Introduction

The purpose of this report is to perform classification on the CIFAR10 dataset under different data scenarios. This dataset initially consisted of 60,000 32x32 images in 10 classes, with 6,000 images per class. There are 50,000 training images and 10,000 test images. Figure 1 illustrates a sample of the images, along with their labels.

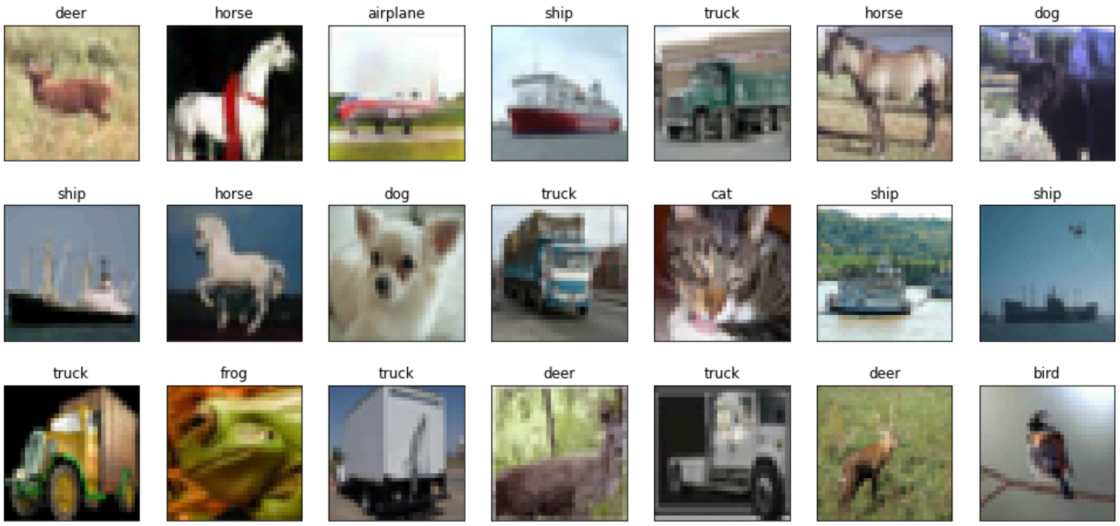

Figure 1. Images and their labels

For our tasks, three distinct datasets have been created:

- The first dataset contains the latter six classes, with 5,000 training samples per class.
- The second dataset contains the first four classes, with 500 training samples per class.
- The third dataset is imbalanced, consisting of the other two datasets combined. Therefore, this dataset contains ten classes, with only 500 samples retained from the first four classes.

For the first part of the project, we have to perform image classification on the six class dataset. Therefore, I will develop a convolutional neural network and train it on the six class data set, and I will use the test data for validation.

For the second part of the project, we have to use the four class and the imbalanced datasets. The purpose is to develop two convolutional neural nets and explore how different model parameters and architectures affect the validation accuracy for each class when trained on each of the datasets. Next, I will discuss possible solutions for situations where we encounter an imbalanced dataset.

The latter part of the project will be the most interesting. I will develop a convolutional neural network and train it on the six class dataset. Then, I will freeze all layers apart from the last one and perform transfer learning on the four class dataset.

Part 1

As stated previously, for the first part of the project I will be using the six class dataset and perform classification on it, and I will be using the test data for validation. For this purpose, I have developed a convolutional neural network with 12 layers using the Keras functional API. I have used four convolutional layers in my network, the first two containing 64 filters and latter two with 32 filters. They all contained the same padding and Relu activation function. After each convolutional layer I have used a pooling layer with default pool size in order to reduce the dimensions of the feature maps. After the input layer and these eight layers, I have used a flattening layer in order to convert the high dimensional tensors into a vector. Then I have used a

dense layer with neurons units and a relu activation function, followed by another dense layer with 6 units and a softmax activation function in order to obtain the probabilities for each class.

