ECOLE
POLYTECHNIQUE
DE BRUXELLES

Pattern Recognition and Image Analysis

CIFAR-3 : Image Classification

Matteo Silvestri

Professors Christine Decaestecker, Olivier Debeir

2024

Table des matières

1 Introduction

Image classification is a fundamental task in computer vision, essential for enhancing applications ranging from digital surveillance to multimedia retrieval systems. This laboratory report explores image classification techniques using a subset of the **CIFAR-10** dataset, a widely recognized benchmark in the evolution of deep learning and image processing. This dataset consists of 60,000 32x32 color images divided into 10 classes, with our study focusing on three :

- 'airplane'
- 'bird'
- 'horse'

The dataset presents a challenge due to its low-resolution images, as we can see from some examples represented in Figure 1.1. The modified version of the dataset contains 18000 images (15000 in the training set and 3000 in the test set), where the three classes are perfectly balanced. For our task, we employ the Histogram of Oriented Gradients (**HoG**), a robust feature descriptor that analyzes local image regions to create histograms of edge directions or gradient orientations. This descriptor effectively captures the shape and texture of objects, simplifying images from 1024 pixels to a 256-value vector (16 orientations × 16 blocks). This transformation significantly reduces data size while preserving essential visual details. We have represented some HoG examples in Figure 1.2, where we can see gradient orientations for each of the sixteen block.

By leveraging these features, the report evaluates the effectiveness of various classification techniques in distinguishing between the selected classes, providing insights into the strengths and limitations of the methods used. The codes on which the following results are based can be found in the [GITHUB REPOSITORY].

