Dog Emotion Classification: A New Approach

Salman Salman

November 25, 2022

Abstract

The facial expressions of animals have been identified as manifestation of emotion (Dolensek et al., 2020). However, the challenge remains how to classify these emotions and understand them. The objective of this report is to conduct a study on the images of dogs and classify the emotions of each dog using deep learning algorithms, and to provide an new approach to solve this particular classification problem. The data set used in this report– Dog Emotion data set– can be found here. Firstly, we will explore the data set and prepare it for classification. Next we will present a basic model (a modified version of LeNet5) and compare it to two pre-trained models; namely, VGG16 and ResNet50. To fetch the data, we will use meta-learning, specifically few-shot learning. Lastly, results and potential optimisations will be discussed. All the code discussed here can be found in the accompanying Colab notebooks, here (LeNet5 model), here (VGG16 mode) and here (ResNet50 model).

1 Introduction

It is often hard to discern what a dog is feeling. And since dogs cannot talk, we cannot ask them either. For this reason, we are left with one option – to guess. But guessing is usually problematic since humans cannot memorise a whole host of emotions and facial expressions, much less those of dogs. Nevertheless, dogs seem to have developed facial expressions that are easy to understand so they could communicate with us [1]. This is where deep learning comes into play (He et al., 2015). In this report, we analyse only four emotions that dogs experience: happy, sad, angry and relaxed. Current studies show that dogs have recognisable facial expressions (Mota-Rojas et al., 2021). Based on this fact, we can train a neural network to learn these facial expressions. Using deep learning, we could develop an algorithm that could be made accessible with the goal of helping humans assess the emotions of their dogs.

For a human with no prior experience with dogs, the chances of correctly gauging the emotions of a dog solely by looking at its face are 25%; essentially guessing one emotion out of four. With some experience, however, the probability will increase. Unfortunately, some humans are simply not good at guessing emotions, not least the emotions of dogs. This report, along with the accompanying code, could serve as the basis for an app that helps dog owners gain a better understanding of their dog's emotions just by scanning its face, especially for new dog owners and people who are not very good at reading facial expressions. Tough this task might be exceedingly difficult for humans, it could be formulated as a classification problem and then solved with a neural network (NN). As a benchmark we will look at the random probability of picking one emotion out of four (i.e. 25%) and compare our results to current results on the same data set. Moreover, we will compare our results to open-source notebooks on Kaggle[2]. Ideally, our NN should outperform both random selection, and at the very least, achieve a similar performance as the Kaggle examples.

We will firstly introduce the problem and formulate it mathematically. Secondly, we talk about data preparation and preprocessing, that is, how the data are arranged and represented. Then we discuss the model we are using. For each model we will list the results and discuss them and compare the models and their respective performances. Lastly, we discuss possible improvements to our models and to the data set in general.

[1] see https://dogsbestlife.com/dog-fun/dog-facial-expression/

[2] see this link for the notebooks: https://www.kaggle.com/datasets/devzohaib/dog-emotions-prediction/code

2 Methodology and Architecture

2.1 Preprocessing and Preparing the Data

The data set used in this report was taken from Kaggle. The original source of the data, however, is unclear[3] The data consists of images of dogs. It comes in four folders, each folder contains images corresponding to one emotion. For example, the angry folder contains images of angry dogs. the sad folder images of sad dogs and so on. It is important to note that the data was *probably* collected by asking humans to classify the emotions of dogs, so it is not perfect and already introduces some bias, since we cannot be sure what the dog is actually feeling; we merely have a consensus. Furthermore, in some images the dogs are small or appear next to other objects. As a result, the learning task becomes somewhat difficult. To make the data legible for a neural network, we need to create features and labels out of each image. This can be modelled mathematically. In the next section, we will look at the mathematical representation of data.

2.2 Mathematical Representation

As is typical in machine learning, the data is divided into features and labels. The feature vector is usually denoted with X and the label vector with y. For images, the feature vector consists of the pixel values for each image. These range from 0 to 255. Moreover, each image is represented using three colour channels. As for the image size, each image has a fixed height and width. In our case, each image has height and width of 224, that is, it is a 224 x 224 matrix. Mathematically, the entire set of image tensors can be written as follows:

$$X = \{x \in \{0, ..., 255\}^{n \times n \times c} | \; n = 224, c = 3\}$$

.

