frequency of pixels from every class is critical. To solve that, the weight map was computed as:

$$w(x) = w_c(x) + w_0 * \exp\left(-\frac{\left(d_1(x) + d_2(x)\right)^2}{2\sigma^2}\right) \tag{23}$$

where w_c is the weight map that balance the class frequencies, d_1 represents the distance to the nearest cell, d_2 the distance to the second nearest cell and $\sigma = 5\ pixels$.

3.3.3 EVALUATING SEMANTIC SEGMENTATION MODELS

3.3.3.1 Intersection over Union (IoU)

As it was explained in the Object Detection chapter, in the case of image segmentation IoU measures the number of pixels in common between the prediction and the ground truth mask:

$$IoU = \frac{Prediction \cap Ground\ Truth}{Prediction\ \cup Ground\ Truth}$$

where ∩ denotes the intersection and ∪ the union of both masks. The next example illustrates the definition above.

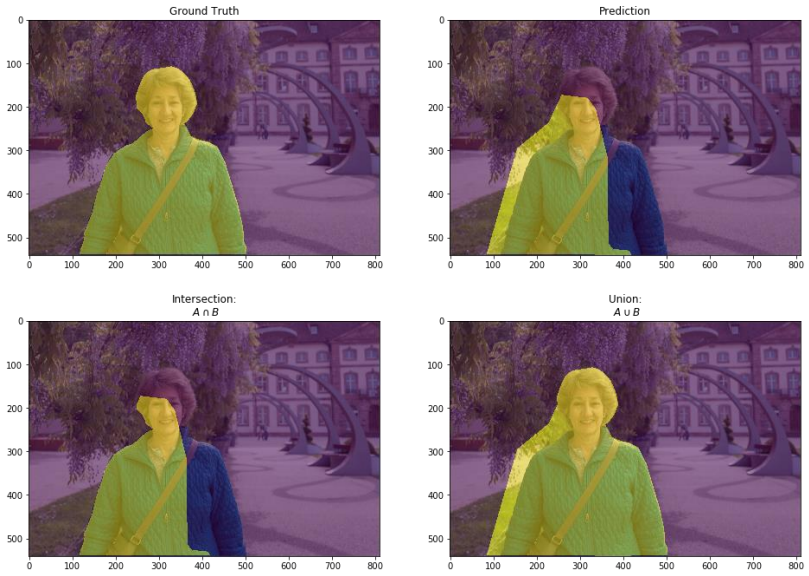


Figure 3-17: Example of how IoU is calculated.

Source: https://www.jeremyjordan.me/evaluating-image-segmentation-models/

3.3.3.2 Pixel Accuracy

It indicates the percentage of pixels in the image that are correctly classified. When considering a binary mask, a true positive represents pixels that are correctly classified to belong to the class while true negative represents pixels belonging to any class.

$$Accuracy = \frac{True\ Positives + True\ Negatives}{True\ Positives + True\ Negatives + False\ Positives + False\ Positives}$$

3.4 INSTANCE SEGMENTATION

Instance segmentation is a combination of multi-class bounding box classification and semantic segmentation. For this task, every object in the image is labeled with a mask (figure below), as semantic segmentation, but every object of the same class has a different mask, allowing the user to count the number of objects of every class.

One common model for instance segmentation is Mask-RCNN, built after Faster-RCNN, that takes each bounding box and masks the object, as it was explained in Chapter 3.3.1.

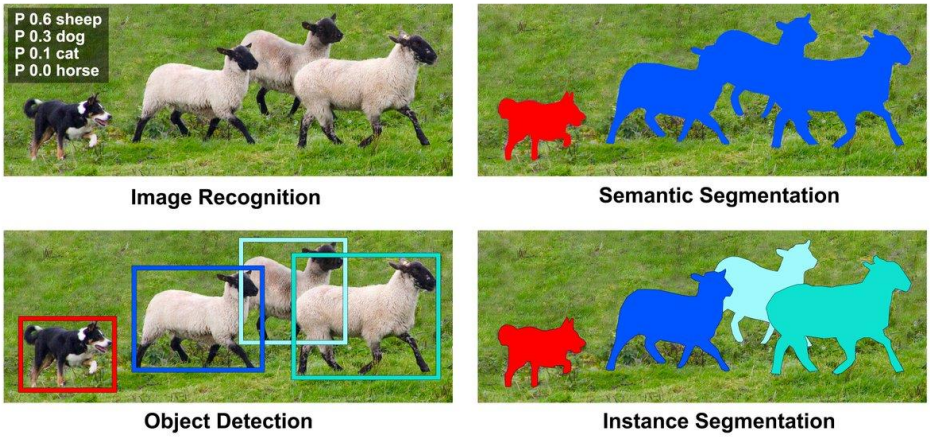


Figure 3-18: Instance segmentation combines object detection and semantic segmentation results.
Source: https://ai-pool.com/d/could-you-explain-me-how-instance-segmentation-works

3.5 APPLICATIONS

As stated in the first Chapter, the applications of convolutional neural networks and computer vision are exponentially growing. Some of the current applications may be obsoleted in a short period of time while new ones will appear in that period. In this chapter, the main applications in the industry are introduced which are developing the state-of-the-art of the models presented in the previous sections:

1. Financial sector:

   Banks and finance services are using computer vision to allow customers scan their IDs and verify the authenticity of the document. This process saves precious time to the customer and the bank.

2. Self-driving cars:

   It is probably the most known application for computer vision. Reduce the number of deaths on road accidents is crucial and reduce the human intervention seems one of the key factors to achieve that. Computer vision algorithms can detect pedestrians, traffic lights, traffic signs and more objects that determine the path of a car and the potential hazards. The development of deep neural networks has enhanced the algorithms that are able to manage self-driving cars. Recently, it seems that the limitation of this technology is the ethical issues associated with car accidents, where the computer may choose between two potential hazard consequences.

3. Facial recognition:

   Used as security scan control, facial recognition replaces the traditional identification card. Because of this, security control can be faster and safer, and more accurate control of the people can be carried out.

4. Biomedical applications:

   Computer vision can help to detect and identify diseases without human intervention or helping it. U-Net is applied for neurons segmentation as it was mentioned in section 3.3.2. The application is non-invasive, free once it is trained and accurate. Therefore, patients can be treated easier and faster.

5. Agriculture:

   "Identify the most efficient growing method, increase yield and reduce waste" [33], or supervise when a crop is ready to harvest are applications of computer vision in agriculture. *SlantView* [34] is applying computer vision to reform regular farms into smart farms.

6. Inventory and warehouse management:

   Computer vision systems scan barcodes and are able to monitor and localize the storage. This is particularly useful for large warehouses where drones are replacing humans for easy tasks, as inventory checks.

4. ACTION RECOGNITION IN VIDEO

Action recognition involves the description and the identification of different actions and goals from video frames "*where different actions could be performed during the time the video is played*" [35]. Although the natural solution for this task is to extend the image classification algorithms exposed in previous sections to multiple frames, action recognition involves several challenges to encounter, these are:

- **Advanced classification architectures**: to accomplish this task, it is crucial to understand the spatiotemporal context of the scene. The state-of-the-art solutions to handle it can be summarized in three main approaches:
  - Process predictions through multiple clips.
  - Two separate networks, one for spatial information and the second for temporal information; or just one network handling both spatial and temporal information.
  - Differentiate between classification and feature extraction tasks or end-to-end training.
- **Huge computational costs**: while image classification takes hours to train a convolutional neural network, the same architecture takes a few days to process video datasets. Depending on the context and the resources, this drawback could become an entry barrier to train a video algorithm from scratch.
- **Capture long context**. To better understand the action from the video, it is crucial to identify and classify the context of the actions. This task involves complex architectures and, therefore, higher computational power.
- **No standard benchmark**. UCF101 and Sports1M have been the most popular datasets for a long time. However, the lack of a standard dataset makes the diversity in training much lesser. DeepMind expect to solve this issue by the recent introduction of Kinetics dataset, which covers 700 human action classes [36].

Video classification and action recognition are growing quickly and particularly for the following applications:

- Automated retrieval.
- Elderly behavior monitoring: recognition of atomic actions, for example: bending, walking and falling activity analysis.
- Video summarization.
- Content-based video retrieval.

4.1 DEEP LEARNING ARCHITECTURES

In 2014, two research papers stablished the backbone for future action recognition algorithms, being the differences in the design choice around the spatial and temporal information described in the previous section. These two papers are:

- **Single stream network** [37]: this research developed the multiple ways of fusing temporal information from consecutive frames. To do so, 2D pre-trained convolutions networks were used with four different configurations:
    - **Single frame:** each frame of the video is taken as an input and the output layer combines every frame at the last stage.
    - **Late fusion:** in this case the final layer fuses two nets with shared parameters spaced 15 frames apart.
    - **Early fusion:** the combination of layers is made in the first layer, convolving over 10 frames.
    - **Slow fusion:** it is a combination of single and late fusion where the convolution is made over multiple stages.

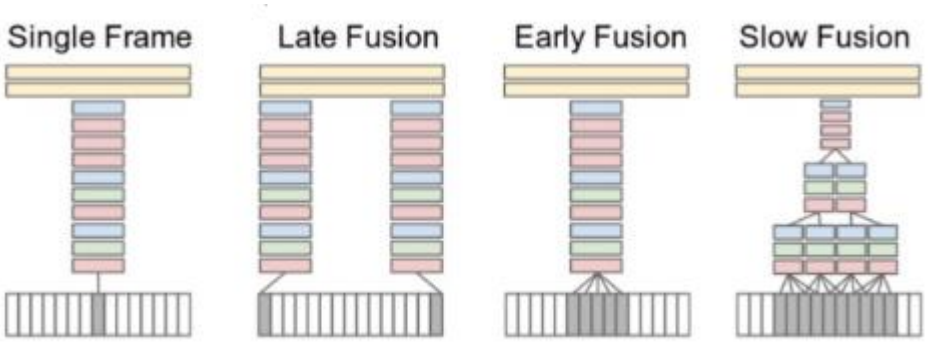


Figure 4-2: Samples of fusion in video architectures. Source: ref [34]

- **Two stream networks** [38]: to overcome the toughness of deep neural networks to learn from motion features, the authors of this paper modelled motion features in the form of stacked optical flow vectors. The main difference is that this model has two separate networks: one for motion information and the other one for spatial context. For the latter, the input is just a single frame of the video, while for the motion net the best performance is achieved when the optical flow is stacked across 10 successive frames. Then, both streams are trained separately and combined at the end using SVM (Support Vector Machine).

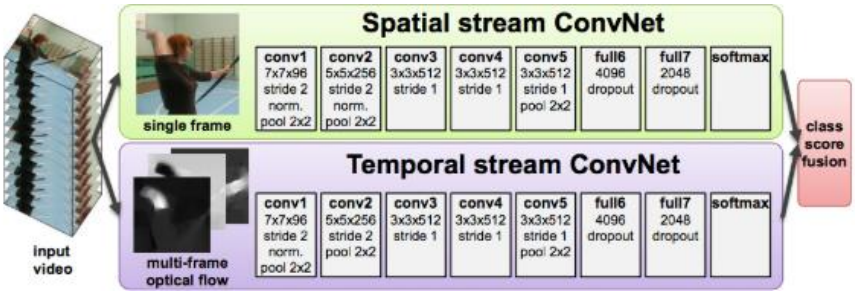


Figure 4-1: Scheme of two stream networks approach. Source: Ref. [34]

However, these two architectures still present three main drawbacks:

- Final predictions are sampled from multiple clips and averaged across the entire video in both architectures becoming inefficient when long range temporal information is needed.

- As the training clips are taken uniformly from the video, the ground truth of each clip is assumed to be the same as the ground truth of the video, which may not be the case if the action that the model is tracking happens only for a short period of time. This problem is known as *false label assignment* [39].

The following architectures are evolutions from the two previously mentioned, and some of them are still being evaluated at the present time.

4.1.1 LONG-TERM RECURRENT CONVOLUTIONAL NETWORKS

Convolution blocks are used as the encoder of the model and feed into Long Short-Term Memory (LSTM) [40] blocks as the decoder. For inputs, 16 frame clips as RGB images and optical flow (as in the two streams networks) are taken. Finally, the prediction is the average of each clip.

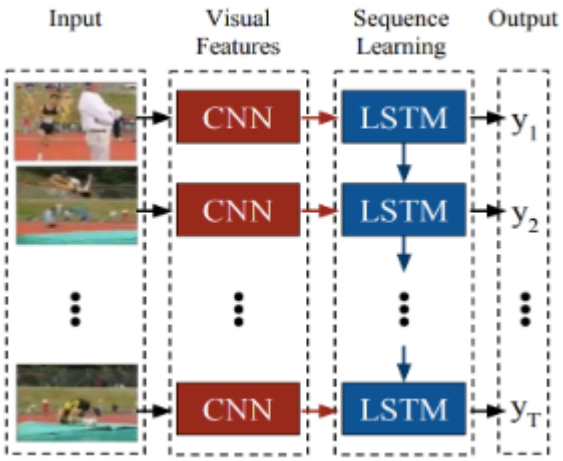


Figure 4-3: Long-short Term Recurrent and Convolutional neural network scheme. Source: ref. [37]

The architecture then is trained end-to-end. This algorithm still presents the inability to capture long range temporal information. To address that, longer clips of 60 frames and lower spatial resolution are two typical solutions.

4.1.2 3D CONVOLUTIONAL NETWORKS

Instead of using 2D convolutional neural networks across frames, 3D convolutions [41] are applied on video as it was a volume. Then, train these vast networks as feature extractors and use a simple linear classifier like SVM on top it.

2-second clips are extracted for each video for training, including the ground truth action reported in the video. Finally, the model uses deconvolutional layers to interpret the decisions.

The main drawback of the algorithm is the huge computational resources needed. To avoid that, the same authors that introduced the algorithm explored the idea of breaking the 3D convolutions into a spatial 2D convolution followed by 1D temporal convolution, obtaining similar performance with less computational requirements.

4.1.3 CONVOLUTIONAL TWO-STREAM NETWORK FUSION

This model [42] takes the two stream networks defined earlier as its basis and demonstrate a performance increment without any increase in the number of parameters. To do that, two approaches were made:

- Spatial net captures the spatial dependency in the video while the temporal network captures the presence of periodic motion for each spatial location. For this purpose, it is crucial to map both spatial and temporal features maps for the corresponding region. Therefore, networks are fused before the end layer rather than at the end as in the two-stream network. The next figure shows the architecture of an 8-layer convolutional two-stream network fusion:

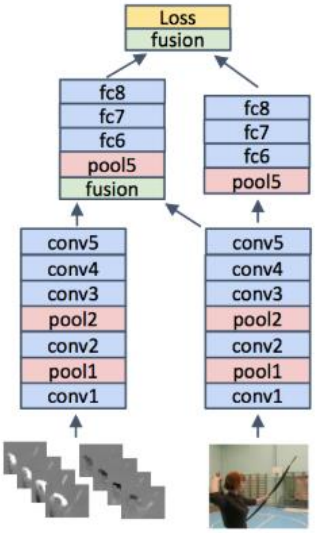


Figure 4-4: Spatian net architecture. Source: ref [34]

- To address the challenge of long-range temporal information, the algorithm combines temporal net output across time frames.

This model has the record in the UCF101 dataset with a score of 94.2% of accuracy.

4.1.4 SLOW-FAST NETWORKS

Facebook AI Researchers analysed that in every video usually exists two distinct parts: dynamics areas where something important is going on and static areas, where frames change very slowly or do not even change.

Therefore, Slow-Fast networks [43] assigns a high resolution Convolutional Neural Network to capture fast and fine motion (Fast pathway), and a low-definition CNN to analyse the spatial semantic content (Slow pathway). The compute cost of the slow pathway is 4 times higher than the cost of the Fast pathway.

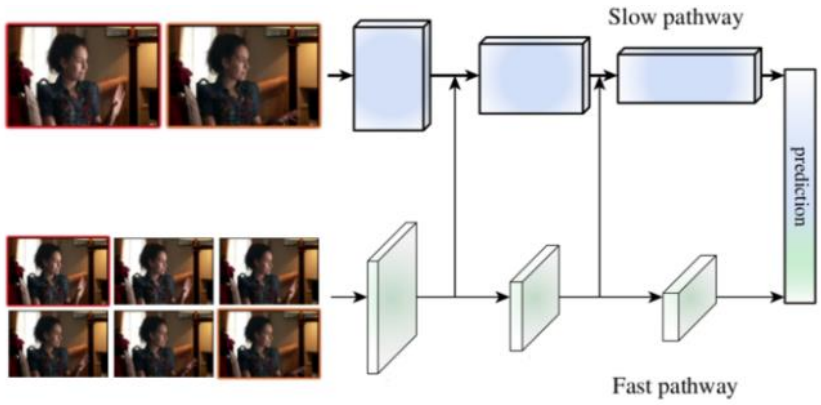


Figure 4-6: SlowFast network structure. Source: ref. [40]

Both pathways use 3D ResNet (see Appendix 4) models, but the difference resides in the stride of each path, being set at 2 sampled frames per second for the Slow path one and 15 sampled frames per second for the Fast pathway. In addition, the channel size of the paths differs also, being the channel size of the Fast path 1/8 of the Slow one. The advantage of doing that is reducing the computational requirements of the Fast path without compromising the performance. Next figure shows the dimension of kernels from a ResNet-50 implementation of the SlowFast algorithm.

stage	*Slow* pathway	*Fast* pathway	output sizes $T\times S^2$
raw clip	-	-	64×224^2
data layer	stride 16, 1^2	stride **2**, 1^2	*Slow* : 4×224^2 *Fast* : 32×224^2
conv_1	1×7^2, 64 stride 1, 2^2	5×7^2, 8 stride 1, 2^2	*Slow* : 4×112^2 *Fast* : 32×112^2
pool_1	1×3^2 max stride 1, 2^2	1×3^2 max stride 1, 2^2	*Slow* : 4×56^2 *Fast* : 32×56^2
res_2	$\begin{bmatrix}1\times1^2, 64\\1\times3^2, 64\\1\times1^2, 256\end{bmatrix}\times3$	$\begin{bmatrix}3\times1^2, 8\\1\times3^2, 8\\1\times1^2, 32\end{bmatrix}\times3$	*Slow* : 4×56^2 *Fast* : 32×56^2
res_3	$\begin{bmatrix}1\times1^2, 128\\1\times3^2, 128\\1\times1^2, 512\end{bmatrix}\times4$	$\begin{bmatrix}3\times1^2, 16\\1\times3^2, 16\\1\times1^2, 64\end{bmatrix}\times4$	*Slow* : 4×28^2 *Fast* : 32×28^2
res_4	$\begin{bmatrix}3\times1^2, 256\\1\times3^2, 256\\1\times1^2, 1024\end{bmatrix}\times6$	$\begin{bmatrix}3\times1^2, 32\\1\times3^2, 32\\1\times1^2, 128\end{bmatrix}\times6$	*Slow* : 4×14^2 *Fast* : 32×14^2
res_5	$\begin{bmatrix}3\times1^2, 512\\1\times3^2, 512\\1\times1^2, 2048\end{bmatrix}\times3$	$\begin{bmatrix}3\times1^2, 64\\1\times3^2, 64\\1\times1^2, 256\end{bmatrix}\times3$	*Slow* : 4×7^2 *Fast* : 32×7^2
global average pool, concate, fc			# classes

Figure 4-5: Table from reference [43]

As it was shown in Figure 4-6: SlowFast network structure, data from the Fast path is fed into the Slow one through lateral connections to improve the performance of the model. Then, at the end of each path, Global Average Pooling is performed to reduce dimensionality. Finally, the result is introduced into a fully connected layer with a SoftMax to classify the action.