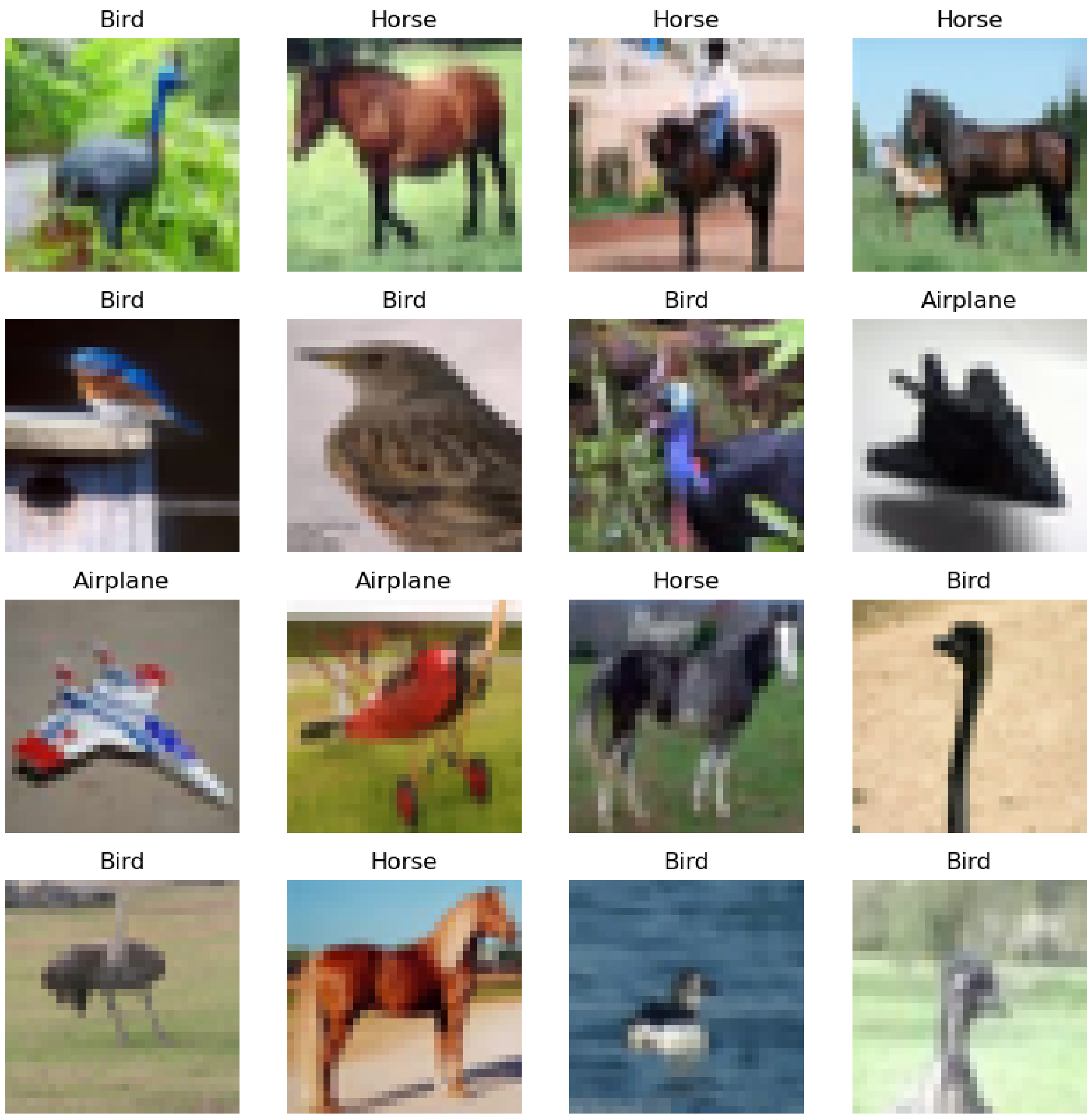


FIGURE 1.1 – Image examples

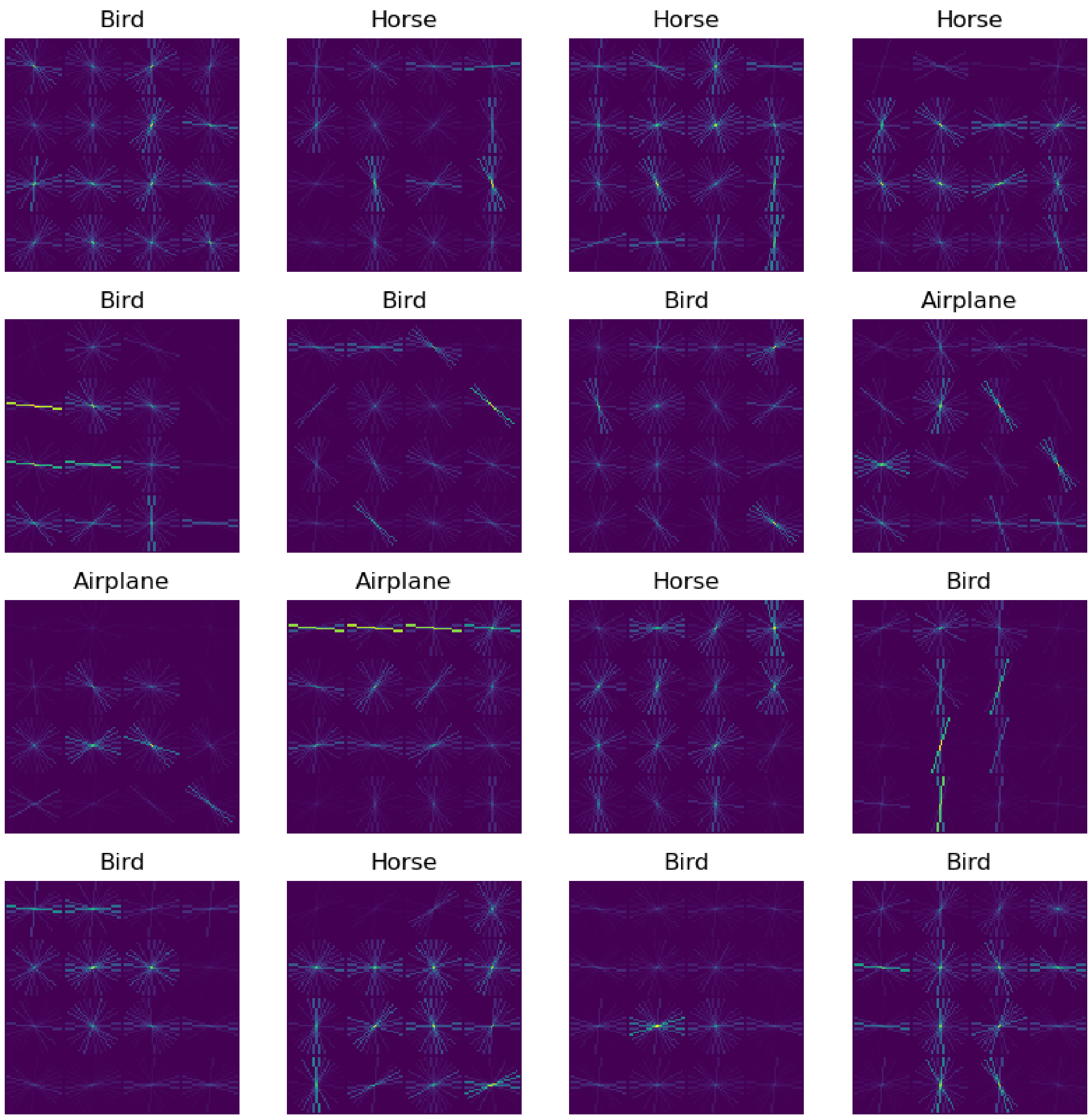


FIGURE 1.2 – HoG examples

2 Classifiers

We will examine numerous methods of performing image classification in this chapter. To implement each of the methods step by step, we'll use the **sklearn** library. The path through image classification is a meticulous one designed to ensure that the model is perfectly suited for the image classification process. It contains the following four primary steps :

1. Use of Standard Model : In this first step, we pick a standard model for our baseline performance. This helps us understand the model's original scalability and limits.
2. Train and Evaluate the Model : In this step, we train the model with our dataset and assess its performance. While following this step, it is simple to discover the model's strength.
3. Hyperparameter Tuning : Involving this step, our model undergoes cross-validation to optimize the model's performance. We use "`sklearn.model_selection.cross_val_score`".
4. Final Evaluation : Additionally, in the final step, we trained the model with optimized hyperparameters and then evaluate it to confirm its improved performance.

The following classifiers will be analysed :

- Ridge
- K Nearest Neighbors
- Decision Trees and Random Forests
- Multi Layer Perceptron
- Support Vector Machines

2.1 Ridge

The Ridge classifier is a robust technique in the machine learning domain used mainly for classification-related tasks. It is derived from the Ridge regression, which was developed to mitigate multicollinearity in regression models through the addition of a L^2 regularization parameter into the loss function. This regularization helps prevent overfitting in models with many predictors and can be summarised as follows

$$\hat{\beta}^R = \text{argmin}_\beta \; ||y - X\beta||_2^2 + \alpha||\beta||_2^2$$

where β_j is the weight for the covariate x_j and y is the target variable. With the Ridge classifier, the penalty added to the coefficients ensures that the coefficients are shrunk towards zero without reaching extreme values. This ensures the model remains robust and generalizes well to unseen data. The method is particularly valuable when working with datasets that have a large number of features, especially when those features exhibit multicollinearity. By effectively managing the complexity of the model, the Ridge classifier balances bias and variance, resulting in reliable classification performance.

