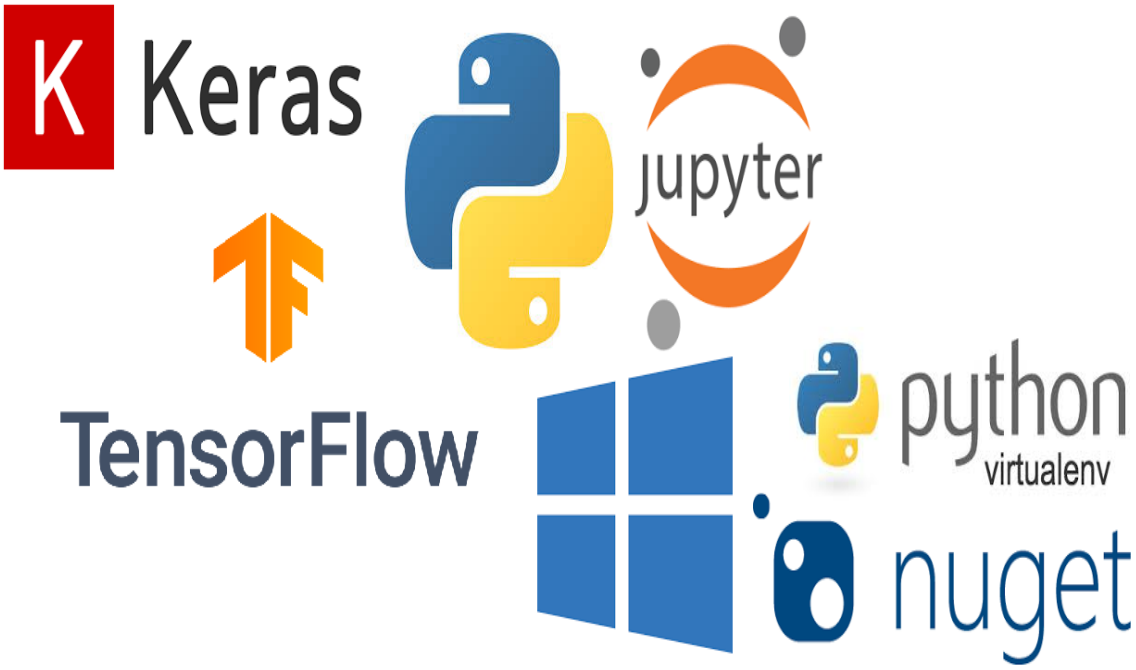
Project Report



Course Name: Large-Scale Computing for Data Analytics

Student ID: 2514007

Student Name: Stefan Faulkner

Introduction

In neural networks, Convolutional neural network (CNNs) is one of the main categories to do image recognition, images classifications. Objects detections, recognition faces etc., are some of the areas where CNNs are widely used.

CNN image classifications take an input image, processes it, and classify it under certain categories (E.g., Dog, Cat, Tiger, Lion). The computers sees an input image as an array of pixels and it depends on the image resolution. Based on the image resolution, it will see h x w x d (h = Height, w = Width, d = Dimension). E.g., An image of 6 x 6 x 3 array of matrix of RGB (3 refers to RGB values) and an image of 4 x 4 x 1 array of matrix of grayscale image.

In this report this is exactly what we will be doing essentially. Our main objective is that we will be using CNNs to classify images from the CIFAR-100 dataset. Note that they are two datasets provided to us which we will be using to train and test our CNNs throughout this report. Firstly, the CIFAR-100 coarse-labelled dataset: 50,000 training and 10,000 testing images, pre-labelled in 20 superclasses. Secondly the CIFAR-100 fine-labelled dataset: 50,000 training and 10,000 testing images, pre-labelled in 100 classes.

We will start by training and testing a CNN to classify the 20 superclasses, adjusting in iterations to arrive at a more optimal network. Afterwards we will retrain it with the goal of increasing its accuracy. Then, we will use this architecture to train a network from scratch for the 100 fine labels. Lastly, we will use a pretrained VGG16 network to do transfer learning and compare performance with previous trained models.

Part 1 – Building and training a Basic CNN model Architecture

```
model1.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 16, 16, 64)        18496
_________________________________________________________________
flatten (Flatten)            (None, 16384)             0
_________________________________________________________________
dense (Dense)                (None, 128)               2097280
_________________________________________________________________
dense_1 (Dense)              (None, 20)                2580
=================================================================
Total params: 2,119,252
Trainable params: 2,119,252
Non-trainable params: 0
_________________________________________________________________
```

Knowing that we have imported and preprocessed the data we are moving on building our fist CNN based classifier on the CIFAR-100 coarse labelled dataset. For the model architecture it follows (This was given to us in our project description):

- A convolution layer with a 3x3 kernel size and 32 filters.
- A max pooling layer with a pool size of 2x2.
- A convolution layer with a 3x3 kernel size and 64 filters.
- A flattening layer.
- A dense layer with 128 units.
- A dense layer with the softmax activation function.

