Object Recognition in Images Using CIFAR-10 dataset

Fatemeh Fathi Niazi *CMSE890 – Project Report*

1 Introduction

The autonomous vehicles industry has the potential to revolutionize transportation yet requires the development of robust and accurate machine learning models capable of navigating the complex real-world environment. Object recognition and classification are critical components of such models, particularly for real-time detection of objects around the road. This project focuses on developing and evaluating a series of convolutional neural network (CNN) models for image classification and object recognition, using the CIFAR-10 dataset as a small-scale evaluation to simulate the demands of the autonomous vehicle industry.

2 Task Definition

To train and test the CNN models, the CIFAR-10 dataset was utilized. This dataset consists of 60,000 images of 32 by 32 pixels belonging to ten different classes, namely airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. The dataset is divided into five training batches and one test batch, each containing 10,000 images. The test batch includes exactly 1,000 randomly selected images from each class, while the training batches contain the remaining images in a random order. It is worth noting that some training batches may contain more images from one class than another, but the proportion of each label in the whole training dataset are equal. The objective of this project is to train a CNN model to accurately classify the 10,000 unseen test data. By obtaining the predicted labels for the test data, the model performance (e.g., accuracy, f1-score, and confusion matrix) can be computed.

3 Algorithm Definition

In this project, a combination of two techniques was employed: Principal Component Analysis (PCA) was employed for dimensionality reduction, and CNN was used for image classification. Given that CNN is known to be computationally expensive, PCA was explored as a means of reducing the number of features in the dataset while maintaining classification accuracy. The objective was to improve the computational efficiency of the CNN model while ensuring the accuracy of the classification remains close to that of the full dataset without any dimensionality reduction. By reducing the number of features through PCA, it was expected that the CNN would perform image classification tasks more efficiently. The ultimate goal of combining these two techniques was to develop an accurate and computationally efficient image classification model capable of being applied to real-world scenarios.

4 Machine Learning Approach

This section provides a detailed overview of the various machine learning (ML) approaches utilized in this study. Specifically, the application of exploratory data analysis (EDA) methods and the construction of pipelines for processing and analyzing the data are discussed in depth. Furthermore, the results of the analyses, including insightful visualizations that effectively illustrate key findings, are presented. Additionally, the report offers a thorough discussion of the outcomes and their implications, drawing on relevant literature to contextualize the findings. As a result, readers will gain valuable insights into the ML approach employed and how it was effectively applied to address the research questions posed in this study.

4.1 EDA methods

EDA is a crucial step in any machine learning project, particularly in the context of image classification using CNNs. Some of the EDA steps undertaken in this project are detailed in the following sub-sections. Notably, as the project was run on a remote computing server (Markov server in Biochemistry Department) rather than a local machine, some errors were encountered when attempting to download and load the dataset directly through the "`tensorflow.keras.datasets`" method. To circumvent this issue, a copy of the dataset in the form of pickle files was transferred to the remote computing server and subsequently read manually within the code.

4.1.1 Analyze class distribution

The CIFAR-10 dataset was utilized in this project, comprising 60,000 images with dimensions of 32 by 32 pixels from 10 different classes: including airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. The dataset is partitioned into five training batches and one test batch, each with 10,000 images. The test batch comprises 1,000 randomly selected images from each class. The training batches contain the remaining images in random order, but the distribution of images from each class in the whole training dataset is uniform, with exactly 5,000 images from each class, as illustrated in **Figure 1**. It is worth noting that some training batches might have more images from one class than another.

4.1.2 Dataset visualization

Figure 2 shows ten randomly selected images from each class in the CIFAR-10 dataset. As this figure shows, the images exhibit variations in angles and perspectives. Additionally, the images have a resolution of 32 by 32 pixels, which might appear slightly blurry to the human eye. Nonetheless, the images contain sufficient details for successful classification by ML algorithms.

