**DLE305 - Deep Learning**

**Assessment 3 - Simulation Assignment**

Tran Van Anh Nguyen - A00050585

Ali Saeed - A00045120

Ganghao Tim Wang - A00074509

**Table of Content:**

Title Page: Page 0

Table of Content:page 1

Abstract: Page 2

Introduction: Page 3

Dataset: Page 3

Methodology: Page 3 - 5

Experimental Setting; Page 5 - 10

Empirical Results: Page 10

Conclusion: Page 10 - 11

References: Page 11

**Abstract:**

For assessment 2 we were individually given the task to make an image classifier using convolutional neural networks. We followed the example given in the article, “Convolutional Neural Networks: A Python Tutorial Using Tensorflow and Keras”. For assessment 3, Ali Saeed, Tim Wang and Van Anh Nguyen are able to work together and improve on the previous model we built individually, enabling the model to be able to recognise lions and tigers in addition to cats and dogs images. We reproduced our previous model in assessment 2 and added a pre-trained model, VGG16 with data augmentation to produce efficient and accurate results.

VGG16 is an object detection and classification model that utilises convolutional networks. It is considered one of the best architectures to date in performing the tasks of classification efficiently and accurately. It has 13 convolutional layers and 3 fully connected layers with very small convolutional filters, making it a distinct architecture from other configurations. The model excels in fine grained spatial information, reducing noise and controlling overfitting. During its showcase, it was able to classify 1000 images of 1000 different categories boosting a score of 92.7% accuracy. Using transfer learning, we will implement this model to improve on our previous model to improve accuracy as well as customising to a much larger data set. (G, 2021)

Using Visual Studio Code as the main editor, we performed data pre-processing on the new dataset. Both datasets we are using came from Kaggle; we downloaded the new dataset, “Lions-Tigers”. We renamed all the images the same way our “cats” and “dogs” datasets were named-by an ordered number system-making it simpler to load the images and pre-process the data. To compare and contrast the differences between our old models and the newly implemented pre-trained model, we tested all the models. This showcases the risks and limitations of the previous models as well as allowing us to understand why and how VGG16 was optimised to perform more efficiently and provide more accurate results.

As we expected, the previous models we implemented to classify the “cats” and “dogs” datasets are no longer feasible in performing classification on the new data. This may be due to a variety of different reasons. Data diversity might be hindered through the previous model’s simplicity; the model needs to be fine-tuned accordingly to capture the increased diversity. We need to optimise the models by fine-tuning its learning rates and training time to adjust to the increased amount of data. It would cost a lot of computational power and hyper-parameter testing. Using VGG16, we discovered how the pre-trained model was able to compute much more accurate results. However, it is still not a perfect model. Further optimisation and fine-tuning need to be tested on the pre-trained model to reduce overfitting and underfitting to produce better and more accurate results.

**Introduction:**

Previously on assessment 2, we were able to analyse and compare a baseline MLP model to a CNN model. We reached the conclusion of how CNN models are able to produce much more accurate results. The previous CNN model still needed optimisation with hyper-parameter testing and more attention to data pre-processing but we were able to see first hand how the model was getting increasingly better with each additional feature.

For this project we will still be improving on the previous image classifier model using Tensorflow’s API. We will add two new data sets, “lions” and “tigers” and use the transfer learning method to implement a pre-trained model. We will then use data augmentation to the trained data. Data augmentation has many advantages including reducing overfitting, one of the major issues with the previous model. It helps generalise data, enable better feature learning and help balance classes that are underrepresented.

**Dataset:**

We will use the original “cats” and “dogs” dataset from Kaggle containing about 12500 images of cats and dogs. We will also adopt the “lions” and “tigers” dataset from Kaggle to improve the diversity and range of our image classifier. Unlike the “cats” and “dogs” dataset we previously worked with, “lions” and “tigers” had a different naming convention, making it difficult to apply the same code and load the data into the model. We renamed both datasets in order by number with their name at the front. ("Lions-tigers," n.d.) (Kaggle.com. (n.d.))