4.2 EVALUATING ACTION RECOGNITIONS MODELS

Models described above divide the video in frames, so that the model is trained on a specific set of clips instead of a continuous video. For that reason, the performance indicators used in action recognition algorithms are very similar to image classification, described in the previous Chapter.

4.2.1 ACCURACY

As it was defined in section 3.2.4.1, accuracy divides the number of correct predictions of each class by the number of samples of that class. For multi-class classification problem, the average of the accuracy of every class is computed.

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ samples}$$

4.2.2 TOP-1 AND TOP-5 ACCURACY

The SlowFast research paper [43] introduced this indicator for the first time. For the following example, with 6 classes and their probability:

- Tiger = 0.4
- Lion = 0.3
- Cat = 0.1
- Dog = 0.09
- Bear = 0.08
- Bird = 0.03

These are the probabilities for a certain video of an animal, where a lion is shown. Top-1 accuracy is the regular accuracy defined previously, and would be 0% in this case. However, Top-5 accuracy would be 100% because the lion probability is among the top-5 guesses.

Top-5 accuracy is particularly useful for datasets with a large number of classes, as the UCF101 (which has 101 classes).

4.2.3 CONFUSION MATRIX

For multi-class classification problem, confusion matrix is also defined by precision and recall (see section 3.2.4.8), adding as many columns and rows as classes exist:

$$precision = \frac{Number\ of\ correctly\ predicted\ samples}{Total\ number\ of\ predicted\ samples\ of\ that\ class}$$

$$recall = \frac{Number\ of\ correctly\ predicted\ samples}{Total\ number\ of\ that\ class\ samples}$$

As it can be seen in the next image, the model of the example gets confused between cats and fish, predicting a cat when there is a fish in the picture:

		True/Actual		
		Cat (🐱)	Fish (🐟)	Hen (🐔)
Predicted	Cat (🐱)	4	6	3
	Fish (🐟)	1	2	0
	Hen (🐔)	1	2	6

Each number represents the number of times (4) the model predicted a cat when there was a cat actually (top-left of the table), then the number of times the model predicted a cat when it was a fish (6) or a hen (3). The same applies to the case where the model predicted a fish and a hen.

It is particularly useful for datasets when two or more classes are very similar and the model struggles to differentiate them.

4.3 APPLICATIONS

The primary goal of action recognition in videos is to interact and help people in a wide variety of situations: from monotonous activities where the system cooperates with the human by doing partially or entirely the activity, to situations where the activity is made by the human and the system enhances and complement his possibilities. In addition to the examples mentioned in section 3.5, action recognition involves more complex tasks:

1. **Intelligent surveillance systems**:

   Detecting suspicious behaviours before it happens can help to reduce the number of these crimes heavily. Computer vision motivation is to do it without human interaction. Detect the thief after it has happened or while it is happening can also help to identify the person and notify it to the police.

2. **Virtual reality**:

   Virtual Reality (VR) and Augmented Reality (AR) main task is to simulate an artificial environment where humans can interact within a computer. It allows the system to interact with the person, recognizing and identifying his/her gestures, modifying this way the environment. It is particularly useful for a wide variety of applications:

   - Maintenance and new learning approaches in industrial applications.
   - Healthcare applications, where surgeons can place surgical incisions more precisely.
   - Video-games and more realistic simulators.

3. **Retail industry**:

   Amazon GO is a *smart supermarket* where the consumer is charged for the products taken of a shelve without any physical interaction with other humans. Cameras across the supermarket caption every human action and identify when the person is taking a product.

5. HYPERPARAMETERS

Hyperparameters are the parameters or variables that are tuned by the user before training and determine the behaviour of the neural network in different scenarios. They vary depending on the model, the dataset and the particular features of the architecture. For this reason, achieving a final model implies vary these hyperparameters until the optimal configuration is reached.

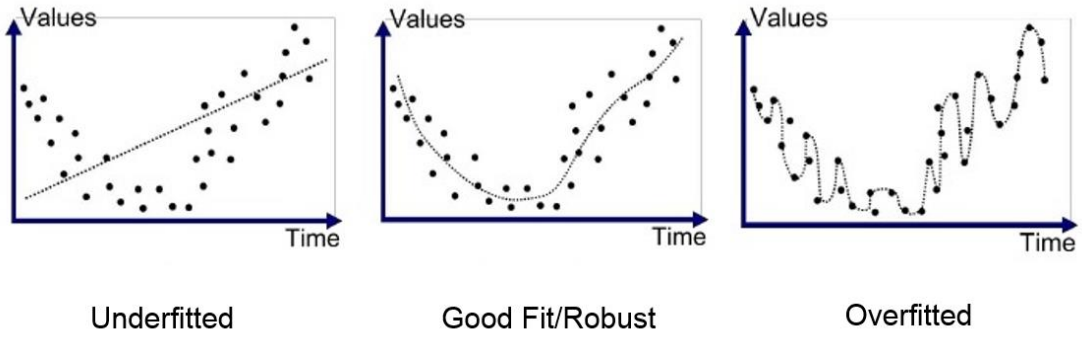
In this section, a general overview of the main hyperparameters used in the project is presented. Later, the behaviour of the model depending on these parameters is tested in Chapter 6

5.1 NUMBER OF HIDDEN LAYERS AND UNITS

Although the architecture of the model is not tuned in the majority of cases, it plays an important role if the task is easy or complex leading to underfitting or overfitting, depending on the case.

"The model's capacity is its ability to fit a wide variety of functions" [44]. Algorithms will perform better when their capacity is related with the true complexity of the task they are performing and the size of the training dataset. Overfitting may occur when the model's capacity is higher than the complexity of the task, while underfitting may occur when the task is complex and the model's capacity too low.

An unequivocal sign of overfitting is the increase of train accuracy at the same time that validation accuracy decreases. That happens when the model learns the training data but it is unable to generalize it. Next picture shows an example where: (left) the model cannot fit the data properly; (centre) the model generalizes and fits the data; (right) the model fits the data precisely but it is not able to generalize it.



Source: *https://medium.com/greyatom/what-is-underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6803a989c76*

To illustrate an example where the model is too complex for the task, with the associated risk of overfitting, two different architectures are trained and the validation loss for each one is plotted. Both models have 3 convolutional layers followed by a fully connected layer with different number of hidden layers and a SoftMax output layer at the end:

- The blue curve has 64 hidden units in the FC layer.
- The green curve has 128 hidden units.

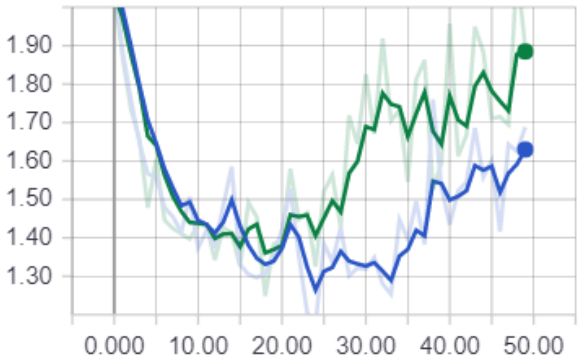


Figure 5-1: Validation loss for two different configurations.

As it can be seen, as the number of epochs increases (x-axis), the validation cost is lower for the blue line, proving that the model fits the data better with 64 units instead of 128, in this case.

5.2 *DROPOUT*

To solve the problem of overfitting in very deep neural networks, dropout technique is applied. By applying dropout, certain nodes or units are "ignored" during forward or backward pass. A model with N hidden units, it has 2^N potential models, which forces the network to learn more robust features.

During training, each node is dropped out with a probability of $1 - p$, being p the probability to be kept. Generally, a dropout value $(1 - p)$ of 20% is a good starting point. A lower value may have minimal effect, while a value too high may lead to under-learning.

5.3 *NUMBER OF EPOCHS*

The number of epochs determines how many times the weights of the network are changed. As in the number of hidden layers, the model behaviour goes from underfitting to overfitting, reaching the optimal value between both behaviours.

There is no right number of epochs for a general situation. The optimal value depends on each model and the only way to know is by training and testing. When the validation and testing loss stops decreasing and starts increasing, the model should stop the training. Next figure shows an example:

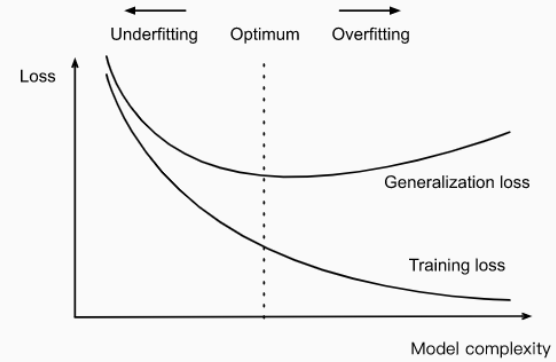


Figure 5-2: Validation loss sample.

5.4 BATCH SIZE

The batch size is the number of training images that the model computes for every iteration. For example, if there are 100 images and the batch size is 10, the model will train for 10 (100 / 10) iterations per epoch.

In terms of computational requirements, as the batch size increases the memory space of the GPU increases too. The immediate effect is the reduction of computational time for each epoch which means that so that the model trains faster.

For a batch size equal to the number of samples, the model knows exactly the best direction to minimize the error. However, it could lead to a local minimum, and the model cannot change its direction once the local minimum has been found. With a 1-sample batch, the model takes longer to reach a local minimum, but it could lead to a global minimum, as the model changes its direction with every iteration.

There is no optimal solution for this task. The batch size is usually chosen by trial and error, as it depends on the dataset, the memory available and the model behaviour. Averaging over batches of 5, 10 and 100 samples will produce better results and understanding of the model than full batch-mode size.

5.5 OPTIMIZER

The objective of training a model is to minimize the loss function defined in section 3.1.7. For every epoch, the weights and bias are updated according to the error calculated with backpropagation. The way the loss function is minimized is determined by the Gradient Descent, which is an iterative machine learning optimization algorithm defined as follows:

$$\theta = \theta - \eta \nabla j(\theta; x, y) \tag{24}$$

where θ is the weight parameter, η the learning rate that will be defined in section 5.6 and $\nabla j(\theta; x, y)$ the gradient of the weight parameter.

The optimizer defines the way these weights are updated, while the loss function determines the direction of the upgrade. Depending on the way is updated, the main optimizers are:

- **Momentum**: It accelerates the Gradient Descent when the weights are being moved in the right direction. It takes the gradient of the current step and the previous time steps as well. Therefore, it moves faster towards convergence. The Gradient Descent defined previously can be rewritten using momentum as follows:

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \nabla j(\theta; x, y) \\ \theta &= \theta - v_t \end{aligned} \tag{26}$$

  In the literature, this optimizer is often named as Stochastic Gradient Descent (SGD) with momentum.

  Nesterov accelerated gradient (NAG): it follows the same concept as momentum but the algorithm is able to 'see' the next step and adjust the speed of the gradient upgrade. For that reason, the Gradient Descent is optimized a bit faster:

$$\begin{aligned} \theta &= \theta - v_t \\ v_t &= \gamma v_{t-1} + \eta \nabla j(\theta - \gamma v_{t-1}) \\ \theta - \gamma v_{t-1} &\ is\ the\ gradient\ of\ looked\ ahead \end{aligned} \tag{27}$$

- **Root Mean Square Propagation (RMSProp)**: It utilizes a moving average of the squared gradient to normalize it and adjusts automatically the learning rate for each parameter. Then, it "*divides the learning rate by the average of the exponential decay of squared gradients"* [45]:

$$\theta_{t-1} = \theta_t \frac{\eta}{\sqrt{(1-\gamma) g^2{}_{t-1} + \gamma g_t + \varepsilon}} * g_t \tag{28}$$

- **Adam**: it is the most used optimizer for computer vision algorithms due to its little memory requirements and high computational efficiency. Adam is related with RMSProp as it uses the exponential moving average of the gradients.

First, it updates the exponential moving average of the gradient m_t and the squared gradient v_t that estimates the first and second moment. β_1 and β_2 are hyperparameters to be tuned, which control the exponential decay rates of the moving averages:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
$$v_t = \beta_1 v_{t-1} + (1-\beta_2) {g_t}^2 \quad (29)$$

m_t *and* v_t *are estimates of first and second moment respectively*

And:

$$\theta_{t+1} = \theta_t - \frac{\eta * \hat{m}_t}{\sqrt{\hat{v}_t + \varepsilon}} \quad (30)$$

Finally, the parameters are updated as follows:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \quad (31)$$

$\hat{m}_t$ *and* $\hat{v}_t$ *are bias corrected estimates of* 1*st and* 2*nd moment respectively*

5.5.1 SELECTING THE OPTIMIZER

With sparse data, where the most critical information is not as frequent as the background, RMSProp can be very useful to adjust automatically the learning rate, penalize the parameters that are frequently updated and give more learning rate to the infrequent parameters.

Adam adds bias-correction and momentum to RMSProp, and generally, it converges faster. For that reason, Adam is the newest and the most common optimizer used in the literature.

For a deeper explanation on these optimizers and their particular features see reference [46].

5.6 LEARNING RATE

The learning rate is the hyperparameter that tells the model how far the weights are moved in the opposite direction of the gradient for each batch.

If it is too high, the model may not converge to any point, or even diverge, as the optimizer overshoots the minimum and makes the model worse. However, if it is too low, the training is more reliable but it may take too many iterations to find the minimum.

The optimal way to tune the learning rate is starting with a high value and see the behaviour of the loss. If the loss does not converge, the learning rate should be decreased until the loss is being reduced consistently. Lots of papers in this field reduce the learning rate by a factor of 10 for the last 10-20 epochs, so the model is adjusted precisely.

The state-of-the-art presents strategies to tune the learning rate, by using step-wise values when the loss stops improving or cyclical learning rates [47], but the most common way to tune it is by trial and error, as it depends on each dataset and the algorithm.

5.7 INPUT SIZE

The input size refers to the dimension of the image or frames and it is directly correlated with the resolution of the picture. Low resolution helps the neural network capture global features while finer details could be lost. In contrast, higher resolution captures more details but global features may get lost.

Usually, higher input sizes increase accuracy until certain threshold. By contrast, the number of parameters to calculate increase proportionally too and some networks may struggle to convergence or take too much time. Thus, the input size depends on the dataset and the algorithm, so the optimal dimension is calculated by trial and error.

For example, in the 3D	64	128	256
Number of parameters (M)	11.1	17.5	34.8
Train time (mins/epoch)	92	270	1186
Accuracy	0.41	0.43	0.44

Table 1: Example of different sizes performance.

The best accuracy is achieved for the highest resolution (256). However, the computational cost and train time is too high compared to the 128-resolution which has almost the same accuracy with less effort.

5.8 TRAIN, VALIDATION AND TEST SPLIT

When training a model, the dataset is divided into three parts: train, validation and test. The train set is the actual data that the model uses to learn; the validation set measures the accuracy of the model while training to calculate the performance with unseen data; finally, the test split is the data used to measure the accuracy and performance indicators after training, that represents the "real" behaviour of the model with unseen data.

Depending on the size of the dataset, the split into train and test can vary. Typically, the 80% of the data is for training and the remaining 20% for testing. Then, the 80% of training is divided likewise into training (80%) and validation (20%). However, when there are few hyperparameters to tune, the validation set can be reduced, as well as when the number of samples in the dataset is insufficient. In contrast, when there are a multiple hyperparameters, it is recommendable to have a larger validation set. This split prevents the model of overfitting, as the user has a live performance indicator while training.

K-fold Cross Validation is very popular for training with small datasets [48]. It splits the training data into k folds, and fits the model using k minus 1 batches, validating with the remaining k fold. Then, the process is repeated until every k fold is used as the validation set.

6. RESULTS

Three algorithms have been chosen from the ones listed above:

- Faster-RCNN for object detection.
- U-Net for semantic segmentation.
- Slow-Faster for video action recognition.

Faster-RCNN have been compared with a YOLO implementation. The objective is to compare both models, that have been trained from scratch, with a consolidated and widely used model as YOLO.

As it was said in the introduction of this project, training these models require huge computational resources. The emergence of the Graphics Processing Units (GPUs) has allowed the Research Community to train, test and develop complex algorithms. However, the accessibility to these resources is usually a limitation. Cloud services are a good solution to this problem, and a brief comparison of them have been included in this chapter.

The performance of the model depends on several factors being one of the most determinant the dataset used. For that reason, a comparison of the state-of-the-art datasets for images and videos has been also included in this chapter.

6.1 TOOLS

NVIDIA is the main GPU developer, so the majority of algorithms have been trained using a NVIDIA GPU. *TensorFlow* and *Pytorch*, the two most common libraries for developing Machine Learning projects in Python, integrate the usage of these GPUs in a very easy way, therefore, the selection of NVIDIA is almost mandatory.

Whether the user has his/her own GPU or not, Cloud frameworks are always a good solution to speed the training and work in more than one model at the same time. Prior to listing all

the services, a brief summary of the main GPUs is detailed below. For further information, and a more extended list, see reference [49]:

GPU	GPU RAM (GB)
P100	16
V100	16
K80	12
Quadro M4000	8
GTX 1070	8
GTX 2060	6

Table 2: List of GPUs considered in this project.

The GPU RAM, or Virtual RAM (VRAM), is used to process graphical applications or training computer vision models like in this case. The higher the VRAM, the larger the memory available for training, which allows larger batch sizes and bigger dimension of the pictures, among others.

Lately, the irruption of laptops has set aside the integration of GPUs in the computer. However, NVIDIA allows the connection of these GPUs externally, which is a very attractive option for training computer vision algorithms with laptops.

Having a GPU is sometimes expensive, depending on the usage and the type of GPU. For that reason, Cloud services are very useful, as the user only pays the time the model is running on the cloud. The mainly Cloud providers, listed from the free options to the most expensive provider, are:

- **Google Collaboratory**:
  Integrated with GitHub and Google Drive, Google Collab uses *Jupyter Notebook* as the code editor. It is very easy and fast to start with famous libraries as Keras and TensorFlow, both developed by Google and integrated in Google Collab with no need of installation. With 12 hours a day of free GPUs usage, and an idle time of 1h, Google Collab is one of the most used Cloud providers nowadays.

With the free package, only the K80 is offered, while the PRO version includes the Tesla P100 and V100. However, the Google Collab PRO is only available in the United States.

- **Kaggle**:

  Very similar to Google Collab, it offers the K80 for free with a 9h limit of usage per day. The main advantage of Kaggle is the possibility of sharing your work with the Kaggle community, that has plenty of tutorials, contests and notebooks available for free.

- **Paperspace**:

  It also uses *Jupyter Notebooks*, and the free version includes 12h a day of usage with the Quadro M4000. Available worldwide, the PRO version includes all the Google GPUs with student discount also offered.