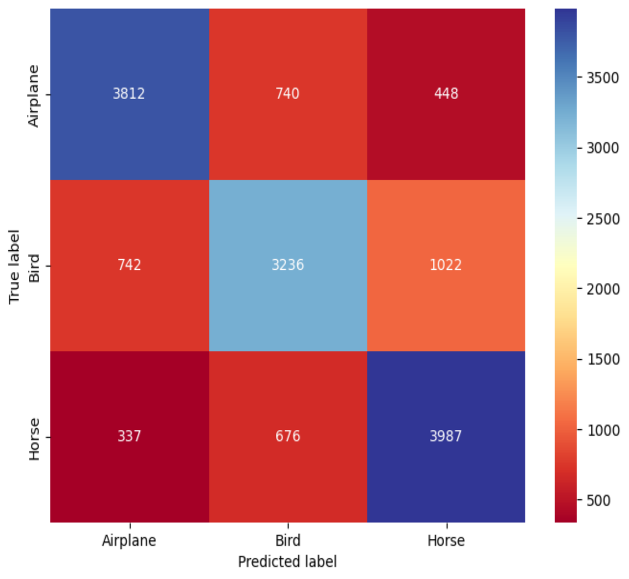
1. We use the default model, where the α parameter is set to 1 by default.
2. After training the model, we report the accuracy for the training and test set.

Set	**Accuracy**
Training	0.736
Test	0.729

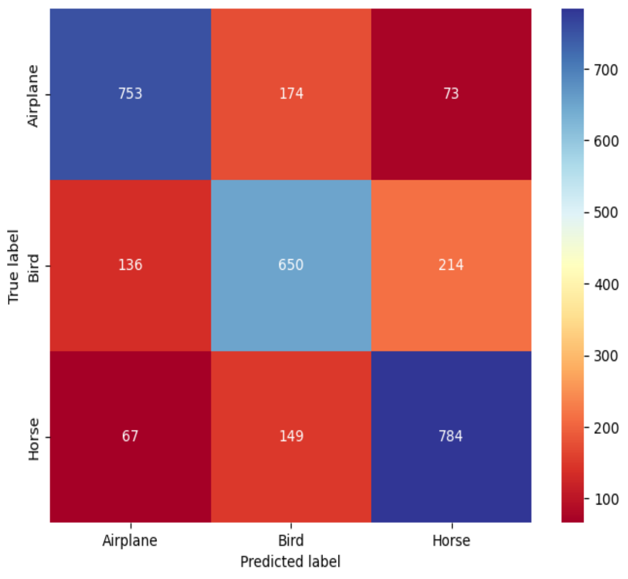
   In Figure 2.1 (a)-(b), we represent the confusion matrices for the two sets, exploring the accuracy for each class in more detail.

3. Using cross-validation to determine which α value maximises accuracy on the validation test, we obtain that the the optimum of 500 values between 0.01 and 1.5 is $\alpha = 0.055$.
4. The optimised Ridge classifier achieves the performance shown in the table. As can be seen, the improvement is there but not very significant. We can see the corresponding Confusion Matrices in the Figure 2.1 (c)-(d).

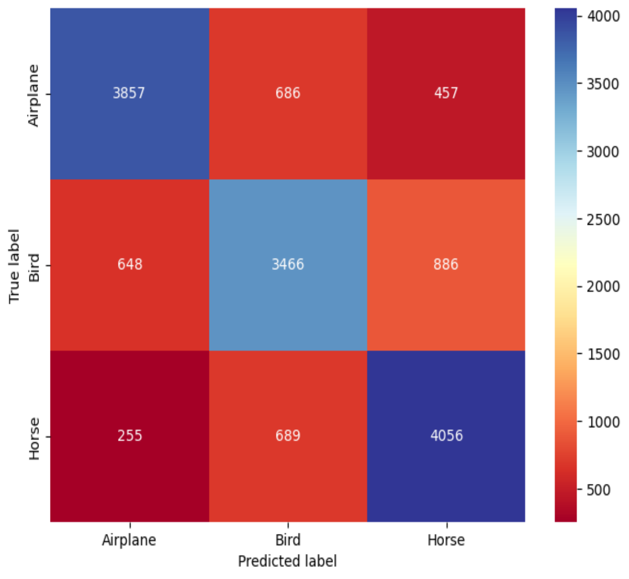
Set	**Accuracy**
Training	0.76
Test	0.744



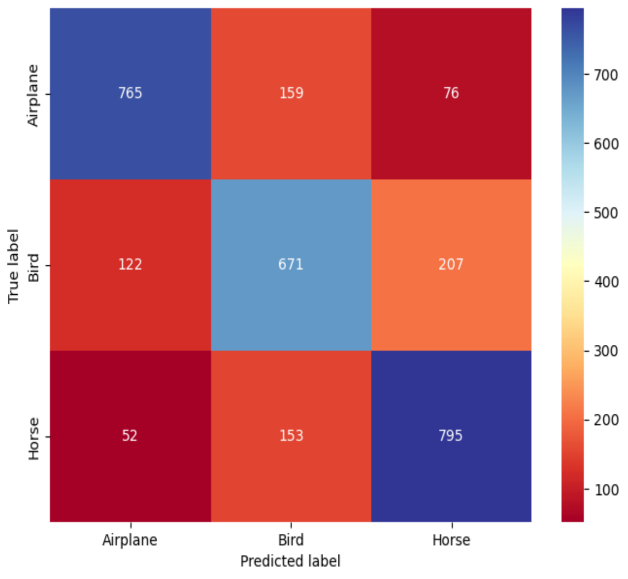
(a) Training-CM for default Ridge



(b) Test-CM for default Ridge



(c) Training-CM for optimised Ridge



(d) Training-CM for optimised Ridge

FIGURE 2.1 – Confusion Matrices for Ridge Classifiers

2.2 KNN

The K-Nearest Neighbors (KNN) classifier is a simple yet powerful algorithm. It operates on the principle of similarity, classifying a new data point based on the majority class among its 'k' nearest neighbors in the feature space. KNN is a non-parametric and instance-based learning algorithm, meaning it makes no assumptions about the underlying data distribution and relies directly on the training data to make predictions. When a new data point needs to be classified, the algorithm calculates the distance between this point and all points in the training set. The 'k' closest points (neighbors) are then identified, and the class most frequently occurring among these neighbors is assigned to the new data point.

1. As a first step, we train the model with $n_{\text{neighbors}} = 1$
2. We report the accuracy for the training and test set. As expected, with only one neighbour the model overfits the data, thus having bad predictive performance.