In terms of optimisers, I chose Adam. This optimisation algorithm is an extension of stochastic gradient descent to update network weights during training. Unlike maintaining a single learning rate through training in SGD, Adam optimiser updates the learning rate for each network weight individually. The Adam optimisers inherit the features of both Adagrad and RMS prop algorithms. The Adam optimiser has several benefits, due to which it is used widely. It is adapted as a benchmark for deep learning papers and recommended as a default optimisation algorithm. Moreover, the algorithm is straightforward to implement, has faster running time, low memory requirements, and requires less tuning than any other optimisation algorithm. (Gupta, 2021)

Once I settled on the Adam optimiser, with a learning rate of 0.001, I started varying the batch size. I ran the CNN for 100 epochs with a batch size of 512. We can see in Figure 2 the results I obtained. The accuracy on test data was 0.82166. Next, I ran the model again, with a batch size of 256 for 100 epochs, as illustrated in Figure 3. This model appears to overfit faster, at around 30 epochs. The test accuracy is slightly better for the model with 256 batch size, at 0.8315. Figure 4 illustrates the accuracy for the same model, with a batch size of 128. The model

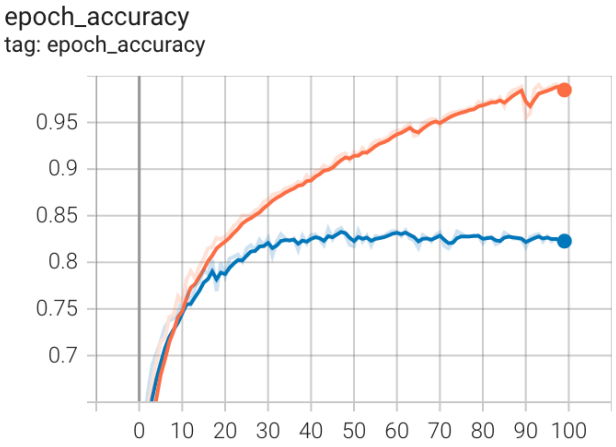

Figure 2. Results with batch size of 512.

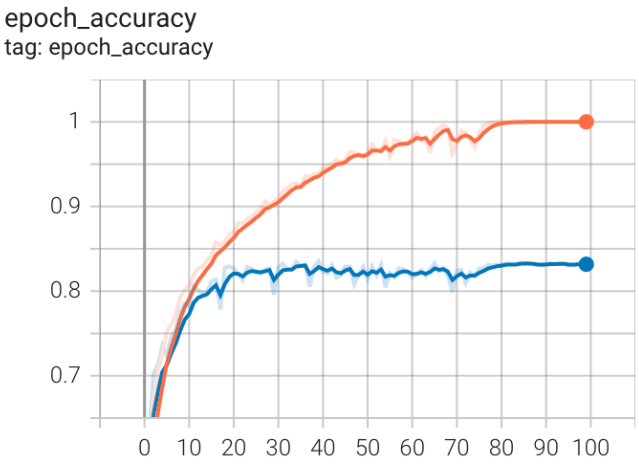

Figure 3. Results with batch size of 256.

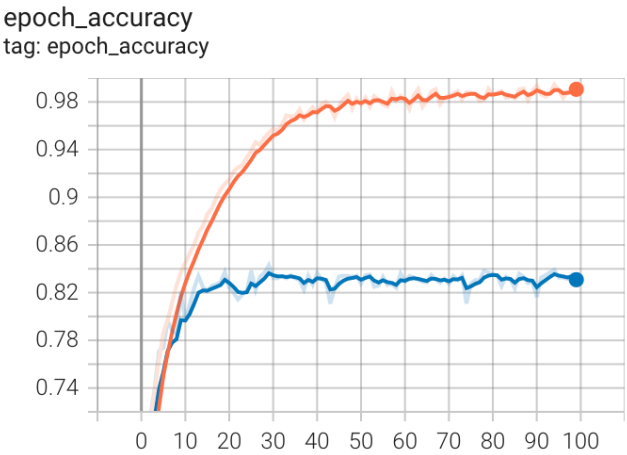

Figure 4. Results with batch size of 128.

converges faster than the other two, at around 20 epochs. The test accuracy is slightly lower than the one with the batch size of 256, at 0.8266. Based on this, moving forward I will be settling for the batch size of 256.