- **Vast.ai**:

  It is a GPU sharing option. Vast matches people who offer their GPU with people that need that GPUs to work. Therefore, the pricing is its best attractive. The list of GPUs available is huge at very economic prices. The main drawback is that your work depends on the host Internet, so if the work is sensitive or very important, it is not recommended.

- **Google Cloud Platform (GCP)**:

  GCP is more powerful and customizable than Google Collab being its main use is for enterprises. It requires configuration and more time to deploy the models. Its pricing, server options and security make GCP a frontrunner for enterprises purposes.

- **IBM Watson Studio**:

  IBM only offers the GPU with a paid version, which is more expensive compared with the previous options listed above. Moreover, the complexity to run a notebook make the IBM one of the least desirable options.

- **Amazon Web Services (AWS)**:

  AWS is the first cloud provider for machine learning is the one with the best GPUs and resources available, thus, the most expensive. AWS is secure, scalable and

reliable but has a lot of options to configure. For enterprises, it can be the best option when the performance is the limitation.

6.1.1 EXPERIMENT

The code in reference [50] has been used to test the Cloud services mentioned above. Results may vary depending on the application, but the graph shown hereafter summarizes the behavior under normal circumstances:

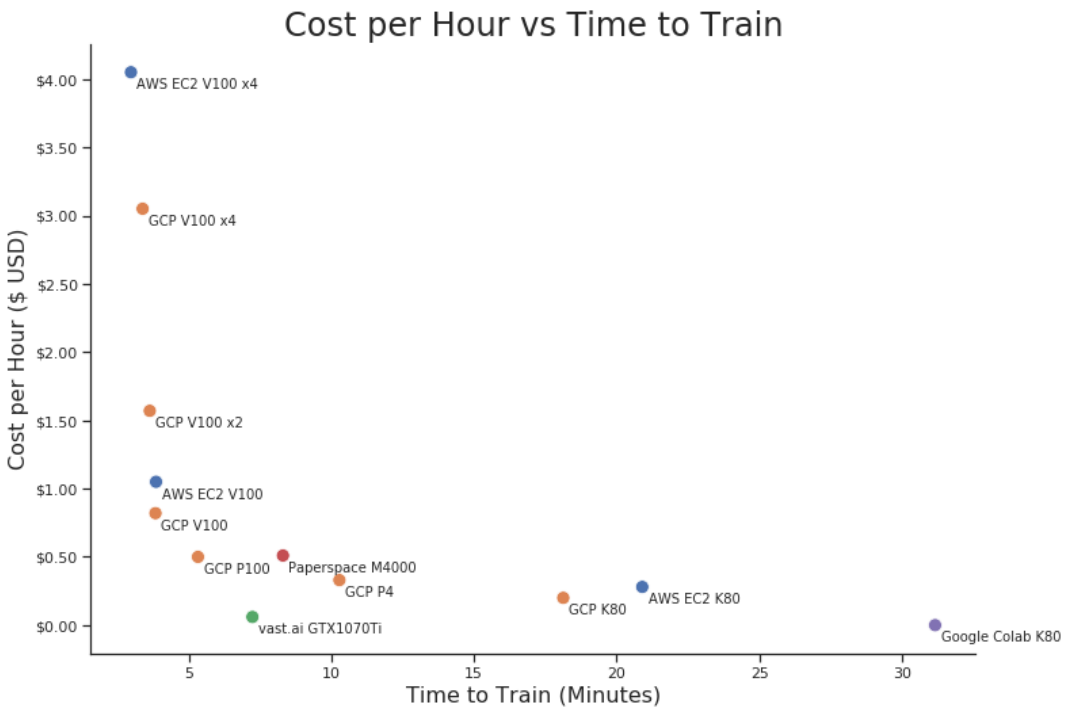


Figure 6-1: Comparison of the different Cloud options. Source: ref. [47]

The balance between cost and performance is achieved in the bottom-left corner of the graph. The *Vast-ai* GTX1070 in this example had a $0.01 cost although it may vary with the demand. Moreover, the *Paperspace* M4000 option was using the paid version to have more memory available.

It can be seen from the graph above that *AWS* is the more powerful resource, while *Google Collab* is the best option if time is not a limitation. Between both edges, *Paperspace* and *GCP* offers competitive performance with affordable prices.

6.2 DATASETS

The dataset is the collection of images or videos used to train the model. It contains images labelled with a class and with extra information depending on the task: in the case of object classification, a .csv file is provided with the dimension of the boxes; while in the case of semantic segmentation includes the pixels belonging to certain class.

The selection of the dataset is crucial depending on the application. For example, to train a traffic light detector, it is very important that the dataset includes a large number of pictures with traffic lights where the object is in different localizations, so the model is able to generalize the object in every picture.

6.2.1 IMAGE DATASETS

As it can be seen in reference [51], there are a huge number of datasets available for image classification. The main ones are:

- **MNIST** (*The Modified National Institute of Standards and Technology*): it was one of the first image datasets and is considered as the state-of-the-art dataset for the majority of the algorithms. It contains hand-written digits.
- **ImageNet** Large Scale Visual Recognition Competition (ILSVRC): Divided according WordNet [52], it contains more than 100,000 images for each class.
- **PASCAL VOC** (*The Pattern Analysis, Statistical Modelling and Computational Learning Visual Object Classes*): It contains only 20 classes for object detection and segmentation, with almost 12,000 images.
- **MS COCO** (*Microsoft Common Objects in Context Dataset)*: only includes object segmentation with more than 330K images. Every year, it hosts a semantic

segmentation competition where new algorithms are submitted and tested. It represents the best dataset for semantic segmentation.

- **Google Open Images**: with almost 9 million labelled images and 600 classes, both bounding box and semantic segmentation are presented in this dataset. It is the ultimate dataset from the listed above and climbing places as the preferred one for bounding box.

6.2.1.1 Object detection dataset

As it was said previously, it is very important that the dataset contains objects in different positions, localizations and with different conditions. Having more than one object in the picture is also important to generalize the model to particular conditions where the input image is not centred around the object. For that reason, Google Open Images dataset has been chosen to train the object detection models.

With an 8.3 rate of objects per image on average, Google Open Images is the dataset with the higher number of objects per image. Next graph shows a comparison between the datasets exposed previously (except MNIST):

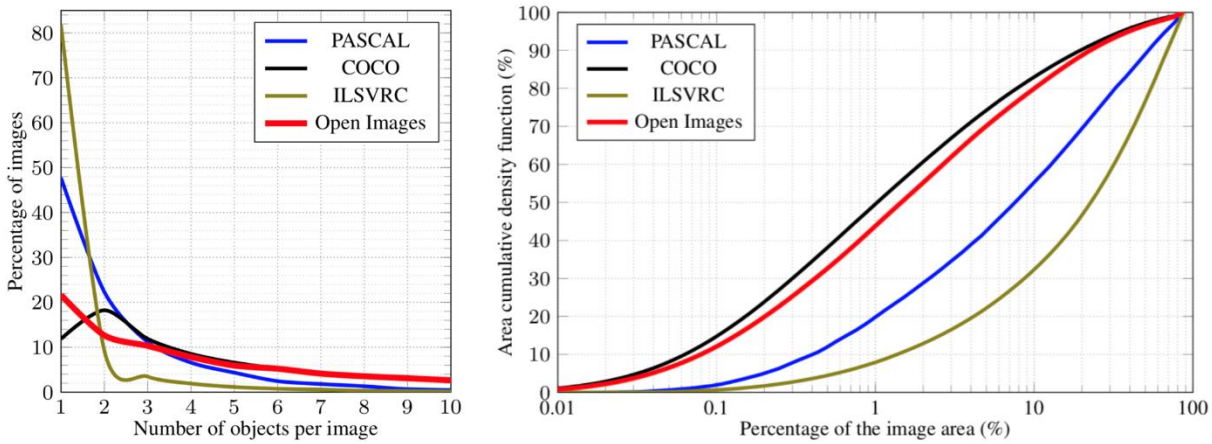


Figure 6-2: Dataset comparison. Source: https://storage.googleapis.com/openimages/web/factsfigures.html

ILSVRC has the lowest number of objects per image. That can be useful for image classification, but in object detection tasks, Open Images offers the most homogeneous distribution of objects per images.

The graph on the right shows that COCO and Open Images offers the best background and foreground balance which is very useful when training the model, especially when the number of classes is high.

6.2.1.2 Semantic segmentation dataset

For the reasons exposed above, Google Open Images was also chosen first to train the semantic segmentation models. However, during the exploration of the dataset, it was seen that some images were not labelled correctly, which can lead to bad performance of the model. The next picture shows an example where it can be seen that not every person in the picture was labelled correctly:Note that the yellow boxes are not part of the semantic segmentation task, and only the woman has been labelled correctly.

Figure 6-3: Google Open Images sample for semantic segmentation task

After that, COCO dataset was chosen as the semantic segmentation dataset, as the second-best option for the reasons exposed previously. An example of the COCO dataset which image is labelled correctly for the class ‘person’ is:

Figure 6-4: COCO sample for semantic segmentation task

6.2.2 VIDEO DATASETS

Reference [53] and [54] summarize the available datasets for action recognition. These datasets vary from the number of classes and samples they contain to the average length of their clips and the resolution. The main datasets used are:

- **HMDB51**: with 6,766 videos of 51 classes, it was the first action recognition dataset of the state-of-the-art.
- **UCF101**: it has 101 action classes and 13,320 videos. The algorithms presented in this project are usually validated with this dataset, to ensure that performance numbers are comparable.
- **Sports-1M**: including 487 sport classes and 1.1M videos, it was the first dataset distributed using YouTube links, with the problems derivate from that: missing videos and copyright violations among other reasons.
- **ActivityNet**: it contains at least 100 videos per category with 203 categories and a total of 28000 videos.
- **Kinetics**: with 600 action classes and over 500,000 videos from YouTube, it is the biggest and the most recent dataset for action recognition.
- **AVA**: the main innovation of AVA dataset is the addition of bounding boxes information to the videos. It contains 80 classes and 57,600 labelled videos from YouTube.
- **YouTube-8M**: over 7 million videos and 4,716 classes form this dataset, although it is not restricted to action recognition only.

6.2.2.1 Action recognition dataset

From the list above, datasets which are extracted from YouTube have been removed to avoid missing videos, copyright licenses and memory limitations, as these datasets are usually too large (1-2 TB). Avoiding very large datasets reduce the training time to hours or even days, compared to days-weeks intervals that take to train a 1TB dataset.

Therefore, datasets considered are: HMDB51, UCF101 and ActivityNet. Next graphs show the number of frames and videos of these three datasets:

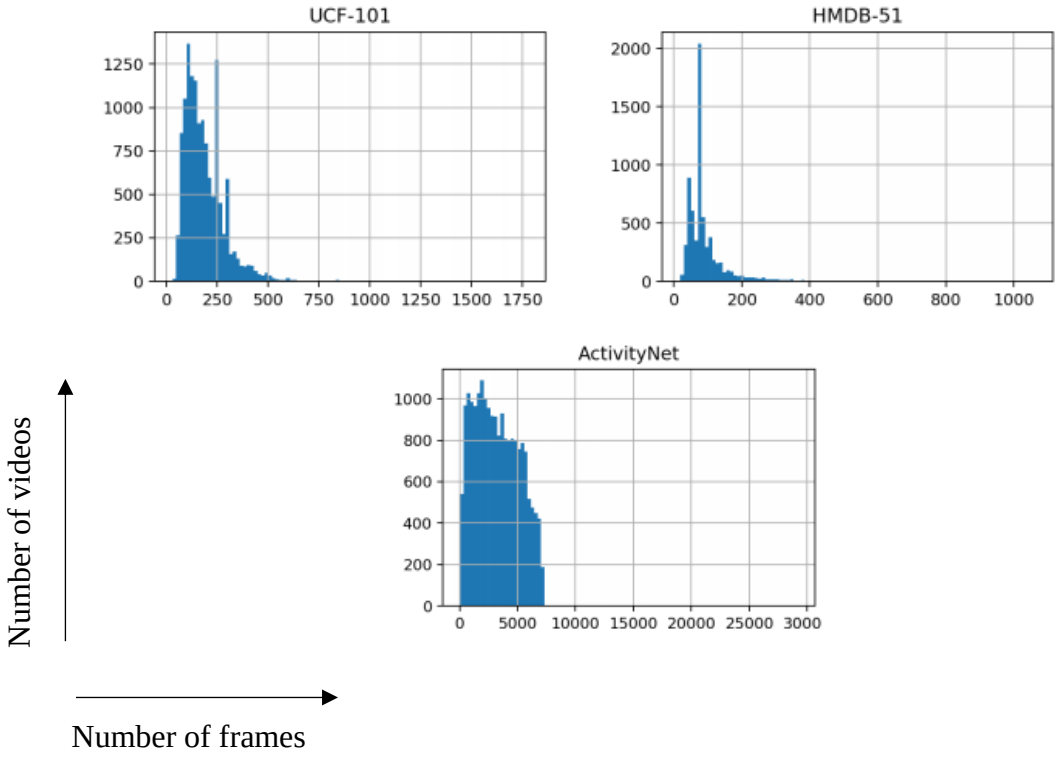


Figure 6-5: Dataset comparison for action recognition.

Note that the scale of the axes is different for each graph. It can be seen that the ActivityNet has the highest number of videos but the largest variance too. In contrast, the number of videos in HMBD-51 is not large enough and may lead to insufficient training samples. UCF-101 has a large enough number of videos with an affordable number of frames for the resources available in this project. For that reason, UCF-101 is the dataset chosen to train the video algorithms of this project.

6.3 IMAGES ALGORITHMS

As it was said previously, bounding box algorithm is trained with *Open Images* dataset, while U-Net is trained with COCO. For that reason, the selected classes must be in both datasets to be compared later. These classes are: people, traffic lights, traffic signs, buses and cars. The reason behind the selection of these classes is to test the model with objects of different sizes and having enough pictures where the maximum number of labels are present at the same time. Thus, the model will struggle sometimes to differentiate them, but it will be more robust. In the other hand, if classes were very easy to differentiate, the train accuracy would be high but the model would fail in complex situations.

6.3.1 FASTER-RCNN

First, the model is trained for classes 'person', 'traffic light' and 'traffic sign'. Then, by transfer learning, classes 'car' and 'bus' are added. The purpose of doing that is simulate a real application, when training a model from scratch is often expensive and time consuming, while using Transfer Learning and adding a class to an already trained model by inference is much more efficient.

Although scripts are going to be attached in the appendixes, the following figure shows a general overview of the structure of the code:

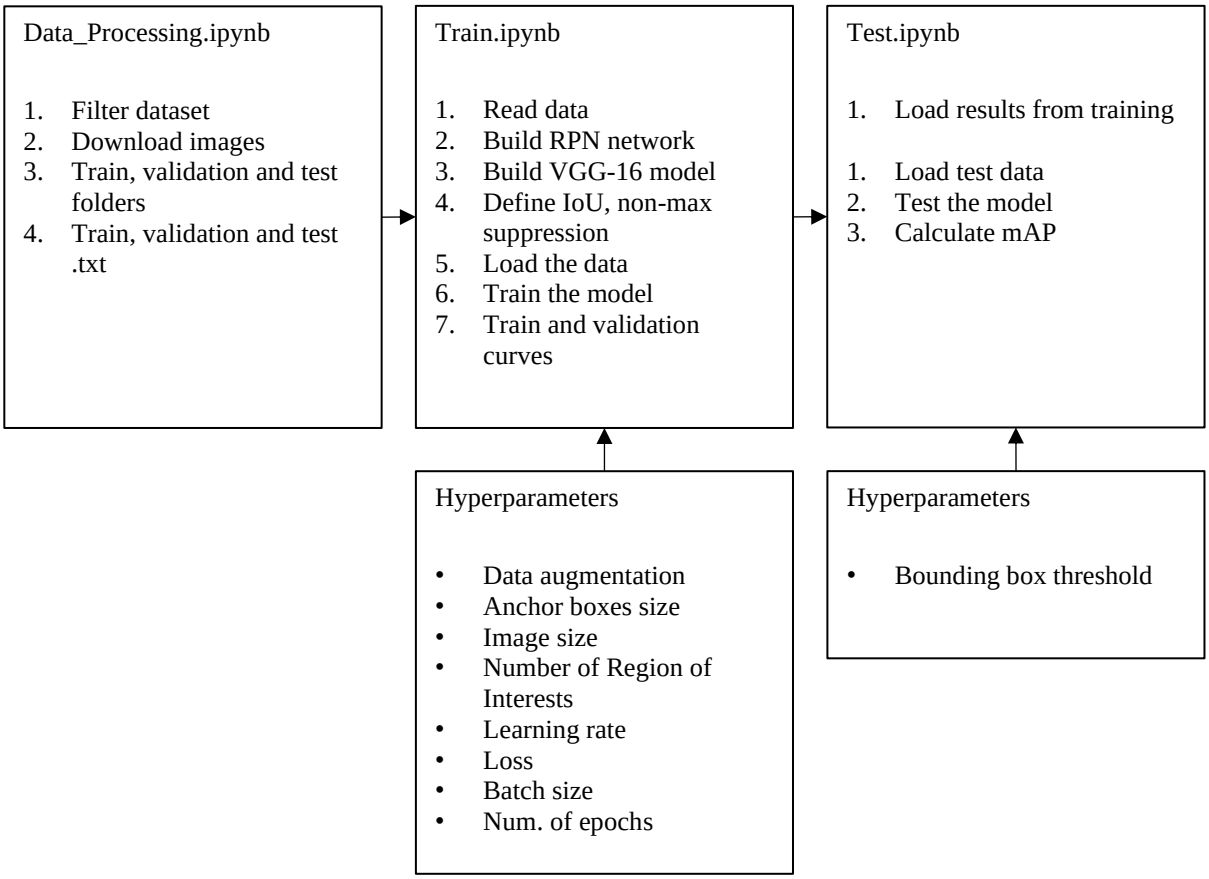


Figure 6-6: Structure followed in the script.

6.3.1.1 Data Preparation

To avoid downloading the entire dataset, which contains around 9M images, the dataset is filtered according to the selected classes. 3 files are used for this purpose:

- *train-images-boxable-with-rotation.csv* contains the image id and url to download.
- *oidv6-train-annotations-bbox.csv* contains the bounding box info with the image id and label name.
- *class-descriptions-boxable.csv* contains the image label name corresponding to its class name.

Then, parameter n defines the number of samples per class to download. If the dataset has less samples than n , the script will download all images of that class.

The script will download n images for each class and saving them in two folders, train and test, according to an 80%-20% proportion. During this process, a dataframe with the image's ids, bounding boxes and images' classes is generated. Finally, a txt file, one for train and one for test, is generated from the dataframe. The next example shows a row of this file:

E:/ Object_Detection/Faster-RCNN/train/0000432bf1241812.jpg,192,2617,296,2729,Person

- *"E:/ Object_Detection/Faster-RCNN/train/0000432bf1241812.jpg"* is the path to the image, where the number starting from 0 represents the image id.
- *"192"* is the bottom-left x coordinate of the bounding box of the object.
- *"2617"* is the upper-right x coordinate of the bounding box of the object.
- *"296"* is the bottom-left y coordinate of the bounding box of the object.
- *"2729"* is the upper-right y coordinate of the bounding box of the object.
- *"Person"* is the class name of the image.