Set	**Accuracy**
Training	1.0
Test	0.694

3. Then we use cross-validation to calculate the optimal number of neighbours for the model. Searching for values between 1 and 50, we obtain $n^*_{\text{neighbors}} = 5$.
4. The optimised model prevents overfitting, although the improvement in predictive performance is small. The accuracies are shown below.

Set	**Accuracy**
Training	0.81
Test	0.722

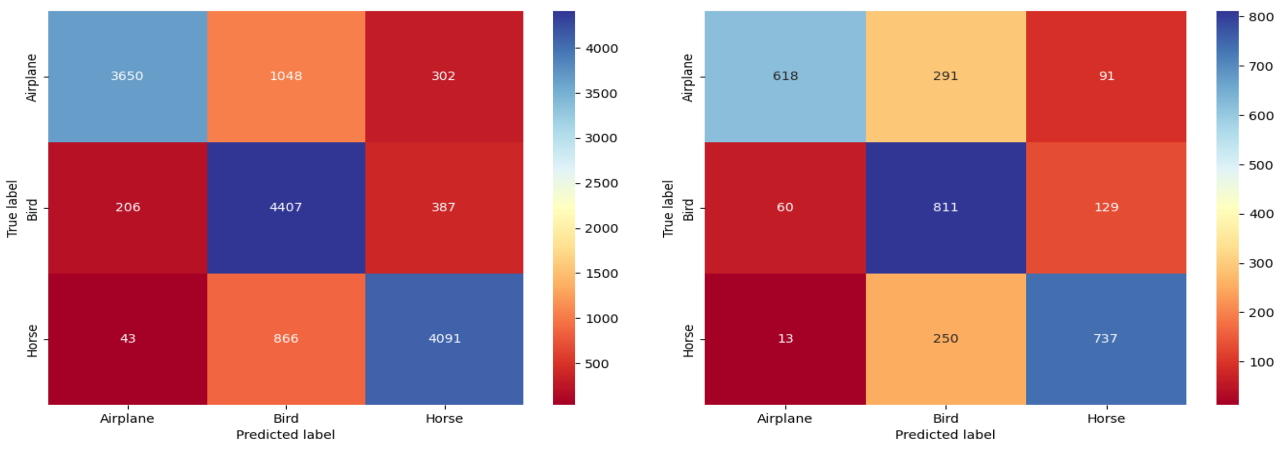


FIGURE 2.2 – Confusion Matrices for 5-NN Classifier (left = Training set , right = Test set)

2.3 Decision Trees and Random Forests

A decision tree is a versatile machine learning algorithm used for both classification and regression tasks. It models decisions and their possible consequences as a tree-like structure of nodes, branches, and leaves. Each internal node represents a decision based on a feature, each branch represents the outcome of the decision, and each leaf node represents a class label or a continuous value. Decision trees are easy to interpret and understand, as they closely mimic human decision-making processes. However, they can be prone to overfitting, especially with complex datasets, as they may create overly detailed models that capture noise in the training data.

Random forests address the limitations of decision trees by building an ensemble of trees. This method constructs multiple decision trees during training and outputs the mode of the classes (classification) or the mean prediction (regression) of the individual trees. Each tree in the forest is built from a random subset of the training data (bagging) and a random subset of features.

The ensemble approach enhances predictive performance and robustness, significantly reducing the risk of overfitting. By averaging multiple trees, random forests provide more accurate and stable predictions compared to single decision trees.

2.3.1 DT Classifier

Decision Trees classify the data by splitting the feature space according to simple, single-feature rules. Scikit-learn uses the CART algorithm for its implementation of the classifier. It is important to anticipate that decision trees may not work well for this task. This is because the nature of decision tree models is not well-suited to the demands of image analysis.

1. We train the default model, where the parameter 'criterion' is set to 'gini' and the parameter 'splitter' is set to 'best' by default.
2. Model performance is reported here.

Set	**Accuracy**
Training	1.0
Test	0.573

3. We use cross-validation to estabilish the best criterion among { 'gini' , 'entropy, ' log-loss' } and the best splitter strategy among { 'best', 'random' }. However, it turns out that the default parameters are already the optimal ones.

2.3.2 RF Classifier

Random Forest classifiers utilize multiple decision trees trained on "weaker" datasets (with less data and/or fewer features), averaging their results to reduce overfitting.

1. We train the default model with n_{trees} set to 100 by default.
2. Here we report the performances of the model. As expected, the bad performances of the single decision tree is not absorbed by the random forest, which achieves the best predictive score seen so far.

Set	Accuracy
Training	1.0
Test	0.768

3. We use cross-validation to calculate the optimal number of trees in the forest. Testing the values $\{20,50,80,100,120,150,180,200\}$, we obtain that $n^*_{\text{trees}} = 150$.
4. The optimised model performs about the same as the default model. Below are the accuracies and Confusion Matrices for the test set for both models.

