CRANFIELD UNIVERSITY

Deep Learning Approaches for Visual Recognition and Object Detection

Student :
Jean Eudes KONAIN
Student No: 451122
Department : Aerospace
Word count : 2565

Lecturers:
Dr Yang XING
Mr Paris CHATZITHANOS
Dr Mariusz WISNIEWSKI

December 11, 2024

Table of Contents

1 Introduction

In recent years, artificial intelligence has revolutionised many sectors thanks to major advances, particularly in the field of computer vision [1]. According to Gaudenz Boesch [2], "*The field of computer vision includes a set of main problems such as image classification, localization, image segmentation, and object detection*"**.** With ever-increasing accuracy, image classification and object detection play a crucial role in various applications, such as facial recognition, security systems, autonomous driving, surveillance and human-computer interaction [2].

In this project, we will explore these two fundamental areas by focusing on image classification using the renowned CIFAR-10 dataset, and on object detection through learning transfer.

2 Task 1: Image Classification on CIFAR-10 Dataset

2.1 Models Training for CIFAR-10 Image Recognition

2.1.1 Fully Connected Layers

Architecture

The architecture of the Fully Connected (FC) model that I have implemented is based on a sequence of dense layers. The model starts with an input layer defining the dimensions of the expected images (32x32x3). This data is then flattened via a *Flatten* layer, allowing the input data to be transformed into a one-dimensional vector compatible with subsequent Dense layers [3].

The model comprises three fully connected layers, with 1024, 512, and 128 neurons, respectively, using the ReLU activation function. This gradual reduction in the number of neurons makes it possible to capture complex features, refine them and focus on relevant information. Finally, a Dense 10-neuron output layer, activated by the Softmax activation function, allows predictions to be made for the 10 categories of the CIFAR-10 dataset.

$$Input \rightarrow \{Flatten\} \rightarrow \{FC_i \rightarrow ReLU_i\}_{i=1}^{3} \rightarrow \{FC \rightarrow Softmax\}$$

Model size and parameters

Applying the formulas seen in class, the size of the model is given in the following table:

Layer	Type	# Input units	# Output units	# Connections	# Weights
Dense	Fully connected	3072	1024	3 145 728	3 145 728
Dense_1	Fully connected	1024	512	524 288	524 288
Dense_2	Fully connected	512	128	65 536	65 536
Dense_3	Fully connected	128	10	1280	1280

Table 1 : Table showing the size of the model in the case of a FC

The figure below shows the number of parameters per layer. This number of parameters depends on the weights and biases. The total number of trainable parameters is the sum of each parameter per layer.

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 3072)	0
dense (Dense)	(None, 1024)	3,146,752
dense_1 (Dense)	(None, 512)	524,800
dense_2 (Dense)	(None, 128)	65,664
dense_3 (Dense)	(None, 10)	1,290

```
Total params: 11,215,520 (42.78 MB)
Trainable params: 3,738,506 (14.26 MB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 7,477,014 (28.52 MB)
```

Figure 1: Number of parameters per layer in the case of an FC

Hyperparameters

The hyperparameters chosen for this model are:

- **Optimizer** : Adam with a learning rate of 0.001
- **Loss Function** : Categorical Cross entropy
- **Assessment metric** : Categorical Accuracy
- **Batch size** : 128
- **Number of eras** : 30

2.1.2 Convolutional Neural Networks

Architecture

The architecture of the Convolutional Network Model (CNN) that I implemented is composed of 3 convolutional blocks with 2D convolutions. The first block applies a convolution with 96 size filters (3x3) and using the ReLU activation function. This layer is followed by a MaxPooling operation (2x2) with "same" padding and stride equal to 2. The second block uses 64 size filters (3x3) and the ReLU activation function. This block is followed by a MaxPooling layer (2x2) with "valid" padding and a stride of 2. The last block uses 32 size filters (3x3, a ReLU activation function and is followed by a MaxPooling (2x2) with stride 1 and "same" padding.

These extracted layers are flattened to be passed through a dense layer of 64 neurons using the ReLU function before the dense output layer of 10 neurons, using the Softmax function to classify the images.

$$Input \rightarrow \{ConvL_i \rightarrow ReLU_i \rightarrow MaxPooling_i\}_{i=1}^{3} \rightarrow \{Flatten\} \rightarrow \{FC \rightarrow ReLU\} \rightarrow \{FC \rightarrow Softmax\}$$

Model size and parameters

Applying the formulas seen in class, the size of the model is given in the following table:

Layer	Type	# Units	# Connections	# Weights
Conv2d	Convolution	86 400	2 332 800	2592
Max_pooling2d	Pooling	21 600	86 400	0
Conv2d_1	Convolution	10 816	9 345 024	55 296
Max_pooling2d_1	Pooling	2 304	10 816	0
Conv2d_2	Convolution	128	73 728	18 432
Max_pooling2d_2	Pooling	128	128	0
Dense_8	Fully connected	64	8 192	8 192
Dense_9	Fully connected	10	640	640

Table 2: Table showing the size of the model in the case of a CNN

The figure below shows the number of parameters per layer. This number of parameters depends on the weights and biases. The total number of trainable parameters is the sum of each parameter per layer.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 30, 96)	2,688
max_pooling2d (MaxPooling2D)	(None, 15, 15, 96)	0
conv2d_1 (Conv2D)	(None, 13, 13, 64)	55,360
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_2 (Conv2D)	(None, 2, 2, 32)	18,464
max_pooling2d_2 (MaxPooling2D)	(None, 2, 2, 32)	0
flatten_2 (Flatten)	(None, 128)	0
dense_8 (Dense)	(None, 64)	8,256
dense_9 (Dense)	(None, 10)	650

Total params: 256,256 (1001.00 KB)
Trainable params: 85,418 (333.66 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 170,838 (667.34 KB)

Figure 2: Number of parameters per layer in the case of a CNN

Hyperparameters

The hyperparameters chosen for this model are:

- **Optimizer** : Adam with a learning rate of 0.001
- **Loss Function** : Categorical Cross entropy
- **Assessment metric** : Categorical Accuracy
- **Batch size** : 128
- **Number of eras** : 30

2.2 From Model Training to Performance Monitoring

2.2.1 Performance Analysis : Fully Connected layers and CNN

In order to evaluate and improve the performance of the different models (FCs and CNNs), I separated the CIFAR-10 data into training data and validation data. This step is essential because you should never evaluate the performance of a model on the data used to train it.

Figure 3 below shows the actual distribution. Of the 60,000 images in the CIFAR-10 dataset, 10,000 are used for the test set. Of the remaining 50,000 images, 80% of the data was randomly divided into a training set and 20% into a validation set. Then, the data were normalized to values between 0 and 1.

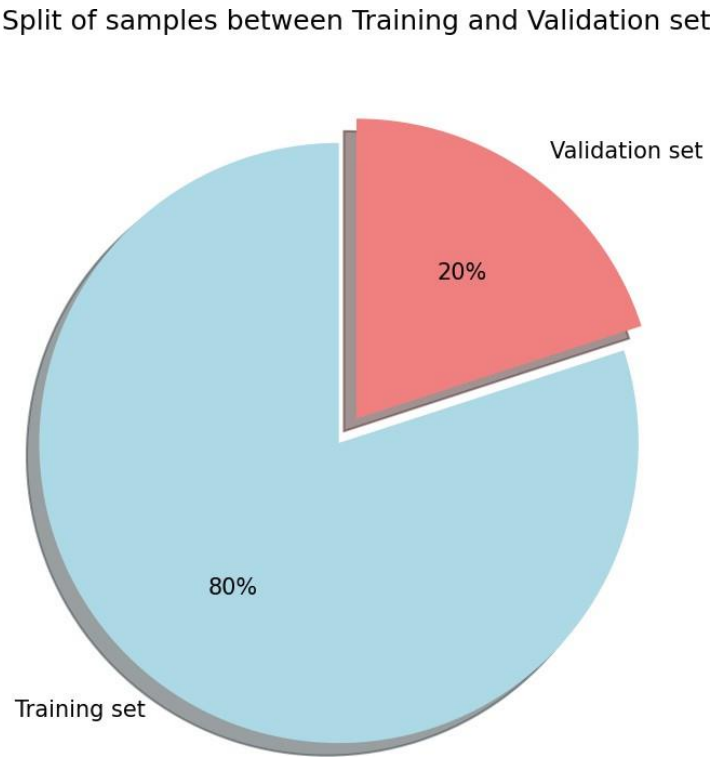

Figure 3: Split of sample between Training and validation set

During training, I used the Categorical Cross entropy loss function [4], suitable for multi-class classification, as it strongly penalizes incorrect predictions by comparing predicted probabilities to real labels. To improve the performance of the different models, my fine-tuning strategy was as follows:

Fully Connected Cases

I increased the number of epochs from 10 to 30, which allowed an increase in the accuracy of the validation data from 47% to 50%. However, this increase has led to an overfitting problem (Figure 4), where the model overfits training data, compromising its ability to generalize. To address this problem, I introduced regularization techniques, including dropouts and L2 penalties (Figure 5). The results are recorded in the table below.