Note that bounding boxes are always rectangles. For that reason, only the two squares mentioned are required to be defined.

6.3.1.2 Training

Before training, the model has to load every bounding box from the txt file and assign it to its corresponding picture. This process takes between 30 to 60 minutes, depending on the GPU, and it is something to considerate when increasing the training set, as the loading time increases exponentially. For that reason, a sample of n=1000 images per class was taken as the base condition.

The first experiment was done using a NVIDIA GTX 2060. The base conditions were limited to the memory of the GPU (6 GB). Later, when the size of the batch and the images were increased, Google Colab GPU was used, as the GTX 2060 memory limit was reached.

The mentioned base conditions are:

- n = 1000 for each class, resulting in: 1904 training images, 479 validation images and 388 test images.
- Region of Interest = 2
- Image size = 150 → Anchor box sizes = [32,64,128]
- Batch size = 1
- Learning rate = 1x10-4
- Augmentation applied: flip over the horizontal and vertical axes, and a rotation of 90 degrees clockwise.

To prevent overfitting, the validation accuracy and the MSE error are obtained. When the validation accuracy starts decreasing while the training accuracy keep improving, the model stops training. Moreover, weights are only saved when the validation accuracy improves.

1. **Learning rate comparison**:

The selection of a learning rate value of $1x10^{-4}$ was due to the paper obtained the optimal results with that value. As this value depends on the samples, the application and particular conditions, several learning rates are tested to increase the performance of the model.

First, in the case of learning rate = $1x10^{-4}$ the training accuracy was too oscillating, as it can be seen in the next graph. Although it increased over the 60 epochs, the behaviour of the accuracy was a clear indicator that the learning rate should be increased.

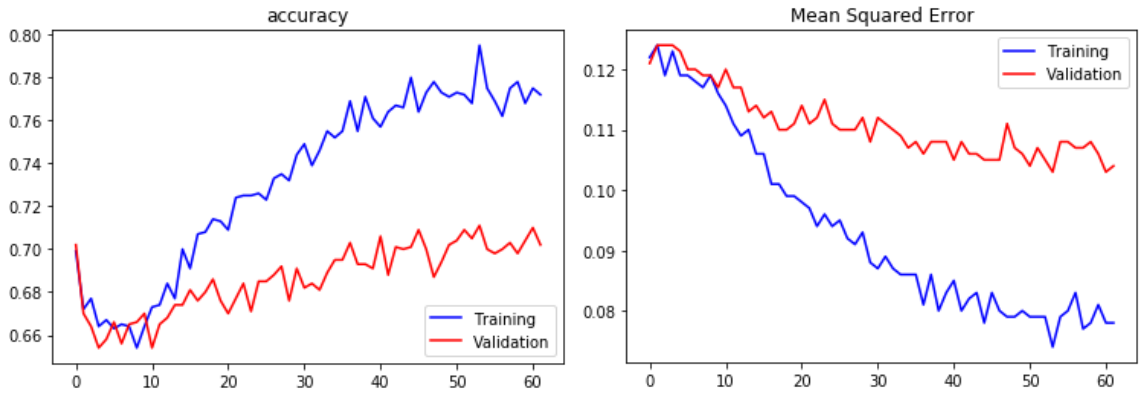


Figure 6-7: Training and validation performance for 1x10$^{-4.}$

Note that the training was stopped at epoch 60 due to the validation accuracy did not improved for the last epochs, and overfitting started to occur.

For that reason, 1x10-5 was tested and results were satisfactory. Both training and validation accuracy increased, as well as the mAP. The maximum value of the validation accuracy was obtained in epoch 59. After that, its decline shows that training has finished.

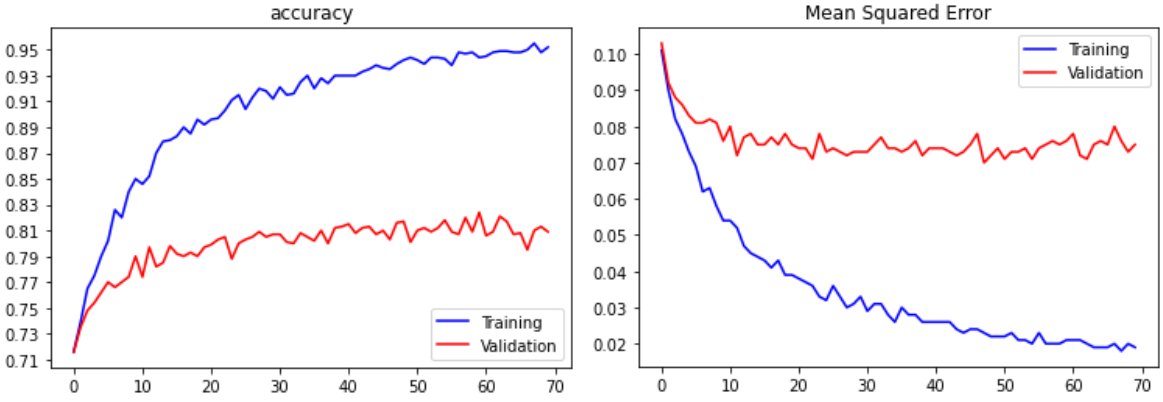


Figure 6-8: Training and validation performance for 1x10^{-5}.

Note that due to the axis interval in the y-axis, it seems that the slope of the curve is flat compared to the previous case, although it is not the case. In the previous example, the accuracy increased 0,05% (0.66 to 0.71) over the 60 epochs, while in this case it increases a 1.1% over the 70 epochs.

Finally, the 1x10-6 value was also tested. Although the accuracy never stopped improving, the pace was lower than in the 1x10-5 case. The training stopped at epoch 70 with a validation accuracy of 0.78.

Learning rate value	Training accuracy	Training MSE	Validation accuracy	Validation MSE	mAP
$1x10^{-4}$	0.77	0.08	0.71	0.11	0.18
$1x10^{-5}$	0.95	0.02	0.82	0.08	0.35
$1x10^{-6}$	0.88	0.04	0.78	0.09	0.25

Table 3: Learning rate comparison.

Therefore, the optimal learning rate is 1x10-5, and it is considered as the starting point for the next simulation.

2. **Batch size comparison**:

Every training took around 14 hours. Therefore, reducing this time is crucial to simulate several conditions. To do that, increasing the batch size is the most common practice.

As it was explained in section 5.4, increasing the batch size reduces the training time per epoch, but it may get stuck in a local minimum, while a batch size of 1 is very slow but it leads to global minimum.

For this simulation, Google Colab was used instead of the GTX 2060 GPU, as it reached the memory limit as soon as the batch size was higher than one. A simulation with the same conditions (batch 1) was made to be able to compare the training times for different batch sizes under the same conditions.

The Google Colab limit is 12h a day. For that reason, although the model needs more time to be trained, the performance is evaluated after the 12h of training. The reason behind that

is that every time the session starts, loading the data takes around 1h, so training for 14 hours would imply more than one day of Google Colab usage and it would be inefficient.

Batch size	Train acc.	Train MSE	Valid. Acc.	Valid. MSE	mAP	Epochs	GPU (seconds per epoch)
1	0.84	0.06	0.79	0.07	0.27	40	P100 (1040s); T4 (1150s)
5	0.80	0.07	0.79	0.08	0.19	126	K80 (347s); P100 (200s)
10	0.81	0.06	0.81	0.08	0.18	360	P100 (120s); T4 (140s)

Table 4: Batch size comparison.

Although the best validation accuracy is obtained with a batch value of 10, the mAP shows that the best model by far is obtained with a batch size of 1.

As it can be seen in the next figures, for a batch value of 5, the model converges around a validation accuracy of 0.75, in contrast to the case of batch 1, where the model converges around 0.79. This can happen when the model thinks that a minimum has been found, unable to know if it is local or maximum. In the case of a batch value of 10, the maximum value for the validation accuracy is achieved, although the oscillations suggest a mAP value differing the validation accuracy, which is the case. The MSE error graphs are shown to corroborate the accuracy curves and notice when overfitting is happening, which is not the case for the graphs below.

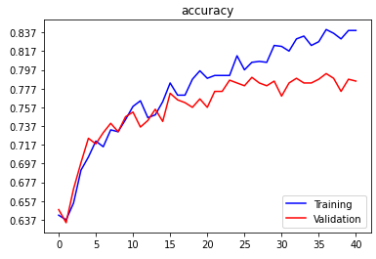
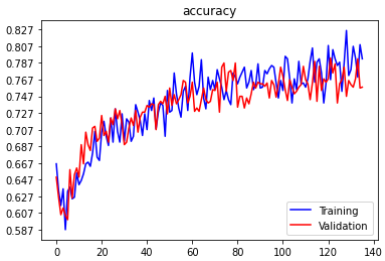
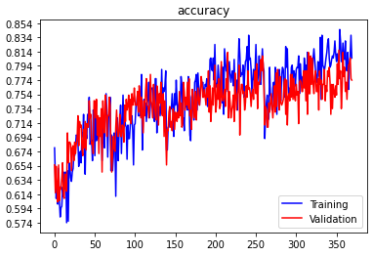
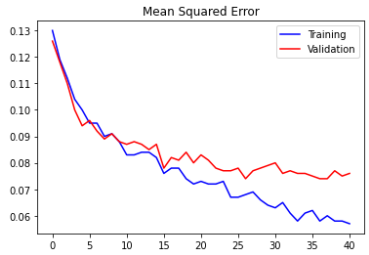
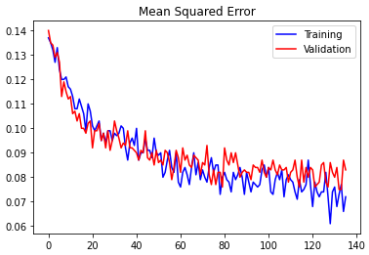
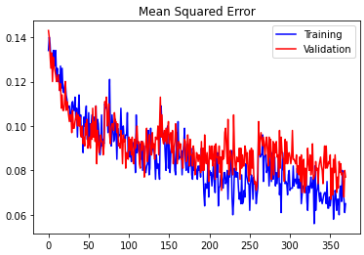
Batch = 1	Batch = 5	Batch = 10
		
		

Table 5: Training and validation curves for the batch size comparison.

3. **Image size**:

The Region Proposal Network divides the image in a 3x3 grid and place boxes in every region. After applying non-max suppression, the remained boxes are divided in positive samples (the ones with objects inside) and negative samples (background). Then, these samples are compared with the ground-truth boxes, the ones with annotations from the dataset. The positive samples that do not match with the ground-truth boxes are taken as hard negative samples.

From the previous sections, it was seen that although the validation accuracy was high, the mAP was still very low. This can happen when bounding boxes are well classified but with bad precision compared to the ground-truth boxes. The effect of increasing the image size can be seen in next table, where the number of positive samples matching the ground-truth boxes increases with the image size.

Image size	Mean overlapping boxes	Training time per epoch	GPU
150	35.56%	1150s	T4
300	68.89%	2685s	K80
600	77.78%	3500s	T4

Table 6: image size comparison.

The main drawback is also the increase in the training time per epoch. With an image size of 600, each epoch takes almost 1h, which is unsustainable considering the resources.

The mean overlapping indicator showed is for only 7 epochs. For that reason, the mAP when testing was very low, if none. However, the increase of overlapping boxes shows that the value of mAP would be very high for the same number of epochs than in previous sections.

6.3.1.3 Transfer Learning

Transfer learning is the concept of sharing and applying the existing knowledge already acquired to a new task or algorithm. Andrew NG, a very famous professor data scientist, in the lecture [55] explains the benefits of transfer learning and how it is going to change the deep learning approaches.

Train a complex model from scratch can be very complex and sometimes unaffordable, due to the limitation of the number of samples, the resources or the time it takes to obtain decent results. For that reason, taking an already trained model and applying to the task at issue can save considerable time.

Although there are numerous strategies to apply transfer learning, below are explained the two main approaches:

- "*Off-the-self Pre-trained Model as Feature Extractors*" [56] :

Almost every algorithm has a fully connected layer at the end of the model to classify every output. Removing this layer and re-train it as a fixed feature extractor for a different application is a common practice that allows using the already trained layers before the final fully connected network.

- *"Fine Tuning Off-the-shelf Pre-trained Models"* [57]:
  In addition to the last layer of the convolutional network, this technique also replaces certain previous layers. As it is known, layers learn finer details as the network gets deeper, from general features to particular details. For that reason, freezing some layers and updating the rest gets good results when the already trained model shares some features with the new one.

In this case, classes 'car' and 'bus' are added to the best model obtained in the previous section, applying the first technique mentioned above, where the last fully connected layer is re-trained. By doing that, reasonable results should be obtained with less time training.

The model is trained for 500 images of each class, instead of 1000 as in the previous section to reduce the training time. The base conditions are:

- 1989 training images, 356 validation images and 299 test images.
- Region of Interest = 2
- Image size = 150 → Anchor box sizes = [32,64,128]
- Batch size = 1
- Adam learning rate = 1x10-5
- Augmentation applied: flip over the horizontal and vertical axes, and a rotation of 90 degrees clockwise.

- Weights taken from the batch-1-model from the batch size comparison.

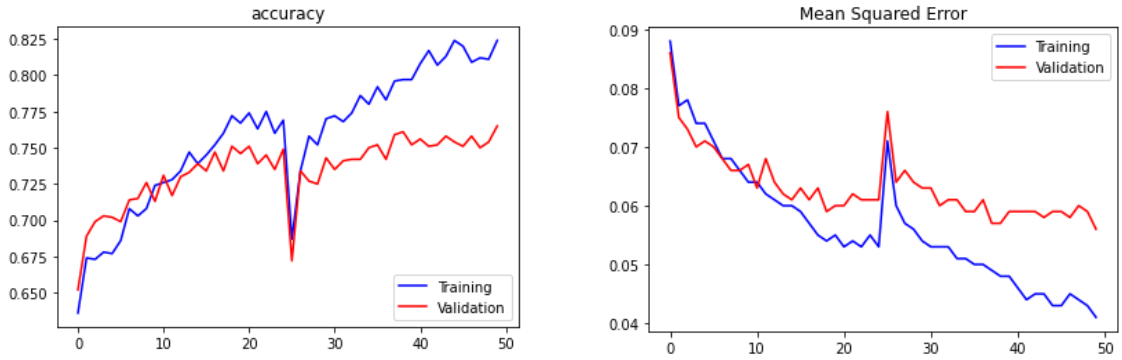


Figure 6-9: training and validation curves for the transfer learning model.

The model has been trained 8h for 50 epochs. The spikes on both graphs are due to a re-allocation of the virtual machine of Google Colab. Although the performance curves show that the model may improve for longer training times, the mAP is higher than the model taken for inference, improving the value from 0.35 to 0.37 with a smaller number of epochs:

- mAP = 0.37
- Person AP: 0.29
- Traffic light AP: 0.17
- Car AP: 0.37
- Bus AP: 0.58
- Traffic sign AP: 0.42

Actually, the best mAP value is for the 'bus' class, which was trained completely by inference. This is also influenced by two aspects: the pictures that contain buses are usually centred around the object, and the shape of the buses is very fixed, in contrast to class 'person', for example, which is more difficult to generalize for the model.

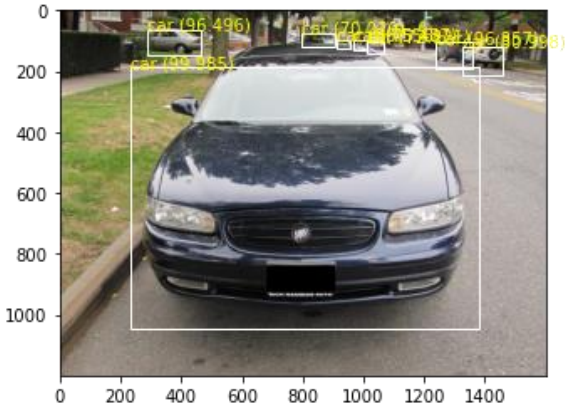
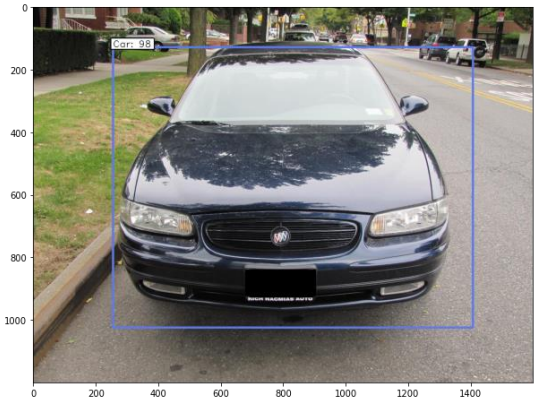
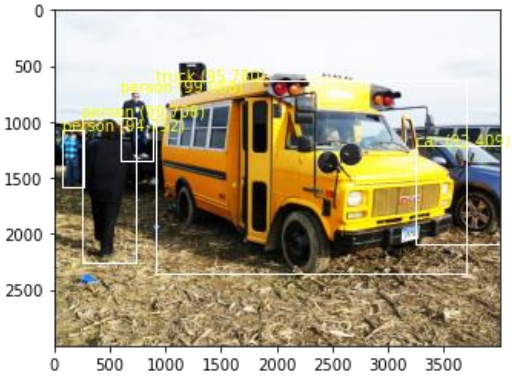
At the end of this project, Appendix 5 plots the architecture of the model and Appendix 6 show some samples of the model's output.

6.3.2 FASTER-RCNN COMPARISON WITH YOLO

YOLO is a widely known object detection algorithm, as it was mentioned in section 3.2.2. Due to the limitations to train more models, a YOLO implementation has been taken to validate the performance of Faster-RCNN. The script includes all the pre-process of data, and the bounding box treatments, except that the model itself has been downloaded from [58]. This model has been trained using a Pascal Titan X, and obtained a mAP of 57,9% in the COCO dataset.

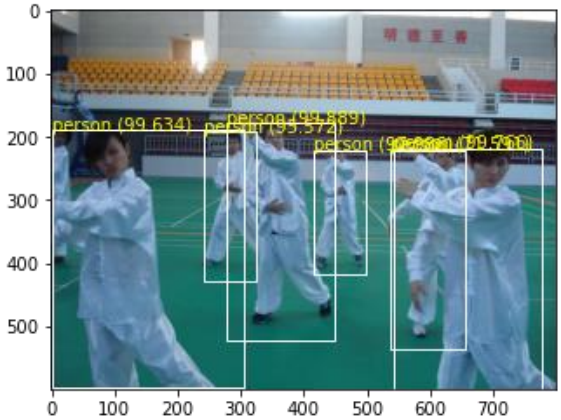
The biggest difference between both models resides when multiple objects are present at the same time. YOLO is able to find very small objects, even when they are inside of a bounding box from a different class, while Faster-RCNN don´t.

To solve that, Region Proposal Network should be improved. Test different regions of interest (RoIs), and include hard negative mining to broaden the regressor classifier are two affordable strategies that may increase the performance of the model.