Next, I will will be investigating which learning rate is the most appropriate for our model. So far, I have used a learning rate of 0.001, therefore I will attempt to use a learning rate of 0.01 and

assess the results. Figure 5 illustrates the results for the model with the higher learning rate. As we can see, we obtain poorer results than previously, as the validation accuracy seems to settle around 0.75 after the first 20 epochs. The loss for the validation data appears to increase after 20 epochs which is a sign of overfitting, therefore less training epochs should be considered. Based on this and the fact that the test accuracy is considerably higher for a learning rate of 0.001, I am inclined to stick to that one.

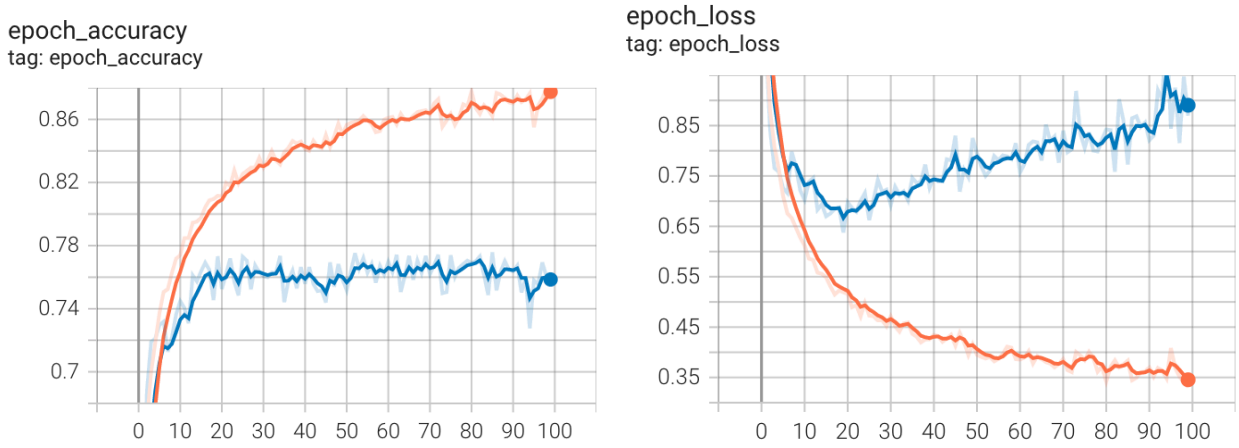

Figure 5. Results for learning rate of 0.01.

I have also considered a learning rate of 0.0001 and trained the model for 200 epochs. Figure 6 depicts the results. As we can see, the validation accuracy seems to settle around 0.82 after 160 training epochs, therefore I think 160 epochs would be the correct choice for this learning rate. The loss seems to steadily decrease on the training dataset, but it seems to stagnate on the validation data.
Lastly, I am presenting the plots for training and validation accuracy, as well as the loss for both datasets in Figure 7. For this model, I have selected a learning rate of 0.001 and I believe the right