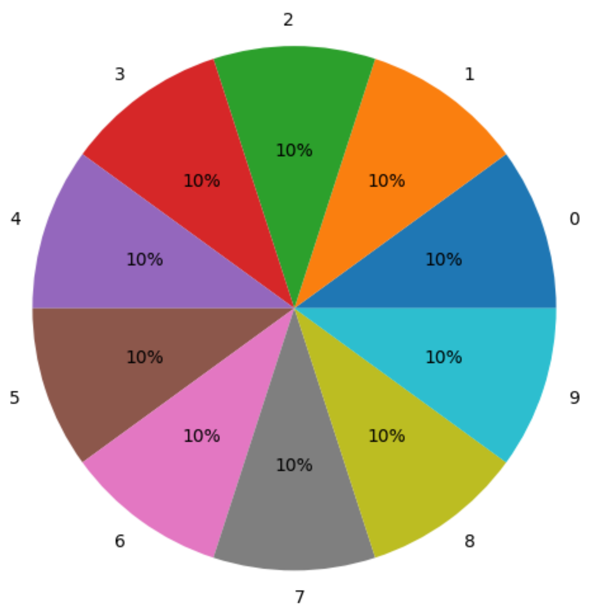

Figure 1. *The proportion of each label in the training dataset.*

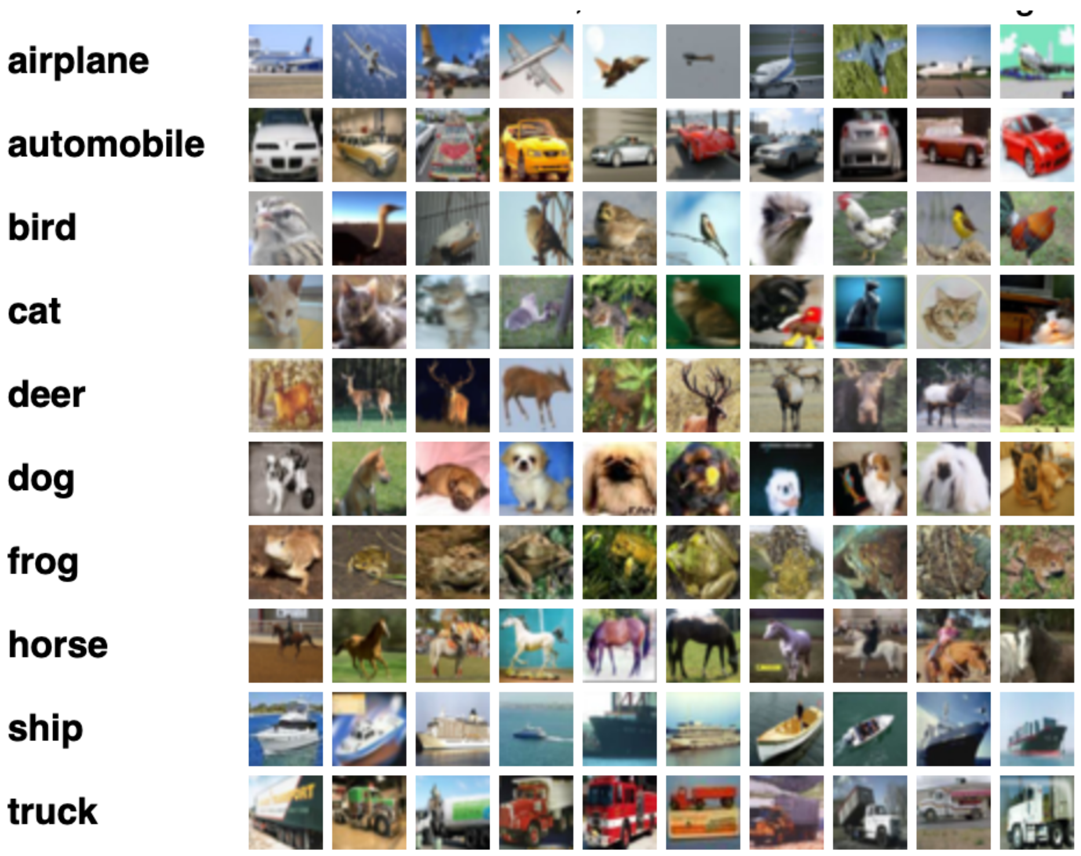

Figure 2. *Different Classes in the dataset, as well as 10 random images from each.*