	Before fine-tuning	**After fine-tuning**
Validation loss	1,650	1,399
Validation accuracy	50,33 %	52,60%

Table 3: Table showing the loss and accuracy on the validation data

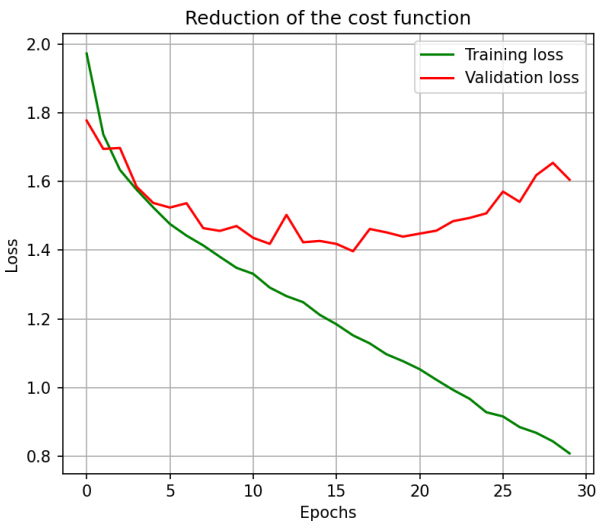

Figure 4: Reduction of the cost function before fine-tuning

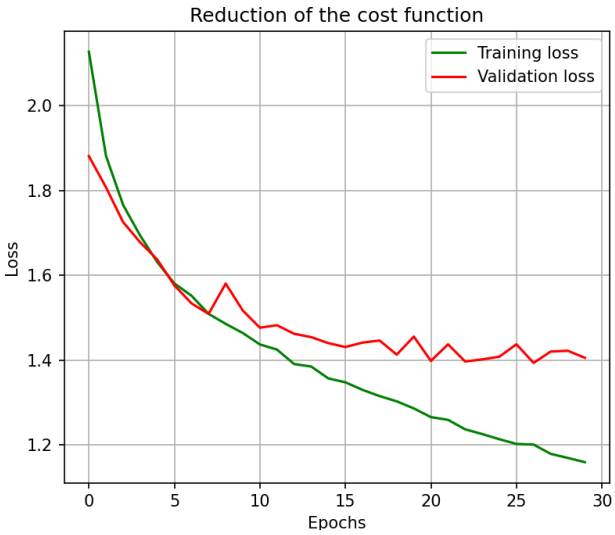

Figure 5: Reduction of the cost function after fine-tuning

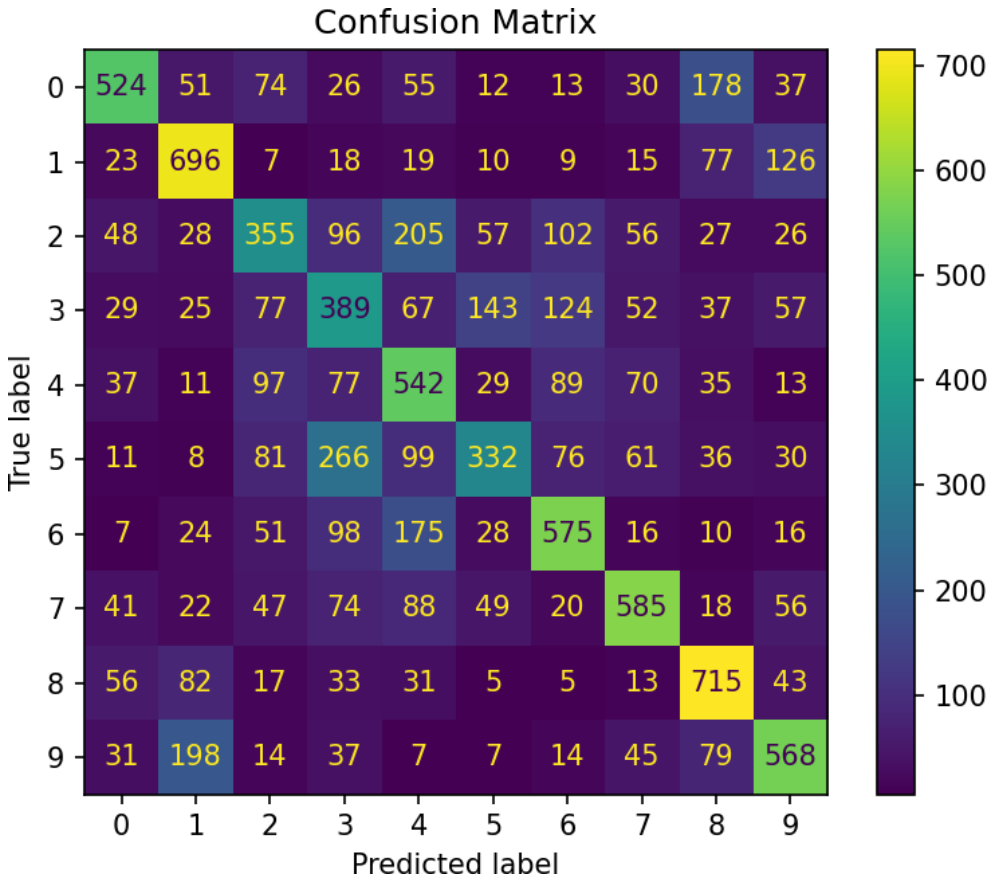

Figure 6: Confusion matrix for FC model

	Precision	Recall	F1-score
Label 0	0,6493	0,5240	0,5800
Label 1	0,6079	0,6960	0,6490
Label 2	0,4329	0,3550	0,3901
Label 3	0,3492	0,3890	0,3680
Label 4	0,4208	0,5420	0,4738
Label 5	0,4940	0,3320	0,3971
Label 6	0,5599	0,5750	0,5673
Label 7	0,6204	0,5850	0,6022
Label 8	0,5899	0,7150	0,6465
Label 9	0,5844	0,5680	0,5761

Table 4: Table showing Precision, Recall and F1-score

In terms of overall accuracy, the model obtains an accuracy score of **52.81%.** However, detailed analysis of the confusion matrix reveals that the model has difficulty predicting some specific labels, including labels 2 (F1: 0.3901), 3 (F1: 0.3680), and 5 (F1: 0.3971). The F1-score reflects the balance between accuracy (accuracy of predictions) and recall (detection of real cases) [5]. On the other hand, the model correctly predicts labels 1 (F1: 0.6490) and 8 (F1: 0.6465).

Convolutional Neural Network Case

I increased the number of epochs from 10 to 30, which increased the accuracy of the validation data from 61% to 67%. To avoid the overfitting caused by this increase (Figure 8), I added regularizations (dropouts, L2 penalty) and I modified the learning rate. Despite this, performance has not improved. As a last resort, I modified the architecture of the model by removing some pooling layers and adding dense layers and regularizations (Figure 7).

$$Input \rightarrow \{ConvL_i \rightarrow ReLU_i\}_{i=1}^{3} \rightarrow \{AveragePooling2D\} \rightarrow \{Flatten\} \rightarrow \{FC_i \rightarrow ReLU_i\}_{i=1}^{2} \rightarrow \{FC \rightarrow Softmax\}$$

The results are recorded in the table below.

