LSC Project

Predicting CIFAR 100 image labels using Convolutional Neural Networks

Introduction

The CIFAR 100 dataset contains a total of 60,000 images and corresponding labels. The data is split in to two partitions with the 'training' partition containing 50,000 images and the 'test' partition containing 10,000 images. Each image is assigned 1 out of 100 labels (known as the 'fine' labels) and these are grouped in to 20 super classes (known as the 'coarse' labels).

As an example, the below image from the training set is characterised as below:

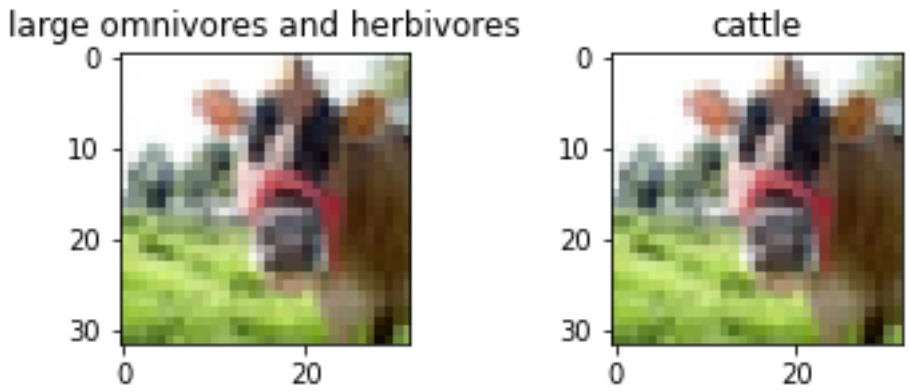

Figure 1 Example of a course label (left) and a fine label (right) on the same image

The purpose of this report is to outline recommendations for building a Convolutional Neural Network (CNN) to correctly classify these images with their appropriate labels. To do this, several approaches will be trailed and evaluated to understand which gives the best predictive accuracy on unseen data (i.e. the 'test' data that has not been used to train the model).

To simplify the model build as much as possible, the CNN models will be validated on the test set to allow test performance to be analysed easily in Tensorboard. Please note that this may introduce some bias and that in a real-world scenario it is advised to have an additional 'validation' set from the training sample.

Part 1 – Building an initial model to predict coarse labels

An initial model has been built using a CNN with the architecture below:

1. Convolutional layer with a 3x3 kernel size and 32 filters which uses a ReLU activation function and 'same' padding'
2. Max pooling layer with a pool size of 2x2
3. Convolutional layer with a 3x3 kernel size and 64 filters which uses a ReLU activation function and 'same' padding'
4. A flattening layer
5. A dense layer with 128 units which uses a ReLU activation function
6. A dense layer with 20 units which uses a Softmax activation function

The model initially used a batch size of 32 training samples and an Adam optimizer with a learning rate of '0.001'. An Adam optimizer was initially selected as it is fast, efficient and less likely to get stuck in a local minima compared to an ordinary gradient descent optimizer.

The model was set to run through 100 epochs and record the validation accuracy (i.e. test accuracy). The 'best' model was then selected based on which recorded the highest validation accuracy.

Performance variables were logged after each epoch within TensorBoard to get view of where the model started to overfit the data:

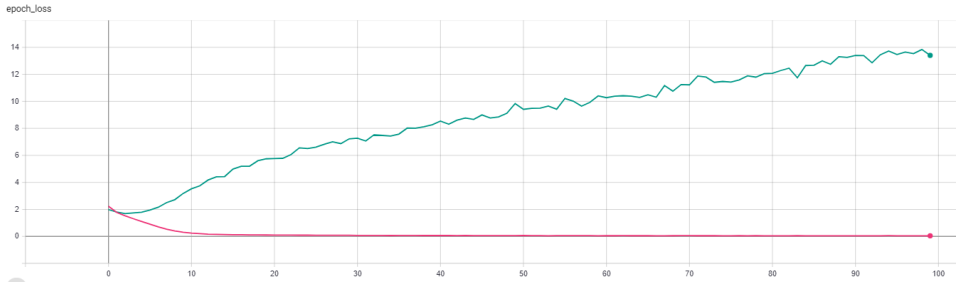

Figure 2 Cross entropy loss over each epoch (Pink: Training, Green: Test)