4.1.3 Preprocessing

Retrieved data from the CIFAR-10 dataset has 3072 features, which includes the flatten version of the 32 by 32-pixel network in three channels of red, green, and blue. Therefore, the retrieved data matrix was reshaped into the `NumRows`×32×32×3, where `NumRows` is the number of images in the dataset. It is noted that the Fortran-like reshaping should be used to maintain the image structure. After reshaping the dataset, **Figure 3** shows a sample image with its corresponding different color channels.

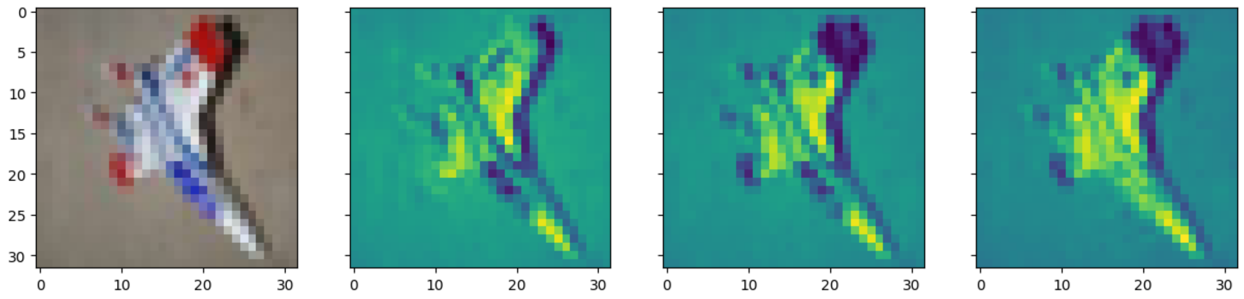

Figure 3. *Different channels of an image from dataset.*

Scaling the input data is an essential preprocessing step for ML projects, even though the neural network (NN) models are theoretically insensitive to it. Scaling enables optimization algorithms to converge more smoothly, avoiding getting stuck in local minimum or jumping from a minimum zone. In this project, the "`sklearn.preprocessing.StandardScaler`" function was used to scale the input dataset. The `StandardScaler` not only scales the data but also attempts to remove the mean from the dataset, centering the data around zero, which makes the data more desirable for algorithms like PCA. Scaling the data in this manner simplifies training CNNs since it makes the optimization process more efficient. If the data is not scaled, then some input features may have a much larger order than others, which can cause numerical instability during the training process, slowing convergence, and, in some cases, even causing the optimization algorithm to fail. Scaling the data ensures that the gradients computed during backpropagation are of similar magnitude, which helps the optimization algorithm converge more quickly and reliably.

4.1.4 Split dataset into training, validation, and testing

The CIFAR-10 dataset was already divided into distinct training and testing datasets, making any additional partitioning superfluous. Nevertheless, in order to achieve peak model performance, a validation dataset was formed by allocating 10% of the training dataset. This methodology permits the model to be trained using most of the data, while also assessing its efficacy on an independent subset during each epoch.

4.2 Machine learning algorithms

As mentioned before, there are two main ML algorithms used in this project: PCA for dimensionality reduction, and CNN algorithm for image classification.

4.2.1 Dimensionality reduction using PCA

Principal Component Analysis (PCA) is a popular technique used for dimensionality reduction in image processing. It is widely used to extract a set of lower-dimensional, more informative features from high-dimensional image datasets. One of the main applications of PCA in image processing is to prepare the image data for classification using CNNs. By reducing the dimensionality of the input data, PCA can help to reduce the computational requirements and speed up the training process of the CNN model. In this way, PCA plays a crucial role in improving the performance of image classification models, particularly in cases where the dataset has a large number of features or high-dimensional image data.