Set	Accuracy
Training	1.0
Test	0.77

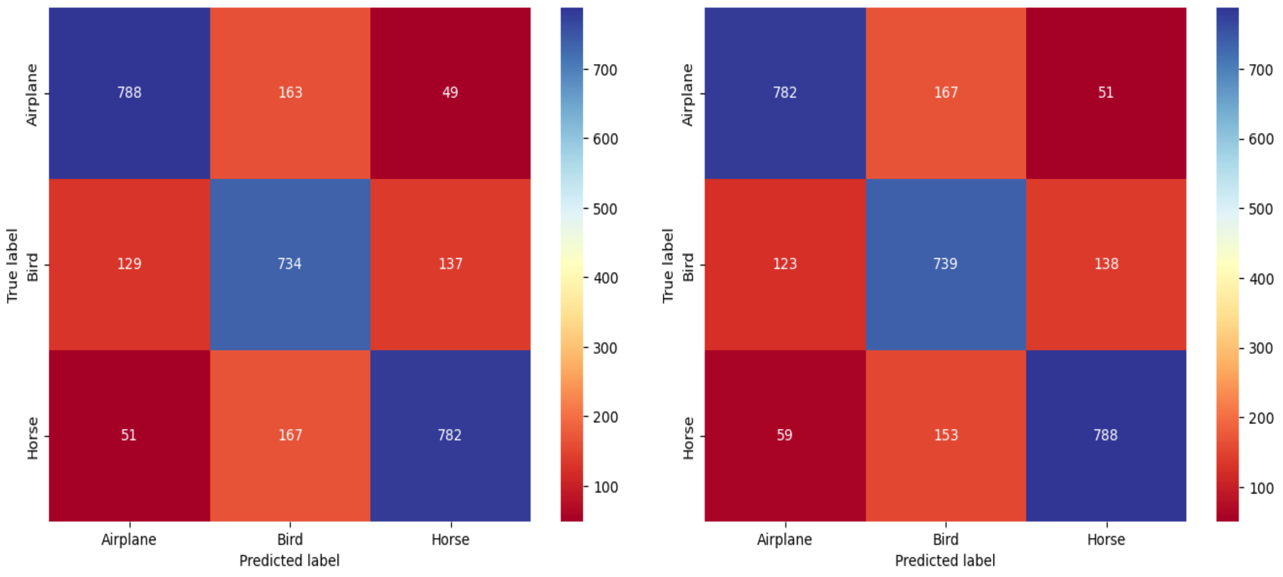


FIGURE 2.3 – Confusion Matrices for RF Classifier in Test set (left : $n_{\text{trees}} = 100$, right : $n_{\text{trees}} = 150$)

2.4 MLP

A Multilayer Perceptron (MLP) is a type of artificial neural network commonly used for supervised learning tasks. It consists of multiple layers of nodes, or neurons, organized in an input layer, one or more hidden layers, and an output layer. Each neuron in a layer is connected to every neuron in the subsequent layer, creating a fully connected network. MLPs use nonlinear activation functions, such as the sigmoid, tanh, or ReLU, which enable the network to capture complex patterns and relationships in the data. During training, MLPs utilize backpropagation, a gradient-based optimization algorithm, to adjust the weights of the connections to minimize the error between the predicted and actual outputs. The key strength of MLPs lies in their ability to model intricate and non-linear relationships, making them suitable for a wide range of applications. However, they require careful tuning of hyperparameters and large amounts of data for effective training.

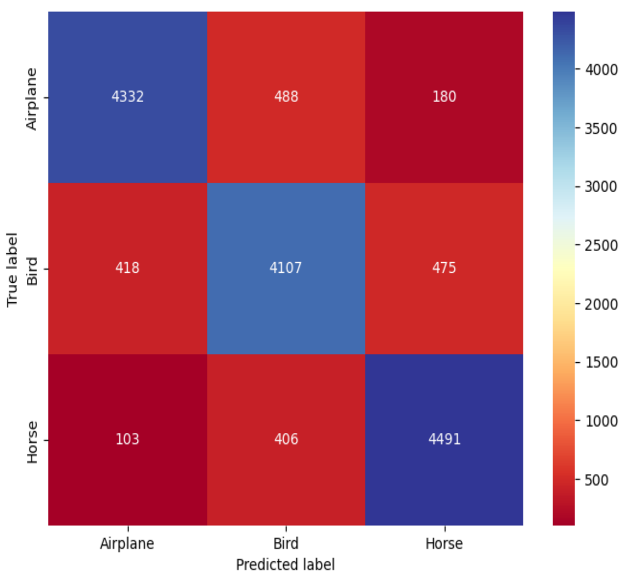
1. We train the default model : one hidden layer with 100 neurons and ReLU activations.
2. We can see from the following table that the performance of the MLP is significantly better than the models seen so far.

Set	**Accuracy**
Training	0.862
Test	0.798

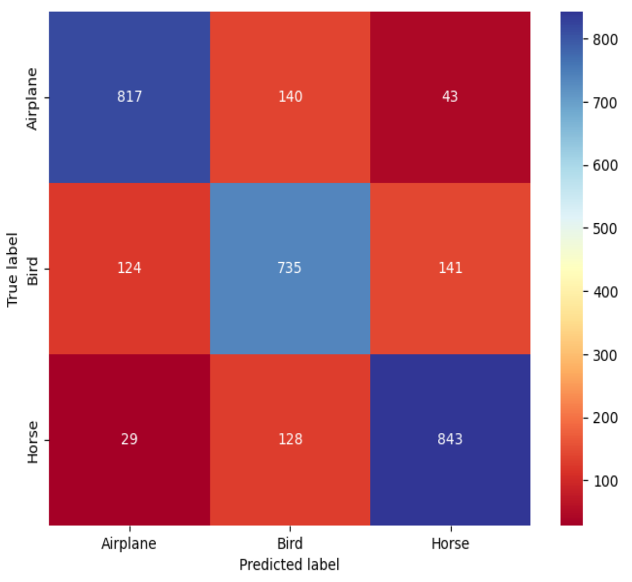
3. We use cross-validation to determine which hidden layer size is best among the values collection {20, 50, 80, 100, 120, 150, 200} and which activation function among the possibilities {'identity', 'logistic', 'tanh', 'relu'} is most suitable. It turns out that the best performance is achieved with a hidden layer of 200 neurons with ReLU activation.
4. The performance of the optimised model is as follows.

