Assignment 2:
Convolutional Neural Network for a Multi-Class Classification of Images Case

Nicole Cristine Dressler
University of Sunderland
Machine Learning and Data Mining (CETM26)

Table of Content

1. Introduction

The field of Machine Learning has grown significantly over the past years, with a special attention to Deep Learning and Neural Networks. Inspired by the brain's biological neural networks, the Artificial Neural Networks (ANN) have come to help in many human processes with its positive results being palpable and its potential only continues to grow.

Deep Learning became vital for society as it aids in complex tasks, such as pattern recognition, data-driven decision making tasks, automation and improvement of efficiency, and natural language processing. It brings an immense advancement, leading to economic growth and scientific and technological improvement of many new products, services and solutions. A critical example can be seen in healthcare, with AI (artificial Intelligence) being used to analyse medical images and genetic data, improving the diagnosis accuracy and enabling early detection of diseases.

Overall, ANNs have a great impact in our society, revolutionising the way we interact with technology. However, it must also be noted the ethical considerations and potential risks that ANNs bring, such as bias, privacy, transparency, and accountability to ensure the responsible and ethical use of ANNs in our society.

This report aims to implement a Neural Network for a multi-class image classification task, to accuracy on object recognition. The Cifar-10 dataset will be utilised for this project, providing a practical understanding of the construction, training and evaluation of a CNN (Convolutional Neural Network) model and its capability of accurately classifying images using Python, Keras and Jupyter Notebooks. This project will compare different types of network, architectural choices, hyper-parameters and other techniques to enhance the performance and generalisation of the model.

2. Background

This section aims to explain the reasoning behind the main techniques used throughout this project, demonstrating literature influence on the techniques selection. As introduced, the task at hand is a multi-class image classification of the CIFAR-10 dataset and the first question that arises for the execution of it is the choice of type of network that will be employed.

From Feedforward Neural Networks, to Recurrent Neural Networks and many other types, the Convolutional Neural Networks (CNN) was the one elected for this project. Chollet et al (2018) introduces convolutional neural networks as a type of deep learning model almost universally used in computer vision applications.

For a better historical contextualisation, Chollet (2017) explains:

In 2011, Dan Ciresan from IDSIA began to win academic image-classification competitions with GPU-trained deepIn 2011, Dan Ciresan from IDSIA began to win academic image-classification competitions with GPU-trained deep neural networks—the first practical success of modern deep learning. But the watershed moment came in 2012, with the entry of Hinton's group in the yearly large-scale image-classification challenge ImageNet. The ImageNet challenge was notoriously difficult at the time, consisting of classifying high-resolution colour images into 1,000 different categories after training on 1.4 million images. In 2011, the top-five accuracy of the winning model, based on classical approaches to computer vision, was only 74.3%. (...). The competition has been dominated by deep convolutional neural networks every year since. By 2015, the winner reached an accuracy of 96.4%, and the classification task on ImageNet was considered to be a completely solved problem.

Chollet (2017) also states that "since 2012, deep convolutional neural networks (convnets) have become the go-to algorithm for all computer vision tasks; more generally, they work on all perceptual tasks". This is the case because of the ability of CNNs to learn relevant features from raw pixel data and capacity to handle big image datasets.

According to Géron (2019):

Convolutional neural networks (CNNs) emerged from the study of the brain's visual cortex, and they have been used in image recognition since the 1980s. In the last few years, thanks to the increase in computational power, the amount of available training data, (...), CNNs have managed to achieve superhuman performance on some complex visual tasks. They power image search services, self-driving cars, automatic video classification systems, and more.

The CNN architecture of its convolutional layer "allows the network to concentrate on small low-level features in the first hidden layer, then assemble them into larger higher-level features in the next hidden layer, and so on (...), which is one of the reasons why CNNs work so well for image recognition” (Géron, 2019).

Considering the arguments outlined above, the Convolutional Neural Network was the type of network that seemed most appropriate, and therefore selected, for the multi-class image classification task presented in this project.

Moving forward we begin with the preprocessing of the data and proceed to the creation of the base model, with its most important building block consisting of adding the convolutional layers, then “to aggressively reduce dimensionality of feature maps and sharpen the located features, we (...) insert a max pooling layer after a convolutional layer” (Buduma, 2017).

Towards the end a “flatten” layer is added to transform the output into a compatible format for the fully connected layers. Additionally, to handle vanishing gradients and improve the model performance we will add dense layers and an output layer with a softmax activation function, since it is, as explained by Chollet (2019), a “multiclass classification problem, your network should end with a softmax activation so that it will output a probability distribution over the N output classes”.

Concluding the base model, we will next create subsequent models to test various architecture choices, changing the number of convolutional layers (the depth of the network), number of pooling layers, and other hyperparameters to using hyper-parameter optimisation techniques to improve the performance and generalisation of the model, which will be discussed in more detail in the methodology section below.

Data augmentation and regularisation techniques will also be included, the dropout method is used as it “is one of the most effective and most commonly used regularisation techniques for neural networks” (Chollet, 2017), “even the state-of-the-art neural networks get a 1– 2% accuracy boost simply by adding dropout” (Géron, 2019). Batch normalisation will also be added, as it “lets the model learn the optimal scale and mean of each of the layer’s inputs” (Géron, 2019). Finally, data augmentation is applied to the models to generate more training data from existing training samples, thus preventing the model from overfit (Chollet, 2017).

3. Methodology & Experiments

The code for this project (Appendix A) was developed in a structured approach. The methodology and experimentation involved will be elucidated below.

3.1 Programming Language and Framework

The programming language chosen was Python, due to its versatility and extensive libraries, it has a great performance for data science. Here are the libraries and versions used:

Python version: 3.11.2 (main, May 6 2023, 17:16:28) [GCC 11.3.0]
Numpy version: 1.23.5
Pandas version: 2.0.2
SKLearn version: 1.2.2
Tensorflow version: 2.12.0
Matplotlib version: 3.7.1
Seaborn version: 0.12.2
Keras version: 2.12.0

3.2 Dataset Description

The dataset used for this project is the CIFAR-10 (Canadian Institute for Advanced Research 10), it became popular in the field of machine learning and computer vision (Krizhevsky, 2009):

> *The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.*

The CIFAR-10 is still a valuable resource for research and practice of training and evaluating image classification models, comparing approaches and performances, especially because this dataset presents a challenge due to its relatively small images and similarity between some classes.

3.3 Code Structure and Implementation

The code implemented used the Sequential API in Keras, a common approach to building deep learning models for its simplicity, sequential flow - which makes it optimal for a CNN model - and for better readability of the code. The holdout method was used for the data splitting into the training set and test set.

For feature engineering techniques, normalising the data was employed to ensure compatibility and value normalisation, aiding the generalisation and convergence of the model Since the pixel values typically range from 0 to 255, dividing the features data scales it to a range of 0 to 1. The target data was then one-hot-encoded, converting the vectors into binary.

The code was developed on a Jupyter Notebook environment, making use of its ease for an interactive development and data visualisation possibilities.