There are multiple optimizers which could have been chosen such as Gradient Descent, Adagrad, Adadelta etc. In the end the chosen optimizer was the Adam Optimizer for its simplicity, fast and being computationally efficient. Essentially, it allows parameters to change rapidly and cross local minima without getting stuck when training.
The idea is to fine tune the model by changing:

- The optimizer learning rate - The learning rate controls how quickly the model is adapted to the problem.
- The batch size: The size of the batch is the number of individual training samples we run through the network in one go.
- The number of training epochs - An epoch corresponds to all training data being shown to the network once.

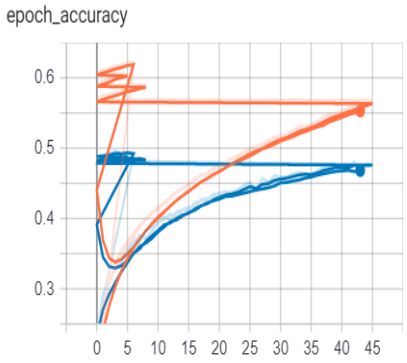
Optimizer Learning Rate

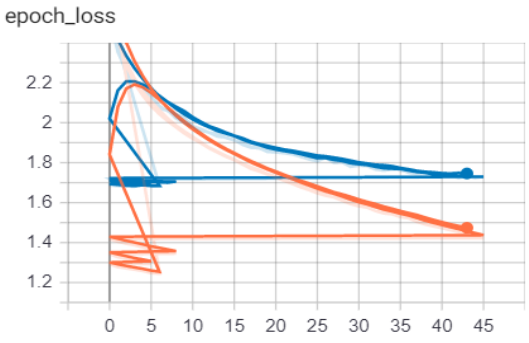
Note that Learning Rates of 0.1, 0.01 ,0.001 and 0.0001 were tested. It was evident that a higher learning rate allows the optimizer to improve in fewer epochs but may reach a suboptimal solution, whereas when we chose a smaller learning rate it could take more epochs to train but can reach a more optimal solution. In our case, a learning rate of 0.0001 minimizes the loss function most rapidly and achieves the highest accuracy of approximately 56% on the training data (although this is not a good indicator of how good our model is), where all other parameters kept equal.

Batch Size

Noticeable changes were present when the batch size was varied/changed. The idea was to iterate through several batch sizes before settling on an optimal number. After doing this we see that the batch size had a huge impact on the training of our model. In other words, the accuracy increases as the batch size increases. When the batch size is small the network weights jumps around and the validation accuracy performs poorly, which may cause the network to learn slowly. Here we will choose a batch size of 512 due to the computational efficiency and validation accuracy when the model was trained.

Tensorboard Analysis





The Tensorboard graphs shown above outline how our model progressed while training. First thing we can notice is that it has like a zig-zag shape around the place till it finally reached a stoppage where the dot is shown. So, we notice it did not have a smooth curve shown. The orange line shows the loss and accuracy for the train data while the blue line shows the accuracy and loss for the validation data. We see from the above graphs our model trained for 44 epochs and finished with a final prediction accuracy of roughly little over 45% on the validation data.

Also note that, we select the number of epochs to train our model to be 100 (this gave us the best accuracy), however, having a call-back with a patience of 3 means that the training will stop if the validation loss has not improved for 3 epochs. This prevents the model from overfitting on the training data.

Lastly, after considering all the hyperparameter adjustments above for this architecture built we arrived at a prediction accuracy of 46.9% on the superclasses of CIFAR-100 data in our assessment.