The labels vector, y, contains the one-hot encoding(s) of the categories. Mathematically:

$$y = \{e \in \{0, 1\}^n | \; n = 4\}$$

The size of each set varies depending on how much data we use. Computational restrictions, such as the amount of RAM available to us, can limit the size of our data. In this report, the 3400 random data point were chosen from each folder. In total, 13600 images, with their corresponding labels. This approach is called few-shot learning[4].

2.3 Preparing the Data

There are a handful of steps used for preparing the data. For images, these are usually shuffling, normalisation, conversion to on-hot encoding, and resizing. After fetching the data the we apply a random shuffle. For normalisation, we simply divide by 255 and all images were resized to 224 x 224. Moreover, the data is split into training, validation, and test sets. For the train/validation/test split, I chose 70/15/15. There is no particular reason for this choice, I started with it, and it seems to work fairly well, so I did not alter it.

2.4 Model

For image classification we typically use a CNN (Gonzalez, 2021). In this report, we will look at three CNN classification models, starting with a simple architecture and work our way towards more complex ones. In each model, we look at three things: the model architecture, accuracy and loss for our data set, and the confusion matrix containing the classification results.

2.4.1 LeNet5

The first relevant model is LeNet5. LeNet was developed by Yan LeCun and is well-known for performing well on image classification tasks (LeCun et al., 1998, Rosebrock, 2016). Figure 1 shows the original architecture of LeNet5.

[3] this presents us with some issues that will be discussed later in the report.

[4] For more examples on few-shot-learning, see https://dl.acm.org/doi/abs/10.1145/3386252

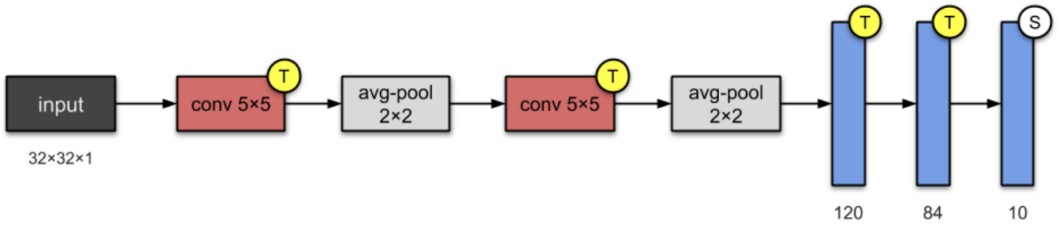

Figure 1: LeNet5 (see source)

Using the model in Figure 1 as a starting point, with dropout and batch normalisation to reduce overfitting and to make sure that the validation accuracy does not get stuck. Figure 2 shows the model after the modifications. The final layer was also changed from having two units to four since in our case we are trying to classify four categories.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 222, 222, 6)       168

average_pooling2d (AverageP  (None, 111, 111, 6)       0
ooling2D)

batch_normalization (BatchN  (None, 111, 111, 6)       24
ormalization)

dropout (Dropout)            (None, 111, 111, 6)       0

conv2d_1 (Conv2D)            (None, 109, 109, 16)      880

average_pooling2d_1 (Averag  (None, 54, 54, 16)        0
ePooling2D)

flatten (Flatten)            (None, 46656)             0

dense (Dense)                (None, 120)               5598840

dense_1 (Dense)              (None, 84)                10164

dropout_1 (Dropout)          (None, 84)                0

dense_2 (Dense)              (None, 4)                 340

=================================================================
Total params: 5,610,416
Trainable params: 5,610,404
Non-trainable params: 12
_________________________________________________________________
```

Figure 2: Model Summary

Results:

This model was trained with the Adam optimiser for twenty epochs and a batch size of 40. The choice for the batch size is the result of trail and error. High values for the batch size (60 and above) yield low validation accuracy and low values lead a fluctuating accuracy. When evaluating the model on the test set we get the following results:

