Assignment 6: Deep Learning

Michael Shepherd, 19059019

November 1, 2019

Question 1

To answer this question, I used the Python3 Keras library. I used the DataImageGenerator to load my images in. This allowed me to specify the shape of the elements of the data set, which means that I did not need to change the image sizes, I only needed to tell the DataImageGenerator to resize them.

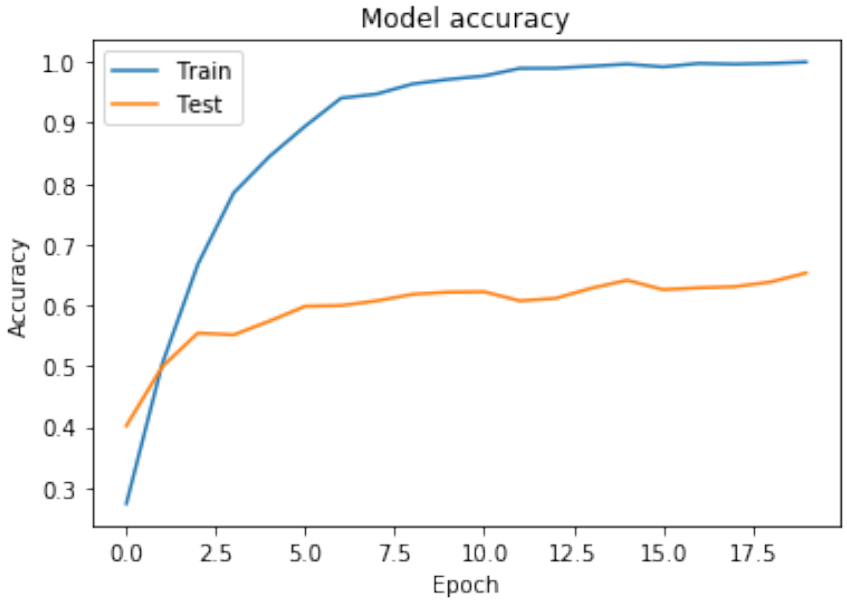
1a

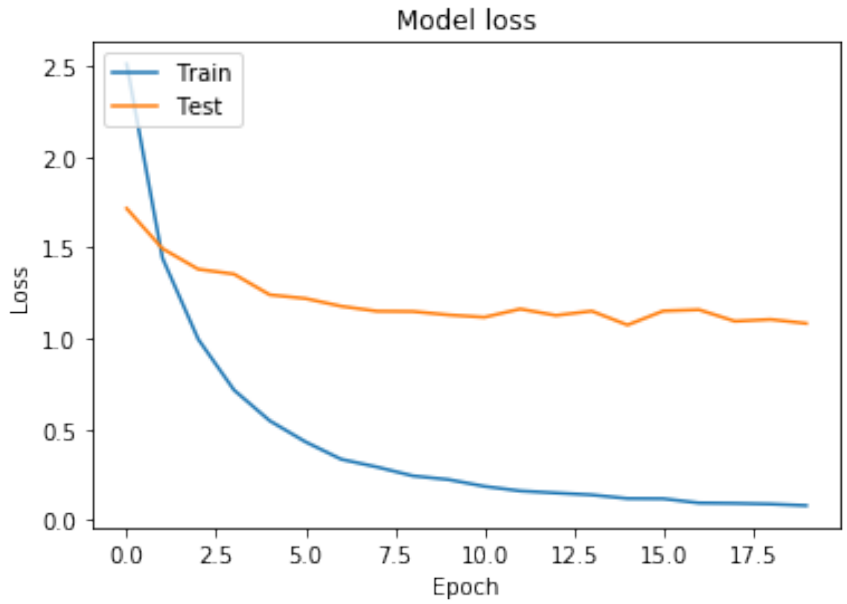
I used Keras sequential models to create this network. This allowed me to easily add the layers to each convolutional stage. To achieve a 60% accuracy with 20 epochs for the model that we were required to make, I made small changes to the hyperparameters of each layer and the batch size. I settled on the following layers. Notably, I used a learning rate of 0.0008, which allowed the model to reach an acceptable level of accuracy. Bellow I have included a listing to describe my model.

```
def one_a_model():
  batch_size = 24
  model = Sequential()
  # Layer 1
  model.add(Conv2D(16, (3, 3), input_shape=input_shape))
  model.add(BatchNormalization(momentum=0.9))
  model.add(Activation('relu'))
  model.add(MaxPooling2D(pool_size=(2, 2)))
  # Layer 2
  model.add(Conv2D(32, (3, 3)))
  model.add(BatchNormalization(momentum=0.9))
  model.add(Activation('relu'))
  model.add(MaxPooling2D(pool_size=(2, 2)))
  # Layer 3
  model.add(Conv2D(64, (3, 3)))
  model.add(BatchNormalization(momentum=0.9))
  model.add(Activation('relu'))
  model.add(MaxPooling2D(pool_size=(2, 2)))
  # Fully connected layer
  model.add(Flatten())
  model.add(Dense(9))
  # Softmax layer
  model.add(Activation('softmax'))
  model.compile(loss=keras.losses.categorical_crossentropy,
                optimizer=keras.optimizers.SGD(lr=0.0008),
                metrics=['accuracy'])
  return model, batch_size
```