One of the major drawbacks of the “lions” and “tigers” datasets in comparison to “cats” and “dogs” is that they are much smaller than our “cats” and “dogs” datasets, containing only about 180 images in each file. This is why we have decided to implement transfer learning using a much faster, more accurate CNN pre-trained model to perform image classifying as it does not require a large dataset to work with to produce accurate results. This also helps us reduce storage size without the risk of overfitting.

**Methodology:**

Our methodology for this project follows the steps of our previous image classifier model. Then we use transfer learning to implement the VGG16 model, fine-tuning the data to improve accuracy through data augmentation. Our methodology to implement the model follows these steps.

We first downloaded the new “lions” and “tigers” datasets from Kaggle. We then renamed both files in order of the animal they represent and in ordered number sequences. This enables them to follow the same naming convention as our “cats” and “dogs” datasets, making data pre-processing easier and more efficient.

Data pre-processing follows the same steps as our previous model. Using the Python Image Library to load our image data as a numpy array. We first calculate the distribution of the most common shapes in the datasets. Especially for this project we are working with a wider range of data diversity, increasing our storage size. We need to be careful not to create noisy data. By calculating the most common shape, we can resize the images and ensure a fixed resolution, ensuring a cleaner dataset whilst not over-distorting the image data.

10% of the dataset is automatically used for validation. The dataset is then resized to the most common shape, and converted to RGB values. Using RGB values for an image classifier helps the model distinguish the colour contrast between the target image and their background. Greyscale would save the model a lot of processing power, however, it does not boost very accurate results as the image classifier model may have trouble distinguishing the background from the actual image.

We can then store them into 4 different numpy arrays, cat\_train\_set, dog\_train\_set, lion\_train\_set and tiger\_train\_set. We repeat this step to create 4 more numpy arrays for the validation dataset namely cat\_valid\_set and so on. Using these stored arrays, we can train our model by creating a training and a labelled dataset. The training dataset combines all the images from the “train” numpy arrays and the validation dataset contains all the labels for the train datasets. Similarly to before we also create two more validation datasets, for train data and the labels of the trained data. Validation datasets help us evaluate the true performance of the model.

Unlike with the previous model where we used binary classification to categorise “cats” and “dogs” as values 0 and 1, we modified the model for multi-class classification using a Keras library to\_categorical. This allows us to sort all 4 datasets into multiclasses: 0, 1, 2, 3 for cats, dogs, lions and tigers respectively.

To evaluate the effectiveness of our new pre-trained model, we tested the same model as the one we previously built in assessment 2 and then compared it to the VGG16 model. To implement the new pre-trained model, we implemented data augmentation to boost accuracy. Using the “ImageDataGenerator” from Keras, we effectively utilise its parameters to flip, rotate, and zoom into images. This data augmentation technique helps generalise the dataset, enabling it to become more robust and adaptive to data changes in different environments. (Awan, 2022)