Set	**Accuracy**
Training	0.9
Test	0.813

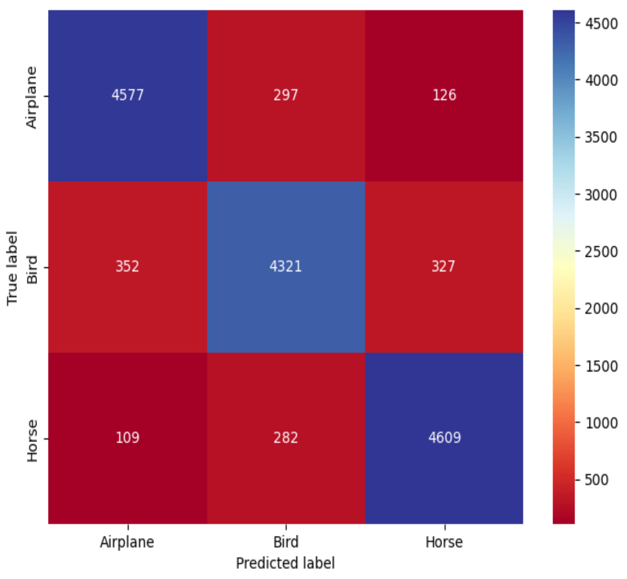
5. As an additional step, we evaluated different MLP architectures by considering more than one hidden layer. In more detail, we analysed the architectures {(20,50,50), (100,50,20), (50,100,50), (200,50,20,20),(50,200,20,50),(100,50,20,100),(50,50,50), (50,100,50)}. Using cross-validation, we obtain that the best one is (200, 50, 20, 20) with which we obtain an accuracy on the Training set equal to 0.862 and on the Test set equal to 0.798. This demonstrates a central point of MLP networks, namely that adding layers and thus increasing parameters does not guarantee better accuracy.



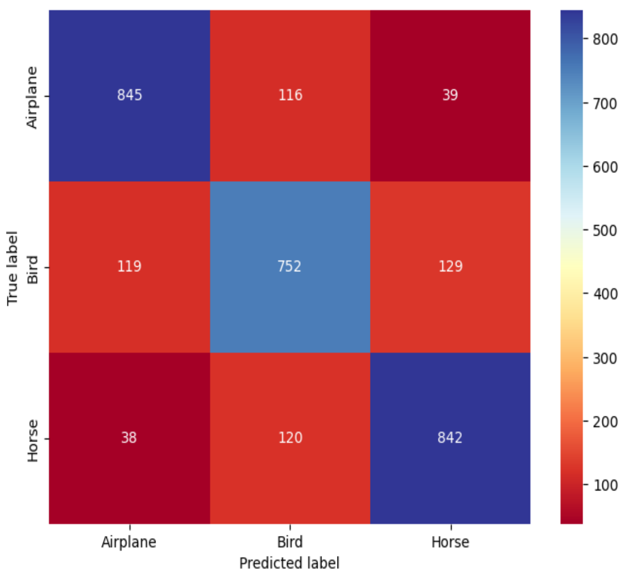
(a) Training-CM for default MLP



(b) Test-CM for default MLP



(c) Training-CM for optimised MLP



(d) Training-CM for optimised MLP

FIGURE 2.4 – Confusion Matrices for MLP Classifiers

2.5 SVM

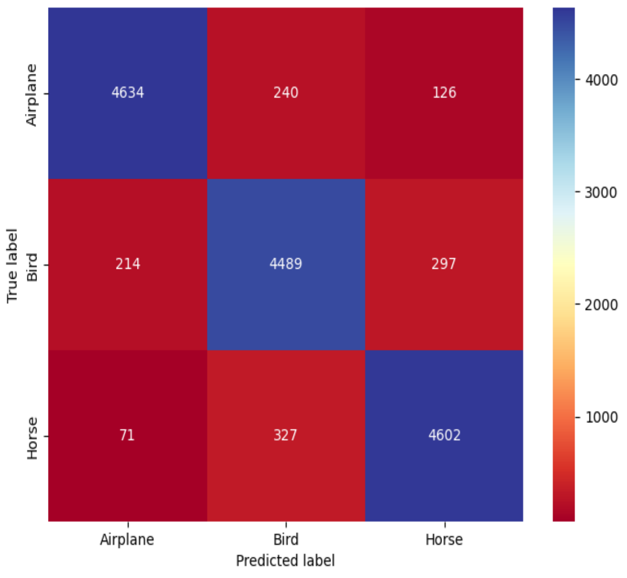
Support Vector Machines (SVMs) are a powerful set of supervised learning algorithms used for classification and outlier detection tasks in machine learning. SVMs are particularly well-known for their ability to handle high-dimensional data and their effectiveness in scenarios where the number of dimensions exceeds the number of samples.The core idea behind SVMs is to find the optimal hyperplane that best separates the data into different classes. This hyperplane maximizes the margin, which is the distance between the closest points of the classes, known as support vectors. By maximizing this margin, SVMs aim to improve the model's generalization ability, ensuring better performance on unseen data. The calculation of this hyperplane is conditioned by hyper parameter C, which governs the penalisation of false positives. Thus, if C is small, the model accepts false positives and the hyperplane divides the two classes less strictly, whereas the opposite happens with large C. SVMs can also be adapted for non-linear classification problems by using kernel functions. These functions transform the input data into higher dimensions, allowing the algorithm to find a linear separating hyperplane in this transformed space.

1. We train the default model with $C = 1$ and and kernel 'rbf'.
2. As expected, this model is very suitable for classification. In fact, its performance beats all those previously seen. For this large dataset, we have 8675 support vectors, distributed across the classes such as (2704,3516,2455).