These parameters and hyperparameters allowed me to train a model with a test set accuracy of $\approx 63\%$ with loss ≈ 1.1. The Training set accuracy

was ≈ 98% with loss ≈ 0.1. We can observe how the accuracies and loss' changed over time in the following graphs. These graphs show us that the training of this model over fits training set. This means that the accuracy for the validation set stagnates at a relatively low accuracy value and the training set accuracy over takes it massively. This would mean that the model is great for classifying the training set, but would struggle far more with unseen data.





1b

To try and solve the problem of over fitting, we augment the training set and introduce regularisation.

Augmentation

To augment the training set, I used the Keras ImageDataGenerator, which gives us many options for data augmentation. I have used the option to flip the data horizontally. This allows the model to see more different data with the same classes as random images of the training set are being flipped throughout the training process.

Regularisation

Regularization is a technique which makes slight modifications to the learning algorithm such that the model generalizes better. This in turn improves the model's performance on unseen data as well[2]. In deep learning, this means that we penalise the weight matrices of the nodes[2].

Dropout as a form of regualrisation refers to ignoring randomly chosen nodes during the training phase[1]. This is very well illustrated by the following images, which were taken from an article by Shubham Jain [2].

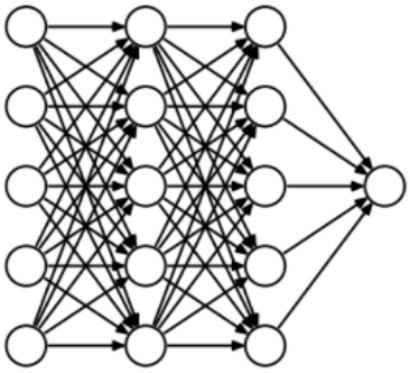


Figure 1: Neural Network

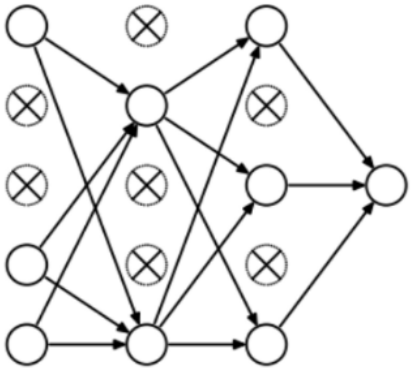
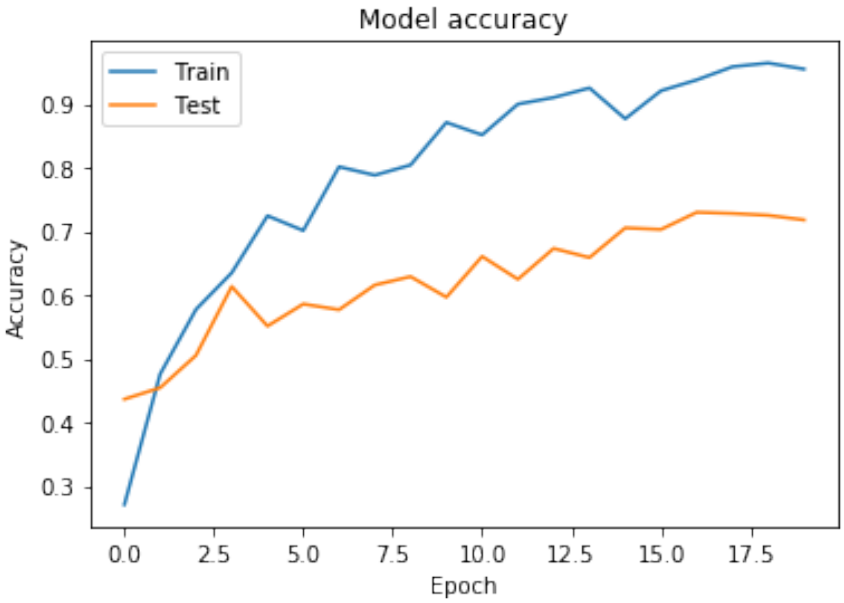