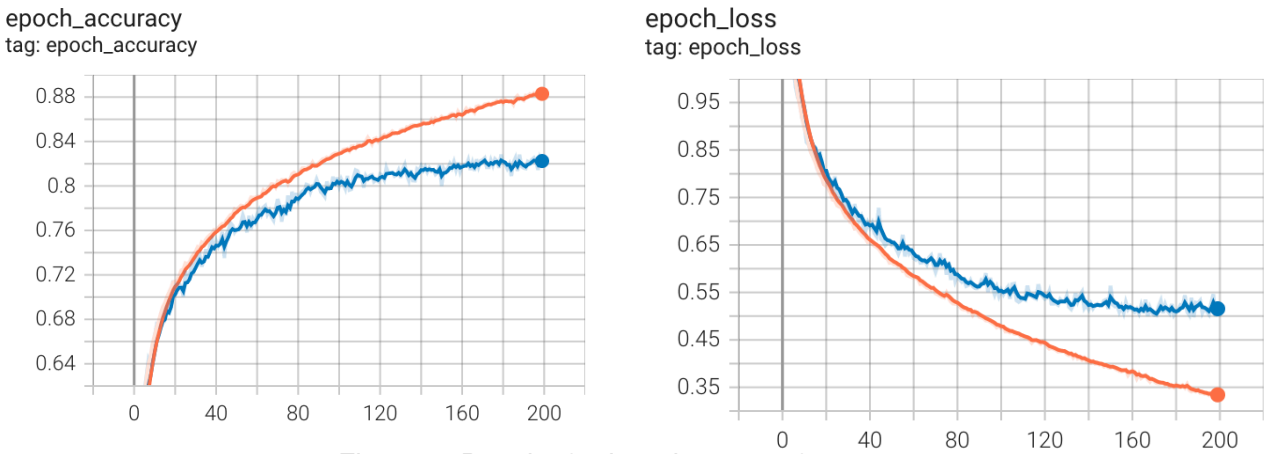

Figure 6. Results for learning rate of 0.0001.

number of epochs to run it is 30. After 30 epochs, the loss on the validation data starts to increase, and the validation accuracy stagnates. The test accuracy for this trained model is 0.8352, which is the best out of all the other models using different learning rates.

As a conclusion, I think a learning rate of 0.001 would be best, with a model trained for 30 epochs.

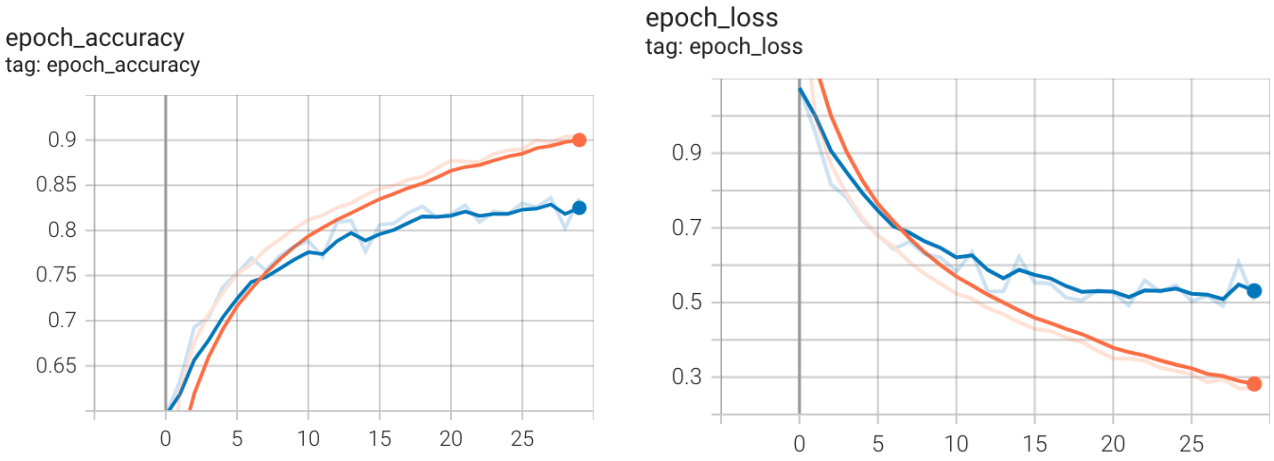

Figure 7. Results for final model trained for 30 epochs with learning rate of 0.001.