```
Test loss:  2.683903932571411
Test accuracy:  0.6274510025978088
```

Figure 3: Test Loss and Accuracy

Given the complexity of our problem, c.a. $62 - 63\%$ is an acceptable accuracy. In this example on Kaggle, the author, using MNetV2, was only able able to achieve a validation accuracy of 46%. Plotting the accuracy and the loss of the model as functions of the epochs we see both the validation and the general accuracy gradually increasing while the losses diverge from each other. The increase in loss is why the validation accuracy converges. From these results, it is reasonable to conclude that our approach is an improvement to the current open-source examples on Kaggle.

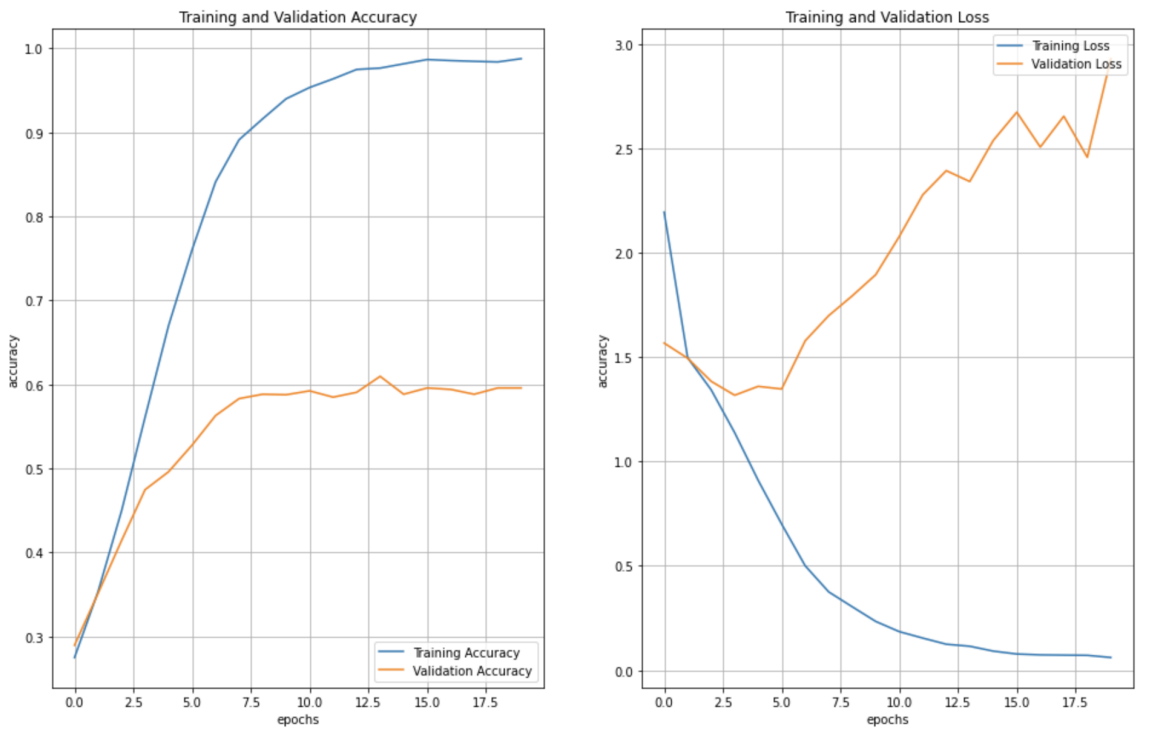

Figure 4: Loss and Accuracy plot

Despite the gradual increase in loss, the model still performs well. The validation accuracy increases with each epoch. The performance of the model is better illustrated with the confusion matrix.

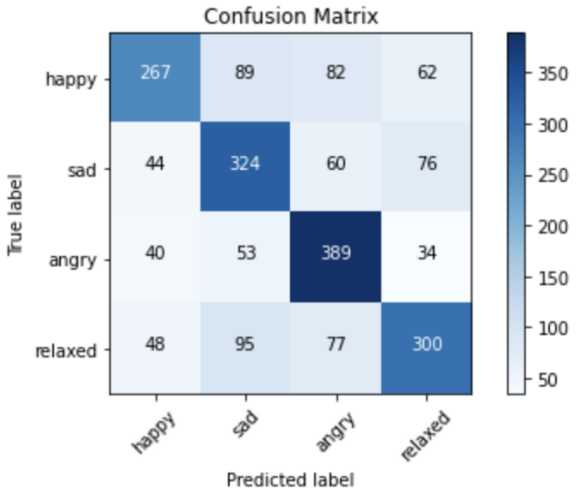

Figure 5: Confusion Matrix

It is clear that the model is better at classifying angry dogs than the rest of the emotions. The reason behind this is the number of images we have for each instances. The angry folder contained the least amount of images, so by randomly choosing over 3000 images we are choosing the same image more than once, which gives the model the opportunity to learn the features better. This strategy delivered better results than using each image only once, which is the strategy used in the Kaggle examples.

We have looked at the strategy presented in this report (i.e. meta learning). Let us look at the results from using each image once without randomly selecting from each folder, i.e., without meta-learning. We observe that the overall accuracy goes down from 62% to 34% (compare Figure 3 to Figure 6).