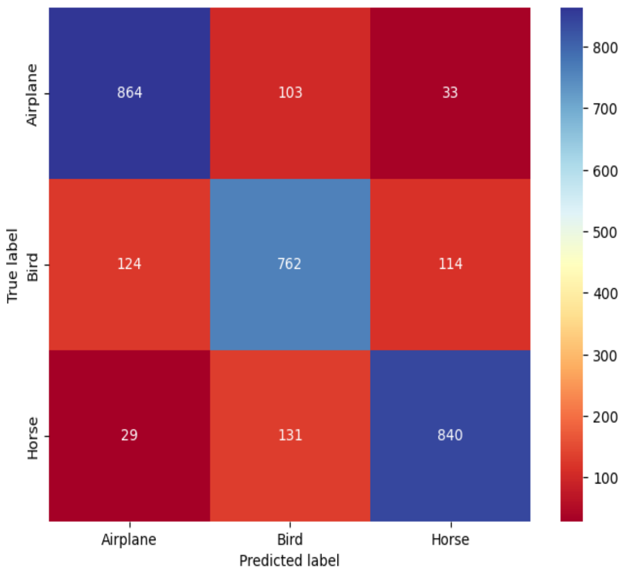
Set	**Accuracy**
Training	0.915
Test	0.822

3. We use cross-validation to determine the best constant C from the following $\{0.1, 0.5, 1, 2, 5, 10\}$ and the best kernel among {'linear', 'poly' (with degree 3), 'rbf'}. We found that the optimal model is achieved with a $C = 5$ and an 'rbf' kernel.
4. The performance of the optimised model is as follows.

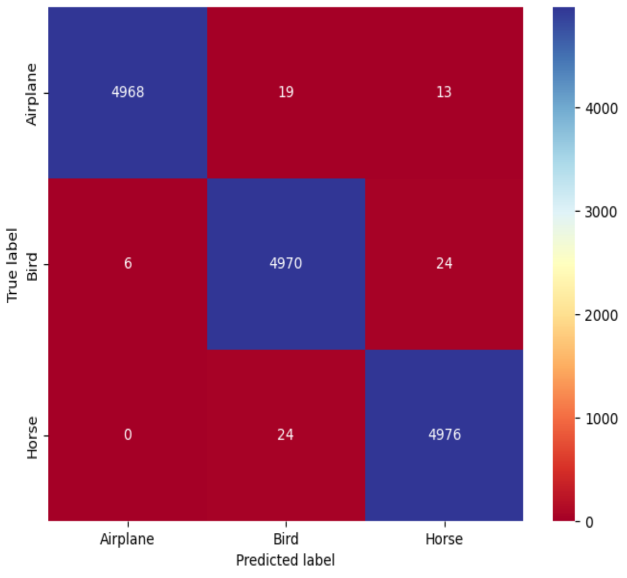
Set	**Accuracy**
Training	0.99
Test	0.83



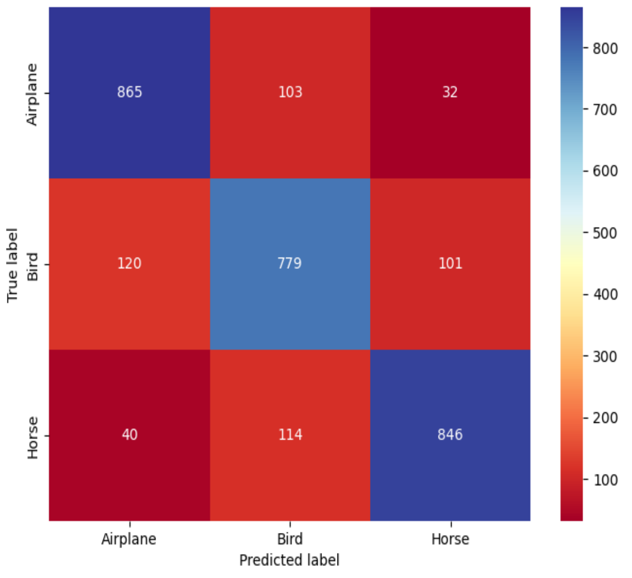
(a) Training-CM for default SVM



(b) Test-CM for default SVM



(c) Training-CM for optimised SVM



(d) Training-CM for optimised SVM

FIGURE 2.5 – Confusion Matrices for SVM Classifiers

2.6 Comparison of results

In this section, we aim to compare the results obtained from the previous section. The table below presents the test accuracy of the optimized models after cross-validation, allowing us to evaluate and contrast their performance effectively.

Model	**Test Accuracy**
Ridge	0.744
KNN	0.722
DT	0.573
RF	0.77
MLP	0.813
SVM	0.83

As we can see, the best approach for this task is to use an SVM model.

3 Deep Convolutional Neural Networks

In our previous analysis, we encountered results' limitations with the models we tested. Therefore, in this chapter, we delve into the capabilities of Convolutional Neural Networks (**CNN**s) for image classification. The first critical point to note is that, unlike previous methods that relied on Histogram of Oriented Gradients (HoG) features, CNNs operate directly on raw image data. This direct approach allows CNNs to capture intricate patterns and spatial hierarchies in images, leading to superior performance. However, this increased capability comes with a significant trade-off. CNNs are massive networks with millions of parameters, requiring substantial computational resources and time for training. The complexity and size of these networks necessitate advanced hardware and optimization techniques to manage the training process efficiently. Despite these challenges, the power of CNNs in extracting and learning complex visual features makes them an invaluable tool in the field of image classification. In Figure 3.1 we represent the architecture of **AlexNet**, one of the first good models for tasks such as CIFAR-10 classification.