Part 2

For the second part of the project I developed a simpler convolutional neural network in order to perform classification on the four class dataset. This CNN consists of 8 layers - one input layer, two convolutional layers with 64 filters, same padding and relu activation function , two pooling layers with default pool size, a flattening layer, a dense layer with 64 neurons and relu activation function and a dense layer with 4 units and softmax activation function.

I used Tensorboard in order to decide on the number of epochs for training, and since after the 25th epoch the loss on the validation data started increasing, and the accuracy was stagnating, I decided that 25 epochs was the appropriate number for training. I used a learning rate of 0.001 and batch size of 256. The accuracy I obtained with this model on validation data is 0.74. Next, I plotted the training and validation accuracy for each class and these can be observed in figure 8 and figure 9 respectively.

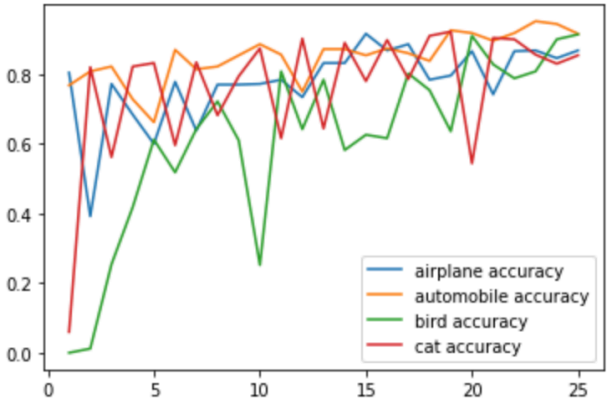

Figure 8. Training accuracy for each class.

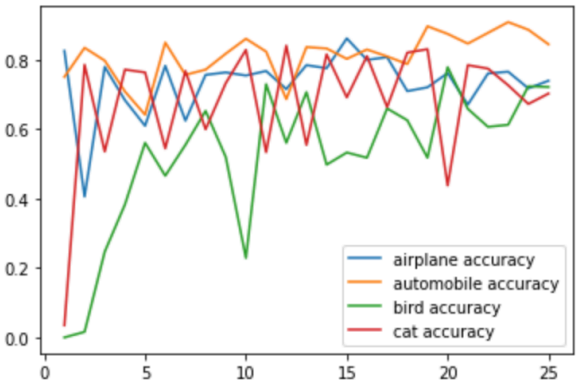

Figure 9. Validation accuracy for each class.

As we can see from the plots, the training accuracy seems to be the worst for pictures of birds and cats, and so is the test accuracy with its values ranging from 0.5 to 0.7. The best performing class both on validation and test accuracy is the automobile with its accuracy reaching 0.8 on validation data.

For the next part, I had to develop a CNN and train it on the imbalanced ten class dataset. We know from the brief that the first four classes contain only 500 samples, whereas the latter six contain 5,000 samples. For this purpose, I have used the same CNN as in the first part of this task, but I changed the last dense layer so that instead of four units it has ten. The first thing I noticed is that the loss is much larger on this dataset (1.446 at its minimum), compared to the one on the four class dataset (0.665 at its minimum). Also, after training for 13 epochs, the validation

loss started to increase sharply, indicating overfitting. Furthermore, there is a very big difference between training accuracy and validation accuracy, indicating that the model doesn't generalise very well. The training accuracy was 0.8 whereas the validation accuracy was 0.56. Figures 10 and 11 illustrate the validation and training accuracy of the model after being trained for 13 epochs as well as the loss on both datasets.

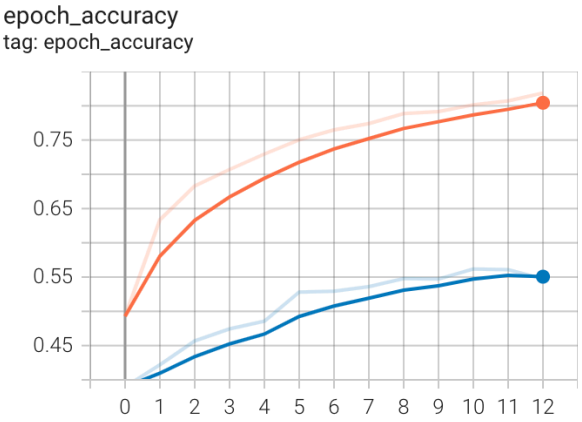

Figure 10. Training and validation accuracy.