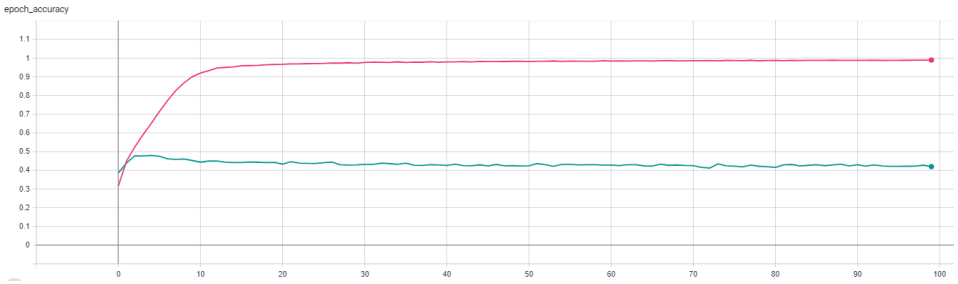

Figure 3 Classification accuracy over each epoch (Pink: Training, Green: Test)

From this we can see that the model reached the optimum validation performance after only 4 epochs which returned a correct classification rate of 0.480. After this the model started to overfit to the training data and validation performance decreased as evidenced by the increasing entropy loss and decreased validation accuracy.

To understand if the Adam optimizer was a good fit for the model, a Stochastic Gradient Descent (SGD) optimizer was fit with the same learning rate as below:

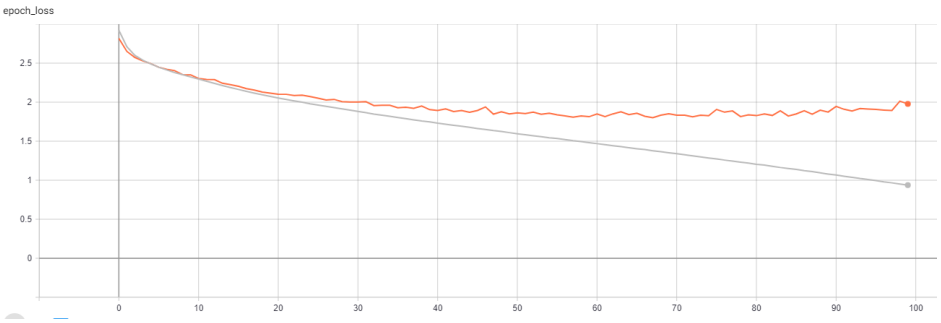

Figure 4 Epoch loss for the SGD optimizer (Grey: Training, Orange: Test)

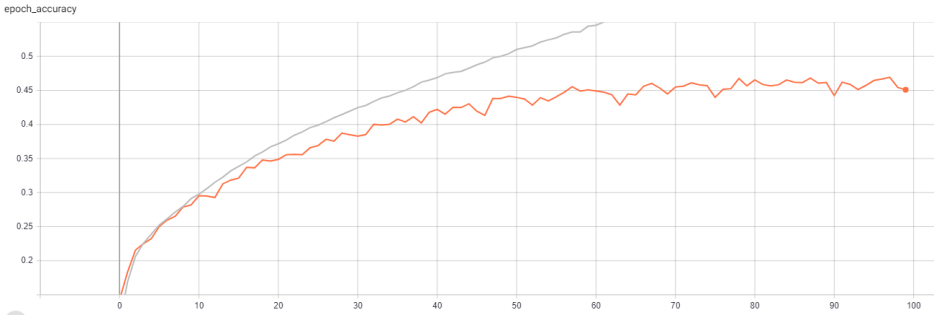

Figure 5 Epoch accuracy for the SGD optimizer (Grey: Training, Orange: Test)

Compared to the Adam optimizer, the SGD optimizer had more comparable epoch loss and accuracy between the training and test data and was not as prone to overfitting to the training data. However, the SGD optimizer was slower to mature and its optimal validation accuracy (0.469) was lower than that of the Adam optimizer (0.480). For the remaining iterations, the Adam optimizer was preferred.