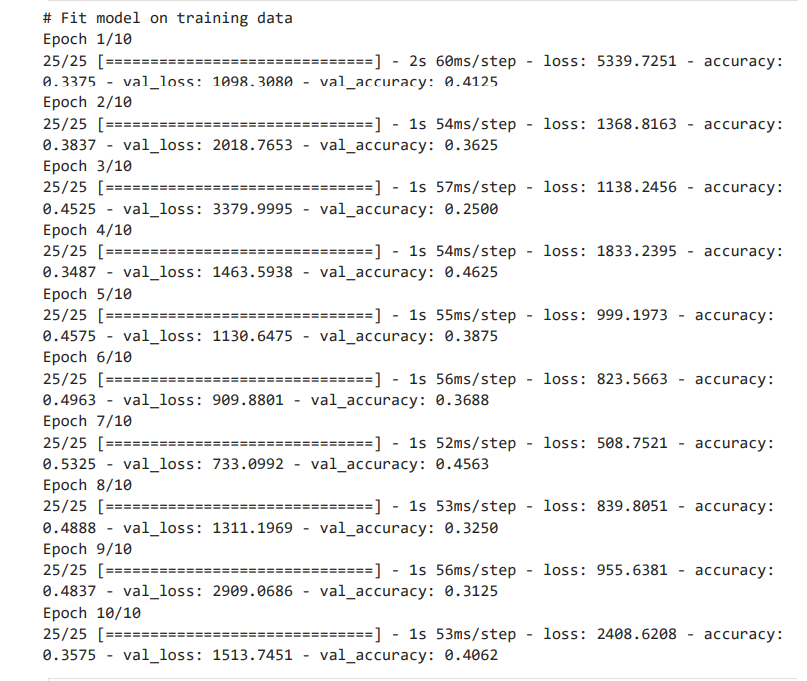
We will use AdamOptimiser as the optimiser, categorical-cross-entropy as our loss function and accuracy as the loss function metrics. Accuracy is the most appropriate metric for our task as it is more commonly used for classification tasks. Previously in assessment 2, we implemented MSE as the evaluation metrics, however we have come to learn that MSE is more suitable for regression tasks. Accuracy provides a much better picture of how well the classifier is performing based on the total correct instances out of the total instances. We can better interpret and evaluate the performance of the model and fine-tune it for better accuracy.

We did use hyper-parameter testing on the model to see the differences in changing the convolutional layers and the fully connected layers in improving the accuracy of the results. We added an additional dropout layer after the fully connected layer to provide more generalisation and prevent overfitting. Batch normalisation was implemented for the Huge Model to help stabilise and speed up training. We then use the VGG16 architecture for feature extraction using transfer learning.

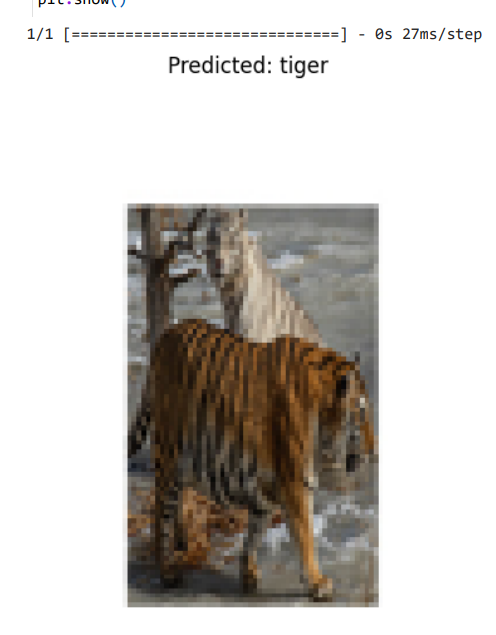
**Experimental Setting:**

MLP and CNN Models before VGG16 and Data Augmentation:

Run-of-the-Mill MLP Model:

To experiment with the effectiveness of using a pre-trained model as well as data augmentation, we need to experiment with the previous models and compare the differences in the results and how we were able to improve on the previous models based on what we learnt. 

Here is the result after 10 epochs using a run-of-the-mill MLP model with our new datasets. The loss is extremely high whilst accuracy is extremely low. The model was gradually getting better, however, it was not a major improvement.