The code has an organisational separation:

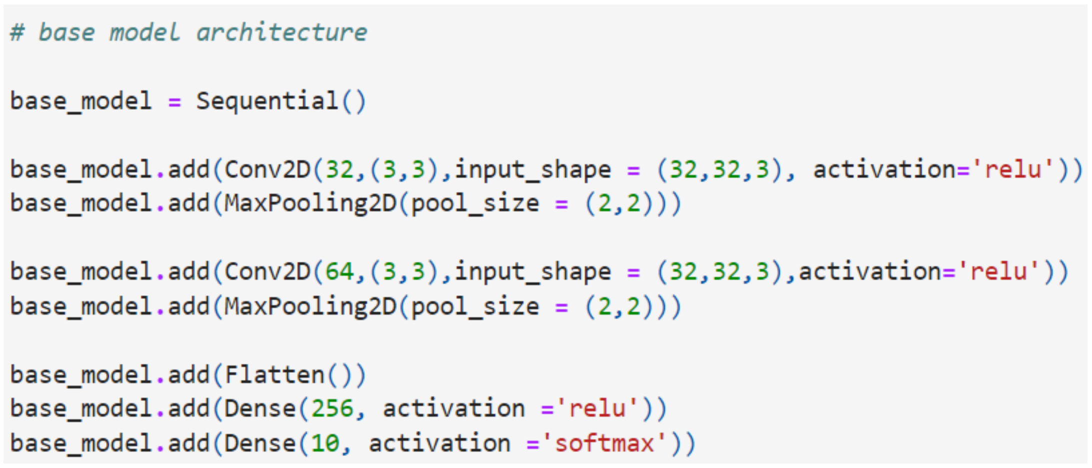
- Importing libraries: get the code requirements and check the versions.
- Loading and preprocessing the Data: importing and exploring the data for the project; and data normalising.
- Models: experimentation by creating and evaluating models with different architectures, each model adds or changes a hyperparameter.
- Data Augmentation: generating more images and evaluating a model fitted on the augmented data.
- Evaluation: comparison of all models, their architecture and performance.

3.4 Models Architecture

3.4.1 Base Model

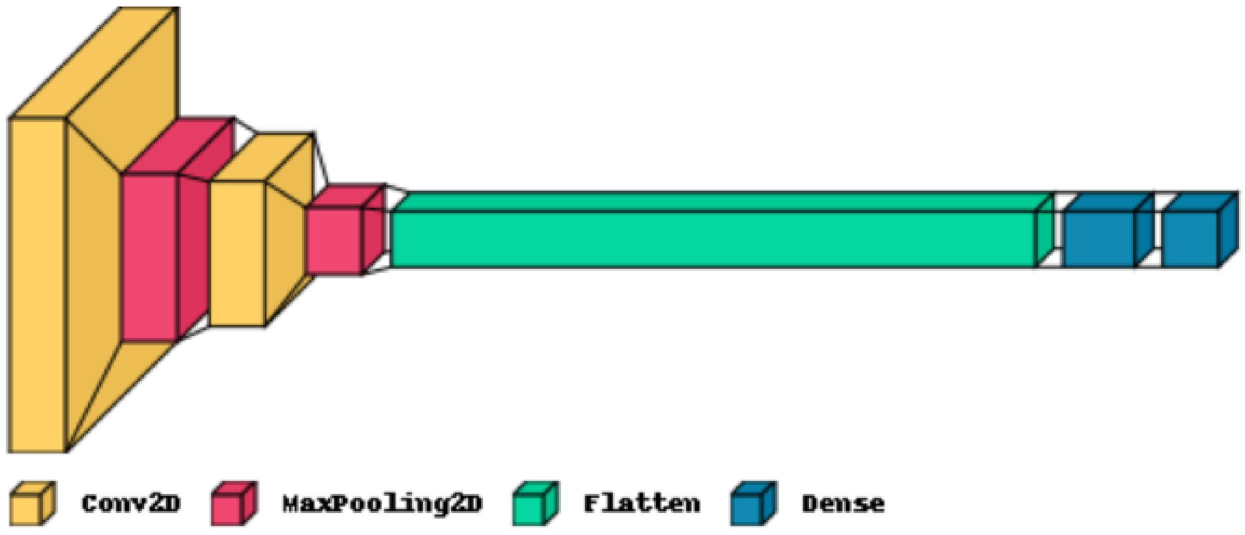
Once the deep learning model for this project was selected - CNN -, a base model was created to improve upon the next steps. This model includes two convolutional layers followed by a pooling layer, a flatten layer, and a dense layer:

Figure 1 - Base Model Architecture



```
# base model architecture

base_model = Sequential()

base_model.add(Conv2D(32,(3,3),input_shape = (32,32,3), activation='relu'))
base_model.add(MaxPooling2D(pool_size = (2,2)))

base_model.add(Conv2D(64,(3,3),input_shape = (32,32,3),activation='relu'))
base_model.add(MaxPooling2D(pool_size = (2,2)))

base_model.add(Flatten())
base_model.add(Dense(256, activation ='relu'))
base_model.add(Dense(10, activation ='softmax'))
```

Figure 2 - Base Model Layers



The first convolutional layer (Conv2D) is the layer utilised to extract features from the image, the format used is 2D since we are working with colour image data, “one channel contains the red pixels, one the green pixels, and one the blue pixels. The convolution kernel moves over both the horizontal and the vertical axes of the image, conferring translation equivariance in both directions” (Goodfellow, 2016). This layer has as hyperparameters 32 filters, a kernel size of (3,3), and the activation being ReLu, as explains Buduma (2017), “the ReLU has recently become the neuron of choice for many tasks (especially in computer vision)”. This layer is followed by a pooling layer, MaxPooling2D, that scales down the size of the image.

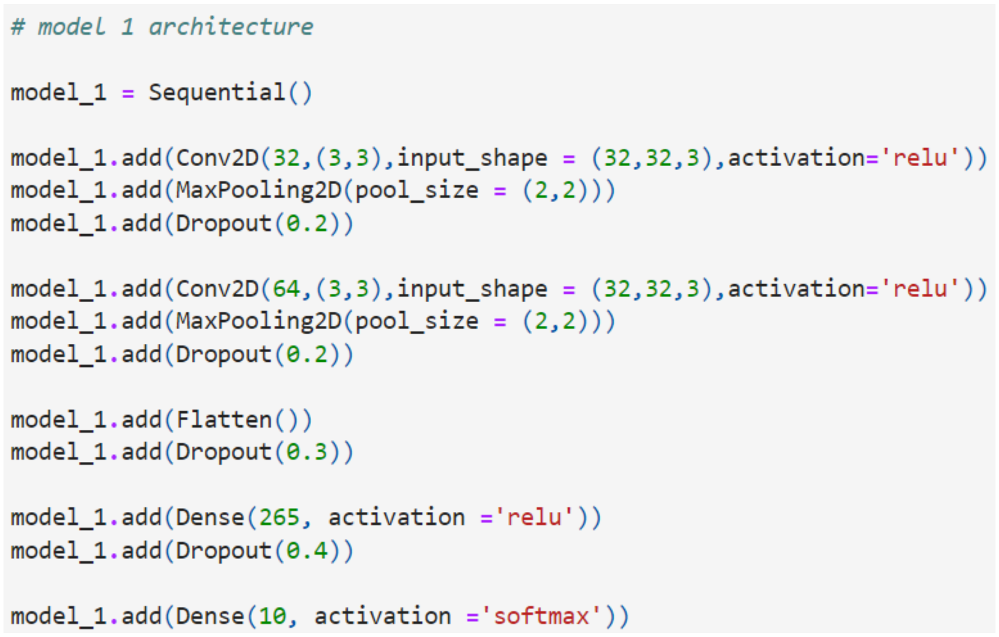
Moving to the second convolutional layer is similar to the first, only increasing the number of filters, it is also followed by a pooling layer.