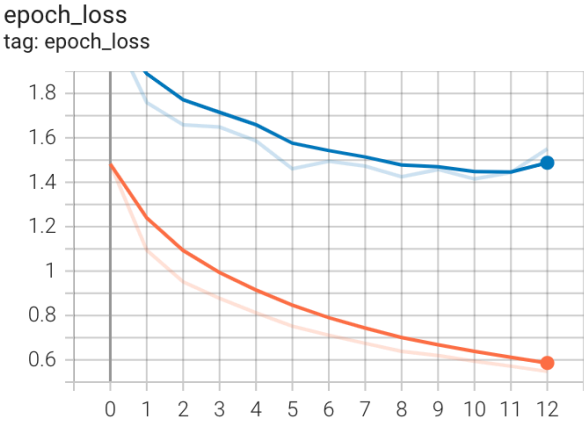

Figure 11. Training and validation loss.

I have also tried a learning rate of 0.01 and a batch size of 128 and the results didn't improve. On the contrary, the validation accuracy did not exceed 0.45 and the validation loss started increasing sharply after 10 epochs.

Next, I have plotted the training and test accuracy for each class, as observed in figures 12 and 13.

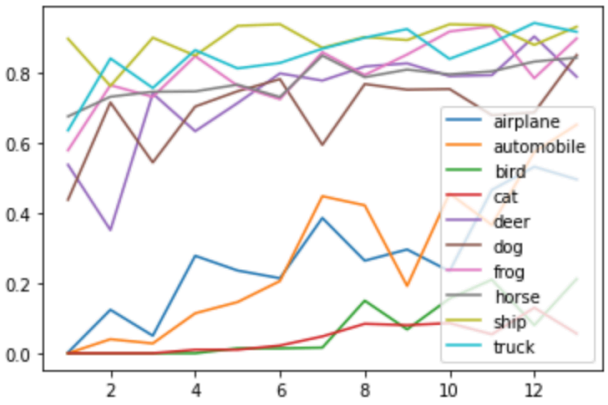

Figure 12. Class accuracy on training data.

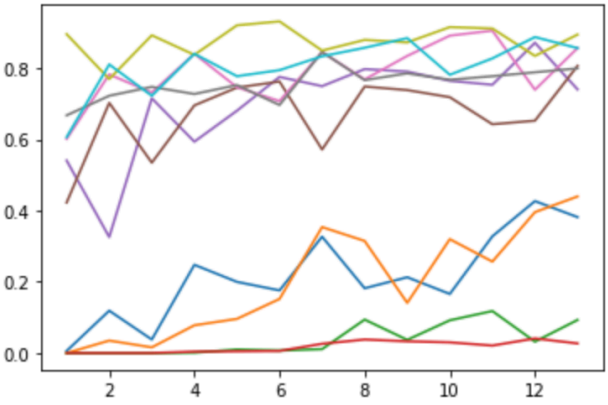

Figure 13. Class accuracy on validation data.

As we can notice from the plots, the first four classes with less training samples tend to perform worst in the beginning and throughout, both on the training and validation data. The accuracy for cat in on the validation data is almost 0 and it is closely followed by bird.

After conducting this experiment I conclude that class imbalance can have a significant detrimental effect on training convolutional neural networks. It affects both convergence during the training phase and generalization of a model on the test set.