	Before fine-tuning	After fine-tuning
Validation loss	1,01	0,83
Validation accuracy	67,19 %	72,44%

Table 5: Table showing the loss and accuracy on the validation data

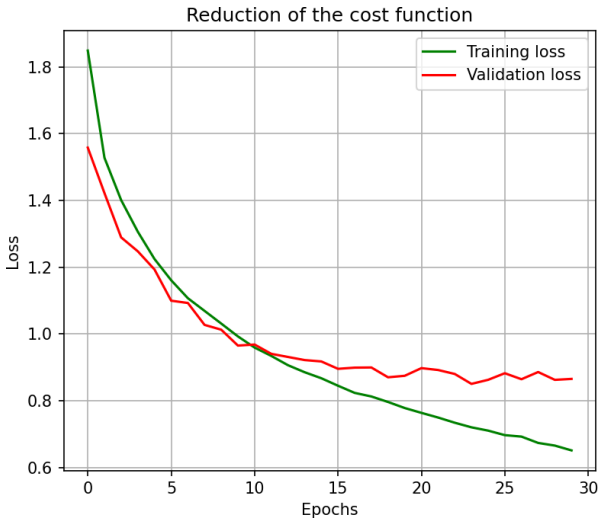

Figure 7: Reduction of the cost function after fine-tuning

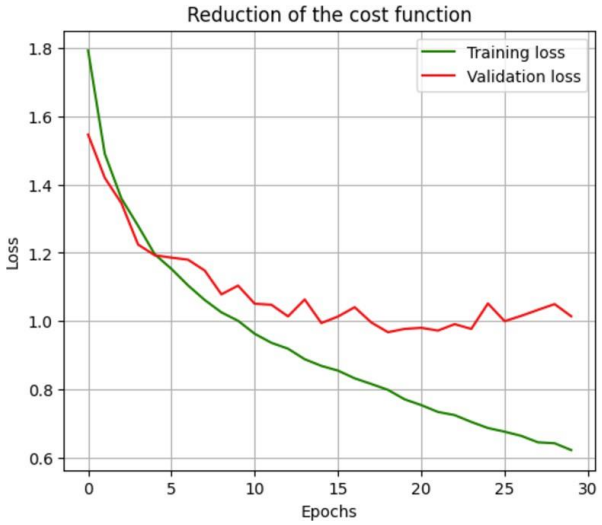

Figure 8: Reduction of the cost before fine-tuning function

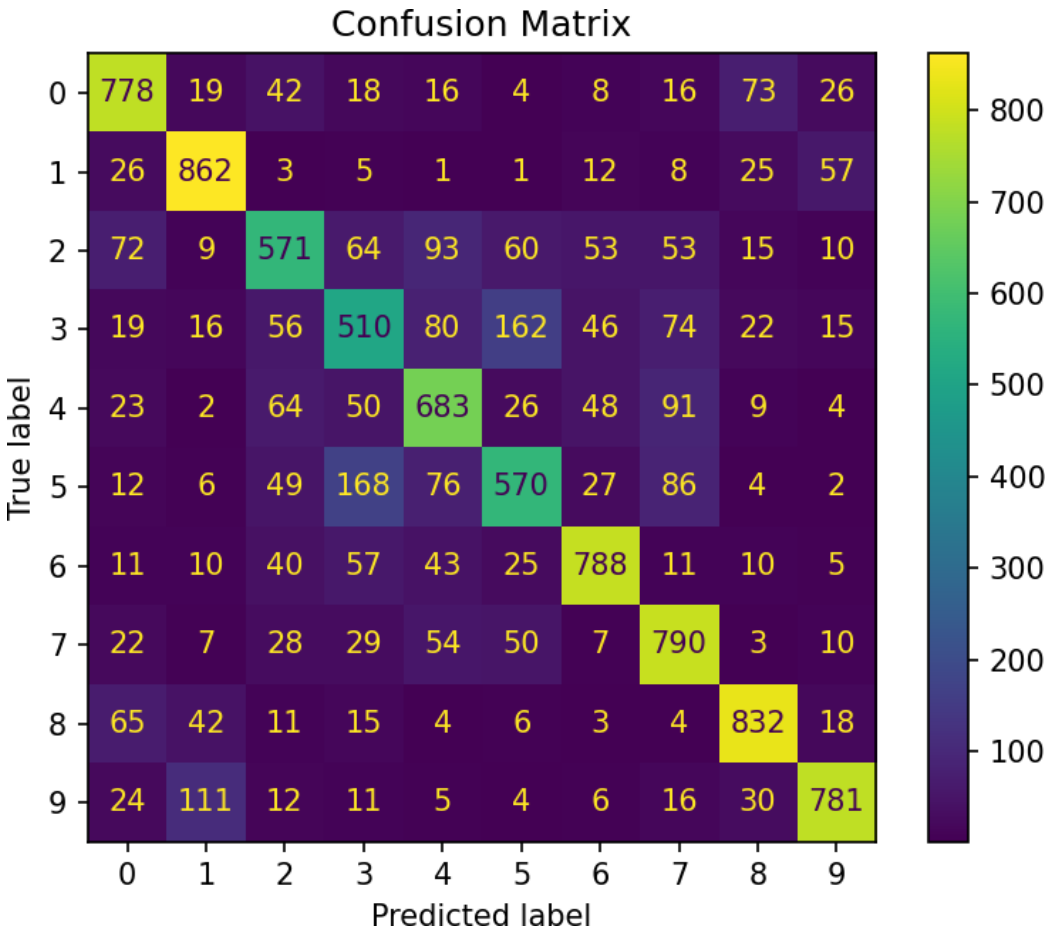

Figure 9: Confusion matrix for CNN model

	Precision	Recall	F1-score
Label 0	0,7395	0,7780	0,7583
Label 1	0,7952	0,8620	0,8273
Label 2	0,6518	0,5710	0,6087
Label 3	0,5502	0,5100	0,5293
Label 4	0,6474	0,6830	0,6647
Label 5	0,6278	0,5700	0,5975
Label 6	0,7896	0,7880	0,7888
Label 7	0,6876	0,7900	0,7352
Label 8	0,8133	0,8320	0,8225
Label 9	0,8416	0,7810	0,8102

Table 6: Table showing Precision, Recall and F1-score

The model achieves an overall accuracy with an accuracy score of **71.65%.** Based on the confusion matrix (Figure 9), the model demonstrates generally correct but uneven performances according to the labels. The best predicted classes are label 1 (F1: 0.8273), 8 (F1: 0.8225), and 9 (F1: 0.8102). Conversely, labels 3 (F1: 0.5293) and 5 (F1: 0.5975) have lower scores. This is explained by a slightly higher dispersion of predictions over several other classes.

2.2.2 Comparison between Fully Connected and CNN Classifiers

Fully-connected networks (FC) and convolutional networks (CNNs) differ fundamentally in their approach, architecture, and performance for image classification. FCs connect each neuron to all input pixels, generating a large number of weights [6]. In the case of CIFAR-10, a 32x32x3 image produces 3072 weights for a single neuron, which leads to a high risk of overfitting. These limitations result in an overall accuracy limited to 52.81% and low F1 scores for some labels, in particular 2 (0.3901) and 3 (0.3680), illustrating difficulties in capturing the spatial dependencies of the different images. On the other hand, CNNs use convolution and pooling layers to extract local features while drastically reducing the number of parameters [6] thus offering a better generalization (accuracy 71.65%, high F1 for labels 1 : 0.8273, 8 : 0.8225, 9 : 0.8102).

2.3 Transfer Learning with Pre-trained Networks

2.3.1 Selection of Pre-trained Model

Among the wide range of pre-trained models (ResNet, AlexNet, VGG, etc.), I chose MobileNet. Compared to other models, MobileNet offers an excellent balance between lightness, speed and accuracy, ideal for resource-constrained systems and especially on small data such as CIFAR-10 (Figure 10). Indeed, according to Wei Wang et al. [7], "*MobileNet is a lightweight network, which uses depthwise separable convolution to deepen the network, and reduce parameters and computation. At the same time, the classification accuracy of MobileNet on ImageNet data set only reduces by 1%*."