To understand how performance varied by batch size, the model was iterated with a smaller batch size of '16' and a larger batch size of '64'. As we had already seen that performance saturated quickly, the models were run for just 20 epochs:

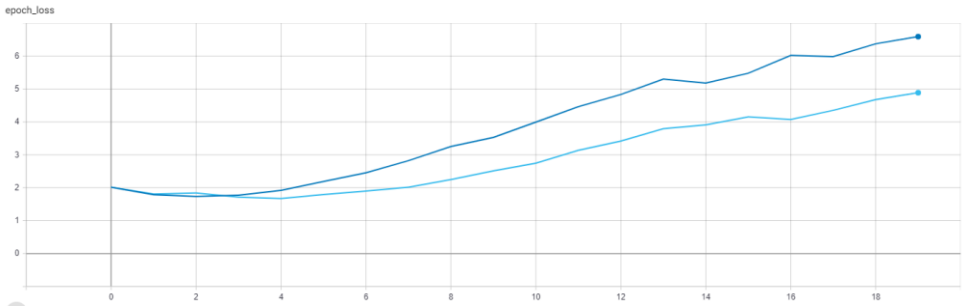

Figure 6 Epoch loss for batch size 16 (dark blue) and batch size 64 (light blue)

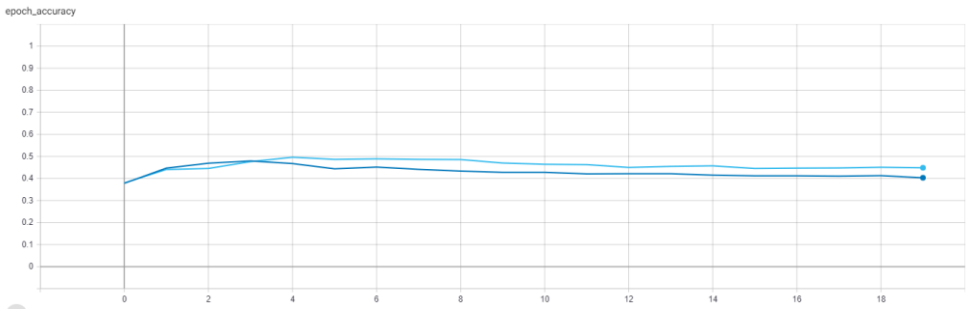

Figure 7 Epoch accuracy for batch 16 (dark blue) and batch size 64 (light blue)

The higher batch size of '64' was slower to overfit and had slightly higher validation accuracy (0.496 vs. 0.4801 with batch size '32'). However, changing this hyperparameter did not have a huge affect on overall model performance over the validation set. For the remaining models, a batch size of '64' will be used.

Next, a different learning rates were used to understand their impact on model performance. The model was iterated with a smaller learning rate of 1e-4 and a larger learning rate of 1e-2.

The learning rate of 1e-2 failed to improve the model beyond random guessing so this was abandoned. However, the learning rate of 1e-4 showed good performance as below:

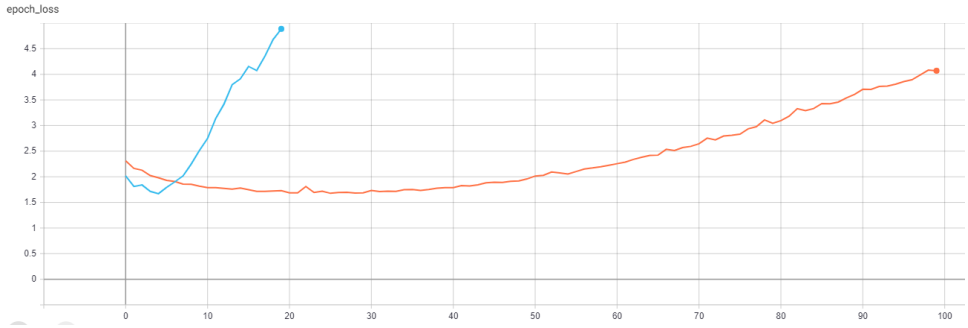

Figure 8 Epoch loss for 1e-3 learning rate (light blue) and 1e-4 learning rate (orange)