Finally, to fully connect the classifier, a flatten layer is added to convert the dimension, and a dense layer, with its first inner layer having 256 neurons. The last layer is the output, using 10 neurons, here the softmax activation is used to predict the class, this activation “will ensure that all the estimated probabilities are between 0 and 1 and that they add up to 1 (which is required if the classes are exclusive)” (Géron, 2019), making it an optimal activation for our task.

3.4.2 Model 1

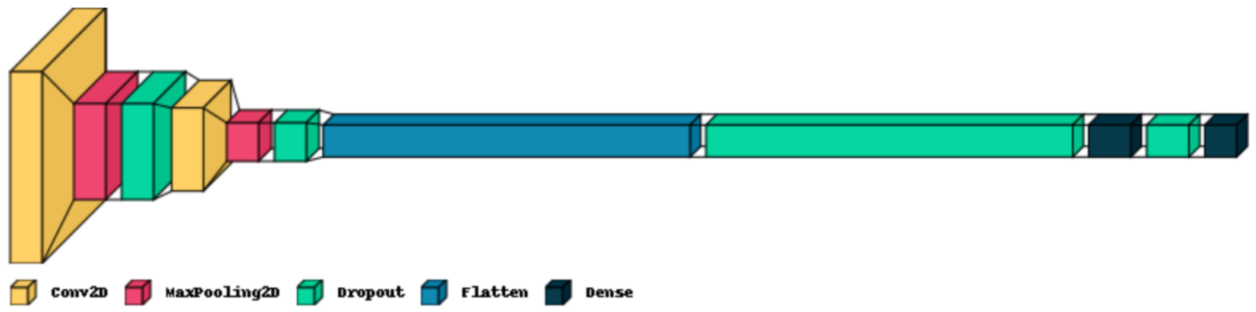
The following model builds upon the base model, adding dropout layers. As explained by Chollet (2017), “dropout is one of the most effective and most commonly used regularisation techniques for neural networks, developed by Geoff Hinton and his students at the University of Toronto''.. The dropout rate is “usually set between 0.2 and 0.5” (Chollet, 2017) and “closer to 40– 50% in convolutional neural networks” (Géron, 2019).

Figure 3 - Model 1 Architecture



```
# model 1 architecture

model_1 = Sequential()

model_1.add(Conv2D(32,(3,3),input_shape = (32,32,3),activation='relu'))
model_1.add(MaxPooling2D(pool_size = (2,2)))
model_1.add(Dropout(0.2))

model_1.add(Conv2D(64,(3,3),input_shape = (32,32,3),activation='relu'))
model_1.add(MaxPooling2D(pool_size = (2,2)))
model_1.add(Dropout(0.2))

model_1.add(Flatten())
model_1.add(Dropout(0.3))

model_1.add(Dense(265, activation ='relu'))
model_1.add(Dropout(0.4))

model_1.add(Dense(10, activation ='softmax'))
```

Figure 4 - Model 1 Layers



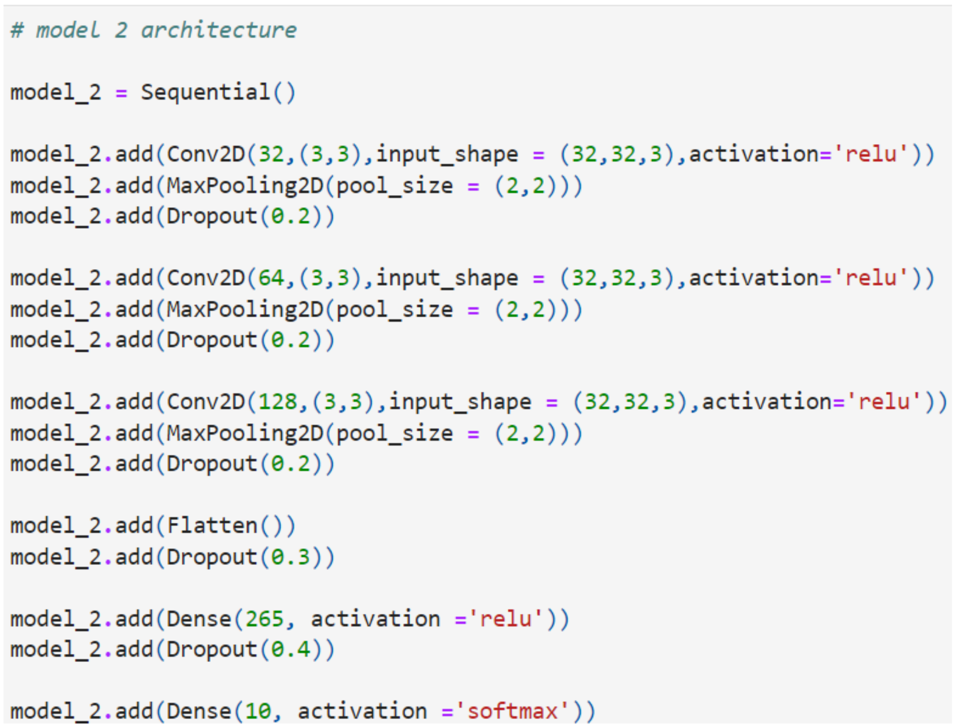
3.4.3 Model 2

The architecture for Model 2, created upon the base model and model 1, counts with a batch normalisation layer added after each Conv2D layer.

"Batch normalisation (Ioffe and Szegedy, 2015) is one of the most exciting recent innovations in optimising deep neural networks, and it is actually not an optimisation algorithm at all. Instead, it is a method of adaptive reparametrization, motivated by the difficulty of training very deep models" (Goodfellow, 2016).

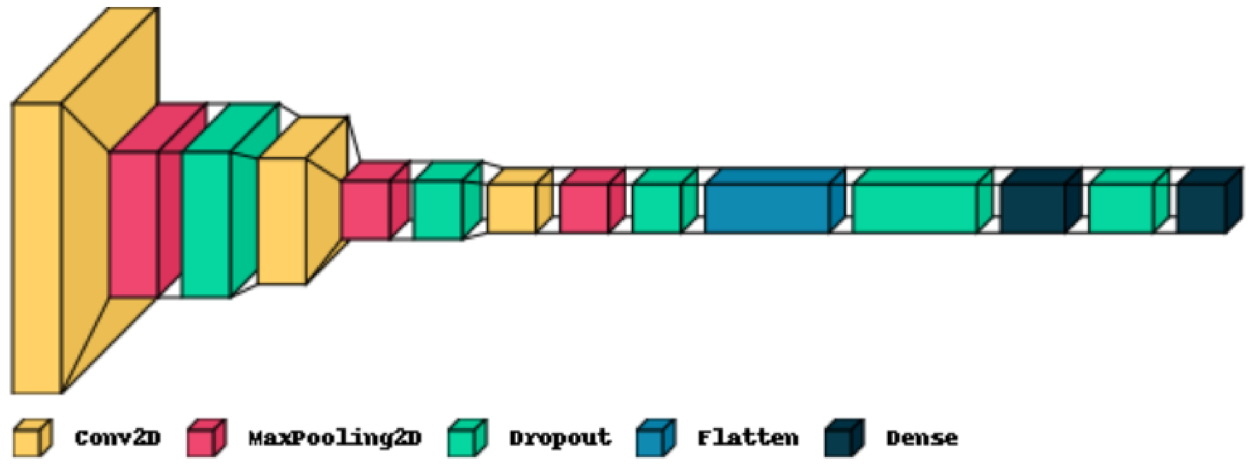
These extra layers should improve the training and performance of the model by normalising the inputs; it addresses the changes in distribution of the network activations during training. It benefits the model by improving the training speed and generalisation of the model.