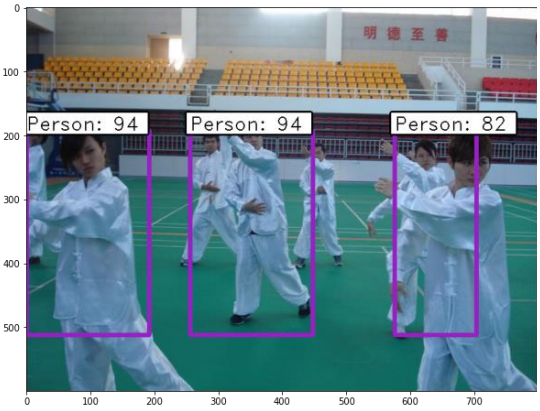
YOLO	Faster-RCNN
	
Car: 96.5; Car 70.0; Car 86.95; Car 99.9	Car: 98.9
	
Stop Sign: 99.9	Traffic sign: 71.5; Traffic sign: 98.6
	
Person: 94.7; Person: 90.7; Car: 92.4; Bus 99.8	Bus: 98.8; Person: 85.6

Bus: 97.9; Person: 88.2; Person: 97.7

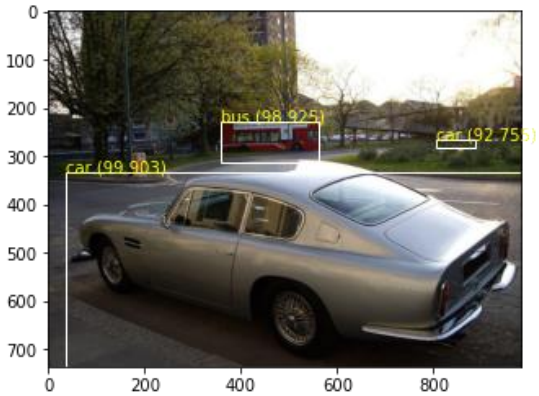
Bus: 98.6



Person: 99.8; Person: 99.6; Person: 99.7;
Person: 95.5; Person: 97.9; Person: 71.5



Person: 94.5; Person: 94.4; Person: 82.4

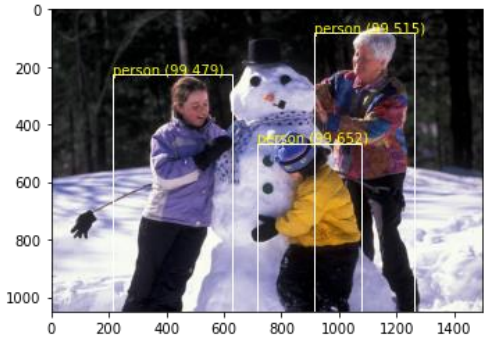


Bus: 98.9; Car: 92.7; Car: 99.9

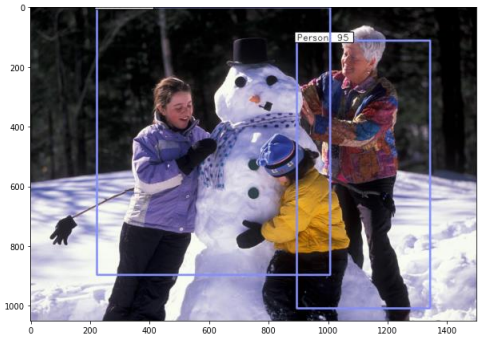
Car: 97.5

Traffic light: 98.7; Traffic light: 97.4

Traffic sign: 99.8; Traffic light: 92.6; Traffic light: 88.7



Person: 99.5; Person: 99.5; Person: 99.6



Person: 99.6; Person: 95.5

N/A

Car: 96.5

6.3.3 U-NET

U-Net is a much simpler model than Faster-RCNN. Not only the number of parameters decreases from 130 million of parameters to 36 million, also the number of processes and the requirements are lower, allowing the model to be trained in just one hour.

As it was said in section 6.2.1.2, U-Net has been trained using COCO dataset. Note that in this dataset, the previous "traffic sign" class now is "stop sign", excluding every traffic sign that is not a "stop".

To reduce the memory usage, instead of training a 5-class model with thousands of images, 5 different models have been trained, one for each class, and joined together to make the final prediction on the 5 classes. For that reason, transfer learning here is not needed, including classes "car" and "bus" since the beginning.

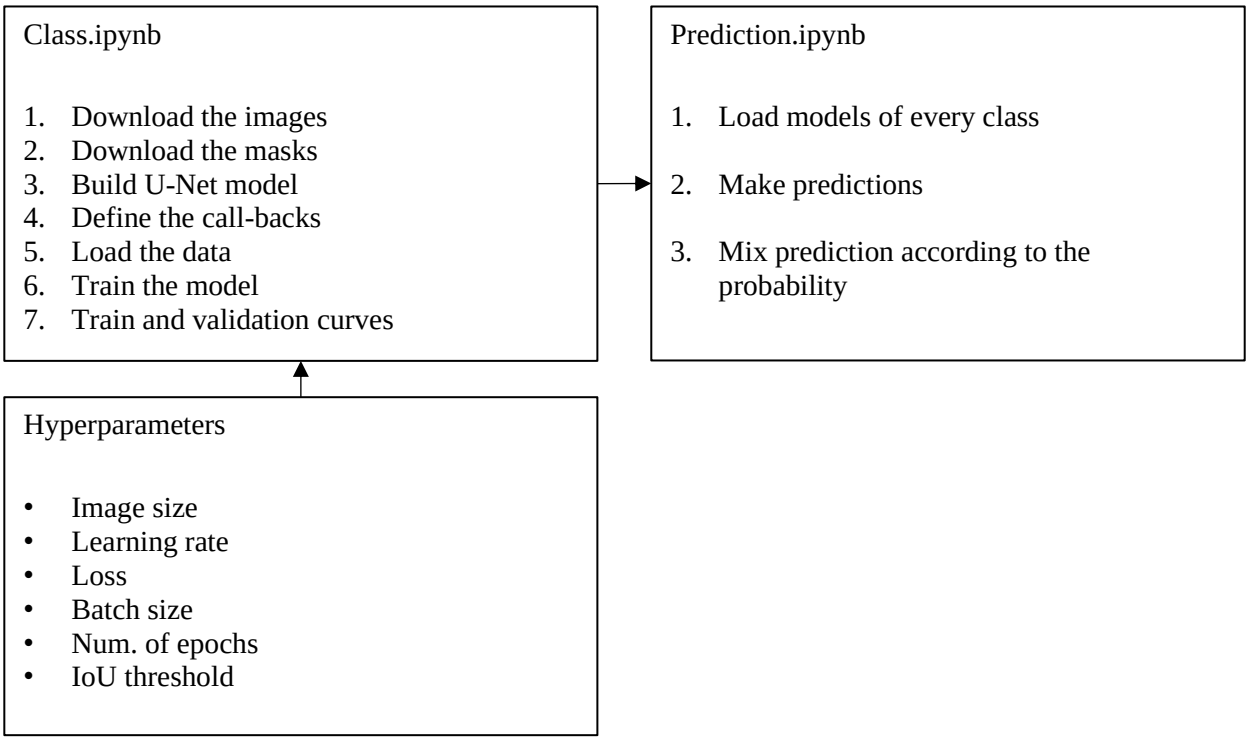


Figure 6-10: Structure followed in the script.

6.3.3.1 Data Preparation

In contrast to Faster-RCNN, where the bounding box was added to the image, U-Net reads a mask to learn the class. A mask is an image of the same size as the real one, whose pixel values are "1" if the pixel is from the desired class and "0" for the background. Next images show an example, where the figure on the right is the image and the figure on the left the mask, for the class "stop sign":

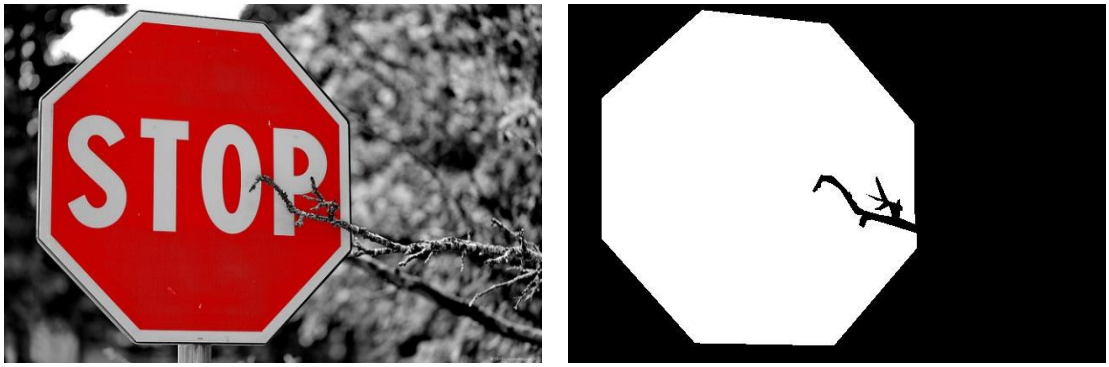


Figure 6-11: COCO mask representation

The whole dataset is downloaded in three folders: train, validation and test. These folders include the true images, while masks are extracted from the real ones according to the "Annotation" files, by means of the COCO API. This API can be downloaded for free and includes several methods to download, plot and attach de masks to the true images.

Then, the U-Net architecture is defined and compile with Adam optimizer, the binary cross entropy as the loss function and the image size as a hyperparameter (128 for the base case).

6.3.3.2 Training

Every model was trained on a NVIDIA GTX 2060 (6GB). First, a model for every class is trained under the base conditions. Later, the two worst classes are trained varying the conditions to find the optimal configuration of the hyperparameters. The conditions to test are the image size, the learning rate and the batch size. Finally, every model is re-trained with the best configuration.

The base conditions are:

- Batch size: 2
- 4000 training images, 800 validation images, 800 images test.
- Adam learning rate = 1x10-4
- images size = 128
- Threshold = 0.5
- Epochs = 30

Class	Training accuracy	Training loss	Validation accuracy	Validation loss	IoU
Person	0.93	0.10	0.88	0.30	0.33
Traffic light	0.99	0.02	0.98	0.06	0.18
Traffic sign	0.99	0.03	0.98	0.08	0.57
Car	0.97	0.06	0.94	0.20	0.15
Bus	0.96	0.06	0.89	0.35	0.49

Table 7: IoU results for the base conditions.

As it can be seen in the table above, the validation accuracy and loss are not related with the Intersection over Union. The IoU is calculated with the test images, and measures the number of pixels that the predicted image and the mask have in common. A value of 1.0 in the IoU is only obtained by the true mask of the image. Therefore, the IoU is an indicator of the performance of the model in working conditions, and in this case, it is not correlated with the validation accuracy. For that reason, the IoU is calculated in each epoch and when it improves, the model is saved. By doing that, overfitting is prevented and the model can be trained with more epochs.

1. **Image size**:

It is important to note that due to the Max Pooling layers and the Up-Sampling convolutions, only multiples of 2 are valid as the image dimension., i.e.: 2,32,64,128,256,512…

As in the Faster-RCNN case, the traffic lights are hard to detect when the image is small. In U-Net, localization is given by the lateral connections to the up-sampling path. However, if the image is too small, this information can get lost. The original paper of U-Net obtained the best results for 512x512 images, so increase the size of the image should improve the performance. This simulation is only made for classes "car" and "traffic light" to save time as they were the worst in the previous test, although every model will be re-trained with the best configuration:

Class	Image size	Training accuracy	Training loss	Validation accuracy	Validation loss	IoU	Total time
Traffic light	128	0.99	0.02	0.98	0.06	0.18	1,5h
	256	0.98	0.05	0.98	0.08	0.21	3,25h
	512	0.98	0.04	0.98	0.06	0.16	10h
Car	128	0.96	0.07	0.94	0.18	0.16	1,75h
	256	0.94	0.13	0.92	0.21	0.21	3,5h
	512	0.94	0.14	0.94	0.18	0.19	10h

Table 8: Results for classes 'traffic light' and 'car'.

For an image size of 512, the memory limit for the GPU was reached. For that reason, the batch size was reduced to 1, and the 3 different image sizes was tested again to compare the effect of the image size with the same conditions.

One drawback of varying the image size is the increase of the total time training the model. While a 128x128 input needs around 1h30, a 512x512 input needs almost 10h. Considering that time, the entire model would need 50h to train, which is unaffordable.

The best configuration in the case of "traffic light" is reached with a 256x256 input, increasing the IoU from 0.18 to 0.21, while for the "car" class, the best IoU is reached with an image size of 600.

Considering the results, the image size is set to 256, as it gets better results than the 128 size-case with an affordable increase of the training time.

2. **Batch size**:

The limitation for increasing the batch was the input size, as it reached the memory limit of the GPU when the image was 600x600. Because the optimal input size is set to 256, test different batches sizes can be a good solution to reduce the training time.

A higher value of the batch size cannot be tested due to memory limit conditions.

Class	Batch size	Training accuracy	Training loss	Validation accuracy	Validation loss	IoU	Training time
Traffic light	1	0.98	0.05	0.98	0.08	0.21	3,25h
	2	0.99	0.02	0.98	0.05	0.24	2,60h
	5	0.99	0.01	0.98	0.07	0.26	2,50h
Car	1	0.94	0.13	0.92	0.21	0.21	3,50h
	2	0.96	0.07	0.94	0.16	0.23	2,80h
	5	0.96	0.07	0.94	0.18	0.26	2,50h

Table 9: results after varying the batch size value.

The best value for the IoU is obtained with a batch size of 5, which also reduces the training time. The reason behind that is because with a higher value of the batch, the model knows the direction to update the gradient faster. With a higher number of epochs, the IoU for a batch size of 1 may get the same result, but it would take longer.

Therefore, the optimal configuration for the batch is 5.

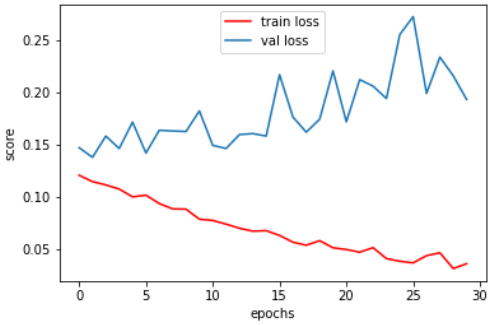
3. **Learning rate**:

Although the training and validation accuracy graphs show that the learning rate is optimal, different values have been tested to prove that statement.

Class	Learning rate	Training accuracy	Training loss	Validation accuracy	Validation loss	IoU
Car	$1x10^{-3}$	0.95	0.11	0.94	0.19	0.24
	$1x10^{-4}$	0.96	0.07	0.94	0.18	0.26
	$1x10^{-5}$	0.98	0.02	0.93	0.36	0.18

Table 10: IoU values for different learning rate values.

In terms of training and validation accuracy, the three learning rates obtain similar results. The difference comes with the losses in training and validation, and the IoU while testing. Next graphs show the losses of each learning rate:

$1x10^{-3}$	

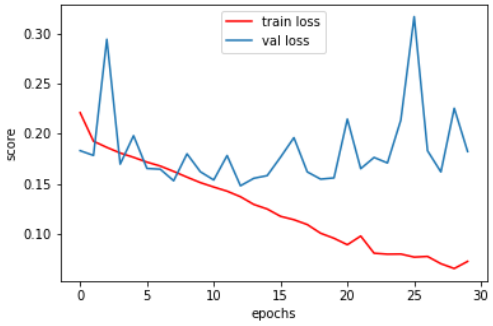
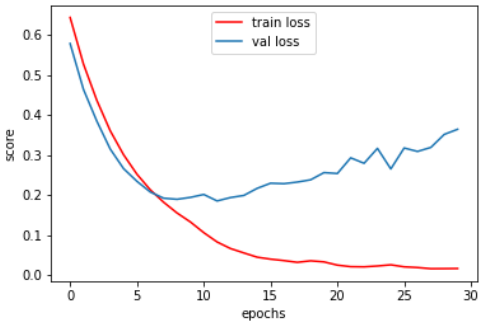
$1x10^{-4}$	
$1x10^{-5}$	

Figure 6-12: Training and validation losses for different leaning rate values.

Figures on the left and on the right are a clear example of overfitting. While the validation loss on the left figure never converges due to a too high value of the learning rate, the figure on the right converges the most and quickly starts overfitting. The lowest value for the training loss is achieved with a learning rate of $1x10^{-5}$, proving that very often a good training performance is not related with good results in validation and test.

Therefore, the learning rate has been fixed to $1x10^{-4}$ and the number of epochs to 30, to prevent overfitting.

4. **Threshold**:

The output of the model is an array with the probabilities for each pixel of belonging to a certain class. The threshold is the hyperparameter that stablish if that probability is enough to be considered or not. A value of 0,5 makes that each probability under this value is considered as background, while above that value is considered as the object that the model is classifying.

A value too low will predict noise while a value too high may miss some relevant information. Find the balance between the two edges is made by trial and error and depends on the model.

Each model is re-trained with the best configuration:

- Image size of 256.
- Batch size of 5.
- Learning rate equals to $1x10^{-4}$.

Tables below show the IoU for different threshold values:

Class	First model	Threshold						
		0.2	0.3	0.4	0.5	0.6	0.7	0.8
Person	0.33	0.44	0.45	0.44	0.44	0.43	0.41	0.38
Traffic light	0.18	0.30	0.29	0.27	0.26	0.25	0.22	0.19
Traffic sign	0.57	0.58	0.59	0.58	0.57	0.55	0.53	0.50
Car	0.15	0.3	0.29	0.28	0.26	0.23	0.21	0.18
Bus	0.49	0.54	0.55	0.54	0.53	0.5	0.47	0.42

Table 11: Threshold comparison.

As it can be seen in the table, the best IoU is achieved with a threshold value of 0.2-0.3. However, the example below shows that when are some objects together, the model may predict as a car, in this case, when there is a person (second picture). On the other hand, when the class is the only object in the picture, it may get better results (first picture), as the predicted area is bigger.

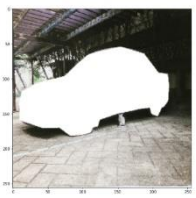
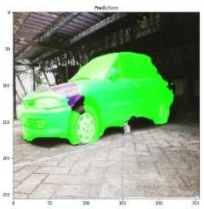
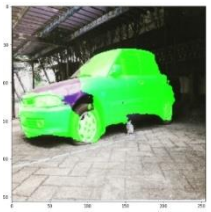
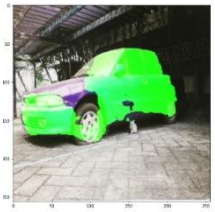
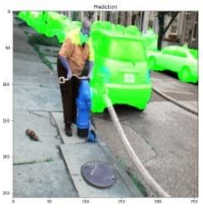
Real	Mask	0.2	0.4	0.6
		 IoU = 0.86	 IoU = 0.77	 IoU = 0.68
		 IoU = 0.77	 IoU = 0.79	 IoU = 0.75

Table 12: Visual representation for the different threshold values.

Note that for the second picture, when the model is assembled, it will predict the person instead of the noise associated with the car, reducing the noise in this case.