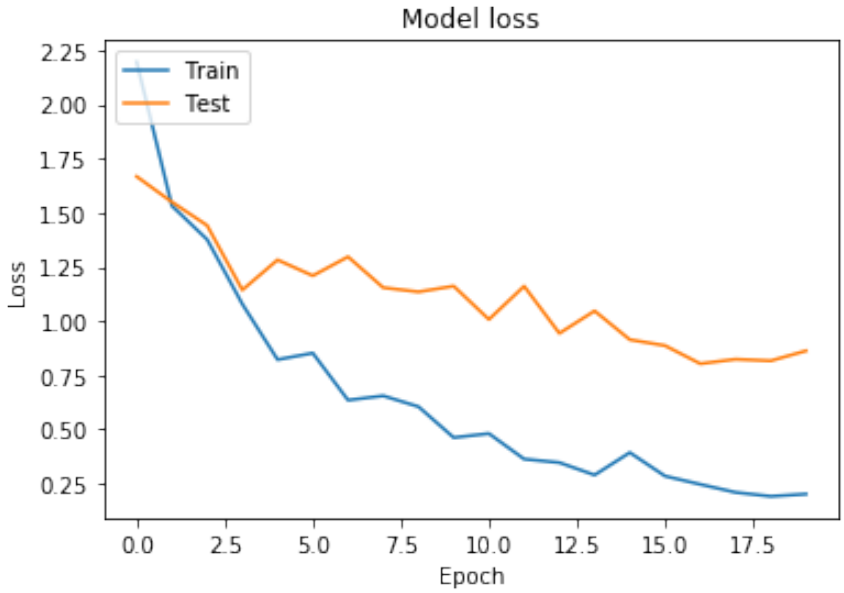
Figure 2: Neural Network With Dropout

Using dropout for regularisation forces the neural network to learn more robust features that are useful in conjunction with many different random subsets of the other neurons[1]. This will slow down the over fitting, as the neural network will take more iterations before it beings to recognise these more robust features and converge. I used a dropout of 0.1.

When I applied both regularisation in the form of dropout and data augmentation in the form of horizontal flips, I was able to achieve a validation accuracy of $\approx 72\%$ with a loss ≈ 0.8. The training set had $\approx 93\%$ accuracy with loss ≈ 0.2.

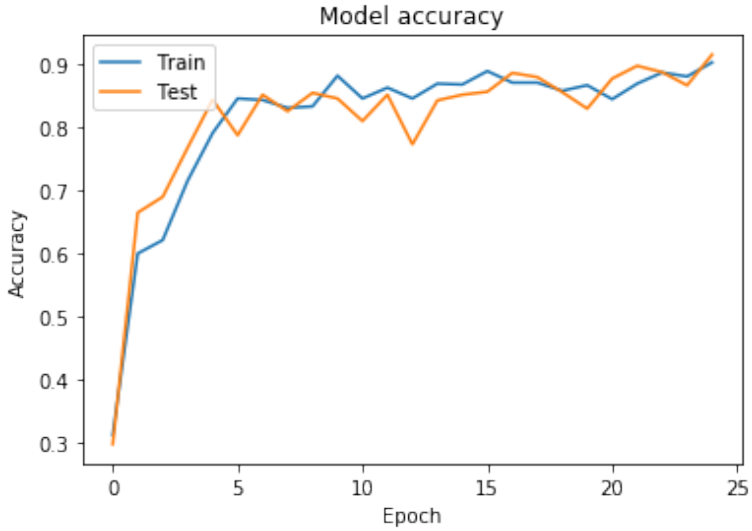
As we can see, the neural network still over fits to the training set, but we also see that the over-fitting is not as dramatic due to the higher accuracy achieved in the validation set. This shows that even though the model is far too good at classifying the training set compared to the validation set, it will still be better that the previous model at handling more general data. This shows us that data augmentation and regularisation together are quite effective at solving the problem of over fitting and help to generalise our model.