Methods for addressing class imbalance can be divided into two main categories. The first category is data level methods that operate on training set and change its class distribution. They aim to alter dataset in order to make standard training algorithms work. These methods include oversampling, which aims to replicate randomly selected samples from minority classes, and

undersampling, which attempts to remove samples from the majority classes until all classes are left with the same number of examples. (Buda, Maki, Mazurouski, 2018)
The other category covers algorithmic level methods. These methods keep the training dataset unchanged and adjust training or inference algorithms. One example of classifier level methods is thresholding, which adjust the decision threshold of a classifier. This is achieved by estimating Bayesian a posteriori probabilities for each class. Other methods include cost-sensitive learning and one-class classification. Methods that combine the two categories are also available. (Buda, Maki, Mazurouski, 2018)

Part 3

In this part of the project, I developed a base model consisting of seven layers - one input layer, two convolutional layers with 128 filters and 64 filters respectively, with relu activation function and same padding, two pooling layers with default pool size, one fattening layer and a dense layer with 64 neurons and relu activation function. Next, I created another model, with the base model as its core and added a dense layer with 6 units and softmax activation function. I used a learning rate of 0.001, the Adam optimiser and a batch size of 256. I trained this model on the six class dataset for 15 epochs and obtained a training accuracy of 0.87 and validation accuracy on the test data of 0.82. The reason why I trained for 15 epochs is the validation loss was starting to increase, indicating overfitting.

In order to perform transfer learning, after I trained the model on the six class dataset, I froze the base model, and then created another model with the frozen base model as its core and added a dense layer with softmax activation function and 4 units. I used the same learning rate and batch size as previously. I then trained this new layer on the four class dataset for 100 epochs and assessed the model's accuracy on the test dataset. I obtained a training accuracy of 0.7325 and a

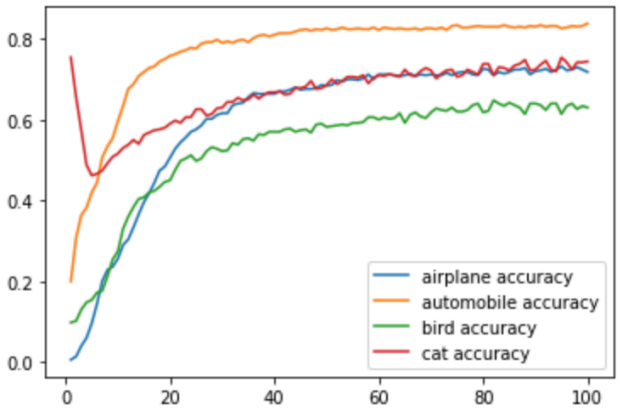

Figure 14. Model accuracy on training data per class.

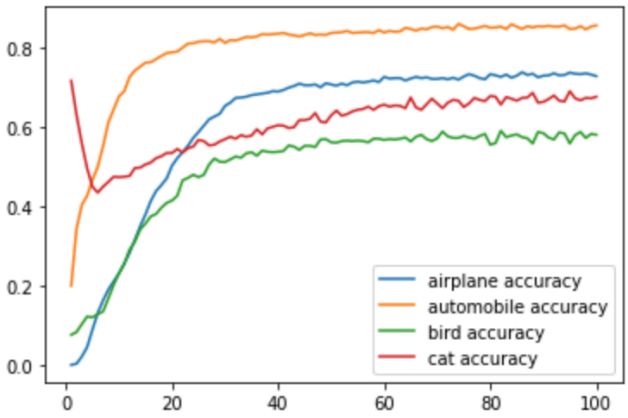

Figure 15. Model accuracy on validation data per class.

validation accuracy of 0.7265. I also plotted the training and test accuracy per class, as presented in figures 14 and 15.

As we can see, the accuracy of each class doesn't vary as much as we have seen on the four class model in part two, the results are much more consistent. Similarly, automobile is still the best performing class with an accuracy of around 0.8, and bird is still the least performing class with an accuracy of 0.5. However, it should be noted that the model had to be trained for much longer in order to achieve these results.

One interesting feature of transfer learning is that the model doesn't seem to overfit after running it for much longer than other models (after about 70 epochs). There is a very small gap between training and validation accuracy and the loss appears to steadily decrease for the validation data as seen in figures 16 and 17.