Network	Depth	Size	Parameters (Millions)	Image Input Size
alexnet	8	227 MB	61.0	227-by-227
vgg16	16	515 MB	138	224-by-224
vgg19	19	535 MB	144	224-by-224
squeezenet	18	4.6 MB	1.24	227-by-227
googlenet	22	27 MB	7.0	224-by-224
inceptionv3	48	89 MB	23.9	299-by-299
densenet201	201	77 MB	20.0	224-by-224
mobilenetv2	53	13 MB	3.5	224-by-224
resnet18	18	44 MB	11.7	224-by-224
resnet50	50	96 MB	25.6	224-by-224
resnet101	101	167 MB	44.6	224-by-224
xception	71	85 MB	22.9	299-by-299
inceptionresnetv2	164	209 MB	55.9	299-by-299
shufflenet	50	6.3 MB	1.4	224-by-224
nasnetmobile	*	20 MB	5.3	224-by-224
nasnetlarge	*	360 MB	88.9	331-by-331

Figure 10: Image showing the different pre-trained models and their parameters. This image is from the Deep Learning For Computer Vision course delivered by Cranfield University.

2.3.2 Visualization of Learned Filters and Feature Maps

The architecture of MobileNet lies in the use of *"depthwise separable convolutions*" or *"depthwise separable filters " "to construct lightweight deep convolutional neural networks "* [7] (Figure 11).

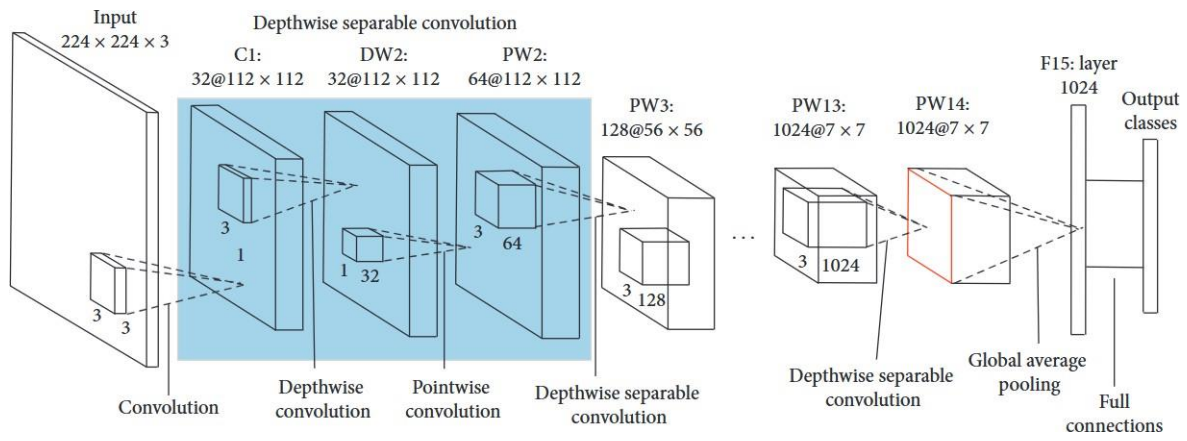

Figure 11: "Architecture of MobileNet" [7]

To adapt MobileNet to my project, I modified the architecture by excluding the fully connected layers present at the end of the model. I added an average pooling layer to reduce the size of the tensor from (7x7x1280) to (2x2x1280). Then, I added 3 dense layers, with 128, 64, and 10 neurons respectively. The first two layers use the ReLU activation function and the last Softmax.

$$MobileNet \rightarrow \{AveragePooling2D\} \rightarrow \{Flatten\} \rightarrow \{FC_i \rightarrow ReLU_i\}_{i=1}^{2} \rightarrow \{FC \rightarrow Softmax\}$$

After training the MobileNet pre-trained model on the CIFAR-10 data, I displayed the learned filters and feature maps of the first convolution layer 'Conv1'.