In this project, the PCA algorithm was utilized to reduce the number of features in each channel of the dataset. The PCA algorithm was applied separately to each channel, resulting in a reduction of the number of features (e.g., from 1024 to 400). Following the reduction, the different channels were stacked together to form a smaller image in RGB. In addition, the idea of using only one channel (in this case red) to classify the images was investigated. In this scenario, the PCA algorithm was applied to only the red channel, and the resulting features were used to train the CNN model.

4.2.1.1 Selecting the number of components in PCA

The dimensionality reduction was carried out using the "`Sklearn.decomposition.PCA`" function. To determine the optimal number of dimensions to reduce to, the elbow method was employed, where the explained variance is plotted as a function of the number of dimensions. The results of the elbow method indicated that reducing the red channel to 236 dimensions could preserve 97% of the information in the training dataset, as shown in **Figure 4**. However, in this project, four different numbers of dimensions per channel were selected, including 1024 (no reduction), 625, 400, and 225.

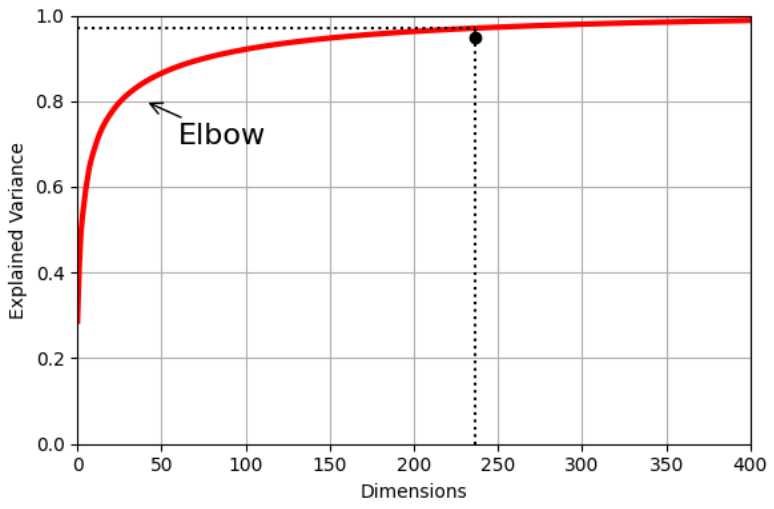

***Figure 4**. Explained variance as a function of the data dimension.*

4.2.2 Image classification using CNN

Convolutional Neural Networks (CNNs) are a type of neural network commonly used in image classification tasks. These models are designed to recognize patterns in image data by analyzing and processing the visual features of an image. Unlike traditional neural networks, CNNs employ convolutional layers, which allow the model to effectively capture spatial relationships between pixels in an image. CNNs have been shown to achieve state-of-the-art performance on a variety of image classification tasks, including object recognition and facial recognition. Due to their high accuracy and efficiency in processing large amounts of visual data, CNNs have become a popular tool in computer vision and machine learning.

In this project, the “`Tensorflow`” library is used for training the CNN model and make predictions. The structure of the CNN models are determined such that each 2D convolution layer is followed by a “`tf.keras.layers.BatchNormalization`” layer. This layer is typically used after convolution layers in deep learning models to normalize the output of the layer and accelerate the learning process by reducing the internal covariate shift. This helps to ensure that the activations going into the next layer are more consistent, which in turn helps the model to generalize better to new data. Moreover, the “`tf.keras.layer.MaxPooling2D`” is also used after each two set of the convolution to reduce the spatial dimensions of the outputs feature map. It also follows to a “`tf.keras.layer.DropOut`” layer to randomly drop some percentage of the neurons to avoid the overfitting.