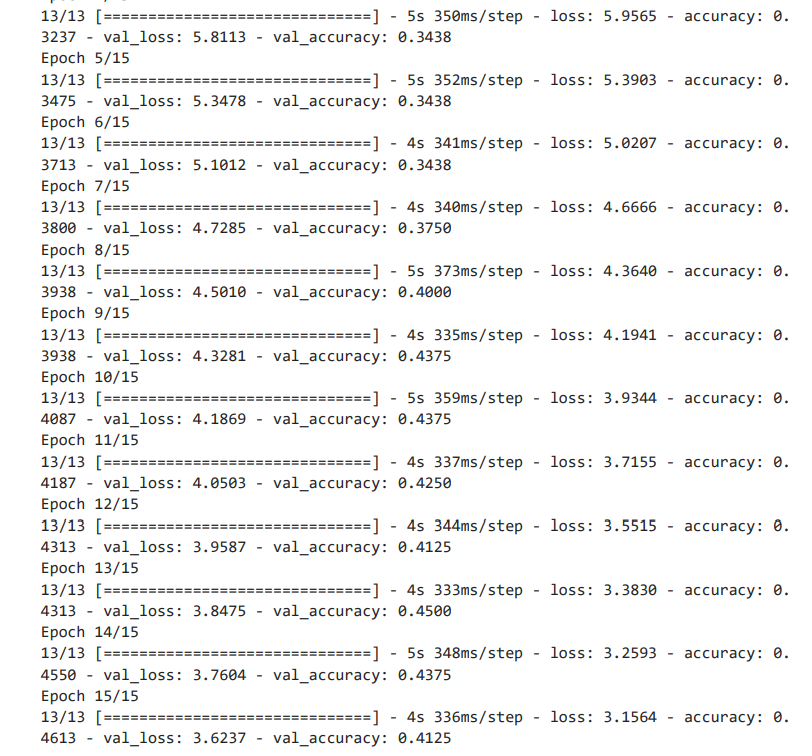


What we can take from these results suggest that with the initial epoch results being extremely poor, this model architecture is not well suited for this task. Validation sets are not doing very well either, however, it did not produce extremely jarring results. We can see that the model is not good at generalising the data. Accuracy results and loss fluctuate with each epoch and the model does not learn consistently. This suggests the model is overfitting and/or the datasets do not have enough data. It is likely that the “lions” and “tigers” datasets do not have enough image data, especially in comparison to the previous “cats” and “dogs” datasets.

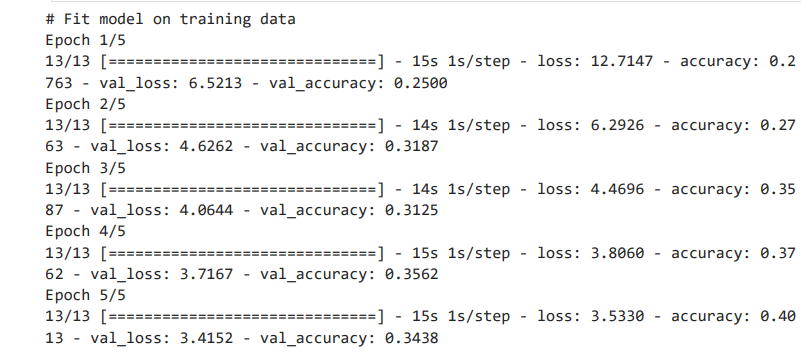
Using matplotlib, we were able to get a visual representation of the model predicting in real time. The library allows you to load and preprocess test images, make predictions using the model created, interpret the results, and finally display the test image and predicted label. Despite earning quite a low score on all prediction metrics, the model was able to correctly identify an image of a tiger as shown here. However these results are not consistent and more hypermeter needs to be done in order to produce consistently accurate results.

CNN Models:

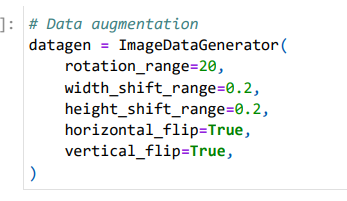
Previously with our assessment 2 tasks, we tested out 3 different convolutional network architectures. We concluded that each model improved as we increased the fully connected layers and made each of our convolutional layers bigger. The best model assumed more fully connected layers and a bigger convolutional layer with more filters. From what we learnt of the previous task, we can utilise our knowledge and apply to this new project. We will quickly analyse the results of each CNN model from our previous project on this new data and compare them to the VGG16 model with Data Augmentation.