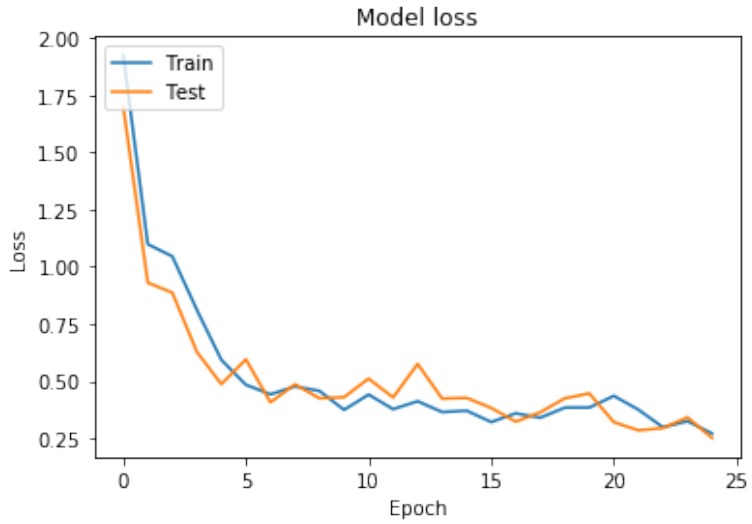




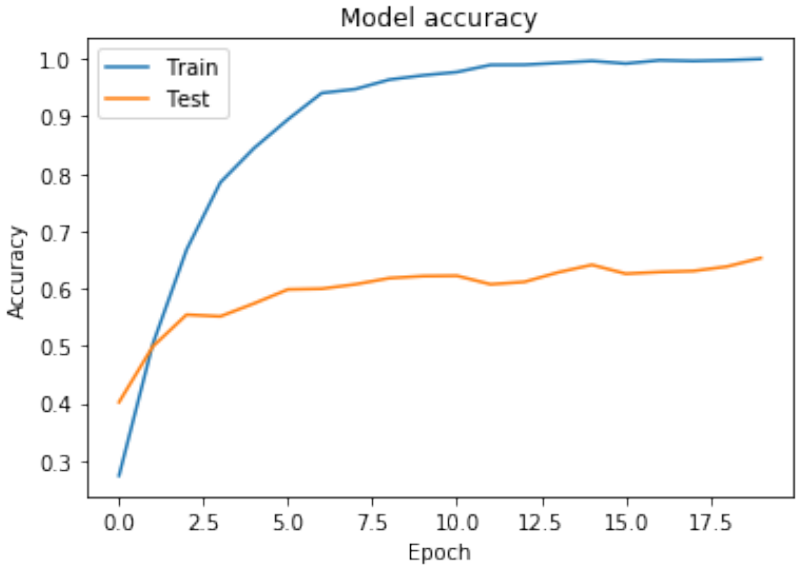
Question 2

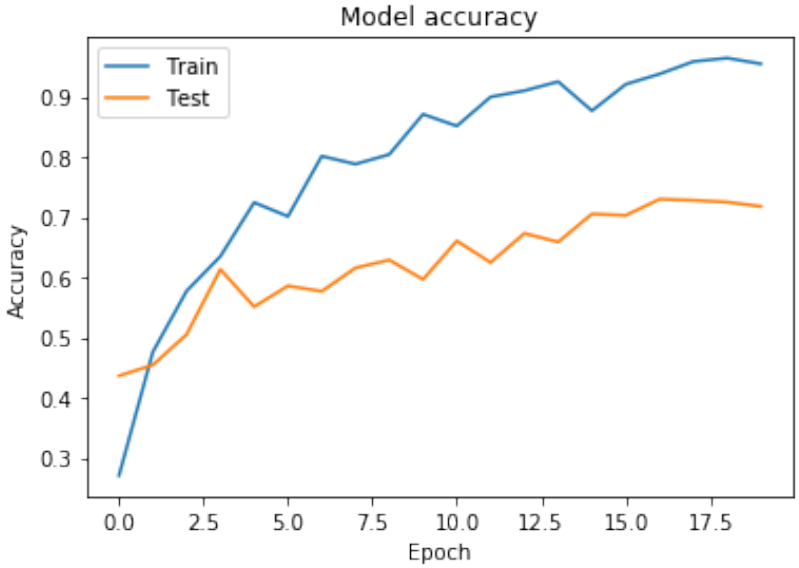
To answer this question, I imported VGG16 from the Keras library. To fine-tune this for our problem, I created a new sequential model with all of the layers from VGG16, except for the last fully connected layer. I replaced the last fully connected layer, which has 9 outputs, with a new fully connected layer that has 9 classes to fit our problem. With a little bit of parameter tweaking, I managed to achieve a validation set accuracy of $\approx 92\%$. More important than the validation set accuracy, however, is the fact that the training accuracy and validation accuracy were consistently similar throughout the training process with the training set accuracy ending up as $\approx 90\%$ too. To achieve this accuracy, I had to alter my batch size to 24 and my learning rate to 0.003. The graphs below shows us that the VGG16 model does very well to remove the problem of over fitting, even with a very small data set like our training set of only 900 images.

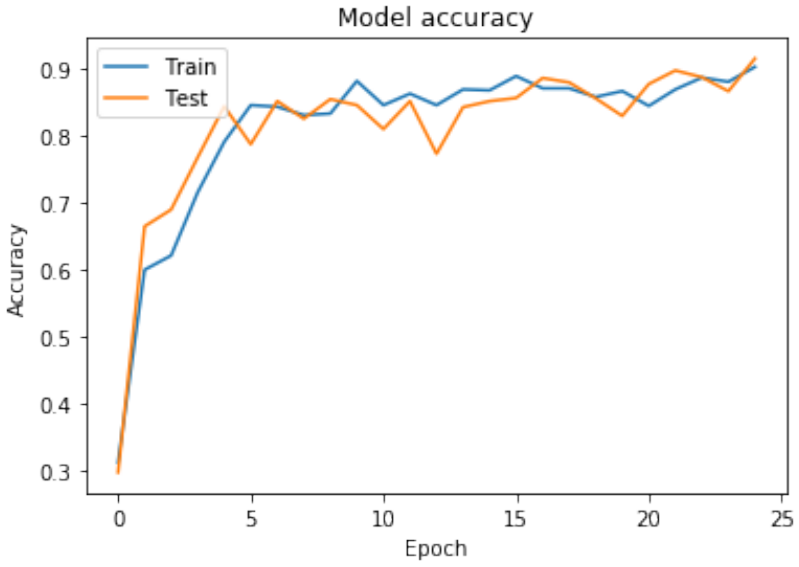




To illustrate the difference between no extra techniques, adding augmentation and regularisation, and using a far more thought out model, here are the accuracy graphs four our three different models. We can clearly see that the general trajectory of the training set accuracy is very similar between all three, but the clear difference is the validation set accuracy. The closing distance between the lines across the three is a great illustration of how a better model reduces over fitting so that it can be more consistently correct for more general problems.


Model accuracy
Train
Test
Accuracy
Epoch
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Model accuracy
Train
Test
Accuracy
Epoch
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Model accuracy
Train
Test
Accuracy
Epoch
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
5
10
15
20
25

References

[1] Amar Budhiraja. *Dropout in (Deep) Machine learning*. URL: https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5.

[2] Shubham Jain. *An Overview of Regularization Techniques in Deep Learning*. URL: https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/.