Assessment 1 - Model Report & Data Analysis

Maxwell Presley

MDS2000185

Introduction

In this assessment the task was to develop multiple CNN and Deep Learning networks to classify sets of images. This report will detail the process and decisions made while developing the networks. It will also show the conclusions made and comparisons between the different networks.

Dataset/Preparation

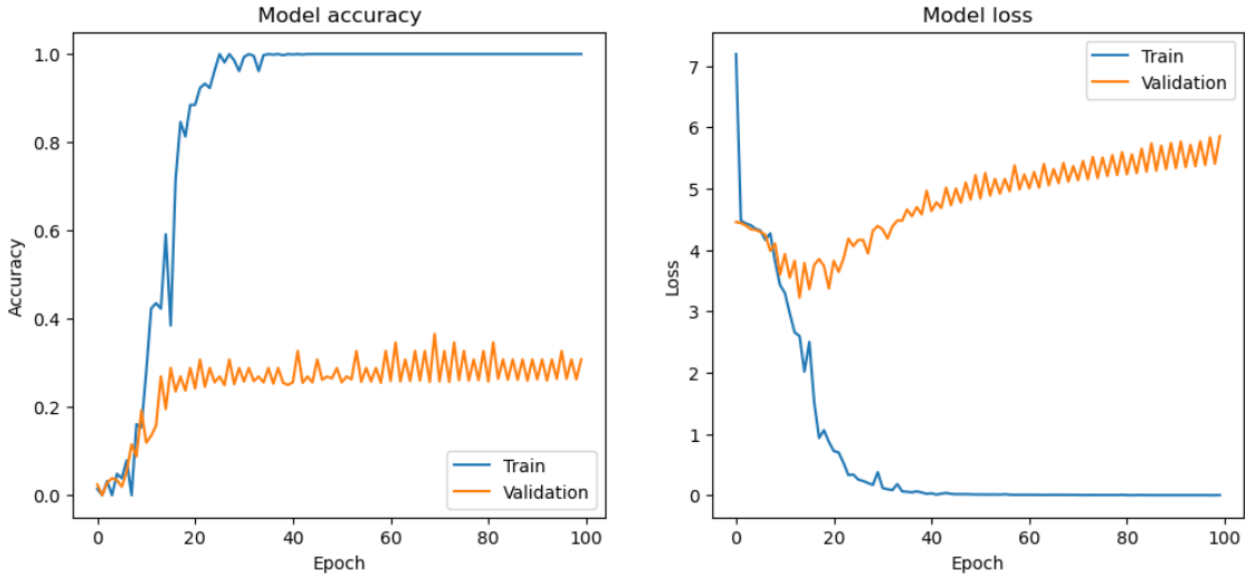
The dataset consisted of 90 classes of different animals. Holding around 5000 images in total. The images were sorted into class folders making it easy to navigate. The images varied in size so they would need to resize them, so they are all uniform. This also made the images more optimal for training. The loop I used cycled through the images in the directory and resized them to 128x128. A size of 256x256 was used but this made the training time too long making it impractical to work with. Next, code was created to check for dupe images as duped images will cause problems in training like imbalances or not accurate training/testing metrics. There were no dupes found. Splitting of the data into train, test and validation was done using train_test_split from sklearn.model_selection. Initially, a split of 80-10-10 was used but this was changed to a more balanced 60-20-20. This better split made the model more accurate.

```
# Walking through the parent directory
for root, dirs, files in os.walk(parent_directory):
    for file in files:
        # Different file types incase of multiple formats
        if file.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp', '.gif')):
            file_path = os.path.join(root, file)
            print(f"Processing file: {file_path}")

            try:
                # Opening the image and resize it
                image = Image.open(file_path)
                image = image.resize(resize_size)

                # Extracting the folder name (animal name)
                folder_name = os.path.basename(root)

                # Creating output directory for each animal type
                save_folder = os.path.join(output_directory, folder_name)
                os.makedirs(save_folder, exist_ok=True)

                # Making the save path
                save_path = os.path.join(save_folder, file)

                # Saving the resized image
                image.save(save_path)
                print(f"Successfully saved resized image to: {save_path}")

                # Clearing image from memory ready for next one
                del image
            # Catching any errors if they occure
            except (IOError, SyntaxError) as e:
                print(f"Error processing image {file_path}: {e}")
```