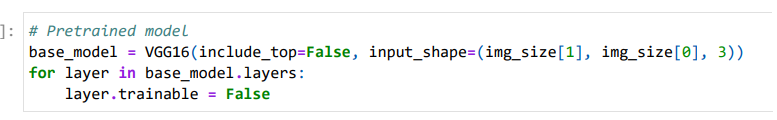
This is our single layer CNN model. As we can see the initial epoch has a pretty height loss metric with very low accuracy. Validation loss and accuracy also reflects the same results. It shows we are not off to a very good start and this model may also not be the best architecture to perform the task for these datasets. 

As we can see from epoch 10-12, the validation accuracy actually decreased instead of increasing whilst the training accuracy is improving. It shows our model might be overfitting. The validation accuracy eventually got better but it started to decrease again. This means overfitting is definitely a concern for this model. Once again we believe it may be due to the significantly smaller dataset we had to work with for lions and tigers. To continue using this model we will have to apply early stopping to prevent overfitting.

Here are the training results of our Huge Model. It contains two fully connected layers, 2 convolutional layers with 128 filters and 256 neurons. Previously, this was the most accurate performing model; however, with our new data, it has the worst loss metrics results that we have seen so far. We can see that after epoch 3, the model was still struggling to learn, extract features and find patterns in the image data. It improved a little after epoch 4 but ultimately increased in validation accuracy even if accuracy increased. This means that the model is not good at generalising data. 

Using Pre-trained Model with Augmented Data:

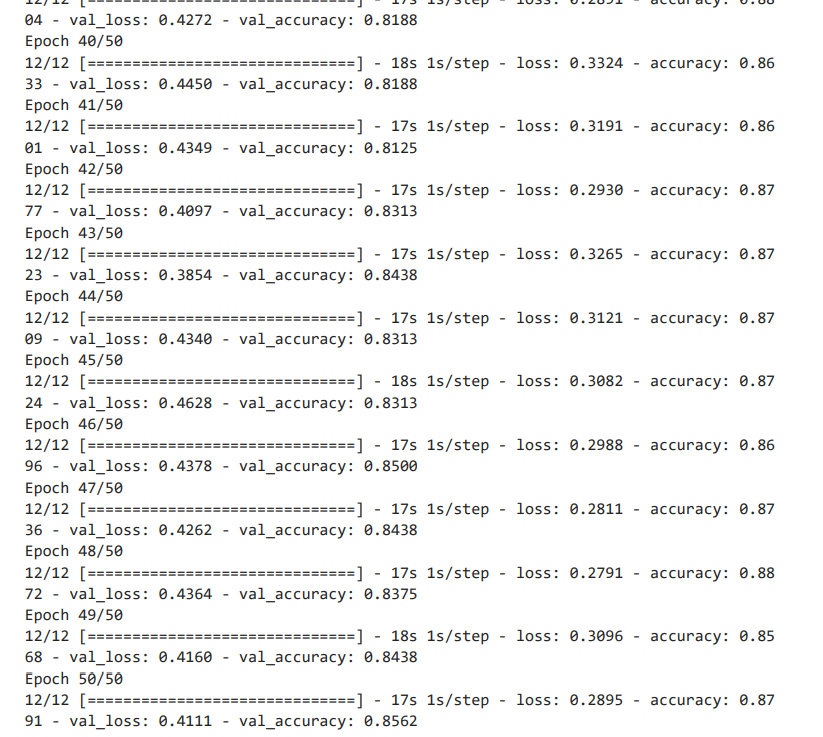
Data augmentation is the process of using the original data and making some minor changes to increase dataset size and data diversity. In the case of image data like what we are working with, this involves flipping, resizing, cropping, zooming. In the following code, the datasets will be augmented by random rotation of images by up to 20 degrees; random horizontal and vertical shifts of the image up to 20% of its width and height; randomly flipping the image horizontally or vertically. Data augmentation can prevent problems that we have been encountering such as overfitting, training dataset being too small (in the case of “lions” and “tigers”), improving model accuracy, and saving time to find new raw data and re-labelling the dataset.