Figure 5 - Model 2 Architecture



```
# model 2 architecture

model_2 = Sequential()

model_2.add(Conv2D(32,(3,3),input_shape = (32,32,3),activation='relu'))
model_2.add(MaxPooling2D(pool_size = (2,2)))
model_2.add(Dropout(0.2))

model_2.add(Conv2D(64,(3,3),input_shape = (32,32,3),activation='relu'))
model_2.add(MaxPooling2D(pool_size = (2,2)))
model_2.add(Dropout(0.2))

model_2.add(Conv2D(128,(3,3),input_shape = (32,32,3),activation='relu'))
model_2.add(MaxPooling2D(pool_size = (2,2)))
model_2.add(Dropout(0.2))

model_2.add(Flatten())
model_2.add(Dropout(0.3))

model_2.add(Dense(265, activation ='relu'))
model_2.add(Dropout(0.4))

model_2.add(Dense(10, activation ='softmax'))
```

Figure 6 - Model 2 Layers

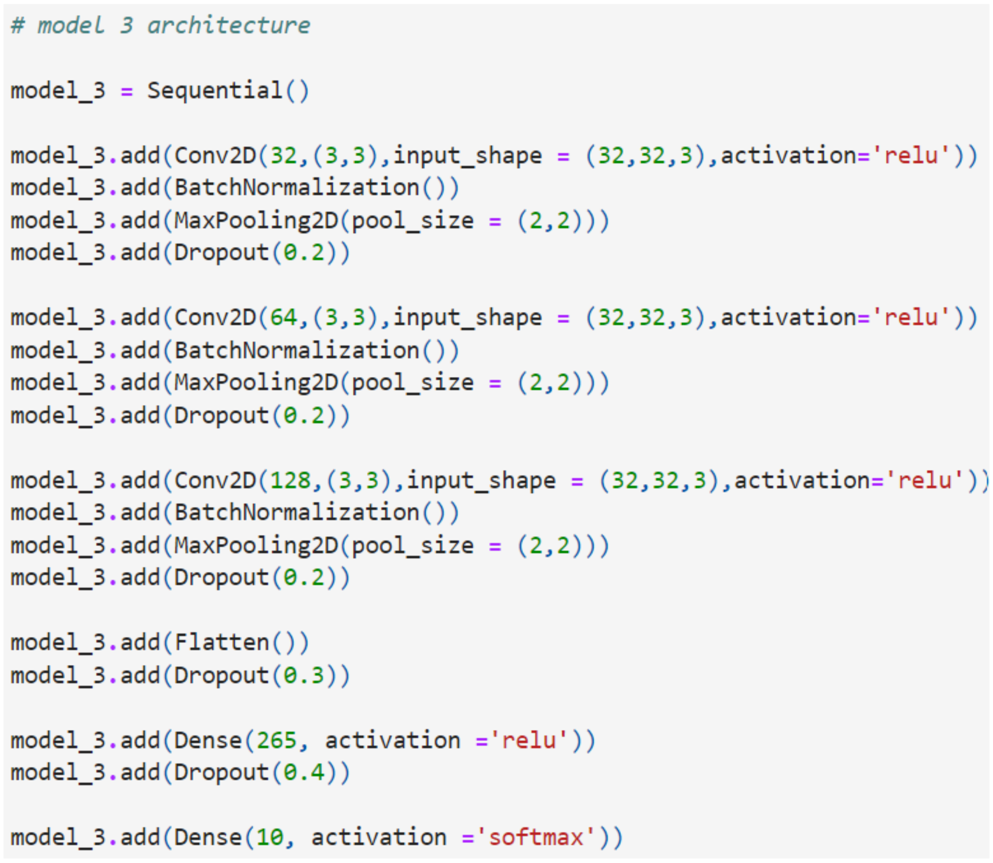


3.4.4 Model 3

After Model 2 was completed we tested giving the model another convolutional layer group to evaluate how much, if at all, these extra layers can improve our architecture. Since the impact of more layers depends on the dataset and specificities of each task or problem the layer was added to experiment with our models performance, if successful it should increase the capacity of the model to capture more complex patterns and features.

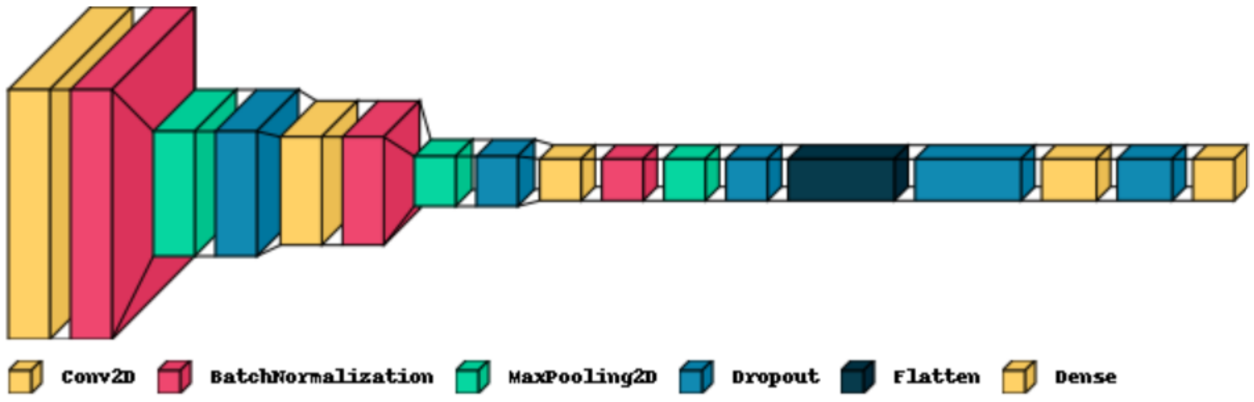
Adding more convolutional layers can, however, lead the model to overfit. This problem should be fairly mitigated by this stage since our model already counts with dropout and batch normalisation layers. The model architecture now is updated as follows:

Figure 7 - Model 3 Architecture



```
# model 3 architecture

model_3 = Sequential()

model_3.add(Conv2D(32,(3,3),input_shape = (32,32,3),activation='relu'))
model_3.add(BatchNormalization())
model_3.add(MaxPooling2D(pool_size = (2,2)))
model_3.add(Dropout(0.2))

model_3.add(Conv2D(64,(3,3),input_shape = (32,32,3),activation='relu'))
model_3.add(BatchNormalization())
model_3.add(MaxPooling2D(pool_size = (2,2)))
model_3.add(Dropout(0.2))

model_3.add(Conv2D(128,(3,3),input_shape = (32,32,3),activation='relu'))
model_3.add(BatchNormalization())
model_3.add(MaxPooling2D(pool_size = (2,2)))
model_3.add(Dropout(0.2))

model_3.add(Flatten())
model_3.add(Dropout(0.3))

model_3.add(Dense(265, activation ='relu'))
model_3.add(Dropout(0.4))

model_3.add(Dense(10, activation ='softmax'))
```