One Convolutional Layer

When creating the first CNN network the image size was 256x256 which caused the model training to be very slow. This is because the image size has more detail than something like 128x128, which can be good for training and getting better results as there is more to learn from however at the cost of taking too long it was not a good option in this case as 128x128 still gave good results at a faster train time. A faster train time was essential for this assessment as there would be a lot of tuning and testing to see what parameters best suited this problem. One parameter that was experimented with a lot was batch size. Batch size is how many images the model looks at in one chunk. A batch size of 32-256 was tested and from this it was found that a batch size of 128 was ideal as it reduced the number of fluctuations the accuracy encountered, ultimately making it more consistent. No noticeable train time difference was noticed. The model achieved a train accuracy of 0.99, a test accuracy of 0.28 and finally a validation accuracy of 0.26. It is very unusual to see train accuracy to go and stay at 0.99, the train increase rate was very experiential. The validation and test accuracy appears to be overfitting but also get stuck in a zigzag that

could indicate no more improvements are being made.



Two Convolutional Layer

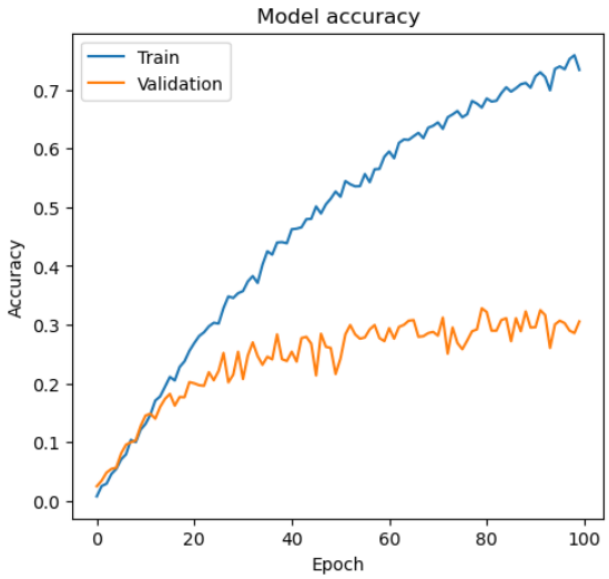
Uping the complexity slightly and increasing the convolutional layers to two saw not much of an improvement. Validation 0.28, Test 0.27 and Train 0.99. Something that were tried to help this was changing the activation from relu to sigmoid. However, this didn't help as it is more for binary problems. Inclusion of autotune was necessary as every second epoch would have a near 0.0 accuracy, introduction of autotune fixed this. The graph for this looked very similar to the first graph.

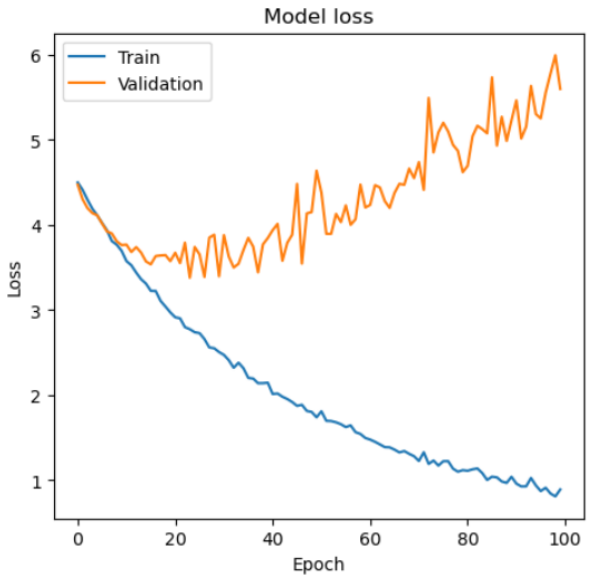
```
# Build the CNN model
model = models.Sequential([
    # Convolutional layer
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(IMG_HEIGHT,
    IMG_WIDTH, 3)),
    layers.MaxPooling2D((2, 2)),

    # Additional convolutional layers
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    # Flatten and dense layers
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(NUM_CLASSES, activation='softmax')
])
```