4.3 Results and discussion

This section focuses on the hyperparameter tuning process of the machine learning models, followed by the selection of the best performing model. The primary objective of this phase is to optimize the hyperparameters of the models to improve their performance on the validation dataset. Once the hyperparameters are tuned, the performance of the selected model is evaluated using different metrics, such as accuracy, precision, recall, and F1 score, to assess the model's ability to classify the images accurately.

4.3.1 Hyperparameter tuning

Hyperparameter tuning is a crucial step in ML that involves selecting the optimal set of hyperparameters for a given model to achieve the best performance. The process of hyperparameter tuning involves systematically searching through a range of hyperparameter values to find the optimal combination that yields the best performance on the validation dataset. The success of a ML project often depends on selecting the best hyperparameters, as poorly chosen hyperparameters can lead to overfitting, slow convergence, or poor performance on new data.

4.3.1.1 Hyperparameter matrix

The following hyperparameters are tuned during this project:

- **Activation function** [“relu” and “selu”]: The rectified linear (“relu”) activation function is commonly used in CNNs due to its simplicity and effectiveness, while the scaled exponential linear ("selu") activation function is known to perform better in deeper networks by reducing the likelihood of vanishing/exploding gradients.

- **Dropout values** [0.0, 0.2, 0.4, and 0.5]: the dropout value specifies the probability of randomly dropping out a fraction of the nodes in a layer during training.
- **Optimization algorithm** ["adam" and "sgd"]: "adam" is an adaptive optimization algorithm that computes adaptive learning rates for each parameter, while "sgd" is a stochastic gradient descent optimization algorithm that updates the model parameters by taking the average of the gradients calculated on randomly selected samples.
- **Initial neuron size** [512, 256]: the number of neurons in the first hidden layer after the convolution. It is noted that the number of neurons from that layer uniformly reduces to 10 neurons in the output layer.
  - Due to the lack of time and for computational efficiency, the initial neuron size was fixed with 512 neurons (conservative side) to reduce the number of runs!
- **Initial filter size** [32, 64]: the number of different filters to be tried on the input image to each convolution layer. It is noted that the number of filters for the convolution layer doubles after each two convolution layers.
- **CNN architecture:** the CNN consists of a convolution part (combination of different 2D convolution, MaxPooling, BatchNormalization, and Dropout layers) and a ANN part (combination of different fully connected hidden layers). In this project, a combination of different layer layouts has been tried:
  - **Number of hidden layers** [2, 3]: the use of 2 and 3 hidden layers in the ANN part has been evaluated as a hyperparameter.
  - **Number of 2D convolution layers** [4, 6]: the use of 4 and 6 convolution layers has been evaluated as hyperparameter.
- **Use of PCA and/or one channel**: in order to evaluate if application of dimensionality reduction with PCA and/or using only one channel of RGB can help in improving the computational runtime and maintain the performance, the following limited cases have also been evaluated. It is noted the full combination of the cases could result in a much bigger running matrix, but the following four cases was evaluated:
  - **Full run in RGB**: using the original data without dimensionality reduction and also feeding all three channels into the CNN.
  - **PCA (625 features) in RGB:** reducing the number of features from 1024 to 625 using the PCA algorithm and also feeding all three channels into the CNN.
  - **PCA (400 features) in R channel:** reducing the number of features from 1024 to 400 using the PCA algorithm and also feeding only red channel into the CNN.
  - **PCA (225 features) in R channel:** reducing the number of features from 1024 to 225 using the PCA algorithm and also feeding only red channel into the CNN.

4.3.1.2 Hyperparameter tuning algorithm

A manual nested loop was implemented to perform the hyperparameter tuning in this project, as the runtime was significantly high, and the manual code could be run for different cases separately. Additionally, to avoid memory shortage errors, the results of each model were saved as an h5 file on a cloud drive, allowing for each model to be trained and evaluated individually. It is noted that, to reduce the computation time, a subset of 10,000 images from the training dataset was used for the hyperparameter tuning process, while the number of epochs was reduced to 20.