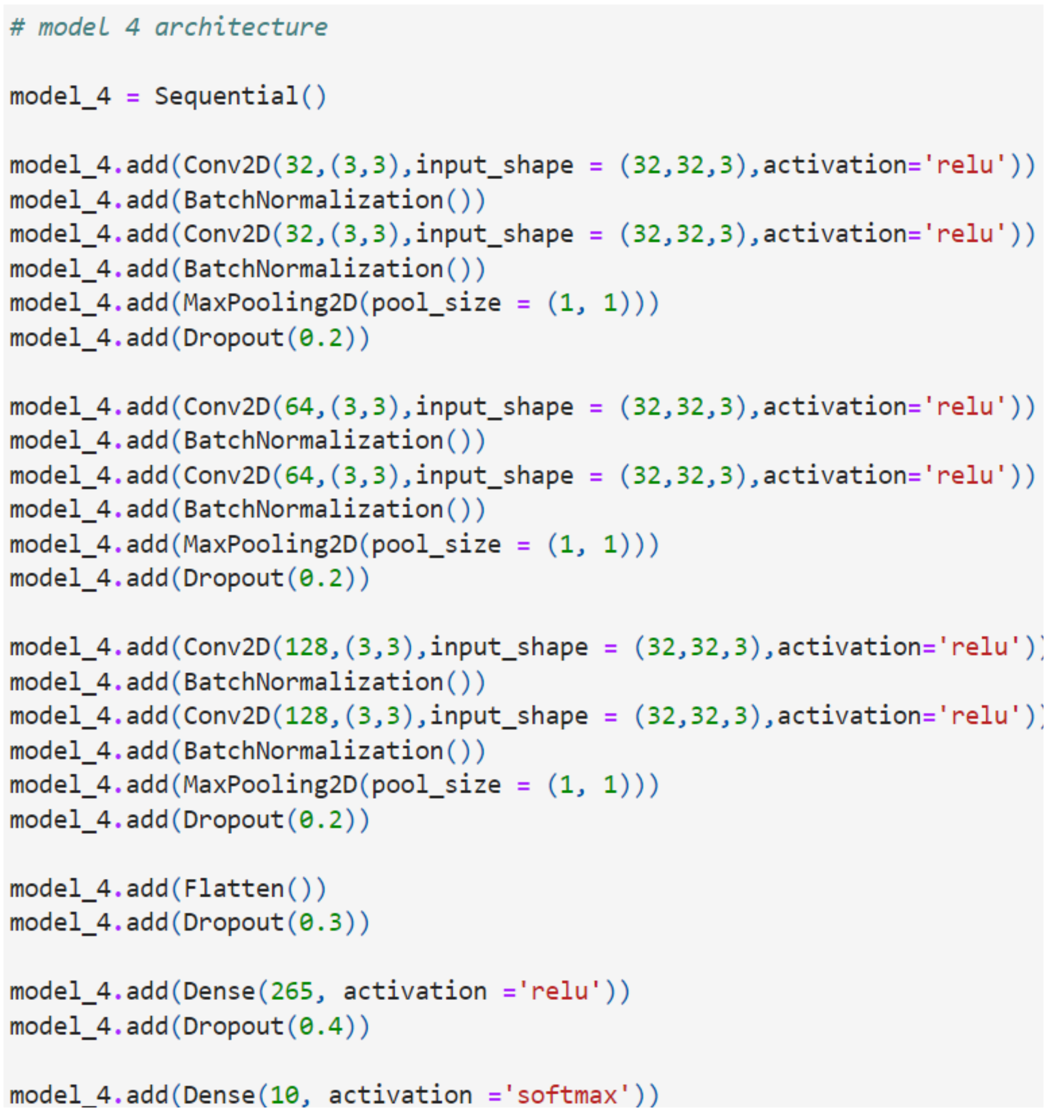
Figure 8 - Model 3 Layers



3.4.5 Model 4

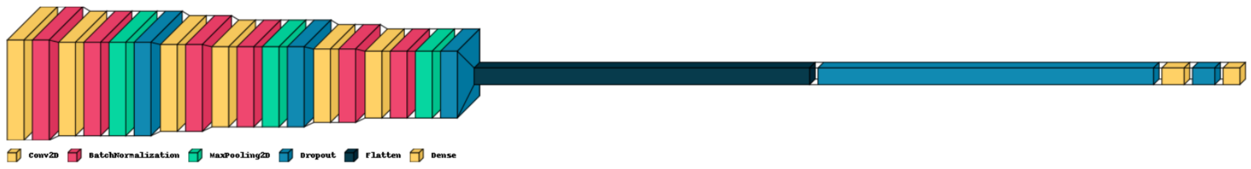
Model 4, built upon the previous models, adds a Conv2D layer inside of each convolutional layers group, it increases the model complexity and ability to capture better features in the data. It can, again, lead to overfitting of the model. Therefore the Model 4 is created to test and experiment with complexity and fitting of the network.

Figure 9 - Model 4 Architecture



```
# model 4 architecture

model_4 = Sequential()

model_4.add(Conv2D(32,(3,3),input_shape = (32,32,3),activation='relu'))
model_4.add(BatchNormalization())
model_4.add(Conv2D(32,(3,3),input_shape = (32,32,3),activation='relu'))
model_4.add(BatchNormalization())
model_4.add(MaxPooling2D(pool_size = (1, 1)))
model_4.add(Dropout(0.2))

model_4.add(Conv2D(64,(3,3),input_shape = (32,32,3),activation='relu'))
model_4.add(BatchNormalization())
model_4.add(Conv2D(64,(3,3),input_shape = (32,32,3),activation='relu'))
model_4.add(BatchNormalization())
model_4.add(MaxPooling2D(pool_size = (1, 1)))
model_4.add(Dropout(0.2))

model_4.add(Conv2D(128,(3,3),input_shape = (32,32,3),activation='relu')
model_4.add(BatchNormalization())
model_4.add(Conv2D(128,(3,3),input_shape = (32,32,3),activation='relu')
model_4.add(BatchNormalization())
model_4.add(MaxPooling2D(pool_size = (1, 1)))
model_4.add(Dropout(0.2))

model_4.add(Flatten())
model_4.add(Dropout(0.3))

model_4.add(Dense(265, activation ='relu'))
model_4.add(Dropout(0.4))

model_4.add(Dense(10, activation ='softmax'))
```