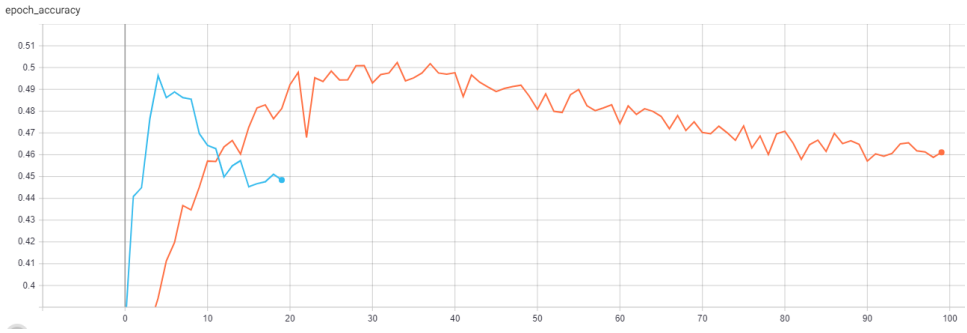

Figure 9 Epoch accuracy for 1e-3 learning rate (light blue) and 1e-4 learning rate (orange)

The smaller learning rate slightly improved accuracy over validation data (0.502 vs. 0.496). However, this also required a larger number of epochs to train and was computationally more expensive. As the performance difference was minimal, the previous learning rate of 1e-3 will be used for future iterations.

Recapping 'best' model at the end of part 1:

- Adam optimizer
- Batch size: 64
- Learning rate: 1e-3
- Epochs: 20 (with best performing epoch selected, determined by validation accuracy)
- Accuracy: 0.496

Part 2 – Optimising the model architecture to increase accuracy

The previous section demonstrated two issues with model performance:

1. The model tended to overfit on the training data showing reduced performance on the validation test when the number of epochs was larger
2. The model could only achieve a ~50% predictive accuracy suggesting there is likely to be some non-random structure to the data that was not captured by the model

Initially 3 dropout layers were added as below to reduce overfitting:

1. Convolutional layer with a 3x3 kernel size and 32 filters which uses a ReLU activation function and 'same' padding'
2. Max pooling layer with a pool size of 2x2

3. **Dropout layer added with a rate of 0.25**
4. Convolutional layer with a 3x3 kernel size and 64 filters which uses a ReLU activation function and 'same' padding'
5. A flattening layer
6. **Dropout layer added with a rate of 0.25**
7. A dense layer with 128 units which uses a ReLU activation function
8. **Dropout layer added with a rate of 0.5**
9. A dense layer with 20 units which uses a Softmax activation function

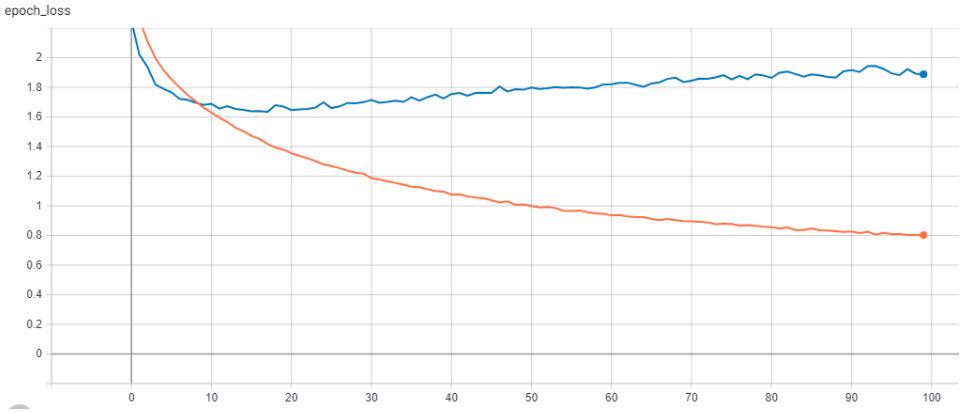

Figure 10 Cross entropy loss over each epoch with dropout layers added (Orange: Training, Blue: Test)