Part 2 – Altering our CNN based Classifier Model Architecture

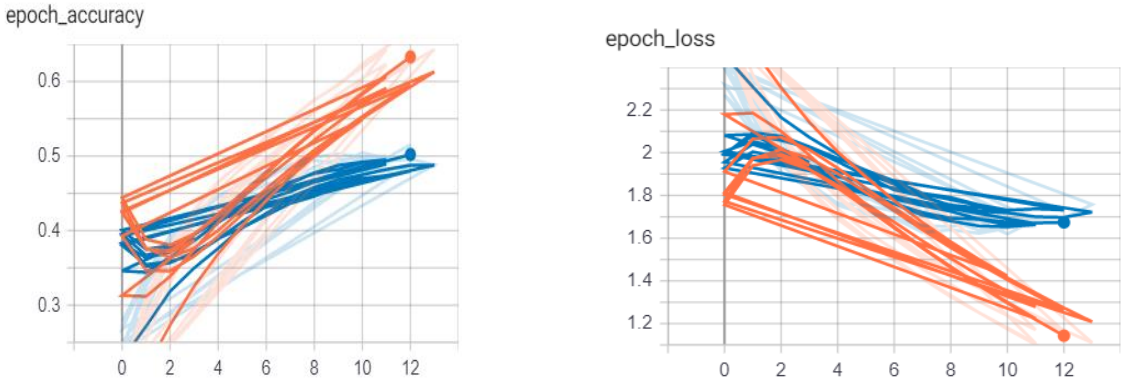
In this part we will take a look on what we can do to go about improving our accuracy of our previous model by altering its architecture. This is so that we hopefully can improve our test accuracy. Changes will be made on the same dataset as we wish to classify based on the 20 superclasses as before. New hyperparameters will be trained to find the optimal network architecture with an increased batch size. This corresponds to **model 3** in our code provided.
Now our new model follows the architecture below:

- Convolution layer with 32 filters, a 3x3 kernel and a relu activation function
- Max pooling layer with a pool size of 2x2
- Convolution layer with 64 filters, a 3x3 kernel and a relu activation function
- Max pooling layer with a pool size of 2x2
- Convolution layer with 128 filters, a 3x3 kernel and a relu activation function
- Max pooling layer with a pool size of 2x2
- Flattening layer
- Dense layer with 256 units and a relu activation function
- Another Dense layer with 256 units and a relu activation function
- Another Dense layer with 256 units and a relu activation function
- Dense layer with 512 units and relu activation function
- Dense layer with 20 units and softmax activation function

Now we have 12 layers where previously we had 6 layers to our initial network in Part 1. The idea with the following above architecture is we need some sort of extra convolution layer with an increased additional of filters which should help us in discovering finer

details from the input. We have also added two additional pooling layers which will help ensure only the most important features propagate through from the previous layer. Finally, we have added multiple dense layers with varying units

Our new model architecture gave us the results below from Tensorboard:



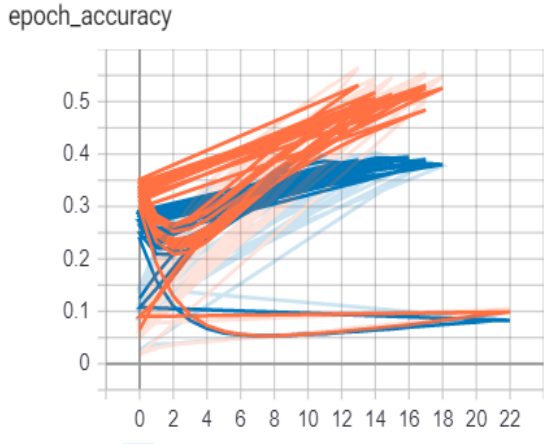
In the results above we are not seeing that zig-zagged shape as before but now we are seeing multiple lines as it was training to provide optimal results. The same concept remains as the orange line shows the loss and accuracy for the train data while the blue line shows the accuracy and loss for the validation data. We see from the above graphs our model trained for only 12 epochs and finished with a final prediction accuracy of roughly little over 51% on the validation data. This is a considerably improvement in our new model architecture as compared to before!

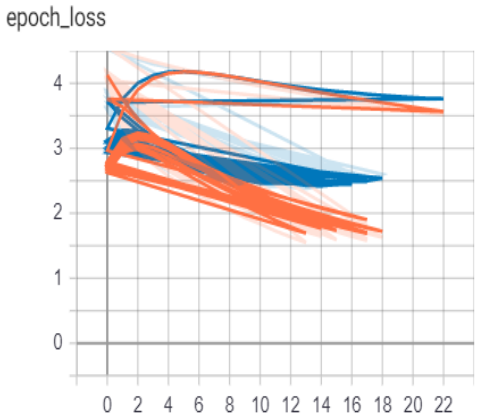
Note that we remained with the Adam Optimizer as for the same reasons stated before for efficiency and its adaptive learning rate. The batch size selected when we iterate through several varying batch sizes was the size of 128 for our new architecture. The learning rate which gave considerably good results was 0.001 which was selected.

Lastly, we noticed from our assessment on the accuracy on unseen data after optimizing the hyperparameters is approximately 52%. This is a great improvement from the initial network from part 1 which had a prediction accuracy of 46.9%. It is highly likely with more finetuning to our architecture we could even get an even greater accuracy.