To sum up, considering a threshold value of 0.4 and the base conditions described in the beginning of this section, the overall IoU of the model obtained for the model is 0.42. Some samples considering all classes in the model are shown at the end of this project, in Appendix 8.

6.3.4 CONCLUSIONS

In this section, two algorithms with two different approaches of image classification have been trained. The first algorithm classified the images and the objects inside the picture adding bounding boxes to the original image, while in the second algorithm the classification was made by creating an extra image with the pixels of the objects of a different colour.

Faster-RCNN model has almost 140 million parameters while U-Net has 36 million. In addition to the computing resources associated with that, the observed impact has been the

overfitting risk associated with U-Net. While Faster-RCNN barely suffers from overfitting, optimize the learning rate and the number of epochs in U-Net is critical for a good performance, which suffers overfitting for longer training times than 4h.

Although Faster-RCNN got worse results than U-Net and needed more time to train, the integration of Faster-RCNN with video algorithms, as SlowFast in this project, proves that these models are powerful and scalable for several applications. The limitation in this project was the time and resources available to train. In section 6.3.1.2, increase the image size proved that it is the reasonable next step to get similar results than in the paper [25].

In the case of U-Net, the size of the model and its simpler complexity made possible to simulate multiple scenarios and test the effect of different hyperparameters. The main advantage of U-Net in comparison with similar algorithms in the state-of-the-art literature is the advantage of learning the context information associated with the image. This is done by the lateral connections between the down-sampling and the up-sampling paths. However, it has been observed that if the input image is too small this information may get lost. Although the applications are different, the same optimal image size has been obtained in this project and in the paper [32], proving that the results shown in the project are reliable.

6.4 VIDEO ALGORITHMS

SlowFast algorithm is tested for video action recognition. The algorithm has been chosen instead of the ones listed in section 4.1 because it is the newest and most innovative, and uses Faster-RCNN to classify the images with bounding boxes. Which has been already trained previously.

The dataset chosen to train the model is UCF101. In this dataset, 101 casual actions are classified. Some of the actions are listed below, to see the full list see reference [59]:

- Basketball shooting.
- Ice Dancing.

- Walking with a dog.
- Writing on board.
- Horse riding.

6.4.1 SLOW-FAST

The 101 classes are considered, with a total dataset size of 6.7 GB. The number of parameters of the model is 36 million, very similar to the U-Net case. Both conditions make possible to train the model with the NVIDIA GTX 2060.

The hyperparameters are defined in a separated file, so they can be varied externally to the SlowFast script. The main scheme of the algorithm is:

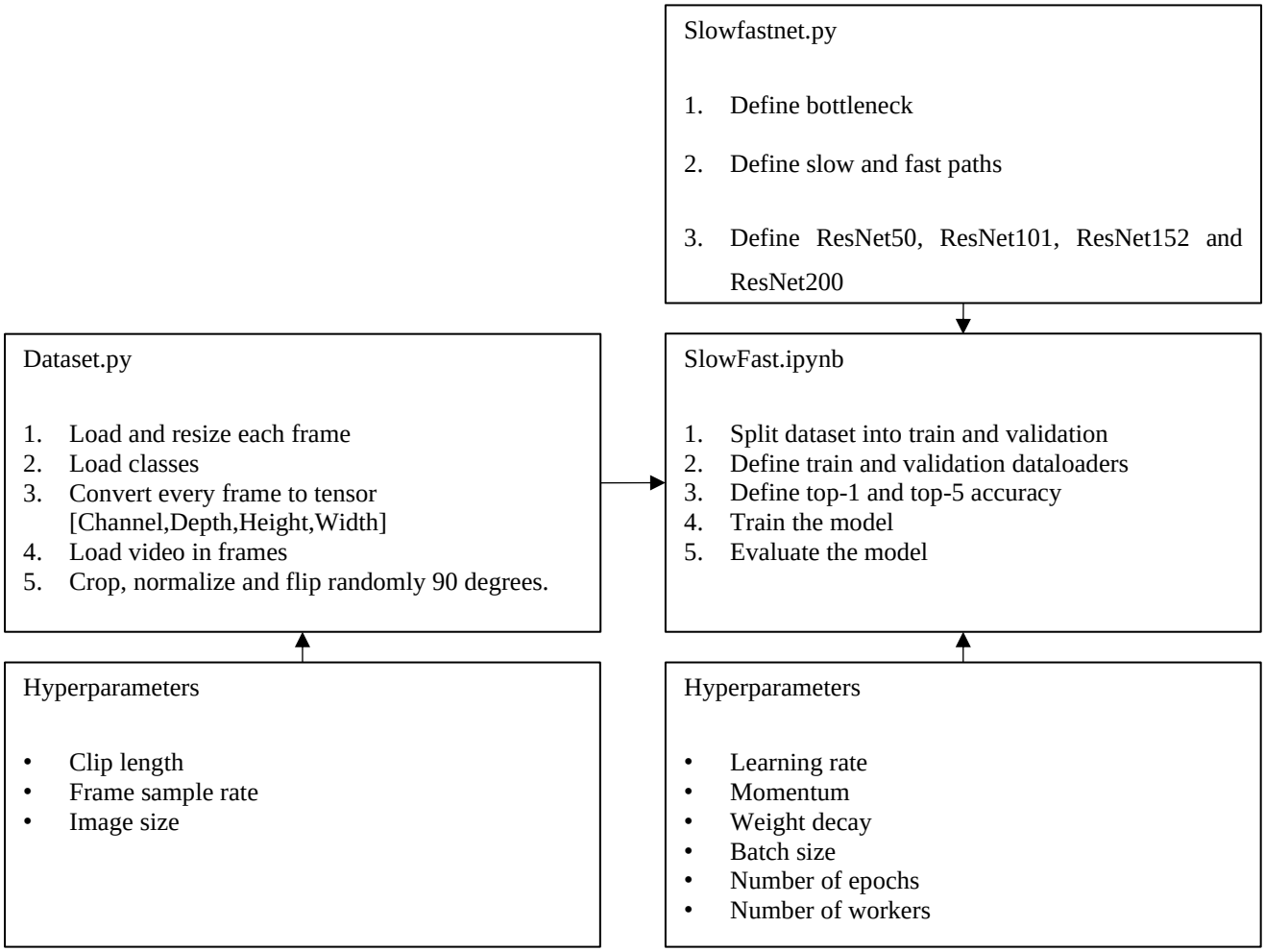


Figure 6-13: Structure followed in the script.

6.4.1.1 Data Preparation

The dataset is divided according to a 70/30% train and validation relation. Each video is between 6 and 7 seconds, and multiple videos can be related with the same scene. For that reason, it is important that the all samples from the same scene are always in the train or in the validation folders, but not in both, otherwise the model would classify the validation videos linking the videos instead of learning their features.

Before training the model, the script splits every video in 64 frames, processing each frame individually, which is the frame rate. It is stored in a buffer and then cropped and normalized into the desired dimension, which is 112x112 in this case. Finally, the buffer with the frames is stored in a 4-dimension matrix, which columns are: Channel, Depth, Height and Width. If the dataset loaded is for training, a horizontal flip is made to some frames randomly to improve the performance of the model.

The clip length, 64 in this case, depends on the dataset and the algorithm. As this dataset only contains videos of 6-10 seconds, a 64 buffer is enough to capture the action and the context of the video. This value has been proved in the literature that work efficiently for this dataset.

6.4.1.2 Training

Top-1 and top-5 accuracy are the performance indicators chosen for training this algorithm, as the official paper [43] did, to be able to compare both results.

To avoid reaching the memory limit, the Convolutional Neural Network of both paths is the ResNet50, with a total number of parameters of 51 million.

The hyperparameters considered in the training are:

- Number of workers = 4
- Stochastic Gradient Descent (SGD) Learning rate = 0.01
- Momentum = 0.9

- Weight decay = 0.00005
- Batch size = 8
- Number of epochs = 40

The number of workers is a hyperparameter defined in PyTorch library only, that limit the number of processes that CPU is executing in parallel, to make sure that the bottleneck of the model are the processes executed in the GPU. Therefore, it is not a hyperparameter associated with the performance of the model but with the performance of the machine. It is set to 4 as it balanced performance with computing demand while training.

SGD optimizer was used in contrast to Faster-RCNN and U-Net, where Adam was implemented. The reason is because the paper implemented it, although Adam has proved to get better results. Momentum and weight decay are also defined to simulate the same behaviour as with Adam. Weight decay tells the optimizer the value to update the learning rate after each epoch. When using Adam, this hyperparameter is set automatically by the optimizer, while defining it manually can prevent the model to get stuck in a local minimum.

Finally, the batch size and the number of epochs have been chosen based on the knowledge from the previous sections: in Faster-RCNN, higher batches reduced the training time, but being careful with oscillated behaviours; in U-Net, 40 epochs have been proved to be a reasonable number of epochs that prevent overfitting.

Top-1 and Top-5 accuracy (see section 4.2.2) have been used to measure the performance of this model, to compare the results with the paper [43]. The results for the base conditions training are shown below:

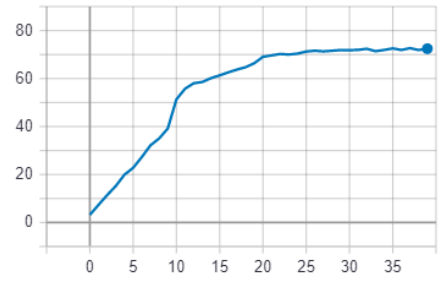
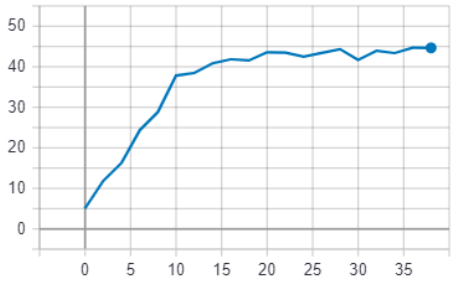
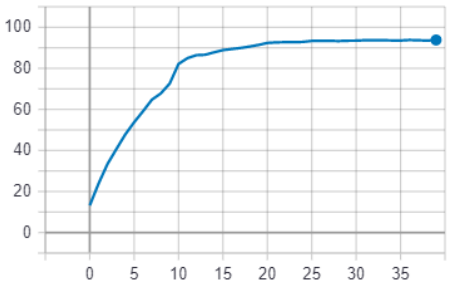
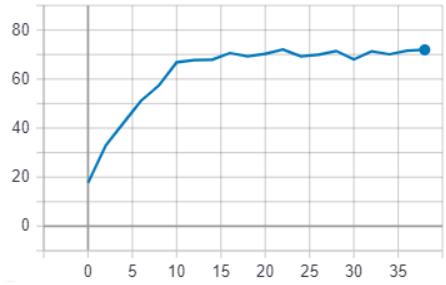
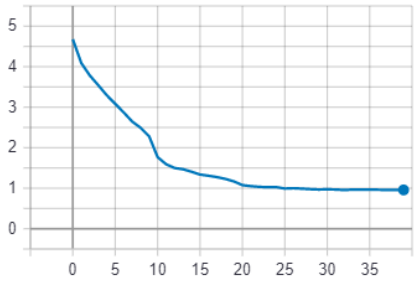
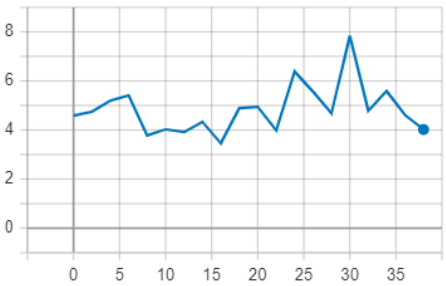
	Training	Validation
Top-1 Acc.		
Top-5 Acc.		
Loss		

Table 13: Results after training for the base conditions.

As it can be seen in the previous graph, the loss for validation oscillates too much, indicating that the learning rate may be too high. Moreover, the validation top-1 accuracy is very low compared to the accuracy obtained in the paper. For that reason, varying the learning rate has been the first approach to improve the validation accuracy.

The optimizer used to train the model has been the Stochastic Gradient Descent (SGD) with a gamma value of 0.1 and 10 steps, meaning that every 10 steps, the learning rate is updated by a 0.1. The first 10 epochs, the learning rate is 0.01, from epochs 10 to 20, the learning rate is 0.001, and so on. The weight decay is $1x10^{-5}$, according to the state-of-the-art literature that suggests that deeper architectures usually require smaller values.

This optimizer has been chosen according to the paper. For that reason, the first iterations try to improve the validation accuracy by varying the learning rate and the weight decay. Later, ReduceLROnPlateau and AdamW are used:

- ReduceLROnPlateau is a learning rate scheduler, which update the value after a determined number of epochs if the optimizer has not improved. The number of epochs is called 'patience' and has been fixed to 10, similar to the 10 steps of the previous case. Also, the value to update the learning rate is 0.1, which is the value by default.
- AdamW [60]: similar to Adam (see 5.5), the term related with the weight decay is normalized in the cost function, which regularizes the update in the weights to get smaller and more slowly changing gradients. The two terms framed with the red boxes are the two terms added to the Adam equation:

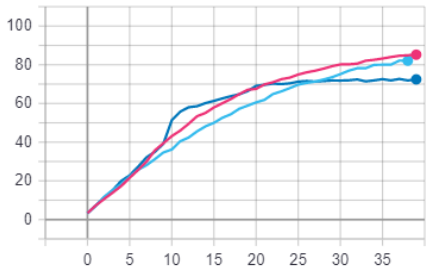
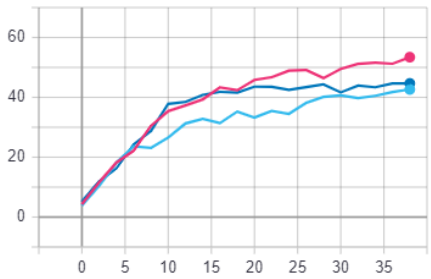
$$x_t \leftarrow x_{t-1} - \alpha \frac{\beta_1 m_{t-1} + (1-\beta_1)(\nabla f_t + \boxed{w x_{t-1}})}{\boxed{\sqrt{v_t}} + \epsilon} \qquad (32)$$

First, as it was said previously, the learning rate was changed to 0.001. Although the shape of the loss improved, the validation accuracy dropped. Then, the weight decay was increased to 0.0005 to try to convergence faster, but the results were almost identical. Later, ReduceLROnPlateu and AdamW were tested. In the case of ReduceLROnPlateu, it improved the training accuracy but it got the same results for validation. Finally, AdamW has proved that in this case, it performs much better than SGD.

Optimizer	Learning schedule	Learning rate	Weight decay	Top-1 Acc	Top-5 Acc	Top-1 Val	Top-5 Val
SGD	Step LR	0.01	$1x10^{-5}$	0.72	0.92	0.45	0.72
	Step LR	0.001	$1x10^{-5}$	0.62	0.93	0.40	0.67
	Step LR	0.001	0.0005	0.62	0.92	0.39	0.65
	ReduceLROnPlateau	0.001	$1x10^{-5}$	0.82	0.97	0.44	0.67
AdamW	N/A	0.001	$1x10^{-5}$	0.85	0.98	0.54	0.78

Table 14: Results for different optimizers.

Next graphs only show the training and validation accuracy and the losses for the base conditions (blue line), the ReduceLROnPlateau (cyan) and the AdamW (pink) cases, to lighten the graph.

	Training	Validation
Top-1 Acc.		

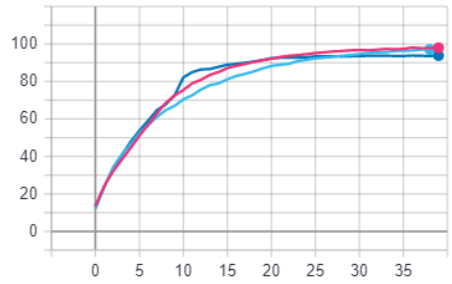
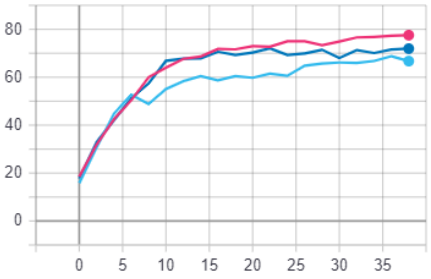
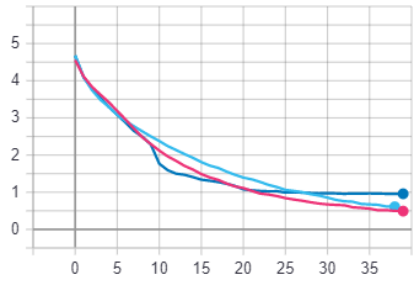
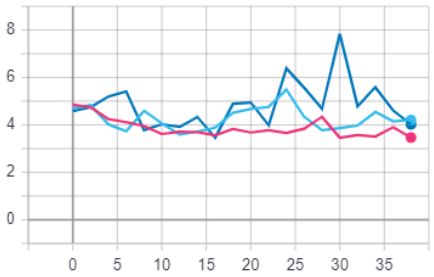
Top-5 Acc.		
Loss		

Table 15: Curves for the base conditions, ReduceLROnPlateau and AdamW optimizers.

Previous to the samples showed above, the train/validation split was made randomly, taking the 70% of the data for training, and the remaining 30% for validation. For that reason, some clips from the same video were taken for training and validation. Although the clip is not the same, the camera and the action are the same or very similar, which simplifies the task of classification, obtaining very good results for validation. Next figure shows the improvement in validation when the model is split randomly in comparison with the best configuration shown above, where the pink curve represents the AdamW optimizer, and the orange graph the random split.

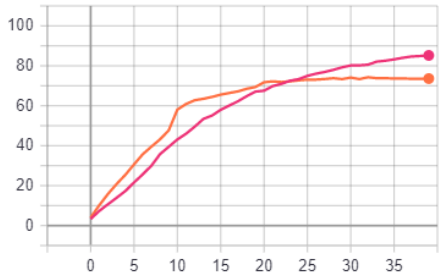
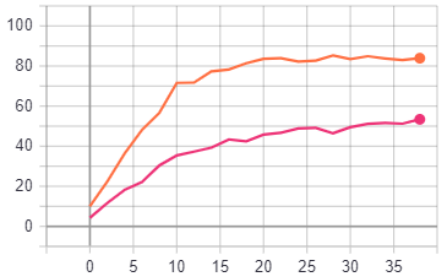
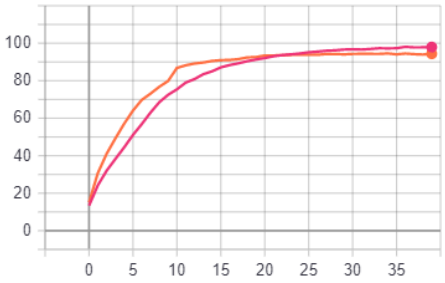
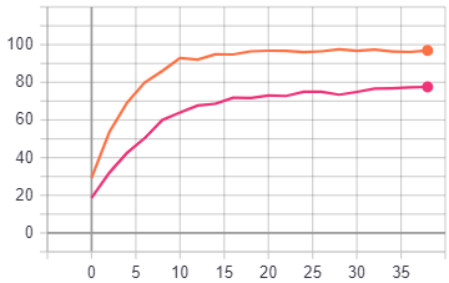
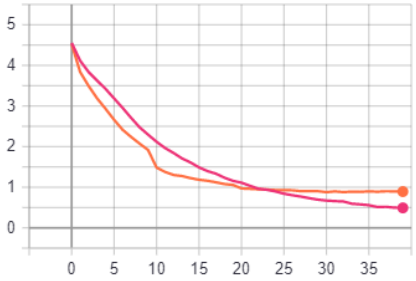
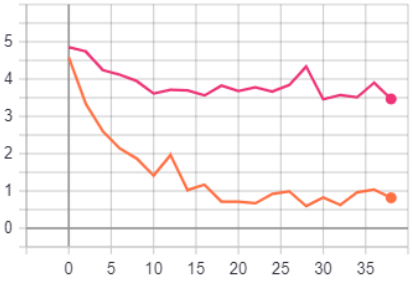
	Training	Validation
Top-1 Acc.		
Top-5 Acc.		
Loss		

Table 16: Comparison for AdamW case and the random train/validation split.