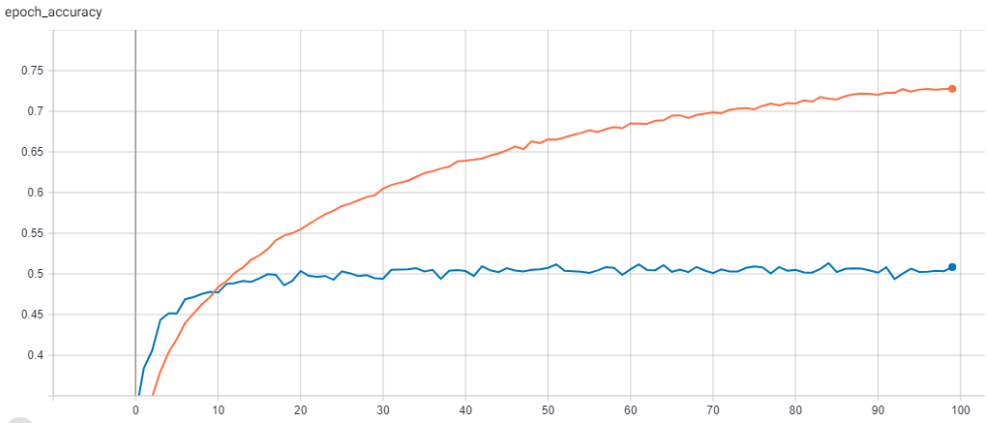

Figure 11 Classification accuracy over each epoch with dropout layers added (Orange: Training, Blue: Test)

As the dropout layers increased the time required for the model to mature, the number of epochs was increased to 100. The dropout layers reduced overfitting significantly and improved the classification accuracy to a maximum of 0.513 (compared to 0.496).

Next 2 additional sequential convolutional layers were added to capture more of the non-linear patterns across the data as below:

1. Convolutional layer with a 3x3 kernel size and 32 filters which uses a ReLU activation function and 'same' padding'
2. **Convolutional layer with a 3x3 kernel size and 32 filters which uses a ReLU activation function and 'same' padding'**
3. Max pooling layer with a pool size of 2x2
4. *Dropout layer added with a rate of 0.25*

5. Convolutional layer with a 3x3 kernel size and 64 filters which uses a ReLU activation function and 'same' padding'
6. **Convolutional layer with a 3x3 kernel size and 64 filters which uses a ReLU activation function and 'same' padding'**
7. A flattening layer
8. *Dropout layer added with a rate of 0.25*
9. A dense layer with 128 units which uses a ReLU activation function
10. *Dropout layer added with a rate of 0.5*
11. A dense layer with 20 units which uses a Softmax activation function

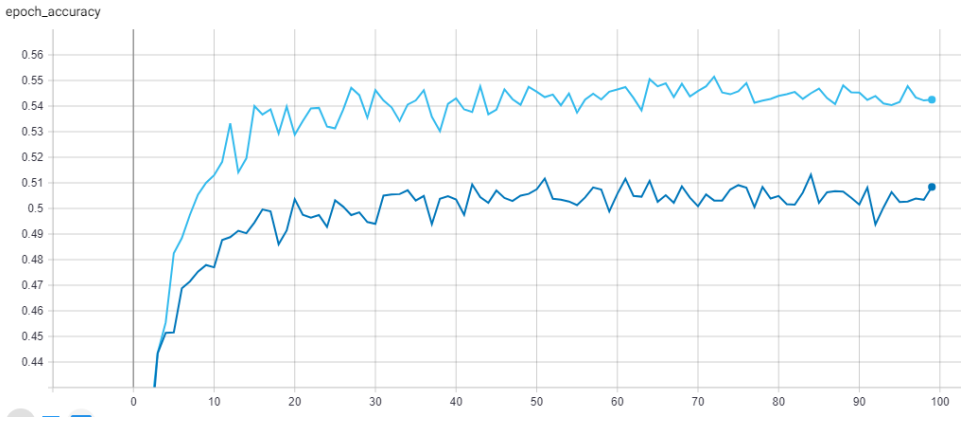

Figure 12 Validation accuracy for the additional convolution model (light blue) vs. the previous simpler model (dark blue)

Adding in the additional convolutional layers improved validation accuracy to 0.552 (compared to 0.513) previously. The performance over epochs showed that the model continued to be less susceptible to overfitting due to the inclusion of the dropout layers.