For the pre-trained model we are using VGG16. VGG16 is a pre-trained model created for object detection and classification. It is one of the most well-known and accurate models for image classification to date. Layer.trainable is set to “False”, this freezes the layers, preventing them from being updated during training. This is because the bottom layer of the model already contains feature extractors. Include\_top is set to “False” so we can change and add our own classification layer. (G, 2021)



Dropout is a regularisation technique that helps prevent overfitting in our image classification model. Setting dropout to 0.5 means we are dropping 50% of neurons of each previous layer after every iteration. This helps the model not rely too much on previous neurons which helps with generalisation and prevents overfitting. BatchNormalisation helps normalise the activations of each layer’s neurons. BatchNormalisation helps generalise data, improve training stability, and helps the model’s learning rates.

For this model, we ran through the data with 50 epochs. We will analyse the results of the last 10 epochs.

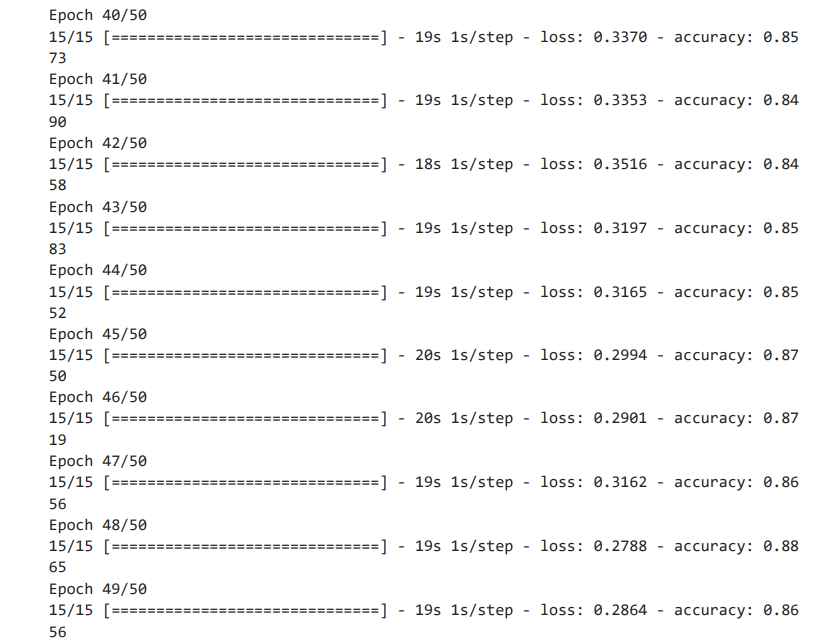
We can already see the significant difference between this model’s results and the previous model’s. Loss is significantly lower than previous and accuracy led a staggering height of 86%. From our last project the highest accuracy we averagely reached was about 75%. Using the VGG16 pre-trained model with Augmented data without any hyperparameter testing or optimisation, we have already reached new heights with the results. Validation accuracy is still a little lower than training accuracy, however, that is normal and is to be expected. 

The model was continuously improving with training and validation accuracy. The model was still learning without remaining stagnant. It was able to optimise its performance. After epoch 45, the results were the same as previous, we had originally thought it may be that the model has reached its optimal performance and could not improve more after. However, it jumped again after epoch 46, showing the model was learning more complex patterns with different data variations.

Larger Model Using Pre-trained Model and Augmented Data:

After the previous model, we knew that our pre-trained model was working well, boosting great accuracy and low loss. It was learning complex patterns and improving after each epoch. For the larger model, we utilised what learnt from the previous model and added some hyperparameters that may help improve the model even further. Clearly from our previous project, making the model bigger helped make it more accurate and efficient.