```
Test loss:  1.4955377578735352
Test accuracy:  0.34282127022743225
```

Figure 6: Test Loss and Accuracy

By looking at the accuracy and loss in each epoch, we can discern that the model's performance has decreased and, moreover, that it became less stable (i.e. it fluctuates more). This can be seen in Figure 7 (compare to Figure 4). Increasing the number of epochs won't improve the results as the validation loss was starting to increase; more epochs are likely to result in a higher loss.

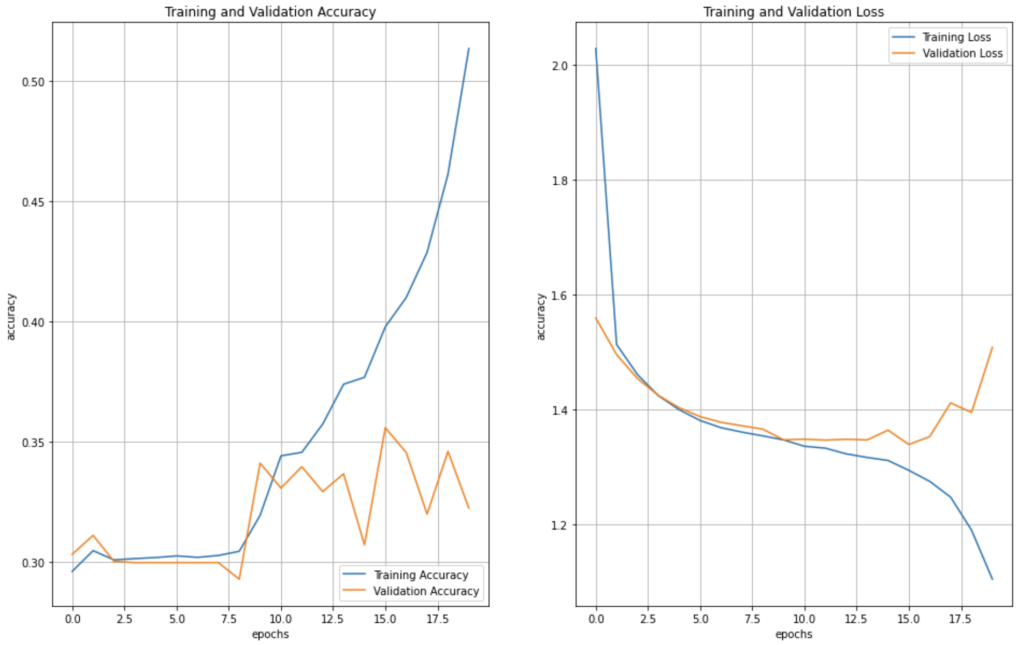

Figure 7: Loss and Accuracy Plot

The confusion matrix looks different as well. The quality of prediction, as the plots show has decreased, thus it is reasonable to conclude that randomly picking a fixed number of pictures from each folder (meta-learning approach using few-shot learning) is more fitting to our problem. Going forward, we will only work with the previous approach–i.e., picking a fixed number of images from each folder– which delivers a much better accuracy than the approach in Figure 7. The purpose of trying out two strategies was to determine which one is more suitable for our data set, and to demonstrate that the approach we are using in this report achieves better results.

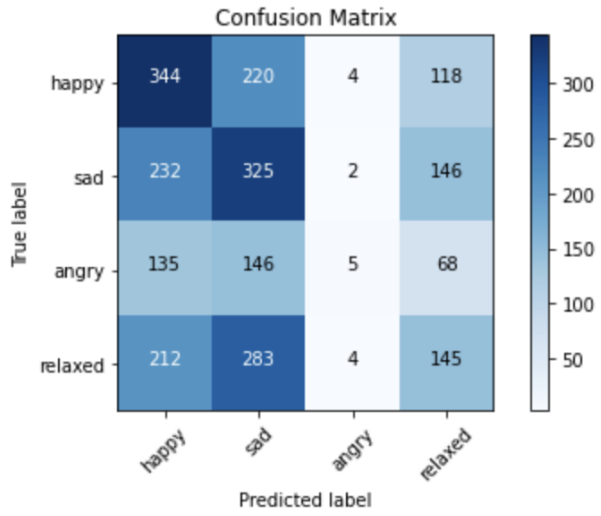

Figure 8: Confusion Matrix

2.4.2 VGG16

VGG16 is a deep convolutional network that won the ImageNet challenge in 2014 (Simonyan and Zisserman, 2014). The VGG16 model has been pretrained to predict on 1000 categories. The resulting weights were saved in Keras Applications, and can be generalised to predict other problems with different categories. Figure 9 shows the original architecture of the model.

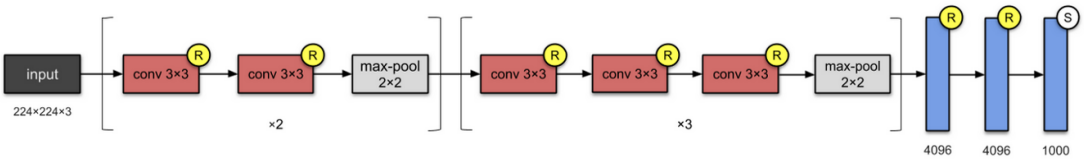

Figure 9: VGG16 Model (see source)

The last layer contains 1000 neurons. To make the model workable on our problem, we need to modify the last layer; i.e. reduce it to 4. The rest of the model will remain the same. It is noteworthy to ensure that the size of our images matches the VGG16 input. Otherwise the model won't work. Initally, I chose 224 as the image size, which happens to match the input size required for VGG16. Figure 10 shows the summary of the mode after modification.

Layer (type)	Output Shape	Param #
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102764544
fc2 (Dense)	(None, 4096)	16781312
dense_1 (Dense)	(None, 4)	16388

Total params: 134,276,932
Trainable params: 16,388
Non-trainable params: 134,260,544

Figure 10: VGG16 Summary

The number of trainable parameters, as can be seen in the figure above, is equal to the number of parameters in the last layer. Meaning, we are only training the last layer while the rest of the model remains untouched.

Results:

Like LeNet5, the VGG16 was trained with the Adam optimiser and ran for 20 epochs while the batch size is different. For this model 90 proved to be the best batch size[5]. Testing the model on the test set, we get the following results:

```
Test loss:  1.219910740852356
Test accuracy:  0.4540441036224365
```

Figure 11: Test Loss and Accuracy

Overall the accuracy has decreased. I tried two different approaches to splitting the data: 70/15/15 and 60/20/20. The latter yielded a slightly better accuracy. The VGG16 network also appears to be less stable when learning the data (see Figure 12).

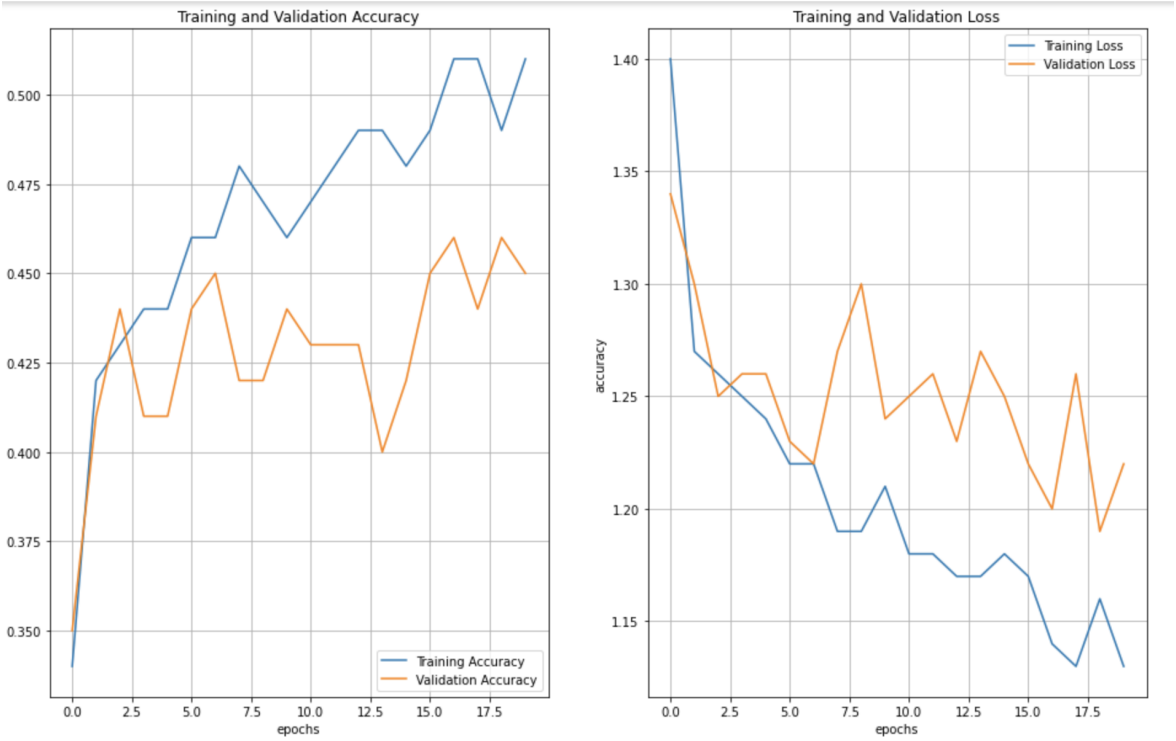

Figure 12: Test Loss and Accuracy Plots

An accuracy of 45% is usually low, however, my result is not any different from others who have used the same model on the same data set[6]. My code is different in the way it fetches the data –i.e. it uses few-shot learning– and augments it differently. Moreover, the VGG16 implementation is different (see the accompanying VGG16 Colab notebook).

[5]This was obtained through a process of trail and error