To optimise the mix of hyperparameters for these models, a random search was implemented testing the combination of the below parameters over 10 trials with 50:

- First convolution filters: 16, 32, 64
- Second convolution filters: 16, 32, 64
- Third convolution filters: 32, 64, 128
- Fourth convolution filters: 32, 64, 128
- Dense units: 64, 128, 256

The random search identified the existing model as having the best parameters out of those tested as below:

- First convolution filters: 32
- Second convolution filters: 32
- Third convolution filters: 64,
- Fourth convolution filters: 64
- Dense units: 128

Recapping 'best' model at the end of part 2:

- Adam optimizer
- Batch size: 64
- Learning rate: 1e-3

- Epochs: 100 (with best performing epoch selected, determined by validation accuracy)
- Additional convolution layers added
- Additional drop out layers added
- Accuracy: 0.552

Part 3 – Predicting fine labels using the existing model and transfer learning

Continuing with the model identified in part 2, the model architecture was applied to predict the fine labels:

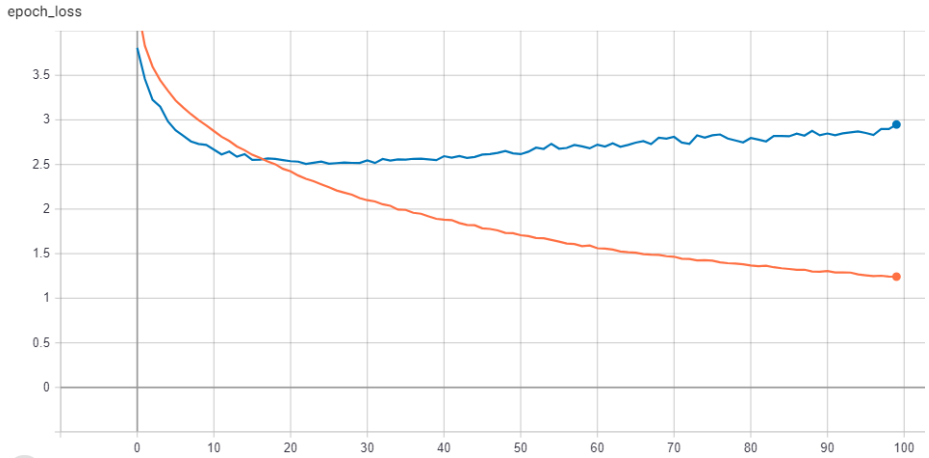

Figure 13 Cross entropy loss over each epoch when applied to the fine labels (Orange: Training, Blue: Test)

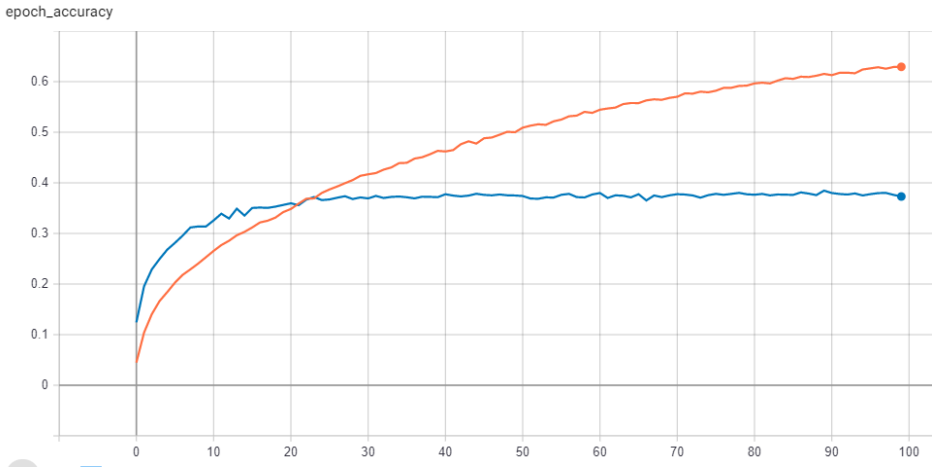

Figure 14 Classification accuracy over each epoch when applied to the fine labels (Orange: Training, Blue: Test)

As the model is now predicting a larger number of labels, a decrease in accuracy is expected. Applying the same model architecture to the fine labels returned a classification accuracy of 0.385 over the test set. The epoch accuracy showed that, like the performance over the coarse labels, the model had little issue with overfitting to the training data.