Figure 10 - Model 4 Layers



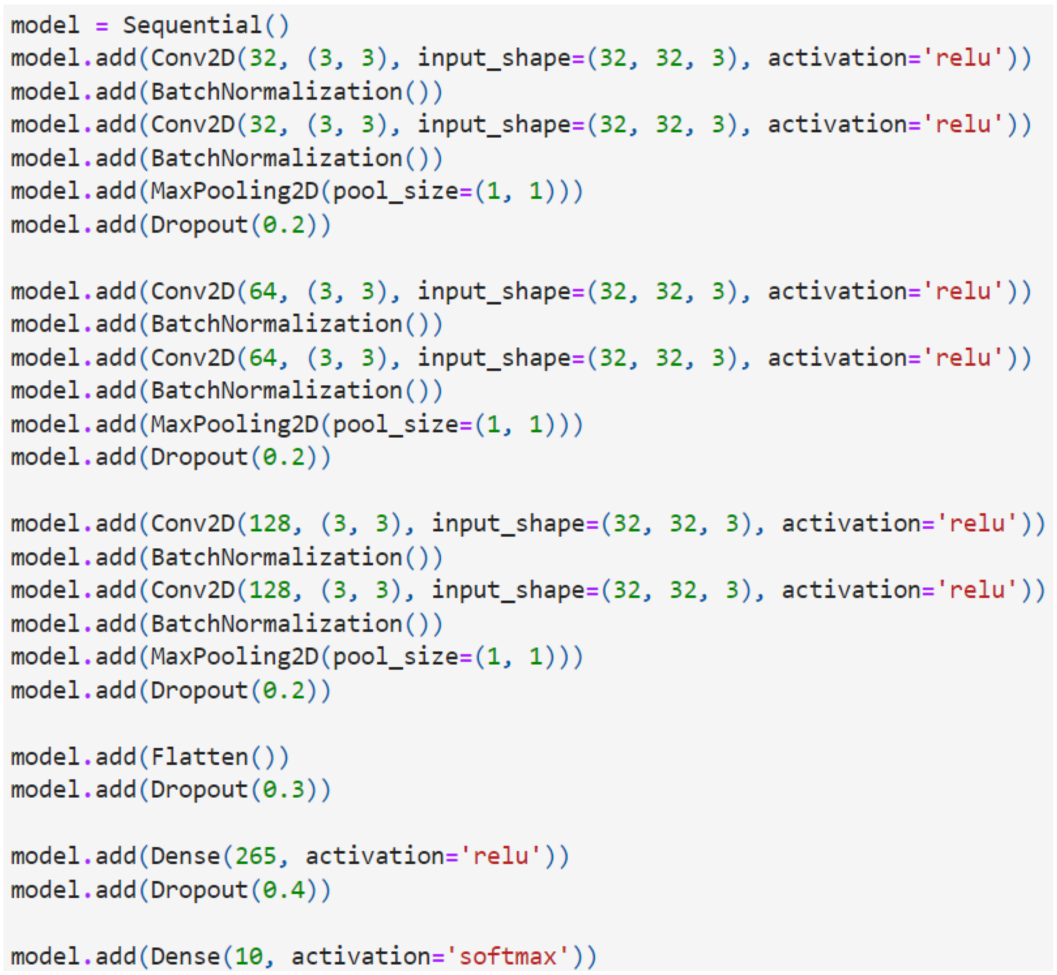
3.4.6 Model 5 - Data Augmentation

With Model 5 data augmentation is introduced to the project, another regularisation technique to prevent overfitting in computer vision. This is because with data augmentation we introduce to the model realistic new images, increasing the training set.

According to Chollet (2017), “overfitting is caused by having too few samples to learn from, rendering you unable to train a model that can generalise to new data. Given infinite data, your model would be exposed to every possible aspect of the data distribution at hand: you would never overfit. Data augmentation takes the approach of generating more training data from existing training samples, by augmenting the samples via a number of random transformations that yield believable-looking images. The goal is that at training time, your model will never see the exact same picture twice. This helps expose the model to more aspects of the data and generalise better”.

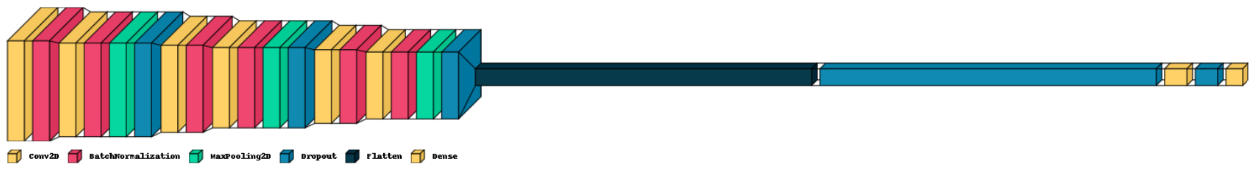
The architecture for model 5 is identical to model 4, the differentiation comes with the training of the model, the 5th model uses the augmented data for its training. The data augmentation features image rotation, width shift range, height shift range and horizontal flip.

Figure 11 - Model 5 Architecture



```
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(32, 32, 3), activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(32, (3, 3), input_shape=(32, 32, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(1, 1)))
model.add(Dropout(0.2))

model.add(Conv2D(64, (3, 3), input_shape=(32, 32, 3), activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(64, (3, 3), input_shape=(32, 32, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(1, 1)))
model.add(Dropout(0.2))

model.add(Conv2D(128, (3, 3), input_shape=(32, 32, 3), activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(128, (3, 3), input_shape=(32, 32, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(1, 1)))
model.add(Dropout(0.2))

model.add(Flatten())
model.add(Dropout(0.3))

model.add(Dense(265, activation='relu'))
model.add(Dropout(0.4))

model.add(Dense(10, activation='softmax'))
```

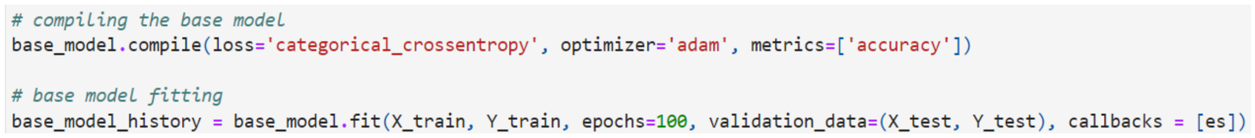
Figure 12 - Model 5 Layers



3.6 Training Procedure

When compiling the models the following parameters were chosen:

Figure 13 - Compile and Fit



```
# compiling the base model
base_model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# base model fitting
base_model_history = base_model.fit(X_train, Y_train, epochs=100, validation_data=(X_test, Y_test), callbacks = [es])
```

Categorical_crossentropy was chosen as the loss parameter, “crossentropy is usually the best choice when you’re dealing with models that output probabilities (...) when it comes to common problems such as classification (...) there are simple guidelines you can follow to choose the correct loss. For instance, you’ll use (...) categorical crossentropy for a many-class classification problem” (Chollet, 2017).

According to Goodfellow (2016), “the most popular optimization algorithms actively in use include (...) Adam. The choice of which algorithm to use, at this point, seems to depend largely on the user’s familiarity with the algorithm (for ease of hyperparameter tuning)”. Adam is also used as the optimizer in many examples by Géron (2019), thus, for this project in hand Adam was chosen as the optimizer.