4.3.1.3 Hyperparameter tuning results

The results of the hyperparameter tunning are saved within 129 trained h5 files, which takes more than 18 hours of runtime in a Markov remote computing server. After that, a Python script was developed to load each of these models and predict the classes for the testing dataset to calculate the performance metrics, including precision, recall, and f1-score. In this regard, **Table 1** shows the best three models in each case (in terms of using PCA/channels) based on their f1-score.

***Table 1**. Hyperparameter tuning results.*

No. conv. layers	No. hidden layers	Activ. func.	Optim. alg.	Dropout value	Ini. filter size	Ini. neuron size	Run time (s)	Recall (%)	F1score (%)
Case: Full run in RGB									
6	**3**	**SeLu**	**adam**	**0.4**	**32**	**512**	**420.3**	**72.57**	**72.31**
6	3	SeLu	adam	0.5	32	512	416.9	71.83	71.46
6	3	ReLu	adam	0.4	32	512	411.1	71.02	70.86
Case: PCA (625 features) in RGB									
4	2	SeLu	sgd	0.4	64	512	427.4	34.49	33.33
4	2	ReLu	sgd	0.4	64	512	502.1	33.47	32.66
4	2	SeLu	sgd	0.5	64	512	321.1	33.40	31.87
Case: PCA (400 features) in R channel									
6	3	ReLu	sgd	0.2	64	512	258.4	33.49	32.57
6	3	ReLu	adam	0.4	64	512	257.1	30.63	29.68
4	2	SeLu	sgd	0.4	64	512	259.3	30.26	29.18
Case: PCA (225 features) in R channel									
4	2	ReLu	sgd	0.4	64	512	201.5	32.04	31.08
4	2	SeLu	sgd	0.4	64	512	202.1	30.89	30.19
4	2	ReLu	sgd	0.2	64	512	203.6	30.76	29.62

As demonstrated in the aforementioned table, the implementation of PCA-based dimensionality reduction resulted in a notable decrease in the performance metrics of the CNN model, even when using 625 reduced features. Consequently, it can be inferred that the use of dimensionality reduction is not advisable, and the final model should consider all data features in its design. One plausible reason for the lackluster performance of the dimensionality reduction approach could be the presence of varying angles of the objects in the images, rendering the majority of the pixels crucial for an effective classification by the CNN.

Based on the results of the hyperparameter tuning, the properties of the selected model as summarized as follows. Also, the model is conceptually illustrated in **Table 2**, which simulated the summary table of the `Tensorflow` module.

- 6 convolutional layers.
- 3 hidden layers.
- SeLu activation function.
- adam optimization function.
- dropout value of 0.4.
- Initial filter size of 32.
- 512 neurons in the first hidden layer.

Table 2. *Best CNN model architecture.*

CNN part	Layer	Output shape	# Params
Part I: convolution	Conv 2D	(X, 32, 32, 32)	896
	Batch Normalization	(X, 32, 32, 32)	128
	Conv 2D	(X, 32, 32, 32)	9248
	Batch Normalization	(X, 32, 32, 32)	128
	Max Pooling 2D	(X, 16, 16, 32)	0
	Dropout	(X, 16, 16, 32)	0
	Conv 2D	(X, 16, 16, 64)	18496
	Batch Normalization	(X, 16, 16, 64)	256
	Conv 2D	(X, 16, 16, 64)	36928
	Batch Normalization	(X, 16, 16, 64)	256
	Max Pooling 2D	(X, 8, 8, 64)	0
	Dropout	(X, 8, 8, 64)	0
	Conv 2D	(X, 8, 8, 128)	73856
	Batch Normalization	(X, 8, 8, 128)	512
	Conv 2D	(X, 8, 8, 128)	147584
	Batch Normalization	(X, 8, 8, 128)	512
	Max Pooling 2D	(X, 4, 4, 128)	0
	Dropout	(X, 4, 4, 128)	0
Part II: ANN	Flatten	(X, 2048)	0
	Batch Normalization	(X, 2048)	8192
	Dense	(X, 512)	1049088
	Dropout	(X, 512)	0
	Batch Normalization	(X, 512)	2048
	Dense	(X, 261)	133893
	Dropout	(X, 261)	0
	Dense	(X, 10)	2620