Three+ Convolutional Layer

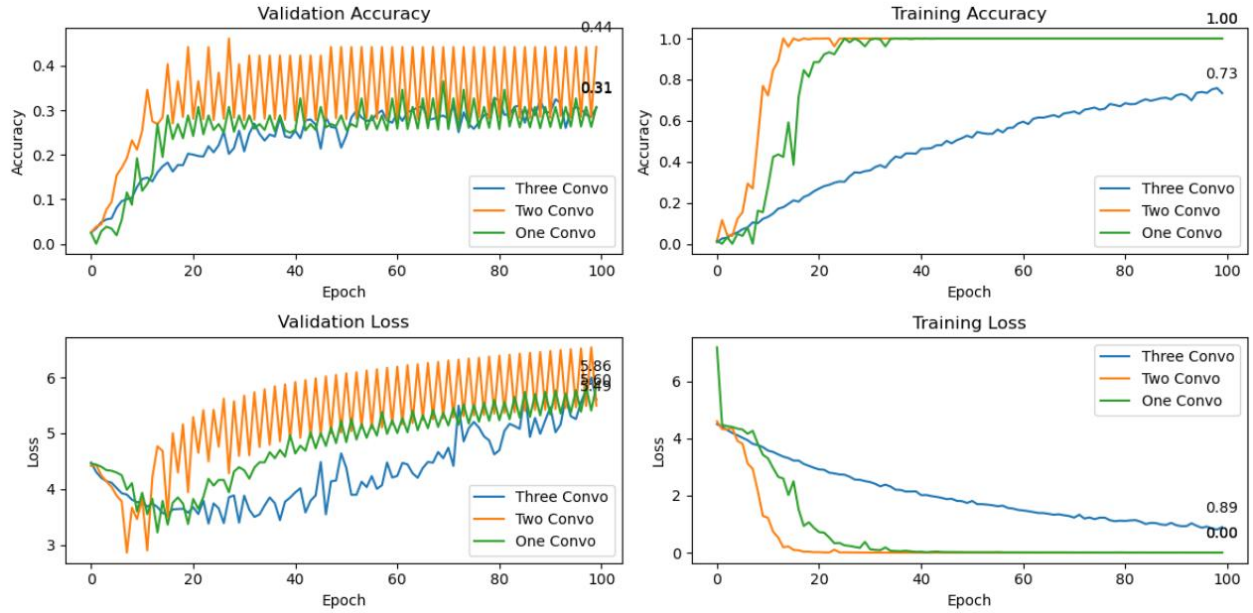
Finding the most optimal and highest performing network is hard as there are a lot of factors to consider. Networks with convolutional layers from 3-5 were tested and

evaluated. The sweet spot was found to be 4 as it gave the best accuracy without taking too much longer to train. 4 layers was very similar to 5 layers but was quicker. Introduction of repeat() aided in training as it stopped the model running out of data for every second epoch. Data augmentation is a good way to make the model more robust by changing the data by flipping, rotating and zooming. Adding this helped make the model more stable and increase overall accuracy. Validation 0.30, test 0.29 and train 0.72.