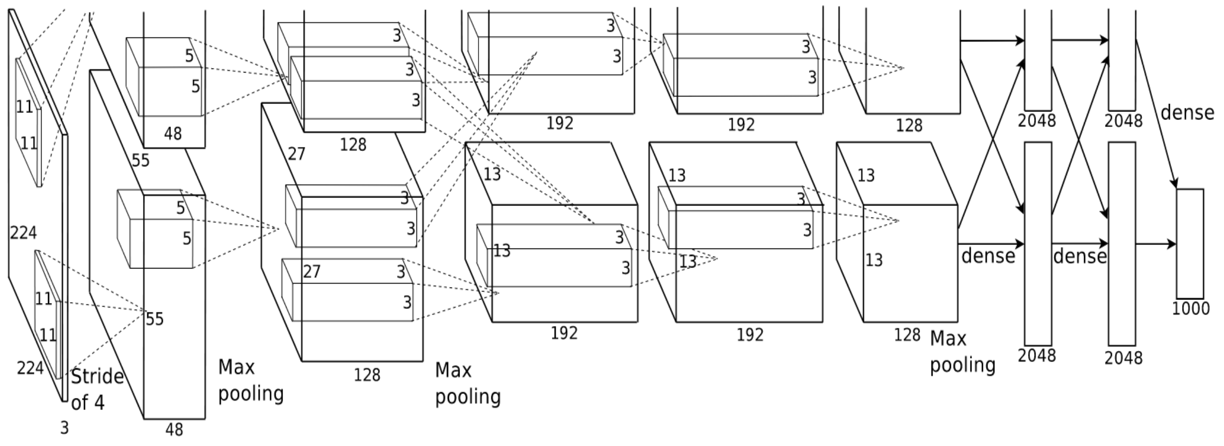


FIGURE 3.1 – AlexNet Architecture

In our case, we implemented the **ResNet** architecture, a groundbreaking model known for its innovative approach to training deep networks. ResNet, short for Residual Network, addresses the problem of vanishing gradients, which often hampers the training of very deep networks.

The core idea behind ResNet is the introduction of residual connections, or skip connections, that bypass one or more layers. These connections allow the model to pass information directly from one layer to another, effectively enabling the network to learn residual functions with reference to the layer inputs, rather than learning unreferenced functions. This makes it significantly easier to optimize deeper networks, as the gradient can flow through these shortcut connections directly, reducing the problem of gradient vanishing. The introduction of residual connections has a profound impact on the field, allowing networks to be substantially deeper without suffering from the degradation problem, where adding more layers leads to worse performance.

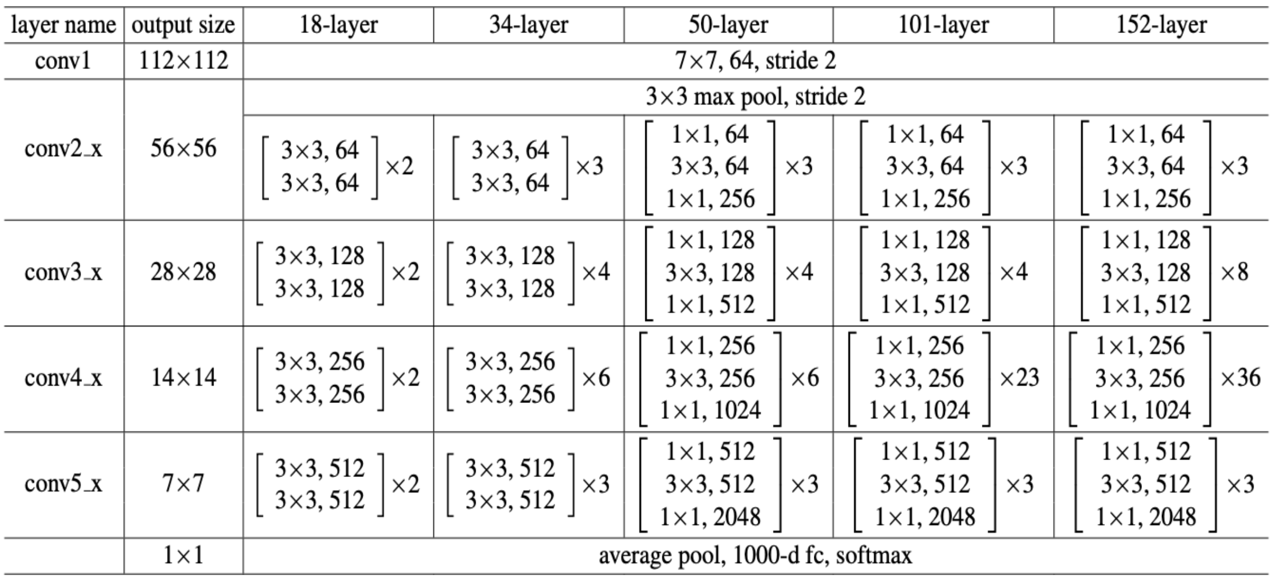
In our implementation, we utilized **ResNet-18** , **ResNet-34** and **ResNet-50** architectures. These variants of ResNet, with 18, 34 and 50 layers respectively, strike a balance between depth and computational efficiency, making them well-suited for our image classification tasks.

In Figure 3.2, we represent the composition of different architectures.

We trained the models on our dataset. The training process involved extensive computational resources to handle the significant number of parameters inherent in these models. After training, we evaluated the models on the test set to measure their performance. The results, including the accuracies and the number of parameters for each model, are summarized in the table below.

Model	**Parameters**	**Train Accuracy**	**Test Accuracy**
ResNet-18	11.2 M	1.0	0.847
ResNet-34	21.3 M	0.98	0.846
ResNet-50	23.5 M	0.95	0.816

In this case, the simplest model is sufficient to achieve the best performance. This result is very interesting, because it shows, as we have also seen in the section on MLPs, that increasing the number of parameters and the complexity of the model does not guarantee an improvement in performance, but even in this case makes it worse.



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				

FIGURE 3.2 – ResNet Architectures

4 Conclusion

In conclusion, our image classification experiment demonstrated an impressive accuracy of 0.83/0.85. This particular example highlights that Machine Learning methods can achieve performance levels comparable to those of Deep Learning approaches, but with significantly lower computational costs. This confirms that Deep Learning is not always the best answer. However, ResNet gives us the best performance and our models seem to hit an accuracy ceiling at 0.85, indicating a need for further improvements to surpass this threshold.

To overcome this limitation, several strategies can be explored. Preprocessing the images to enhance contours or applying segmentation techniques to isolate objects from their backgrounds could provide the models with clearer and more distinguishable features, potentially improving their accuracy. These steps would allow our models to better understand and classify the images, thereby pushing beyond the current performance barrier.

Lastly, it is crucial to possess a comprehensive understanding of various models to select the most suitable one for a given task. This knowledge enables leveraging the strengths of each model, ensuring optimal performance and efficiency. By combining this expertise with advanced preprocessing methods, we can push beyond current performance limits and achieve even higher results.