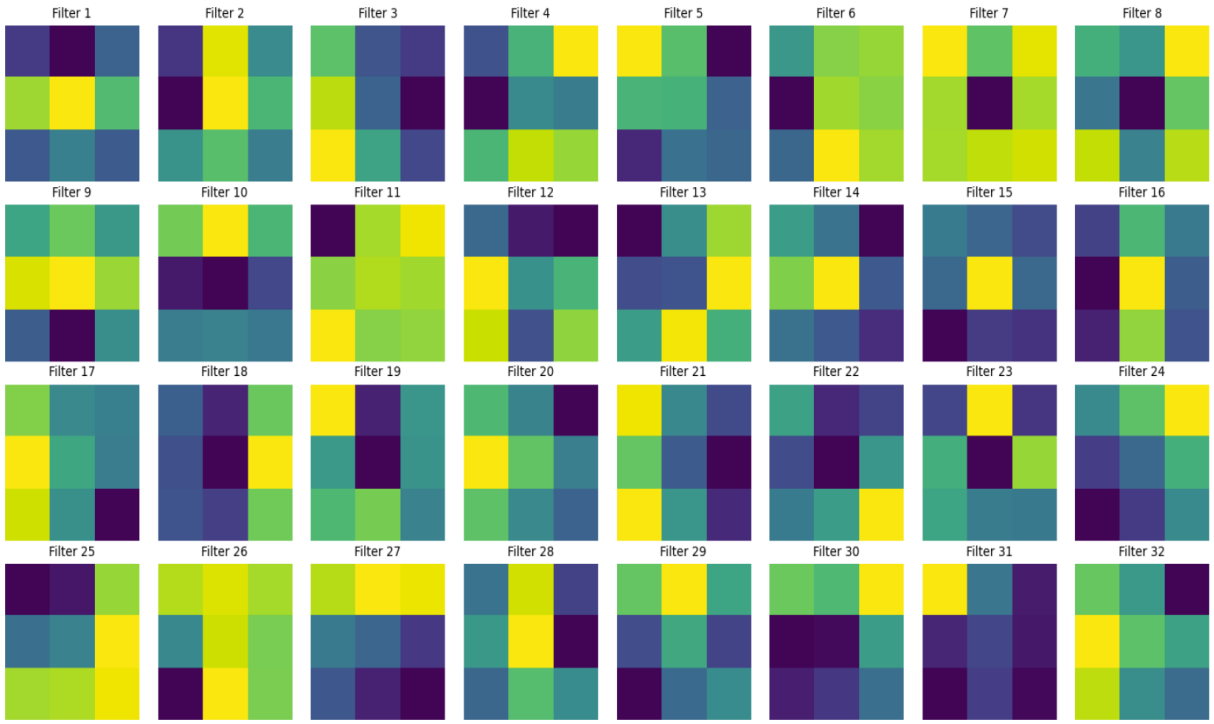

Figure 12: Visualization of Conv1 layer filters

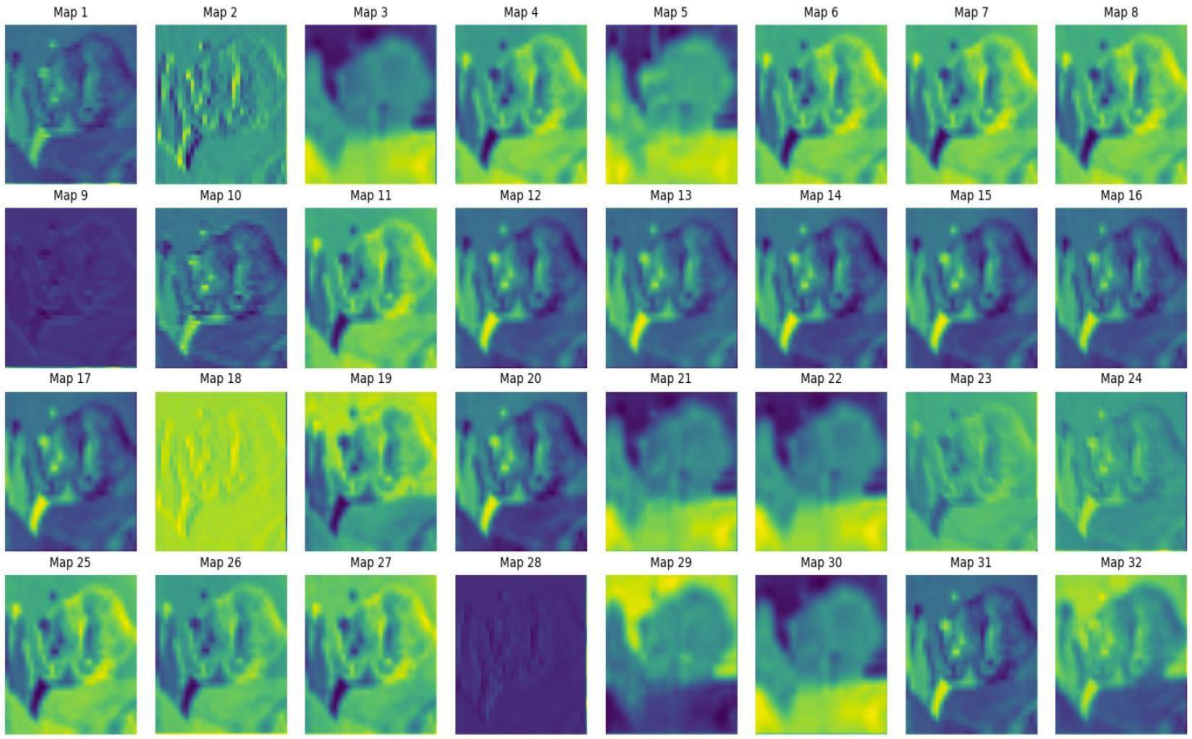

Figure 13: Visualization of Conv1 feature maps

2.3.3 Fine-Tuning Strategy

At the end of the training phase, I obtained an accuracy of **81.36%** on the validation data (Table 7). However, the model was overlearned and did not generalize perfectly to new data (Figure 14).

To solve this problem, my fine-tuning strategy consisted of first unfreezing the last convolution layers of the MobileNet and training them to evaluate their impact on the model's performance. However, the results obtained have been catastrophic. So I decided to freeze these layers again and change the value of the batch size. According to Pavlo M. Radiuk [8], "*the batch size parameter has a crucial effect on the accuracy of image recognition. The greater the parameter value, the higher the image recognition accuracy*". In a second phase, I added regularization techniques, in particular dropouts and L2 penalties (Figure 15). Due to a fairly long compilation time, I opted for a fairly low number of epochs (epochs = 5) to get results quickly, hence the low improvement before and after fine-tuning. The results are recorded in the table below.

	Front fine-tuning	**After fine-tuning**
Validation loss	0,60	0,57
Validation accuracy	81,36%	82,25%

Table 7: Table showing the loss and accuracy on the validation data

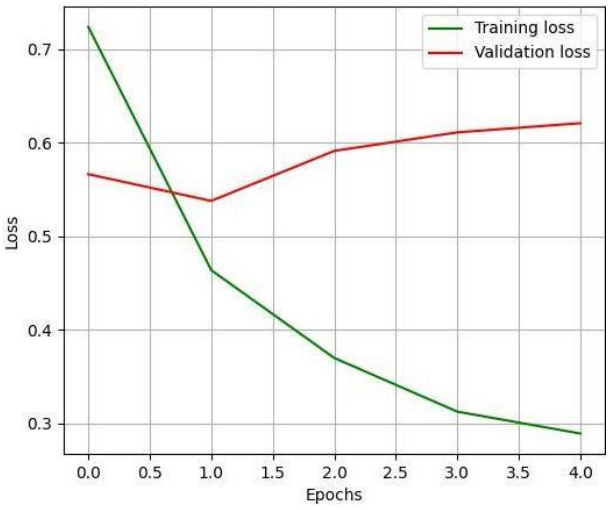

Figure 14: Reduction of the cost before fine-tuning function

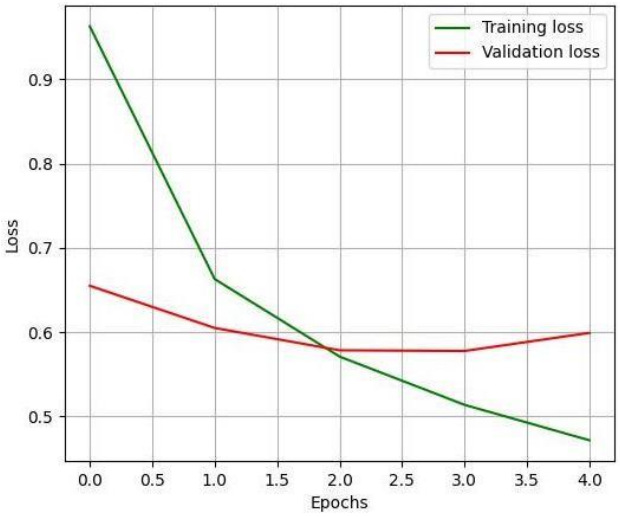

Figure 15: Reduction of the cost function after fine-tuning

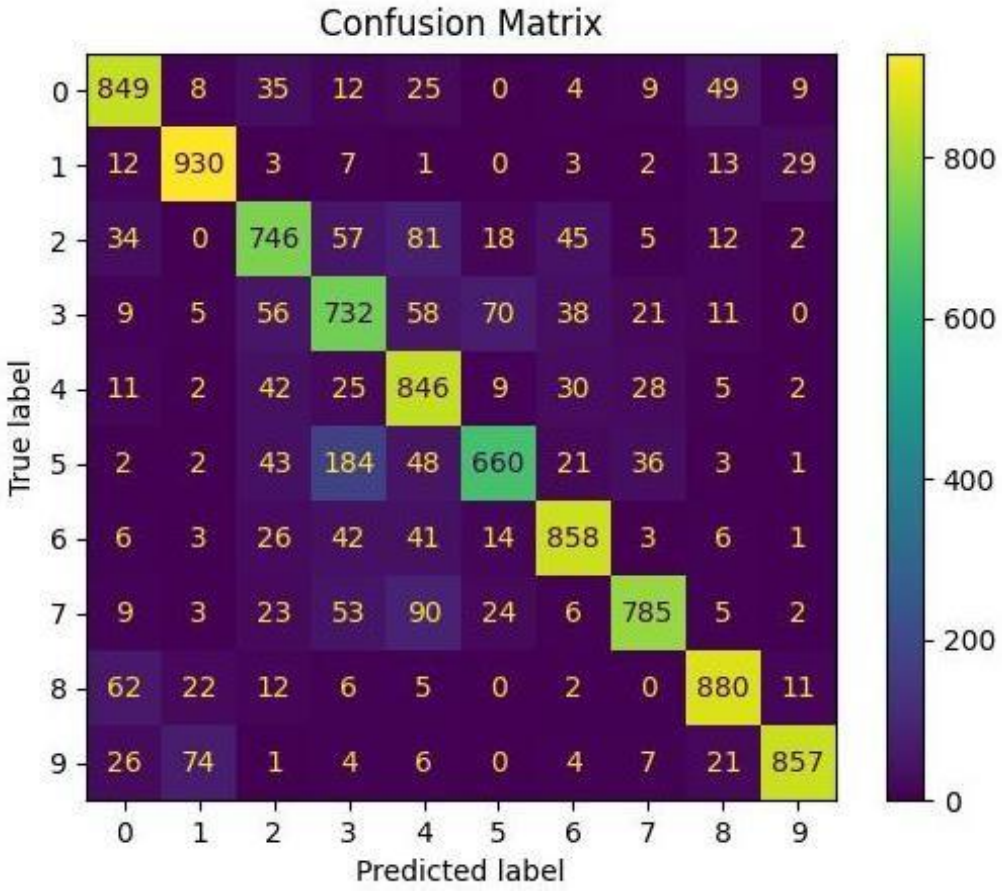

Figure 16: Confusion matrix for pre-trained model

	Precision	Recall	F1-score
Label 0	0.8324	0.8490	0.8406
Label 1	0.8866	0.9300	0.9078
Label 2	0.7558	0.7460	0.7509
Label 3	0.6524	0.7320	0.6899
Label 4	0.7044	0.8460	0.7687
Label 5	0.8302	0.6600	0.7354
Label 6	0.8487	0.8580	0.8533
Label 7	0.8761	0.7850	0.8281
Label 8	0.8756	0.8800	0.8778
Label 9	0.9376	0.8570	0.8955

Table 8: Table showing Accuracy, Recall, and F1-score

Overall, the model achieves an accuracy of **81.43%,** which is a good result compared to the FC and CNN models. In addition, it is also seen that the F1 scores are all above 68%. However, the model cannot correctly predict label 3 (F1: 0.6899). By analyzing the confusion matrix, we notice that the model makes particularly errors in the prediction of labels 3 and 5. This confusion could be due to similarities between the two labels.