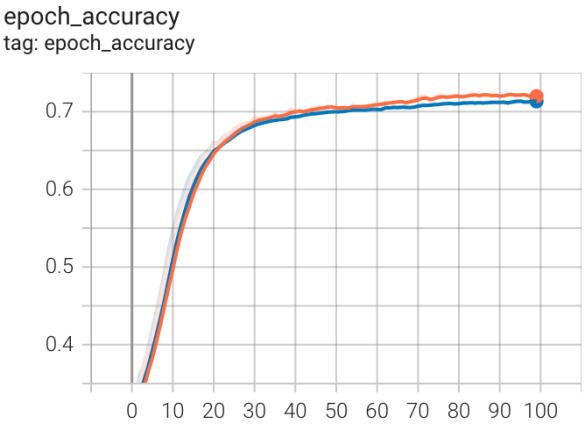

Figure 16. Model training and validation accuracy.

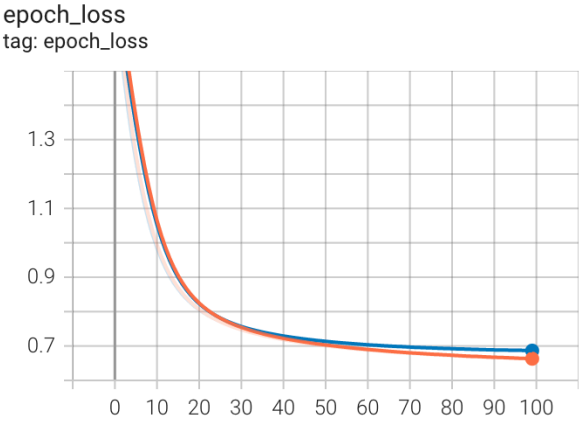

Figure 17. Model training and validation loss.

However, it should be noted that overall, the model does not outperform the model in part two, it actually performs slightly worse. The reason behind this could be that we tried to classify different items, that the model had not seen before during training. As a result, the weights trained from the initial task were different from our target task.

Conclusion

As a conclusion, I have fit several convolutional neural networks on different datasets. In the first part of the project I worked with the six class dataset and initially experimented with different batch sizes, learning rates and training epochs. In the end, I developed a CNN with 12 layers, which obtained 0.835 accuracy on the test data.

In the second part of the project I had to perform classification on the four class and an imbalanced ten class dataset. For the four class dataset, which only contained 500 samples for each class, I developed a CNN which obtained 0.74 accuracy on the test dataset. Next, I used a similar CNN (with the only difference being the last output layer, which contained 10 units instead of four) in order to classify the ten class dataset. I obtained very poor results, with an accuracy that only scored 0.56, and a very high loss on validation data. Unfortunately, the model did not generalise well. After exploring my findings, I also presented a few options for dealing with imbalanced datasets, such as oversampling of the underrepresented classes or thresholding.

In the third part of the project I created a model which I trained on the six class dataset and then froze, in order to perform transfer learning and use the model in order to classify the four class dataset. Although, in theory, transfer learning should have been successful, because it was perfectly suited to our situation in which we only had 500 samples for each class in the four class dataset, and we had 5000 samples for each class in the dataset we used for training, it didn't perform that well. I obtained a lower validation rate by 2% compared to the model in the second part of the project. My only assumption as to why it did not perform that well is that I tried to classify different items, which the network had not seen before during training.

References

Buda, M., Maki, A. and Mazurowski, M., 2018. A systematic study of the class imbalance problem in convolutional neural networks. *Neural networks*, 106, pp.249-259.

Gupta, A., 2021. *A Comprehensive Guide on Deep Learning Optimizers*. [online] Analytics Vidhya. Available at: <https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/#:~:text=The%20results%20of%20the%20Adam,for%20most%20of%20the%20applications.> [Accessed 4 July 2022]