For training, AdamW optimizer performs better than SGD, as expected. However, the validation accuracy increases to an 85% for top-1 and almost 99% for top-5 accuracy. That happens because the model has already seen a clip very similar, so it behaves as the model would be classifying the training dataset.

The paper [43] obtained an accuracy of 75.6 for similar conditions in Kinetics-400 dataset. While the random split training got even higher values than the paper, the AdamW case is a

bit far from that results. Varying the clip length and increasing the frame size are two options that may get better results, but unaffordable due to memory limit conditions in this project.

To approach the bounding box problem in video classification, Faster-RCNN can be used prior to the SlowFast model. The strategy followed in this project has been to classify a frame and fixed the box during 16 frames, to avoid longer processing times. The output of the video, for 'Apply Make Up' class is as follow:

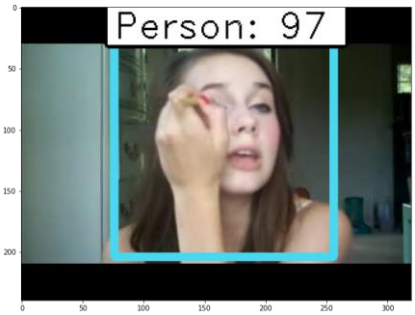


Figure 6-14: Frame after applying Faster-RCNN, for the class 'Apply Eye Make Up'.

The SlowFast architecture of the model followed in this chapter can be found in Appendix 9.

6.4.2 CONCLUSIONS

SlowFast algorithm is very recent (Oct. 2019) but has already prove its good performance with difficult tasks such as action recognition and context understanding. Its integration with bounding boxes algorithms, as Faster-RCNN, makes this model very attractive for several applications.

However, video datasets are usually very large and training with them is expensive and tedious. In this case, it has been preferred to take a smaller dataset to be able to train and understand the concepts related with action recognition. The same architecture and script can be applied to a larger dataset, expecting similar results.

In this case, training the algorithm by inference instead of random weights (from scratch) is crucial, as the datasets and algorithm are usually more complex. In this case it was not

necessary as UCF-101 is a relatively small dataset compared to the datasets mentioned in (section 6.2.2). UCF-101 is a widely known dataset in the state-of-the-art literature, and it has been chosen for that reason. However, training the model with the AVA dataset would allow the model to be useful for more common applications, and inference would play an important role.

7. FINAL CONCLUSIONS

This project has approached the challenge of understanding how convolutional neural networks work when they are applied to Computer Vision tasks. A review of the main algorithms, the state-of-the-art literature of Computer Vision and the different approaches to solve the image classification problem has been developed, training and comparing some models to deeply understand them.

Very long training times and the requirement of an appropriate GPU for machine learning has been the constraints during the development of the project. Computing resources and their availability are the bottleneck of computer vision nowadays, and optimize the resources and the algorithms are one of the challenges that this project has approached. However, these limitations have helped to explore the optimal solution for each task, considering each assignment individually.

Finally, the familiarity with popular tools and libraries, such as TensorFlow, Pytorch and Google Colab, has been very valuable, not only for Computer Vision tasks, but to Machine Learning in general.

8. WAY FORWARD

This section presents the next steps to be followed in the future to increase the scope of the algorithms and apply them to more sophisticated applications.

1. **Instance segmentation**: as it has been explained in section 3.4, after object detection and semantic segmentation, instance segmentation combines both approaches in the output. Considering the already trained Faster-RCNN model, it seems reasonable to add the Mask-RCNN model at the end of the algorithm, and approach the instance segmentation task.
2. **Object detection with action recognition**: AVA dataset is a dataset for video action recognition that includes bounding box annotations of every video. Using this dataset for training Faster-RCNN combined with SlowFast will allow the model to be applied to more complex applications.
3. **Case study**: Apply the algorithms developed in this project to a real example, which consists of a video application that recognize the people on it and measure the distance between them. This system can be used for public places where the distance must be ensured to preserve the health of the people due to the COVID-19 situation.

PART II. APPENDIX

1. SUSTAINABILITY

All United Nations Member States agreed, in 2015, the 17 Sustainable Development Goals that every Member should take into account to reduce poverty and inequality, and improve health and education. These goals are designed to achieve a more sustainable world for all of us, and every country should align its policies with them.

This project is linked most heavily with the ODS number 9, which is: Industry, innovation and infrastructure. Although the algorithms developed by this project do not exist physically, infrastructure now is not limited to physical structures as in the past. Industry 4.0 is shaping the world, the machines and elements are now interconnected, storing the information mainly in the 'Cloud'. This allows that people remotely can see as they were in the place in question, reducing travelling and the costs associated with that, and leading to a more sustainable industry.

Robots and autonomous applications are now leading this change in the shape of the industry. From busy assembly lines with hundreds of people, or offices full of paperwork, Industry 4.0 allow to re-allocate those people to more value-added positions and substitute the physical paper to virtual instances.

This project analyses algorithms that can help machines to work more precisely, reducing the defects and optimizing the existing resources, or being operative only when a certain condition exists, saving lot of energy associated with inactivity. That is why this project can be aligned also with goals number 12, "Responsible Consumption and Production" and number 13 "Climate Action", which outline the necessity of a more sustainable usage of the energy and resources.

There exist lots of applications mentioned throughout this project, but it is our duty, as a society, to find the applications and situations where the technology can help to reduce the gap between those more in need and the rich countries.

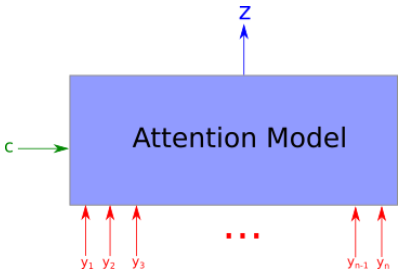
The article [61] studied the impact of Industry 4.0 solutions in Smart Factories, compared to traditional ones, and stated that Industry 4.0 can potentially reduce the additional energy expense by a factor of 75%. If applied to every factory around the world, the global impact of Industry 4.0 is promising.

2. NEURAL NETWORKS WITH 'ATTENTION'

The integration of 'attention' mechanisms in neural networks algorithm has made a large impact on neural computation. For instances, Faster-RCNN method described in the first chapter uses this technique to better approach the Region Proposal Network in the early stages of the algorithm.

The origin of this idea stems from the need for mechanisms to focus on the most pertinent piece of information, instead of using all of it, which would lead to higher computational time being irrelevant to the neural response.

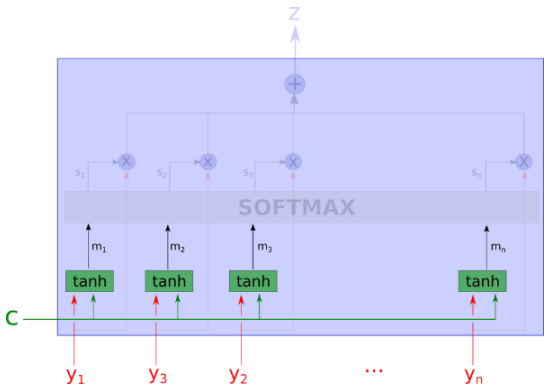
An attention model takes y_n arguments and a context c, and returns the vector z which represents a summary of y_n focusing on the information given by the context c .



To obtain the vector z, it is necessary to break down the black box of the previous image:

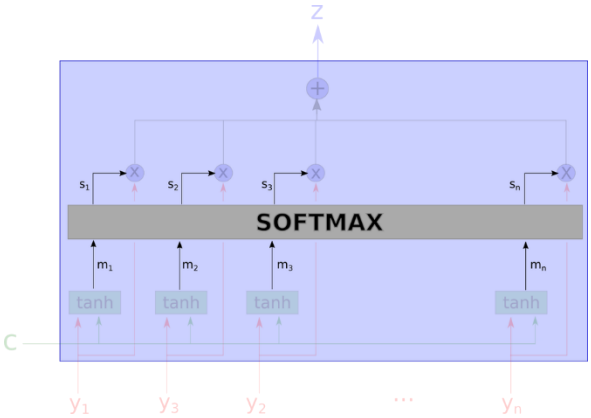
1. The network computes each argument y_i without looking at the other y_j as follows:

$$m_i = \tanh\left(W_{cm}c + W_{ym}y_i\right)$$



In this case, tanh is showed as the example, although Relu or sigmoid functions can be equivalent used.

2. Then, each weight m_i is computed using a *softmax* activation function, leading to s_i:



where $\Sigma s_i = 1$

The higher the value of s_i, the higher the relevance of each variable according to the context.

3. Finally, vector z is the weighted arithmetic mean of y_i , where:

$$z = \Sigma s_i * y_i$$

One major advantage of this model is that the output z can be visualized, being able to understand where the model is looking at. In the next example, when the weight of the image is high the pixel is whiter:

The model described is an example of what is called *soft attention*, because the context vector is calculated as a weighted average of the encoder hidden states. On the other hand, it is possible to use attention scores to select a single hidden state, what is called *hard attention*. However, more complex techniques are required, it is explained in the paper [62].

3. VGG-16

Authors K. Simonyan and A. Zisserman proposed in the paper [63] the convolutional neural network model VGG16, that achieved the accuracy record in ImageNet competition the year it was released, in 2015. The main innovation that VGG16 model introduced was the use of the same 3x3 kernel-size filters for every layer, increasing the computational efficiency overall.

1. **Architecture of the model**:
   - The dimensions of the filters are 3x3, which is the smallest size to capture left – right – up – down motions.
   - The stride of the convolution is fixed to 1 pixel.
   - Same padding – the spatial resolution is preserved after each convolution.
   - 2x2 Max-Pooling layer with a stride of 2.
   - Three Fully-Connected (FC) layers: the first two FC layers with 4096 channels and the last layer with 1000 in this case, that represents the number of classes.
   - SoftMax at the end of the last FC layer.
   - ReLU activation function after each hidden layer.

2. **Applications**:

   VGG-16 model is used mainly for image classification problem. Its potential as classification model has been proved with different algorithms; Faster-RCNN applies VGG-16 after the Regression model, for example. However, the large number of weights that need to be tuned require high computational resources. For this reason, simpler architectures are recommended for easy classification tasks.

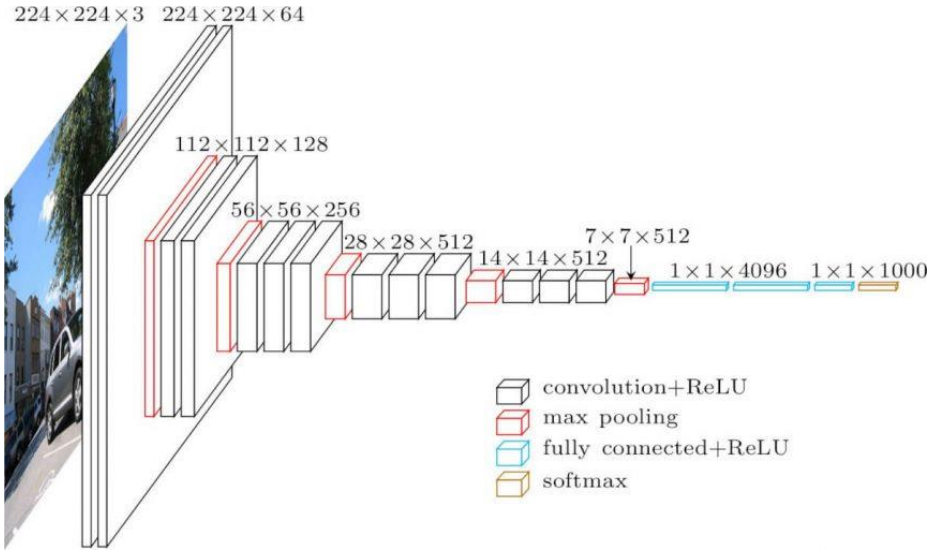


Figure 3-1: VGG-16 structure. Source: ref. [27]

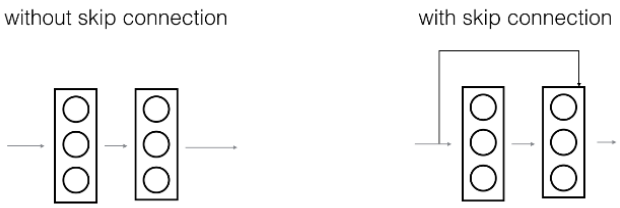
4. RESNET

ResNet is the acronym for Residual Network, and it is one of the most used networks as the backbone of many computer vision algorithms. It was developed in 2015, when it won the ImageNet challenge that year. The innovation of ResNet is the ability to train very deep neural networks, addressing the problem of vanishing gradients.

Vanishing gradient problem refers to the case when gradient is back-propagated to early layers, making the gradient too small until it vanishes, suturing the gradient and even degrading rapidly. ResNet solves this effect by skipping connections.

1. Skip connections

In contrast to the classic approach where the layers are stuck together one after the other, ResNet stacks convolution the same way adding the original input to the output of the convolution block. Next figure shows both approaches:



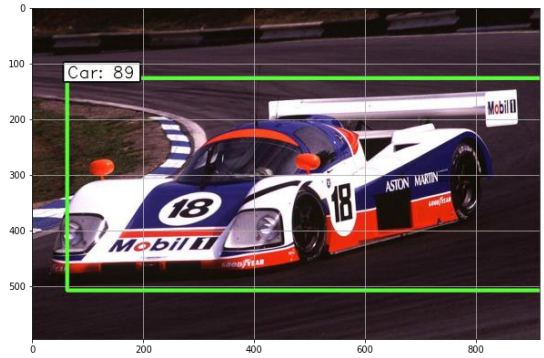
If the skipped connection has the same size as the output, it can be done as shown above, while if the sizes are different, a convolution and a batch normalization must be done to the skip connection in order to add it to the output.

Skip connections performs well because the model learns an identity function that ensures that higher layers perform at least as good as the lower layers.

There are several ResNet implemented depending on the application, being the value it follows the number of layers of the model, where Resnet50, ResNet101 and ResNet152 are the most popular.