It may be possible to improve the model using transfer learning by taking a pre-trained network and amending it to apply to the CIFAR 100 dataset. To test this, the VGG16 model has been imported and used to predict the fine labels:

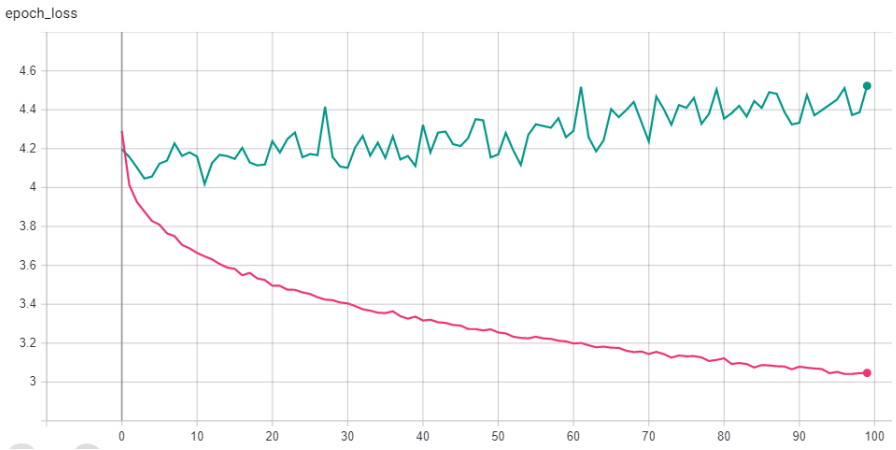

Figure 15 Cross entropy loss over each epoch for the VGG16 model when predicting fine labels (Pink: Training, Green: Test)

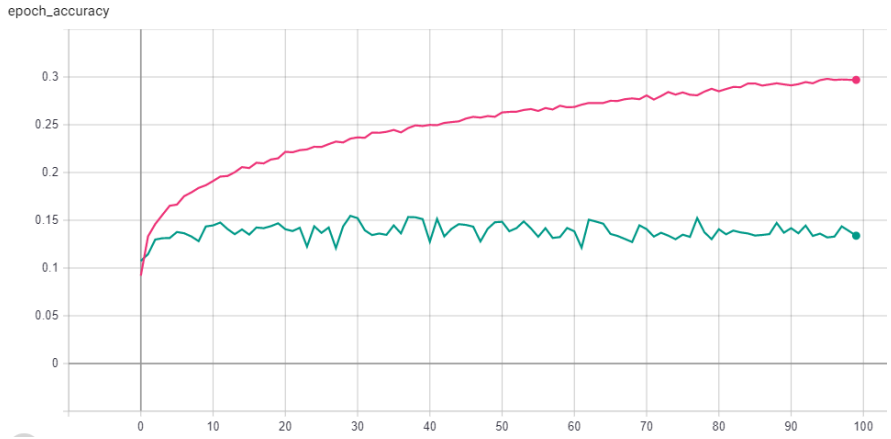

Figure 16 Classification accuracy over each epoch for the VGG16 model when predicting fine labels (Pink: Training, Green: Test)

The VGG16 model struggled to reach the same classification accuracy reaching a high of 0.155 (compared to 0.385 from the previous model). This could potentially be improved further by spending more time pre-processing the images so they are better fit to the VGG16 model.

Summary and recommendations

The developed model attained 0.552 predictive accuracy on the coarse labels and 0.385 predictive accuracy on the fine labels.

To help benchmark, a random guess would return a 0.05 predictive accuracy on the coarse labels and 0.01 predictive accuracy on the fine labels which suggests the model that has been developed has good predictive performance. However, this still leaves a high volume of pictures that are predicted incorrectly suggesting that the model could likely be improved further.

There are three recommended routes for further iteration:

1. Add additional convolutional layers to capture more non-random structure within the data
2. Test a greater number of hyperparameter combinations (though the computational cost quickly gets too high to run on a local machine within a reasonable time limit)
3. Experiment with transfer learning from other networks such as VGG19, ResNet and Xception with more time spent pre-processing the images