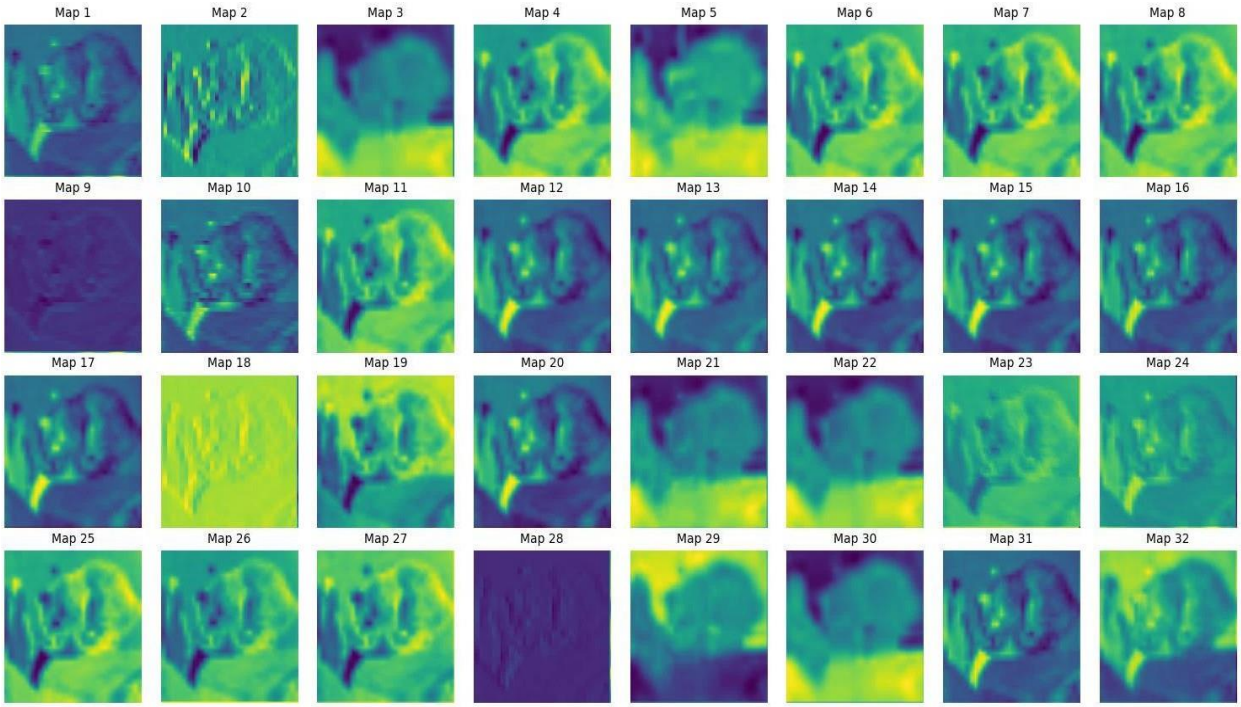

Figure 17: Visualization of Conv1 feature maps after fine-tuning

Comparing the two feature maps before and after fine-tuning, there is a slight change in brightness and contrast on the feature map obtained after fine-tuning. Perhaps this could mean that the model has improved its weights.

3 Task 2: UAV Object Detection Using Transfer Learning

3.1 Data Preparation

3.1.1 Sample Selection and Visualization (3x4 Subplots)

With the provided database, containing UAV images, I displayed 12 samples from the training and test sets, with their bounding boxes (bboxes). In this context, the label is represented in the form [x y width height] where (x, y) are the coordinates of the centre of the bounding box and ″width", ″height" are the width and height of the bounding box, respectively.

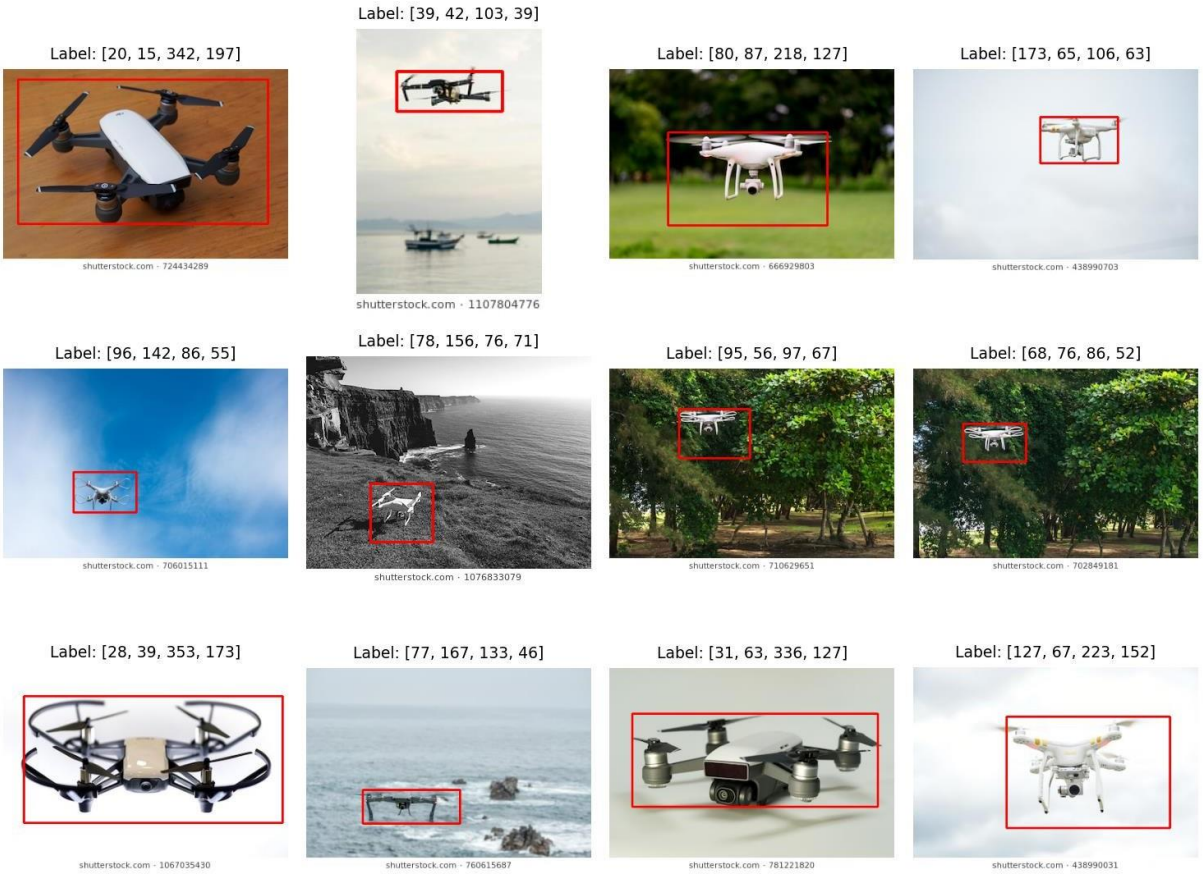

Figure 18 : Visualization of 12 images with their bounding boxes

3.1.2 Data Preprocessing

Before training the different models, I followed several steps of data preprocessing. First, I resized the images to the 224x224x3 format to ensure compatibility with the architectures of the models and adjusted their respective bounding boxes. Then, I normalized the data to have values between 0 and 1. I converted the images in JPG format into a python table, making

them easier to manipulate. After mixing the data, I separated it into 3 separate sets: a training set, a validation set, and a test set, according to the proportions specified in the figure below.