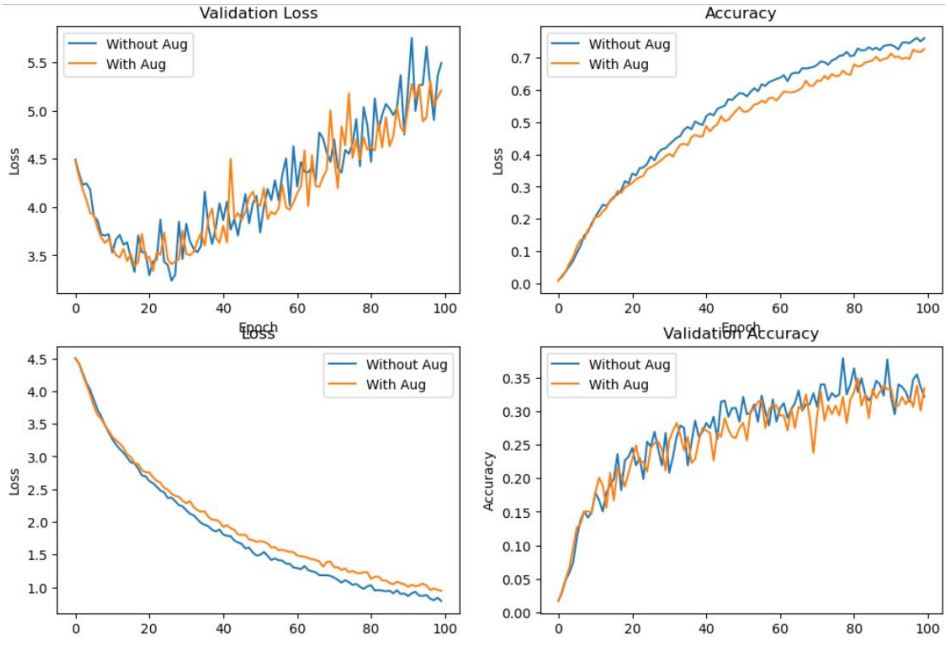




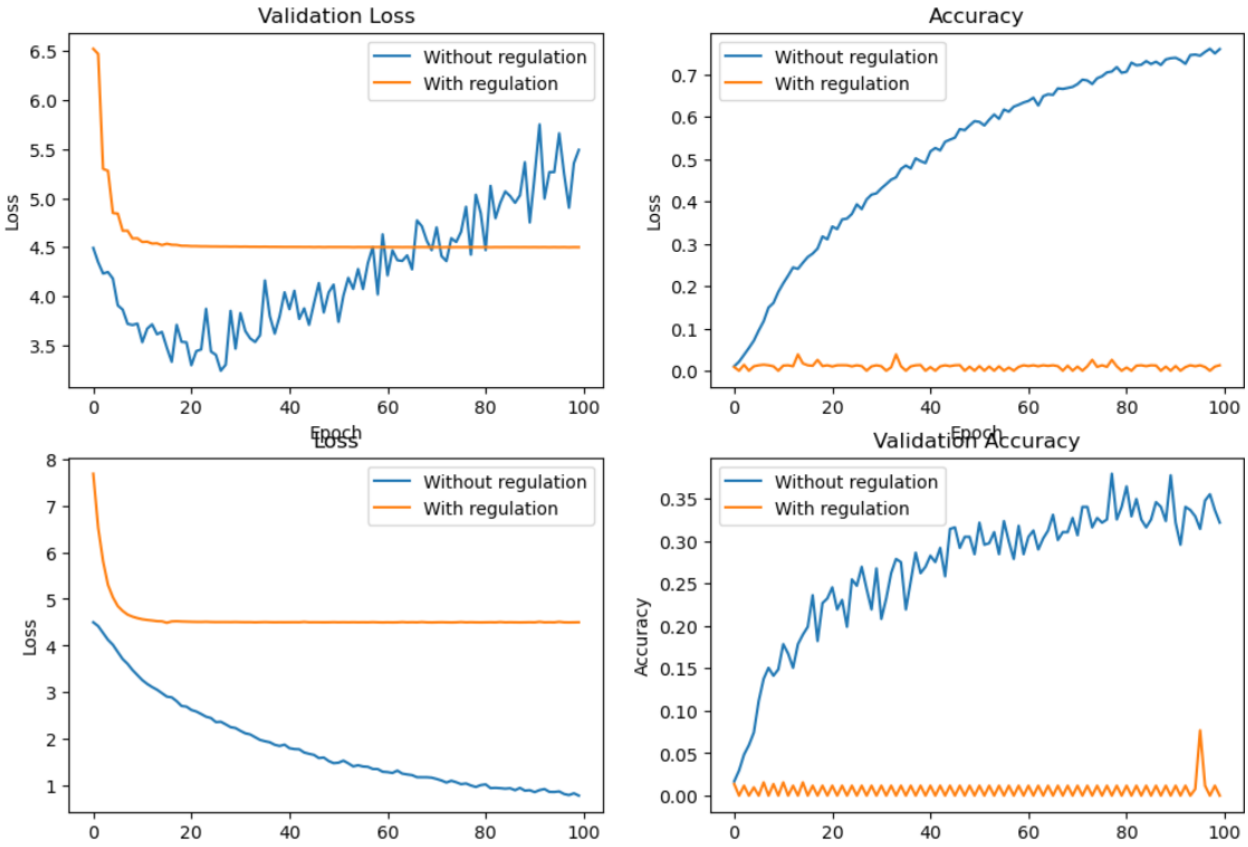
From the 3 networks being graphed it is clear that none of them are able to perform well unfortunately. There is a lot of over-fitting and problems. A larger dataset could help this problem. It would be good to test out even more complex models but due to the limitations that were faced, it wasn't practical to do so.



Comparing augmentation to no augmentation took place and it appeared that they were very similar performing with no augmentation performing slightly worse. It was expected that augmentation was to make a much better impact than it did. More parameters were edited but none gave any tangible results change.



Regularization and drop out can be used to increase the performance of a network. L2 is the method that was chosen. The model didn't take well to it as it can be seen from the graph. The network failed to learn despite changing the values of drop out and Regularization.



When choosing how many epochs to train the model for it depends on a few things. The models were trained on 100 epochs as anymore would take too long to train. There wasn't any need for an early stop for if the accuracy doesn't increase but it was tried. 40 epochs was tried when the image size was 256x256 as 100 took too long at that point.

Conclusion

In conclusion, the networks trained saw a slow by steady improvement over development time. The three networks performed similarly but as the complexity went up so did the performance. 5 epochs was the limited. The networks could have performed better if the dataset was larger or even of better quality (not that the quality was bad tho). With more time and computational resources, the network could have been developed more to a point where the accuracy is much better. Changing the split of training, test and validation did see a increase in accuracy but only marginally.