The models were trained on 100 epochs but the models never actually reached that number during their fitting, that is because the EarlyStopping function was added to the code as an advanced deep learning best practice (Chollet, 2017):

> *When you’re training a model, there are many things you can’t predict from the start. In particular, you can’t tell how many epochs will be needed to get to an optimal validation loss (...) You can use the EarlyStopping callback to interrupt training once a target metric being monitored has stopped improving for a fixed number of epochs.*

3.7 Control and Reproducibility

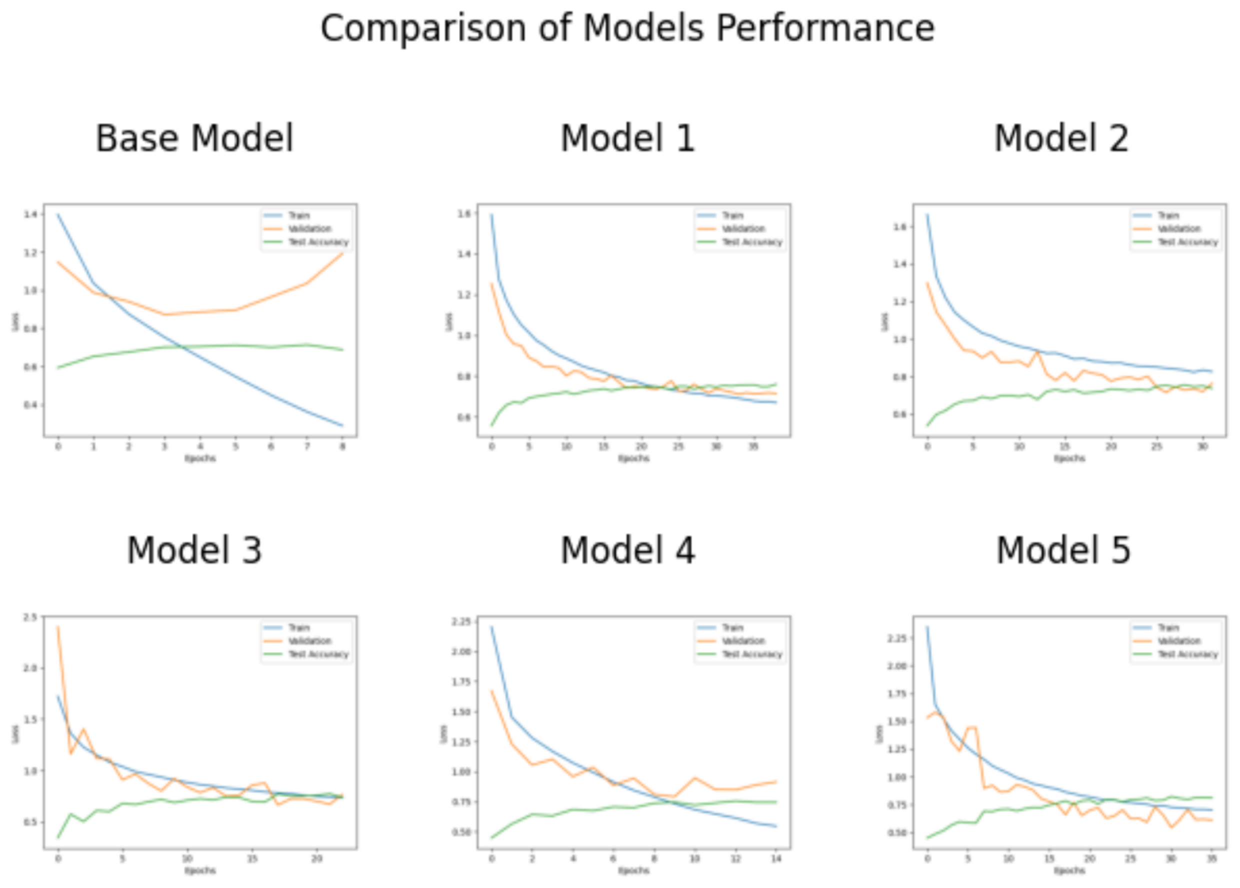
A remote repository was created on GitHub using Git as backup and made available as a public repository: https://github.com/ndressler/Masters_U.Sunderland/blob/main/Multi-class%20Image%20Classification%20of%20the%20CIFAR10%20using%20Convolutional%20Neural%20Network%20(CNN).ipynb

The code for this project was also included as a supporting document (see Appendices).

4. Results

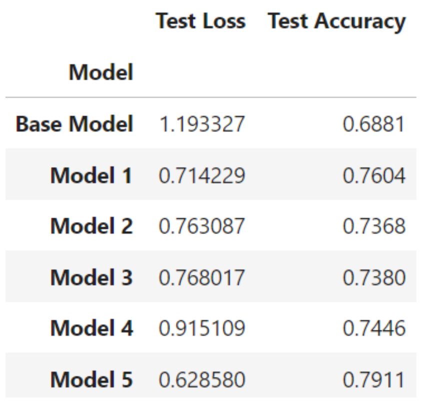
Here are the results for the performance of each model:

Figure 14 - Performance Graph



On the table below (Table 1), we see the results of each model by their test loss and test accuracy. Is also important to note the epoch that the early stopping - when the models performance stopped to improve - took place. For the base model: epoch 9; Model 1, epoch 39; Model 2, 32; Model 3, 23; Model 4, 15; and finally model 5 at epoch 36.

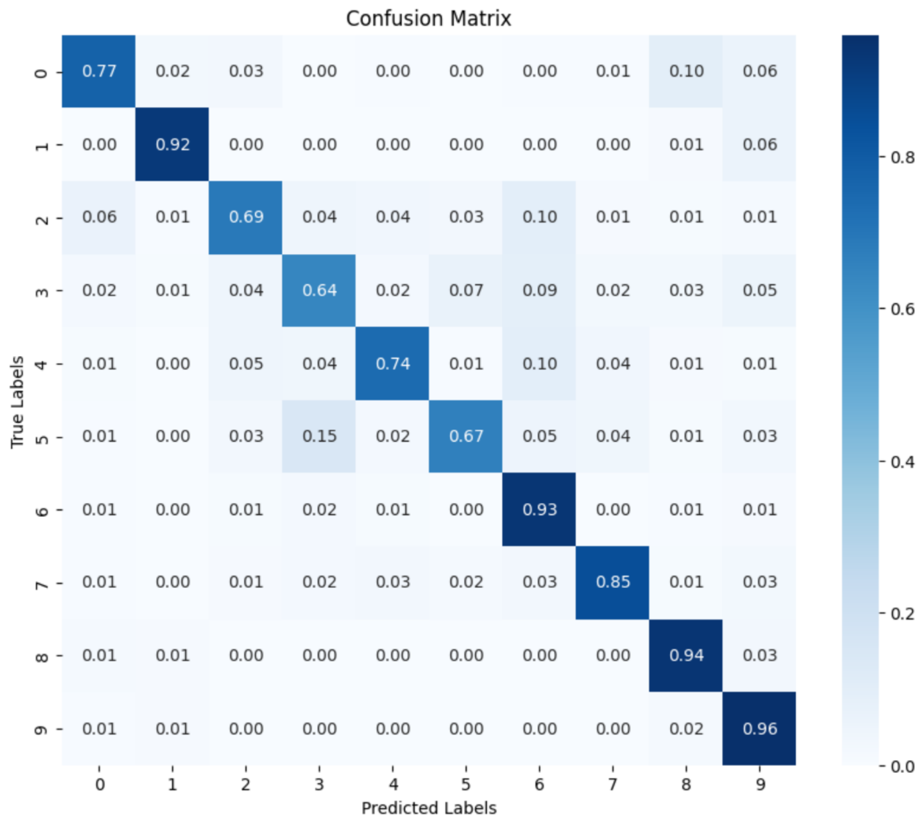
Table 1 - Models Performance



	Test Loss	Test Accuracy
Model		
Base Model	1.193327	0.6881
Model 1	0.714229	0.7604
Model 2	0.763087	0.7368
Model 3	0.768017	0.7380
Model 4	0.915109	0.7446
Model 5	0.628580	0.7911

Multiple models with different hyperparameters were made to provide a comparison between the different constructions of the model by evaluating their accuracy. The best results were seen in model 5, having the lowest test loss and highest test accuracy. A heatmap of predictions from the ebay performing model was also generated to show its accuracy.