We added another fully connected layer to allow the model to capture more complex relationships in the data. We added another dropout layer to provide more regularisation to prevent overfitting. BatchNormalisation was applied after the fully connected layer to help stabilise and speed up training.

We will analyse the last 10 epochs again as we are iterating over the datasets 50 times. Looking at these results, there is not a major difference in the results with the initial model architecture. Loss and accuracy did not fluctuate greatly between the two models; some iterations were better than the model and some were not. Nothing noteworthy has changed with the previous pre-trained model. This may indicate the model has reached its optimal performance. To try and reach new heights, very delicate hyperparameter testing and fine-tuning may need to be in place to reach a better accuracy and efficiency. 

**Empirical Results:**

After experimenting with our previous models built for assessment 2 and the pre-trained model with augmented data, we can safely conclude that we have significantly improved our model and optimised it for image classification. Despite our attempt to improve on the model by using the methods as previous, making the model bigger and adding more layers, the model reached its optimal performance with the initial test. We reached 87% accuracy with this model, significantly higher than our models of assessment 2.

Transfer learning is a great method to utilise trained features from an optimised model, saving storage size. It is an extremely effective method for our dataset because it was so prone to overfitting.However, despite our efforts to improve on the model, nothing noteworthy changed. We will need to fine-tune the model even more by doing hyperparameter testing. We did not want to randomly add more layers to the model due to a time constraint, overall runtime and RAM size.

One of the main problems we encountered whilst doing this project was our mistake in using binary classification for 4 datasets. We initially assigned multi-class values to a binary method which operates on 0 and 1. It took a lot of time away from our project, hindering optimisation and hyperparameter testing. However, that was fixed when we noticed our mistake and changed our classification and loss function to categorical classification and categorically cross-entropy instead. The main problem with our previous models is that it was very prone to overfitting. We fixed these problems with our pre-trained model using augmented data, dropout layer and batch normalisation. We reached new heights with the results and achieved a great loss to accuracy ratio.

**Conclusion:**

This project allowed all three of us to work together and discussed how we could improve on our previous codes and produce an image classifier that was more accurate and efficient. We realised the limitations of our previous code and why it was not working for the new datasets.

The new datasets were significantly smaller than the previous datasets, making it unsuitable for our previous models. Our previous CNN models were not able to learn complex patterns in the image data and it could not adapt to the data diversity and image variation. It was also extremely prone to overfitting due to how much smaller the dataset was compared to the previous model and it did not have the right fine-tuning to improve on its accuracy and efficiency.

Luckily we were able to overcome these issues with academic readings and research as well as our own efforts to make a better and more accurate model. We implemented the correct measures to overcome overfitting and stabilising the dataset after each iteration. The project provided our group with insights to problem solving, group collaboration and critical thinking and analysis skills to improve on our previous models.

**Word Count:** 3487

**References:**

* Awan, A. A. (2022, November). A Complete Guide to Data Augmentation. Learn R, Python & Data Science Online | DataCamp. <https://www.datacamp.com/tutorial/complete-guide-data-augmentation>
* G, R. (2021, September 23). Everything you need to know about VGG16. Medium. <https://medium.com/@mygreatlearning/everything-you-need-to-know-about-vgg16-7315defb5918>
* Lions-tigers. (n.d.). Kaggle: Your Machine Learning and Data Science Community. <https://www.kaggle.com/datasets/akrsnv/lions-and-tigers>
* Luciano, S. (n.d.). Convolutional neural networks: A Python tutorial using TensorFlow and Keras. KD Nuggets. https://www.kdnuggets.com/2019/07/convolutional-neural-networks-python-tutorialtensorflow-keras.html
* Github.com. (n.d.). A tutorial on convolutional neural networks with TensorFlow eager API. https://github.com/StrikingLoo/Cats-and-dogs-classifier-tensorflow-CNN
* Kaggle.com. (n.d.). Dogs vs. cats: Create an algorithm to distinguish dogs from cats. https://www.kaggle.com/c/dogs-vs-cats