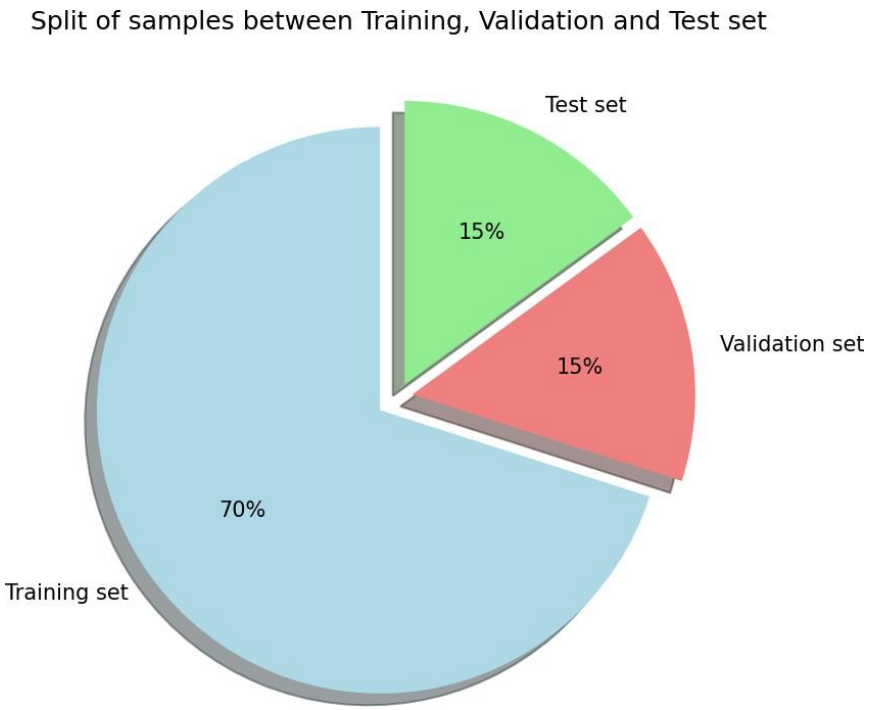

Figure 19: Split of samples between Training, Validation and Test set

3.2 Object Detection and Transfer Learning

3.2.1 Model Comparisons: YOLOv8 vs. SSD

YOLOv8

The architecture of YOLOv8 is based on 3 core blocks:

- Backbone
- Neck
- Head.

The backbone is *"the deep learning architecture", made up of several CNNs, "that acts as a feature extractor of the inputted image*" [10]. It is *"optimized for both speed and accuracy, incorporating depthwise separable convolutions or other efficient layers to minimize computational overhead while retaining representational power"* [9].

The neck *"combines the features acquired from the various layers of the Backbone module"* [10] or alternatively *"refines and fuses the multi-scale features extracted by the backbone"* [9]. The head *"predicts the classes and the bounding box of the objects which is the final output produced by the object detection model"* [10].

SSD (Single Short MultiBox Detection)

The SSD model architecture is built on the MobileNetV2 backbone, as the base network for feature extraction, which has been pre-trained on the ImageNet database without the final classification layers (Fully connected layer). Then, I introduced a succession of convolution layers and dense layers to predict bounding boxes. Finally, I applied *"a technique known as non-maximum suppression"* [11] to filter out redundant bounding boxes and to keep the best ones. "*This ensures only the most likely predictions are retained by the network, while the more noisier ones are removed*" [11].

YOLOv8 vs. SSD

In terms of architecture, YOLOv8 and SSD uses 3 core blocks (backbone, neck and head) but uses different neural network architectures for each block. While YOLOv8 uses architectures like "*depthwise separable convolutions"* [9] for backbone, SSD uses classic architectures like " *(VGG, ResNet, MobileNet) that extracts feature maps from the input image. These feature maps capture different levels of abstraction"* [12]. In addition, YOLO uses a single grid to simultaneously predict classes and boxes for all objects in one pass, while SSD leverages multiple layers for multiscalar predictions adapted to various object sizes [12].

Finally, YOLO and SSD share benefits such as the simplicity of their unified architecture making it easy to train and deploy. However, both have difficulties in detecting very small objects, as they are not well represented in high-level feature maps [12].

3.2.2 Transfer Learning Strategy and Evaluation

To evaluate the performance of these two models, I used, in the case of the SSD, metrics such as IoU which allows to compare predicted and true bounding boxes. It results from the division between the overlapping area of the two boxes and the area of their union [11].

$$IoU = \frac{Area\ of\ Overlap}{Area\ of\ Union}$$ [11]

In the case of YOLOv8, I used metrics like Precision, Recall, mAP50, and mAP50-95. The results are recorded in the table below.

Precision	**Recall**	**mAP50**	**mAP50-95**
0.943	0.83	0.92	0.489

Table 8: Results Achieved for YOLOv8

These results show that the YOLOv8 model correctly predicts bounding boxes with an accuracy of 94.3%. This means that of all the model's predictions, 94.3% are correct, with few false positives, demonstrating a reliable ability to locate and identify detected objects.

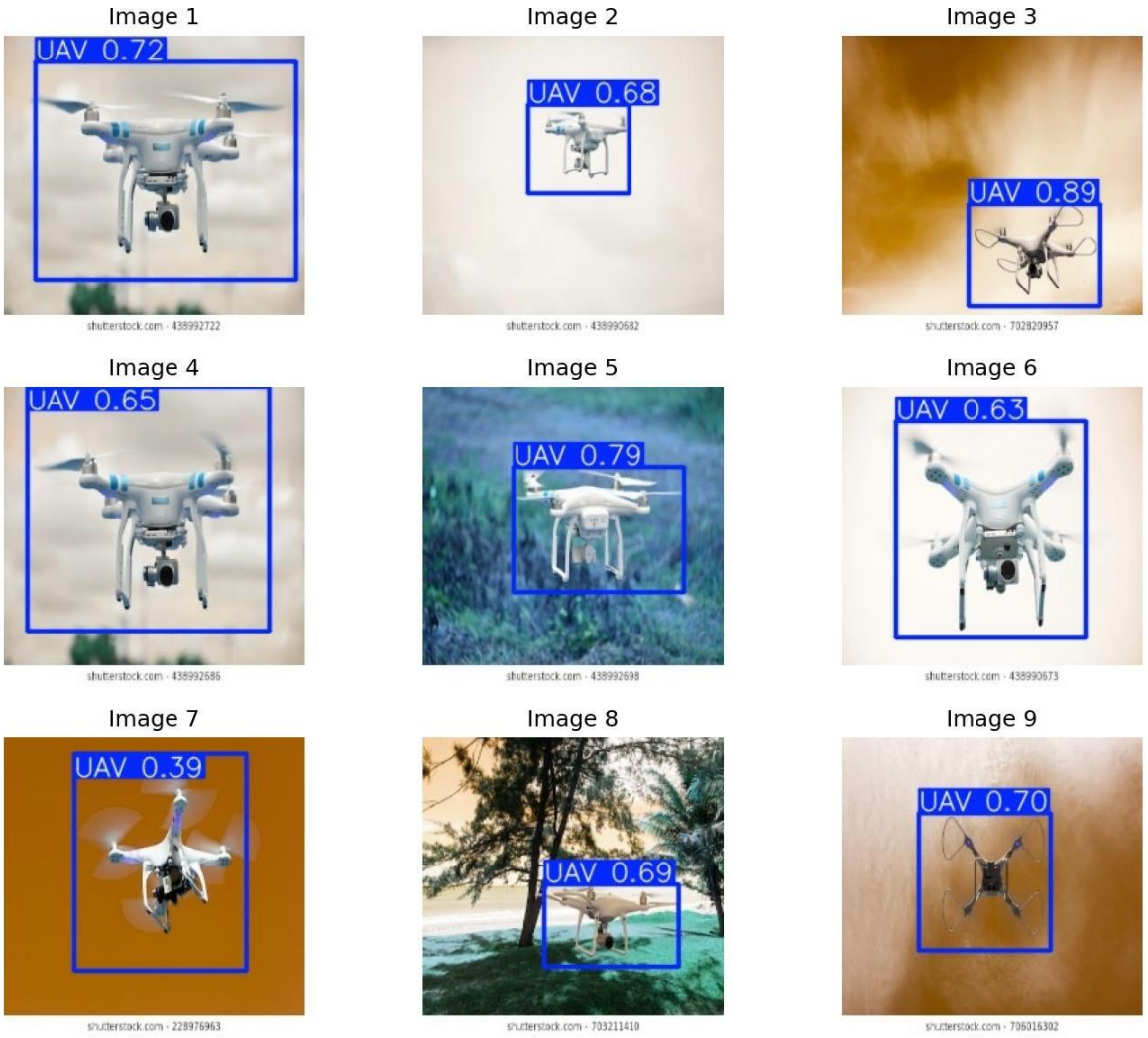

Figure 20: Prediction of bounding boxes in the case of YOLOv8

In the case of the SDD model, I obtained an IoU score of 0.31. Overall, the model manages to predict bounding boxes fairly well, but it does make some errors. This indicates that the model is not yet capable of predicting the location of objects in the image with a high degree of accuracy.