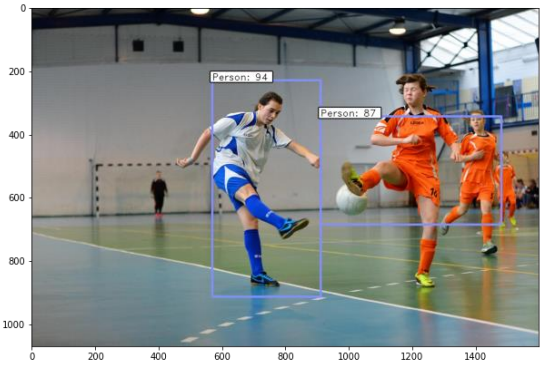
5. FASTER-RCNN ARCHITECTURE

6. FASTER-RCNN RESULTS

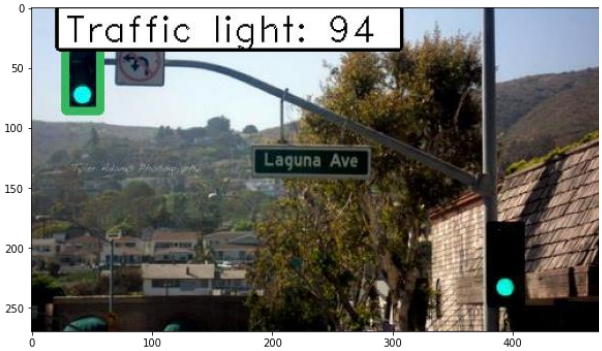


Car: 89.4

Bus: 87.2

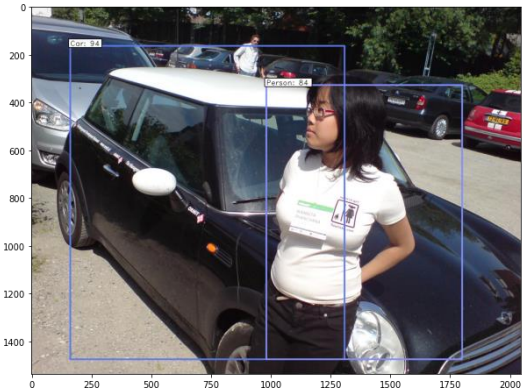


Person: 94.2; Person: 87.2



Traffic light: 94.7

Person: 97.9



Car: 94.7; Person: 84.3

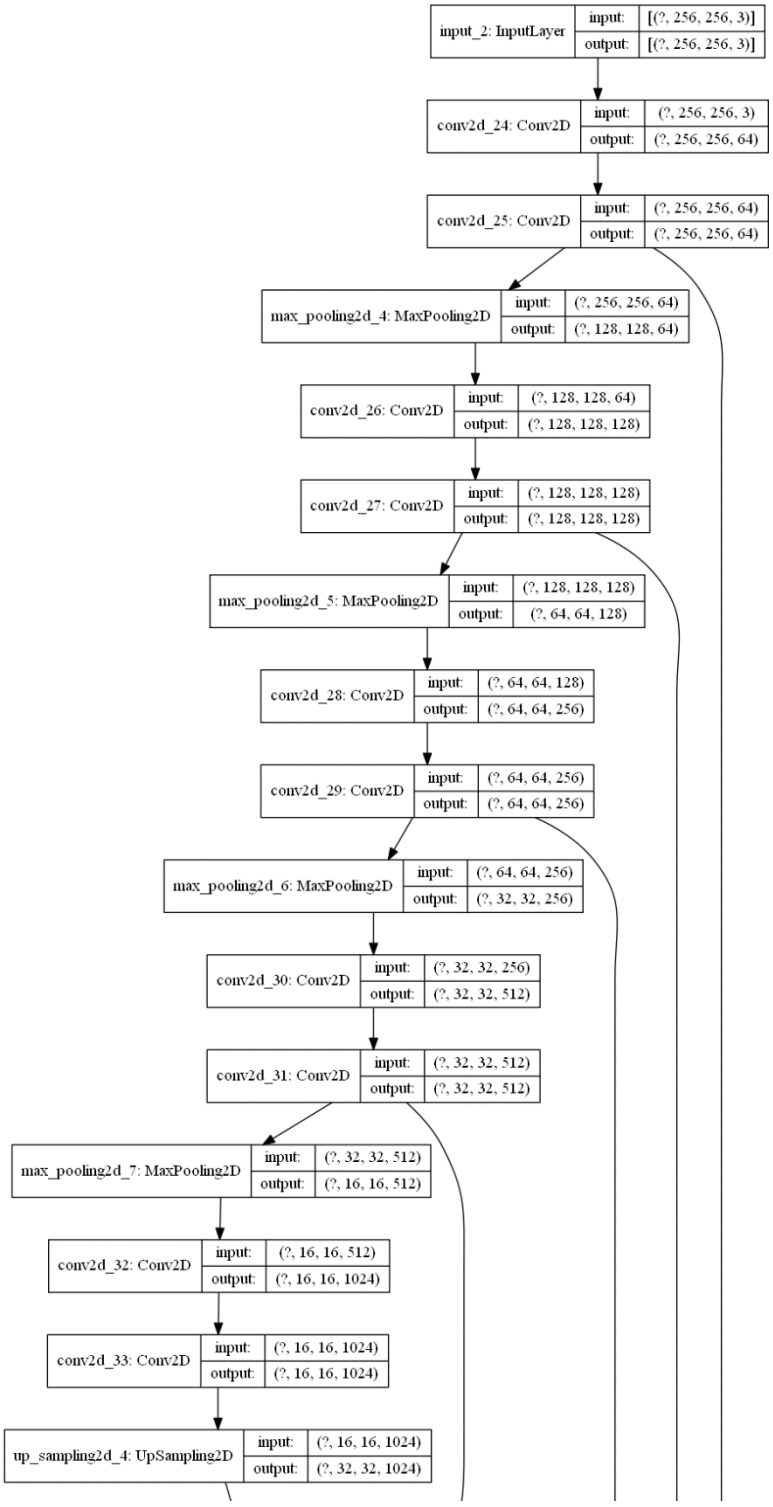
Traffic sign: 99.9; Traffic sign: 99.6

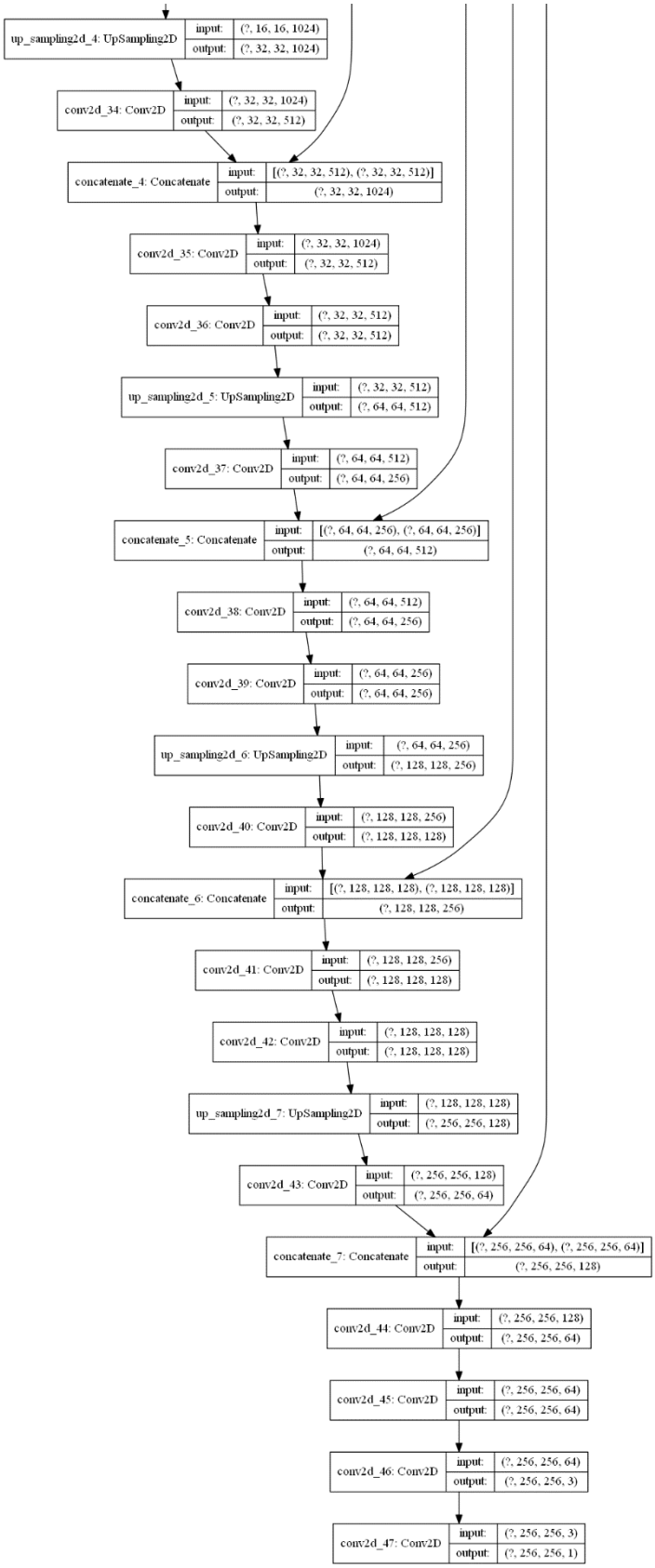
Bus: 97.8

Bus: 99.2; Bus: 76.5; Traffic light: 75.5

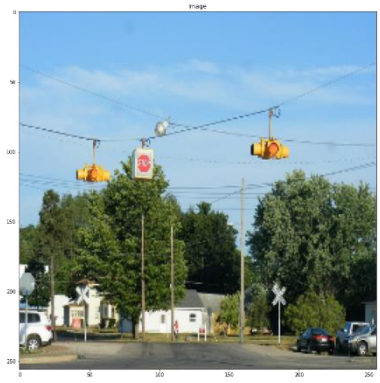
Person: 84

7. U-NET ARCHITECTURE




up_sampling2d_4: UpSampling2D | input: (?, 16, 16, 1024) | output: (?, 32, 32, 1024)
conv2d_34: Conv2D | input: (?, 32, 32, 1024) | output: (?, 32, 32, 512)
concatenate_4: Concatenate | input: [(?, 32, 32, 512), (?, 32, 32, 512)] | output: (?, 32, 32, 1024)
conv2d_35: Conv2D | input: (?, 32, 32, 1024) | output: (?, 32, 32, 512)
conv2d_36: Conv2D | input: (?, 32, 32, 512) | output: (?, 32, 32, 512)
up_sampling2d_5: UpSampling2D | input: (?, 32, 32, 512) | output: (?, 64, 64, 512)
conv2d_37: Conv2D | input: (?, 64, 64, 512) | output: (?, 64, 64, 256)
concatenate_5: Concatenate | input: [(?, 64, 64, 256), (?, 64, 64, 256)] | output: (?, 64, 64, 512)
conv2d_38: Conv2D | input: (?, 64, 64, 512) | output: (?, 64, 64, 256)
conv2d_39: Conv2D | input: (?, 64, 64, 256) | output: (?, 64, 64, 256)
up_sampling2d_6: UpSampling2D | input: (?, 64, 64, 256) | output: (?, 128, 128, 256)
conv2d_40: Conv2D | input: (?, 128, 128, 256) | output: (?, 128, 128, 128)
concatenate_6: Concatenate | input: [(?, 128, 128, 128), (?, 128, 128, 128)] | output: (?, 128, 128, 256)
conv2d_41: Conv2D | input: (?, 128, 128, 256) | output: (?, 128, 128, 128)
conv2d_42: Conv2D | input: (?, 128, 128, 128) | output: (?, 128, 128, 128)
up_sampling2d_7: UpSampling2D | input: (?, 128, 128, 128) | output: (?, 256, 256, 128)
conv2d_43: Conv2D | input: (?, 256, 256, 128) | output: (?, 256, 256, 64)
concatenate_7: Concatenate | input: [(?, 256, 256, 64), (?, 256, 256, 64)] | output: (?, 256, 256, 128)
conv2d_44: Conv2D | input: (?, 256, 256, 128) | output: (?, 256, 256, 64)
conv2d_45: Conv2D | input: (?, 256, 256, 64) | output: (?, 256, 256, 64)
conv2d_46: Conv2D | input: (?, 256, 256, 64) | output: (?, 256, 256, 3)
conv2d_47: Conv2D | input: (?, 256, 256, 3) | output: (?, 256, 256, 1)

8. U-NET RESULTS




POOPDECK
STOP

POOPDECK
STOP

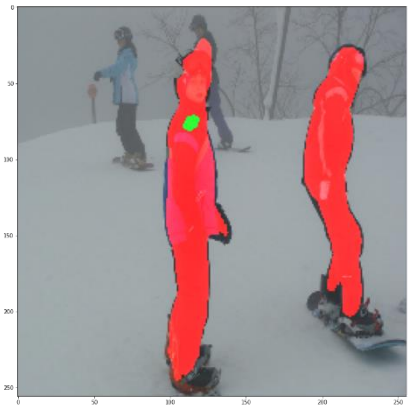
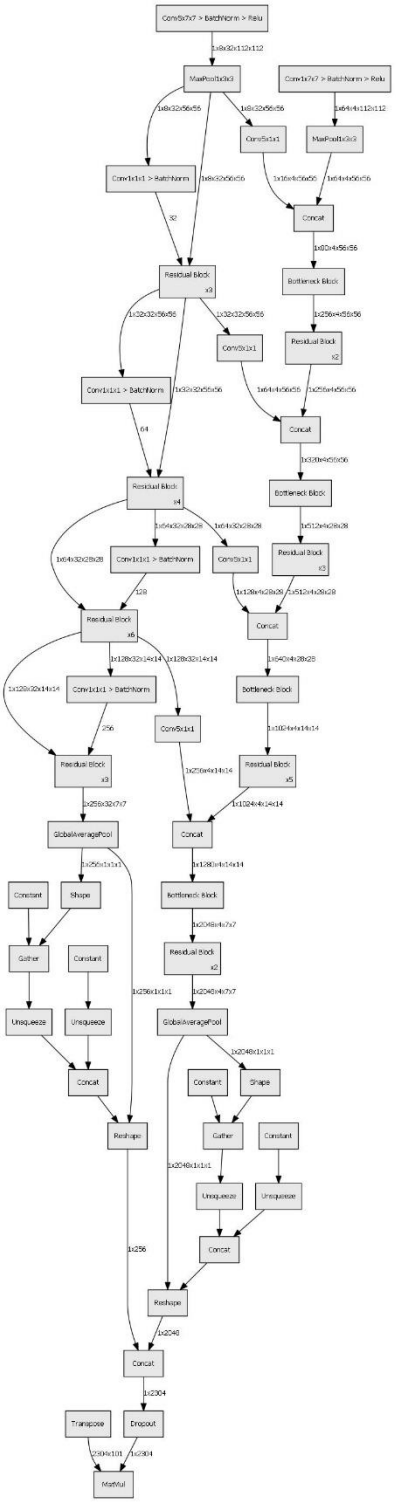



Image
Image
Image
WELCOME
STOP
WELCOME
STOP


Image
Image
Image


Image
Third Street Promenade
Image
Image

9. SLOWFAST ARCHITECTURE



BIBLIOGRAPHY

[1] S. Ren, K. He, R. Girshick and J. Sun, Faster R-CNN: Towards Real-Time Object, 2016.

[2] O. Ronneberger, P. Fischer and T. Brox, U-Net: Convolutional Networks for Biomedical, 2015.

[3] C. Feichtenhofer, H. Fan, J. Malik and K. He, SlowFast Networks for Video Recognition, 2019.

[4] "HostingTribunal," 2020. [Online]. Available: https://hostingtribunal.com/blog/big-data-stats/#gref.

[5] D. H. H. a. T. N. Wiesel, Receptive fields of single neurones in the cat's striate cortex, 1959.

[6] L. G. Roberts, Machine perception of three-dimensional solids, 1963.

[7] C. Maklin, "Towards Data Science," 2019. [Online]. Available: https://towardsdatascience.com/machine-learning-part-17-boosting-algorithms-adaboost-in-python-d00faac6c464.

[8] A. Krizhevsky, I. Sutskever and G. E. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, 2010.

[9] D. G. Lowe, Distinctive Image Features from Scale-Invariant Keypoints, 2004.

[10] H. Bay, T. Tuytelaars and L. V. Gool, SURF: Speeded Up Robust Features, 2006.

[11] D. G. Viswanathan, Features from Accelerated Segment Test (FAST), 2011.

[12] Prabhu, "Medium," 2018. [Online]. Available: https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148.

[13] "missinglink.ai," 2018. [Online]. Available: https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/.

[14] "missinglink.ai," 2018. [Online]. Available: https://missinglink.ai/guides/neural-network-concepts/backpropagation-neural-networks-process-examples-code-minus-math/.

[15] P. Ramachandran, B. Zoph and Q. V. Le, Swish: a self-gated Activation Function, 2017.

[16] "ML Glossary," 2017. [Online]. Available: https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html.

[17] F. Schroff, D. Kalenichenko and J. Philbin, FaceNet: A Unified Embedding for Face Recognition and Clustering, 2015.

[18] P. JAUMIER, "Towards Data Science," [Online]. Available: https://towardsdatascience.com/backpropagation-in-a-convolutional-layer-24c8d64d8509.

[19] W. M. U. S. Y. W. G. W. Kaidong Li, Object Detection with Convolutional Neural Neworks, 2019.

[20] A. Karpathy, G. Toderici and S. Shetty, Large-scale Video Classification with Convolutional Neural Networks, 2014.

[21] R. Girshick, J. Donahue, T. Darrell and J. Malik, Rich feature hierarchies for accurate object detection and semantic segmentation, 2014.

[22] R. Girshick, Fast R-CNN, 2015.

[23] J. Uijlings and K. v. d. Sande, Selective Search for Object Recognition, 2012.

[24] T. Grel, "deepsense.ai," 2017. [Online]. Available: https://blog.deepsense.ai/region-of-interest-pooling-explained/.

[25] K. H. R. G. a. J. S. Shaoqing Ren, Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks, 2016.

[26] J. Redmon and A. Farhadi, YOLO9000: Better, Faster, Stronger, 2016.

[27] C. Szegedy, W. Liu and Y. Jia, Going deeper with convolutions, 2014.

[28] T.-Y. Lin, P. Dollar and F. A. R. (FAIR), "Feature Pyramid Networks for Object Detection," 2017.

[29] S.-H. Tsang, "Towards Data Science," 2019. [Online]. Available: https://towardsdatascience.com/review-fpn-feature-pyramid-network-object-detection-262fc7482610.

[30] Q. Dong, X. Zhu and S. Gong, Class Rectification Hard Mining for Imbalanced Deep Learning, 2017.

[31] R. Khandelwal, "Towards Data Science," 2019. [Online]. Available: https://towardsdatascience.com/computer-vision-instance-segmentation-with-mask-r-cnn-7983502fcad1.

[32] P. F. a. T. B. Olaf Ronneberger, U-Net: Convolutional Networks for Biomedical Image Segmentation, 2015.

[33] K. Cheung, "Algorithm-xlab," 2020. [Online]. Available: https://algorithmxlab.com/blog/computer-vision/#Computer_Vision_Applications_in_the_Financial_Sector.

[34] M. Barre, "SlantRange," 2017. [Online]. Available: https://blog.slantrange.com/using-agriculture-drones-to-move-from-analog-to-digital-farming.

[35] R. Ghosh, "Qure.ai Blog," 2018. [Online]. Available: http://blog.qure.ai/notes/deep-learning-for-videos-action-recognition-review.

[36] "DeepMind," 2017. [Online]. Available: https://deepmind.com/research/open-source/kinetics.

[37] A. Karpathy, G. Toderici and S. Shetty, Large-scale Video Classification with Convolutional Neural Networks, 2014.

[38] K. Simonyan and A. Zisserman, Two-Stream Convolutional Networks for Action Recognition in Videos, 2014.

[39] Z. Lan, Y. Zhu and A. G. Hauptmann, Deep Local Video Feature for Action Recognition, 2017.

[40] J. Donahue, L. A. Hendricks, M. Rohrbach and S. Venugopalan, Long-term Recurrent Convolutional Networks forVisual Recognition and Description, 2016.

[41] D. Tran, L. Bourdev and R. Fergus, Learning Spatiotemporal Features with 3D Convolutional Networks, 2015.

[42] C. Feichtenhofer, A. Pinz and A. Zisserman, Convolutional Two-Stream Network Fusion for Video Action Recognition, 2016.

[43] C. Feichtenhofer, H. Fan, J. Malik and K. He, SlowFast Networks for Video Recognition, 2019.

[44] Y. B. a. A. C. Ian Goodfellow, Deep learning book; Machine Learning Basics, 2016.

[45] R. Khandelwal, "Medium," 2019. [Online]. Available: https://medium.com/datadriveninvestor/overview-of-different-optimizers-for-neural-networks-e0ed119440c3.

[46] S. Ruder, "Sebastian Ruder," 2016. [Online]. Available: https://ruder.io/optimizing-gradient-descent/.

[47] L. N. Smith, Cyclical Learning Rates for Training Neural Networks, 2017.

[48] Sanjay.M, "Towards Data Science," 2018. [Online]. Available: https://towardsdatascience.com/why-and-how-to-cross-validate-a-model-d6424b45261f.

[49] "NVIDIA Developer," [Online]. Available: https://developer.nvidia.com/cuda-gpus.

[50] J. Hale, "Kaggle," 2018. [Online]. Available: https://www.kaggle.com/discdiver/best-values-in-deep-learning-cloud-providers/.

[51] "Cambridge Spark," 2018. [Online]. Available: https://blog.cambridgespark.com/50-free-machine-learning-datasets-image-datasets-241852b03b49.

[52] P. University, "IMAGENET," [Online]. Available: http://image-net.org/explore.

[53] A. Miech, "Video Dataset Overview," 2019. [Online]. Available: https://www.di.ens.fr/~miech/datasetviz/.

[54] "GitHub," [Online]. Available: https://github.com/jinwchoi/awesome-action-recognition#action-recognition-and-video-understanding.

[55] A. NG, "YouTube," 2018. [Online]. Available: https://www.youtube.com/watch?v=wjqaz6m42wU.

[56] D. (. Sarkar, "Towards Data Science," 2018. [Online]. Available: https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a.

[58] J. a. F. A. Redmon, "YOLOv3: An Incremental Improvement," 2018. [Online]. Available: https://pjreddie.com/darknet/yolo/.

[59] "University of Central Florida," [Online]. Available: https://www.crcv.ucf.edu/data/UCF101.

[60] I. Loshchilov and F. Hutter, Decoupled Weight Decay Regularization, 2019.

[61] N. Mohamed and J. Al-Jaroodi, "Leveraging the Capabilities of Industry 4.0 for Improving Energy Efficiency in Smart Factories," 2019. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8632912.

[62] K. Xu, J. L. Ba and R. Kiros, Show, Attend and Tell: Neural Image Caption.

[63] K. S. a. A. Zisserman, Very deep convolutional networks for large-scale image recognition, 2015.

[64] J. Wang and Y. Ma, "Deep Learning for Smart Manufacturing: Methods and Applications," 2018.