4.3.2 Tuned model

Upon completion of the hyperparameter tuning process, a CNN model was trained using the optimized hyperparameters on the entire training dataset consisting of 50,000 images. The training process was completed in 7,154 seconds (almost 2 hours), and the accuracy curve is presented in **Figure 5**. The plot reveals that the accuracy of both the training and validation datasets increases simultaneously, indicating that the model is not overfitting.

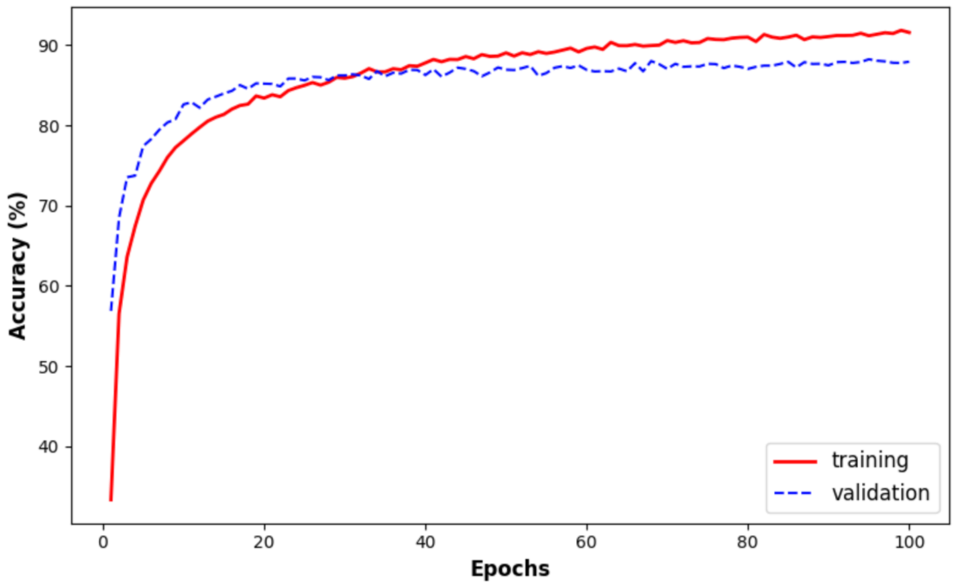

Figure 5. Accuracy evolution during training of the best model.

In order to evaluate the performance of the trained CNN model, it was used to predict the class of the images in the testing dataset. The results were then compared against the actual labels. For this purpose, **Figure 6** shows the confusion matrix of the predictions for the testing dataset. As shown in the confusion matrix, classes that are mistakenly classified instead of each other are the dogs and cats.