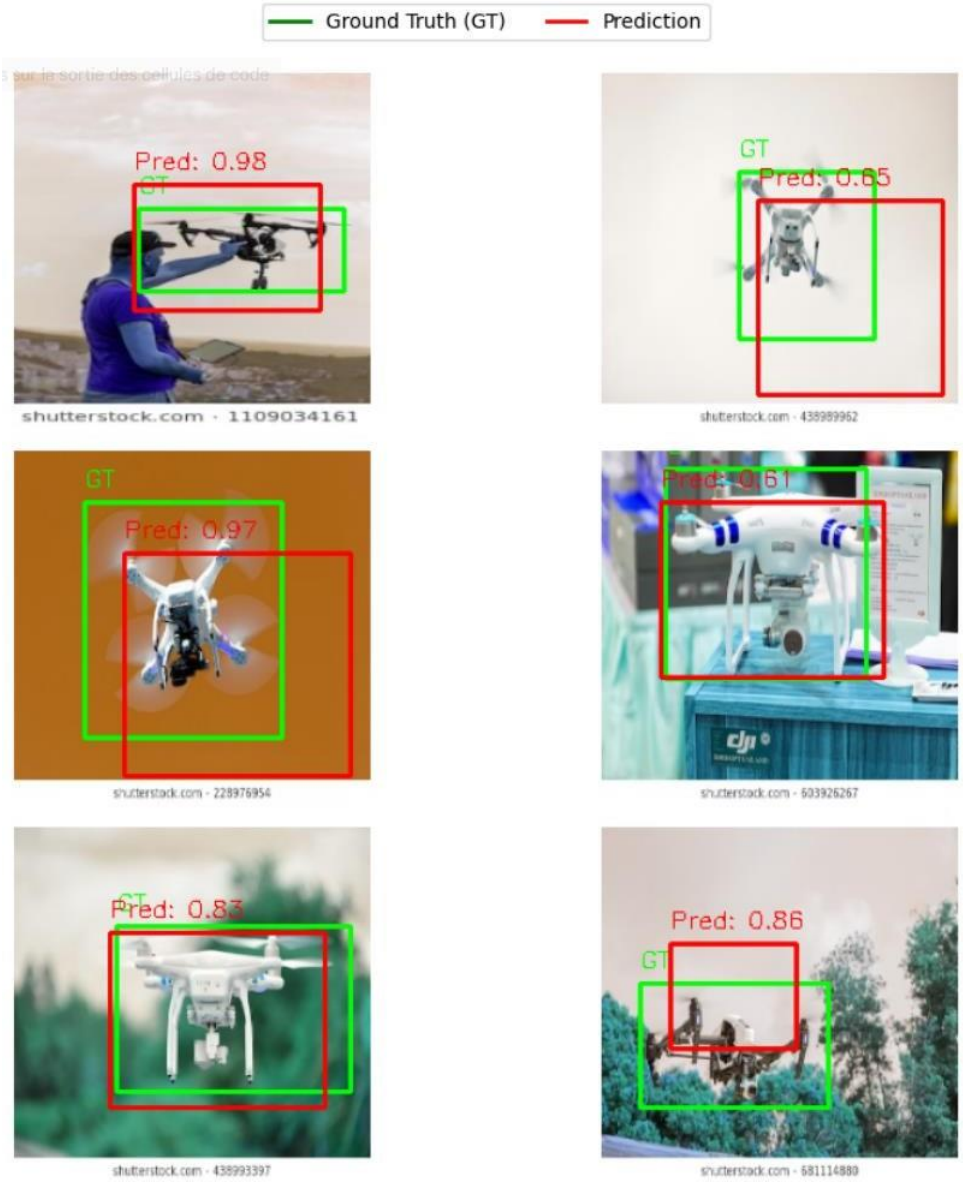

Figure 21: Prediction of bounding boxes in the SSD case

4 Conclusion

To sum up, the purpose of this project was to explore the key areas of Deep Learning for computer vision, namely image classification and object detection. The first part of this project focused on image classification by comparing the Fully Connected (FC) and Convolutional Neural Network (CNN) models, and by using a pre-trained model. According to Gaudenz Boesch [2], "*the use of convolutional neural networks (CNN) is the state-of-the-art method for image classification*". The reason for this is that the CNN model, with an accuracy of 71.65%, significantly outperforms the FC model, which achieves only 52.81%.

Finally, in the second part of the project, transfer learning techniques for object detection were implemented, in particular using the YOLOv8 and SSD models, in order to accurately detect the position of UAVs in an image.

5 References

[1] Adib Bin Rashid, MD Ashfakul Karim Kausik, AI revolutionizing industries worldwide: A comprehensive overview of its diverse applications, Hybrid Advances[Internet]; 2024 [cited 2024-12-10] Volume 7, Available from https://doi.org/10.1016/j.hybadv.2024.100277

[2] Gaudenz Boesch. A Complete Guide to Image Classification in 2025 [Internet]. Viso.ai; 2024 October 18 [cited 2024-12-10] Available from https://viso.ai/computer-vision/image-classification/

[3] Prudhviraju Srivatsavaya. Flatten Layer — Implementation, Advantage and Disadvantages[Internet], Medium; 2023 October 04 [cited 2024-12-10] Available from https://medium.com/@prudhviraju.srivatsavaya/flatten-layer-implementation-advantage-and-disadvantages-0f8c4ecf5ac5

[4] Maxim Sorokin, Categorical Cross-Entropy: Unravelling its Potentials in Multi-Class Classification[Internet], Medium; 2024 January 17 [cited 2024-12-10] Available from https://medium.com/@vergotten/categorical-cross-entropy-unraveling-its-potentials-in-multi-class-classification-705129594a01

[5] Ellie Frank, Understanding the F1 Score [Internet], Medium; 2023 April 29 [cited 2024-12-10] Available from https://ellielfrank.medium.com/understanding-the-f1-score-55371416fbe1

[6] Pooja Mahajan. Fully Connected vs Convolutional Neural Networks[Internet], The Startup; 2020 October 23 [cited 2024-12-10] Available from: https://medium.com/swlh/fully-connected-vs-convolutional-neural-networks-813ca7bc6ee5

[7] Wang, Wei, Li, Yutao, Zou, Ting, Wang, Xin, You, Jieyu, Luo, Yanhong. A Novel Image Classification Approach via Dense-MobileNet Models. Mobile Information Systems[Internet], 2020 [cited 2024-12-10]; Volume 2020, 8 pages. Available from https://doi.org/10.1155/2020/7602384

[8] Pavlo M. Radiuk. Impact of Training Set Batch Size on the Performance of Convolutional Neural Networks for Diverse Datasets. Information Technology and Management Science[Internet]; 2017 [cited 2024-12-10], Volume 20, Issue 1, 5 pages Available from https://doi.org/10.1515/itms-2017-0003.

[9] Muhammad Yaseen. What is YOLOv8: an in-depth exploration of the internal features of the next-generation object detector. arXiv [Internet]; 2024 [cited 2024-12-10], 10 pages Available from https://arxiv.org/pdf/2408.15857v1

[10] Abin Timilsina, YOLOv8 Architecture Explained ! [Internet], Medium; 2024 March 17 [cited 2024-12-10]. Available from https://abintimilsina.medium.com/yolov8-architecture-explained-a5e90a560ce5

[11] Eddie Forson. Understanding SSD MultiBox — Real-Time Object Detection in Deep Learning [Internet], Towards Data Science; 2017 November 11[cited 2024-12-10] Available

from: https://towardsdatascience.com/understanding-ssd-multibox-real-time-object-detection-in-deep-learning-495ef744fab

[12] Nikita Malviya. Comparative Analysis of YOLO and SSD [Internet], Medium; 2023 August 06 [cited 2024-12-10] Available from: https://medium.com/@nikitamalviya/comparative-analysis-of-yolo-and-ssd-7433e249d429