Part 3a – Training a network on the CIFAR-100 Fine Labels

Now for the final part of the report will train a network to classify the CIFAR-100 images according to the 100 fine labels. The idea remains the same as part 2 the use the same architecture but only changing the last dense layer to accommodate the100 labels as compared to before where we had 20 coarse labels.





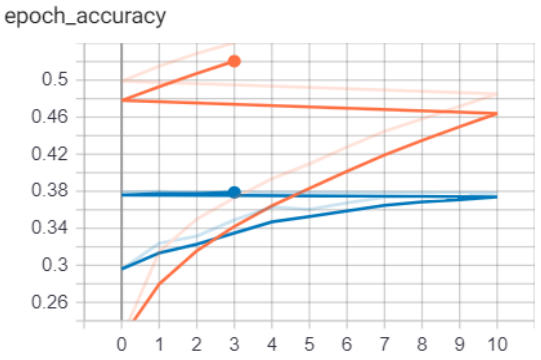
The model performance on this architecture was very poor compare to part 2 which was trained on the coarse labelled test. We first notice two things; the training data accuracy was low which was 53.8% compare to part 2 where we had little bit over 66%. Next, we notice it performs poorly on the unseen data where we are getting a decrease of 37% where before we had an accuracy of approximately 52%. The reasonable cause for this is since we increase the number of labels to a 100 it's a lot harder to correctly classify it. In the final section we will look on how would this this compare to the performance of the original VGG16 on the same dataset, whose layers will be frozen.

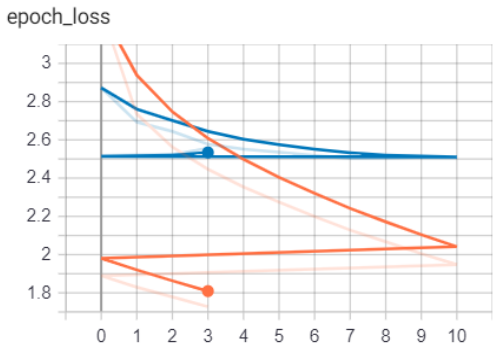
Part 3b – Using Transfer Learning using a pretrained VGG16 network

VGG16 is a convolutional neural network developed by Simonyan and Zisserman from the Oxford Visual Geometry Group. It was used to win the ImageNet competition in 2014 and is currently one of the best general image recognition models. The model achieves 92.7% top-5 test accuracy in ImageNet, which is a dataset of over 14 million images belonging to 1000 classes. It was one of the famous model submitted to ILSVRC-2014. It makes the improvement over AlexNet by replacing large kernel-sized filters (11 and 5 in the first and second convolutional layer, respectively) with multiple 3×3 kernel-sized filters one after another. VGG16 was trained for weeks and was using NVIDIA Titan Black GPU's.

Now, the aim is to use transfer learning by taking the VGG16 Network and apply it tour project. We did this by freezing the layers where we would only need to backpropagate and update the parameters in the top layers. This will consist of a flatten layer, a dense layer with 512 units and a final dense layer of 100 units to finally classify the images.

The VGG model was trained for 100 epochs with the Adam optimizer, learning rate 0.001 and batch size of 128 which gave optimal results. The result of our training progress is shown below via Tensorboard:





Based on our assessment on the testing data we had an accuracy of 38%. This shows us that even with a pre-trained VGG16 network with the layers frozen it did not make much of a difference (in other words there is little to no improvement) when comparing to part 3a. In fact, you can see from above that both the training and the validation data accuracy is very similar as to part3a. The main difference we see here is that this took less steps to

train as we can see that it stopped after 3 epochs. Note here testing data and validation data can be used interchangeably.

Conclusion/Discussion

Overall, we can see that the model from Part 2 performed the best. In other words, it was able to achieve the highest level of accuracy on the test sets of any of the models trained, between the coarse and the fine-labelled CIFAR datasets. If we had more time a lot more exploration could have been done for example, we could have probably unfrozen the layers to see if we would have achieved a higher accuracy rate on our unseen data. There are also many more combinations which could also have been considered when finetuning the layers but in our case this would come with more time and of course if you are more experienced in building these type of model it would be much easier in manipulating the layers to get an improved accuracy.

References

1) https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148
2) https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-cifar-10-photo-classification/
3) https://towardsdatascience.com/step-by-step-vgg16-implementation-in-keras-for-beginners-a833c686ae6c
4) https://neurohive.io/en/popular-networks/vgg16/