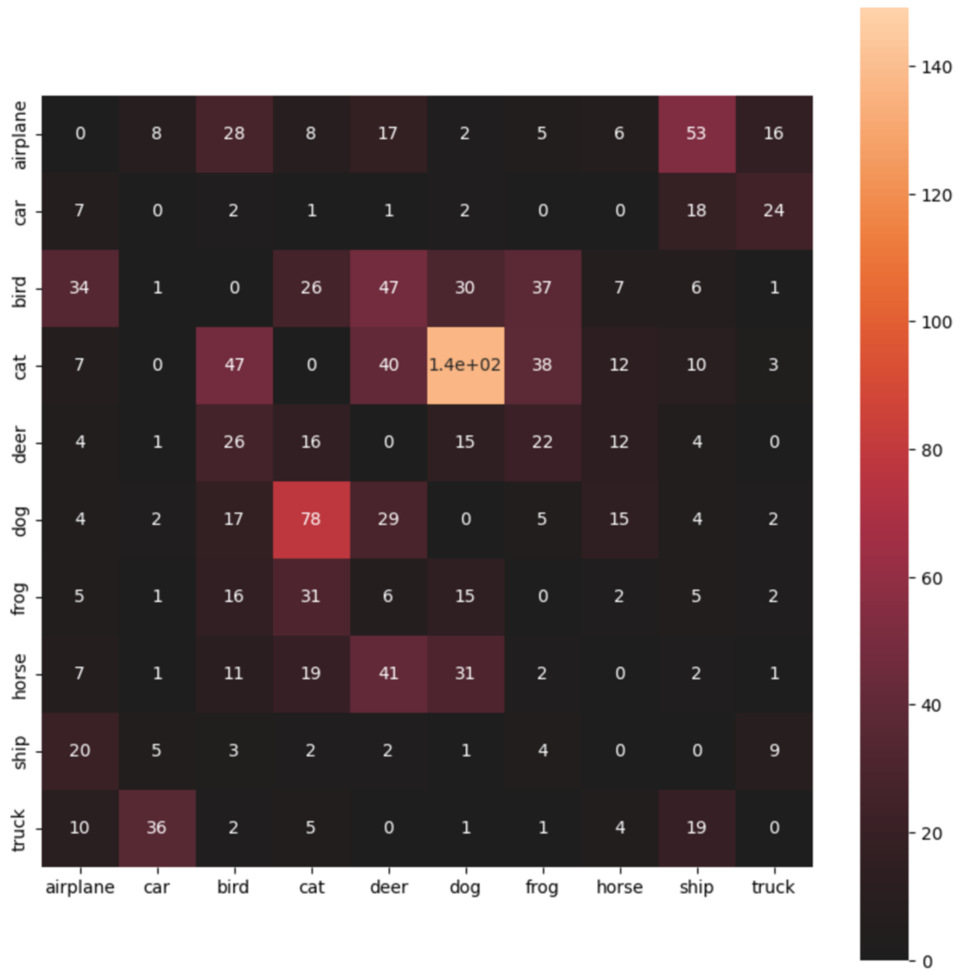

Figure 6. *Confusion matrix for the best CNN model.*

The model performance has been calculated using the classification performance metrics. The precision, recall, and f1-scores for the testing dataset was 87.47%, 87.42%, and 87.37%, respectively. The relatively high f1-score indicated the good performance of the model in classifying the images in the testing dataset. Moreover, the precision and recall scores are close to each other, which means that the model is predicting both the positive and negative classes with similar accuracy, without a significant bias towards one class.

5 Summary and conclusions

The objective of this project was to design a machine learning (ML) model for image classification using the CIFAR-10 dataset. To achieve this, a convolutional neural network (CNN) model was selected, and in order to enhance its computational efficiency, the possibility of applying the PCA algorithm for dimensionality reduction and the use of only one-color channel were also examined. Additionally, a manual hyperparameter tuning process was conducted to identify the best possible set of hyperparameters for the CNN model that can yield the highest classification accuracy.

The findings of this study indicate that utilizing all color channels in the optimal model is more effective. On the other hand, the PCA algorithm causes a substantial reduction in the accuracy and f1-score of the model. Therefore, the chosen model utilizes all pixels to ensure maximum efficiency. The CNN model with the optimized hyperparameters achieved an accuracy of 91.58% on the training dataset and a relatively high f1-score of 87.37% on the testing dataset.

6 Reference

Géron, A. (2017). Hands-on machine learning with scikit-learn and tensorflow: Concepts. Tools, and Techniques to build intelligent systems.

Krizhevsky, A., & Hinton, G. (2009). Learning multiple layers of features from tiny images. Technical report, University of Toronto.

Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. In Advances in neural information processing systems (pp. 91-99).

7 Appendix: Source code

The codes are available on my GitHub account using this link.