Figure 15 - Heatmap of Model 5



	0	1	2	3	4	5	6	7	8	9
0	0.77	0.02	0.03	0.00	0.00	0.00	0.00	0.01	0.10	0.06
1	0.00	0.92	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.06
2	0.06	0.01	0.69	0.04	0.04	0.03	0.10	0.01	0.01	0.01
3	0.02	0.01	0.04	0.64	0.02	0.07	0.09	0.02	0.03	0.05
4	0.01	0.00	0.05	0.04	0.74	0.01	0.10	0.04	0.01	0.01
5	0.01	0.00	0.03	0.15	0.02	0.67	0.05	0.04	0.01	0.03
6	0.01	0.00	0.01	0.02	0.01	0.00	0.93	0.00	0.01	0.01
7	0.01	0.00	0.01	0.02	0.03	0.02	0.03	0.85	0.01	0.03
8	0.01	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.94	0.03
9	0.01	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.96

5. Discussion & Conclusion

To recapitulate, the project's goal was to solve a multi-class image classification task. A Convolutional Neural Network was built and many models were compared by optimising its hyperparameters such as number of layers, number of filters for the convolutional layers, adding different regularisation techniques and so on, comparing different types of architectures for the problem at hand.

First, to discuss the evaluative methodology of the task, the priority in this scenario is to achieve the lowest loss with the highest accuracy on the validation test, meaning that the optimal model created prioritises accuracy to be able to predict with certainty the correct label for the images.

The experiment of this project was intended to increase the complexity of the model and improve its generalisation, meaning, for each step a measure of either preventing overfitting - by adding regularisation techniques - or underfitting - by potentialising its predictive capability on new data.

The order of model enhancements was thought as: first, a base model - a very basic CNN to build upon and compare future incrementations; second, adding dropout layers for regularisation - preventing overfitting; third, adding one more convolutional cluster to prevent underfitting; forth, adding batch normalisation layers, again a regularisation technique; fifth, adding another Conv2D layer to each convolutional cluster - making the model more complex; and lastly, the model was trained on augmented data, aiding the generalisation of the model.

As the results made clear, the base model is overall the worst performing, having the highest loss, lowest accuracy. After that, the dropout method was seen to make significant improvement, by lowering the loss in around 23% and 10% when analysing accuracy.

The validation loss fluctuates with the progression of the models as does the accuracy score, the higher the loss the more our model is overfitting, which is the case between models 1 to 4, until the data augmentation was implemented. The results from Model 5, with data augmentation proved to be the best overall.

We can notice that adding complexity to the model did not necessarily improve its performance as is noticeable in models 2 and 4, when more convolutional layers were added. The inclusion of batch normalisation made no sustainable difference for the model, as could be explained since the model was normalised during preprocessing.

Ultimately, the methods that most improved our model were data augmentation and dropout layers in models 1 and model 5, these models also had the longest run in epochs, meaning that they continued to improve much after other models had already stopped improving. Another interesting point is that adding to the models' complexity did not seem to bring the expected improvement results. In conclusion, from the first to the last model we see an improvement of 47% in test loss and 14% in accuracy.

The CNN model created for this project was developed on a very average performing computer, it can be greatly improved by running and testing an even more complex model. More layers can be stacked; the number of units and filters can be higher; different activation functions such as leaky_relu, elu, prelu and so on, and different optimizers such as SDG, can be experimented on; different rates for dropout, learning rate, and so on.

The model can certainly be made more complex - constantly, of course, being attentive to overfitting. It's always necessary to take into consideration aspects other than the model itself such as the computing power available, and the equilibrium between a satisfactory result and the time and power spent on the creation of the model.

6. Reflection on Ethical use of AI

The field of Deep Learning has grown exponentially over the past years, and with its progress, many deeds never thought possible are now a reality. The rapid advancements in computer vision have made significant developments in our society.

One major example of AI models on computer vision is in the field of transportation, so much so that now cars are able to travel without the need for a human driver behind the wheel, providing safer, more efficient, and environmentally friendly transportation options.

According to many statistics gathered by Gitnux (2023), autonomous cars bring a great positive impact to our society by improving significantly road safety; providing an important increase on mobility accessibility - especially for the elderly and people with disabilities; reducing considerably traffic emissions; improving productivity and lowering stress levels by reducing commuting time; and numerous further advantages.

However, this technology raises many ethical dilemmas, like data privacy and cybersecurity. In light of the fact that a vast amount of personal data is collected and processed by the vehicles and all technology they equip is liable to hacking, the risk of a cyberattack for gaining access to sensitive data, not to mention more serious infringements and even becoming a threat to life, cannot be ignored.

Moreover, in case of an accident with a self-driving car, it is necessary to determine the liability, if it should be aggregated to the owner of the vehicle, or perhaps the company responsible for the algorithm. In case the liability goes to the vehicle rather than the “driver” itself, another challenge arises, that is of the need for a licence to operate said vehicles. Perhaps the driving licence as we know will have alterations on allowance and also on the group of people that is allowed today to possess one. According to Joshi (2022), Germany already is on its way to a clearer overview on those points, it ruled that human lives should be prioritised above anything else.

Given the heightened attention and relevance surrounding autonomous vehicles, this example was brought to elucidate the subject of data ethics and the role artificial intelligence has on it. Ultimately, the use of the AI technology must always come along with a serious attention and investigation to its ethical consequences and challenges, for as the first cannot exist in harmony with society without the second.

7. References

Buduma, N. (2017) Fundamentals of Deep Learning. O’Reilly Media, Inc.

Chollet, F. (2018) Deep Learning with Python. New York, NY: Manning Publications.

Géron, A. (2019) Hands-On Machine Learning with Scikit-Learn, Keras & Tensorflow. O’Reilly Media, Inc.

Gitnux. 2023. Self Driving Cars Statistics And Trends in 2023. Gitnux, [Online, viewed 21/06/2023]. Available at: https://blog.gitnux.com/self-driving-cars-statistics/.

Goodfellow, et al. (2016). Deep Learning, MIT Press.

Joshi, N. 2022. 5 Moral Dilemmas That Self-Driving Cars Face Today. Forbers, [Online, viewed 21/06/2023]. Available at:
https://www.forbes.com/sites/naveenjoshi/2022/08/05/5-moral-dilemmas-that-self-driving-cars-face-today/.

Krizhevsky, A. and Hinton, G. (2009). Learning multiple layers of features from tiny images.

Krizhevsky, A. and Hinton, G. (2009a). CIFAR-10 and CIFAR-100 Dataset, Toronto.edu. Available at: https://www.cs.toronto.edu/~kriz/cifar.html.

Moro, S., Laureano, R. and Cortez, P. (2014) Bank Marketing Data Set, UCI Machine Learning Repository. Available at: https://archive.ics.uci.edu/ml/datasets/bank+marketing.

8. Appendices

Appendix A: Code for Assignment

The code developed for this assignment is available on the Supplementary Documents.