[6]see this notebook

Let us now look at the resulting confusion matrix. Figure 13 is the corresponding matrix to VGG16.

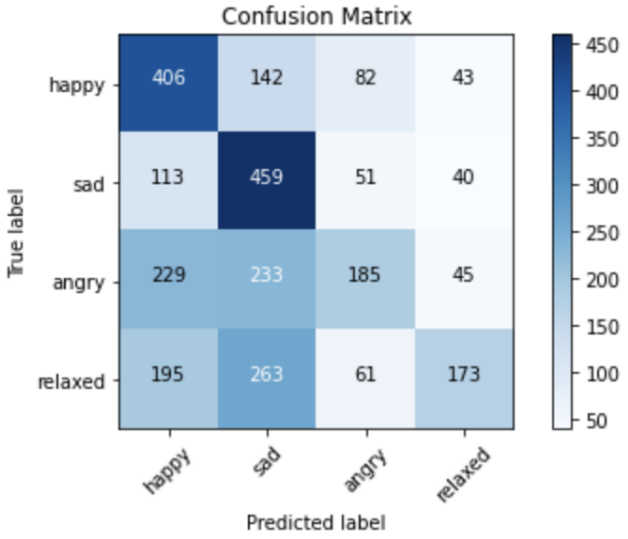

Figure 13: Confusion Matrix (VGG16)

The mode does well on happy and sad, though not so well on angry and relaxed. If we compare these results to Figure 5, we conclude that the LeNet model is more balanced and less biased, which also shows in the the plots (Figures 9 & and 4) and the overall accuracy is also lower.

2.4.3 ResNet50

ResNet50 is available on Keras as well. The algorithm is based on a paper from 2016 (He et al., 2016). Visually, ResNet50 looks as displayed in Figure 14 below.

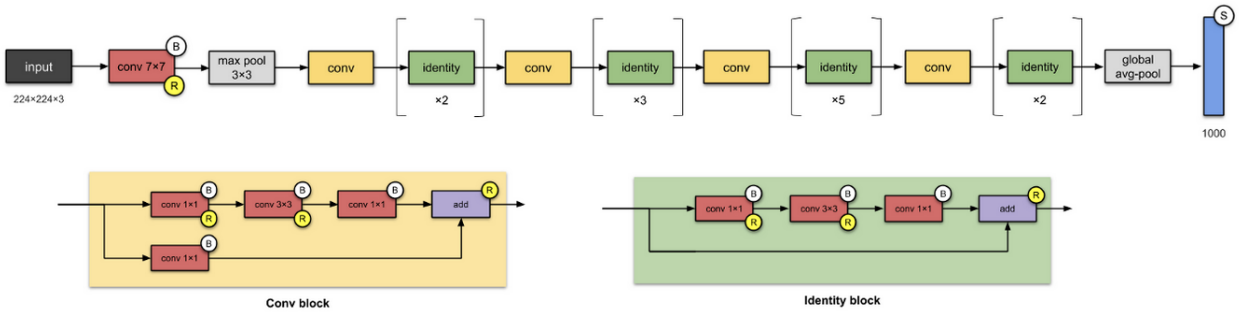

Figure 14: ResNet-50 (see source)

The model we are training utilises the pre-trained weights from Keras, flattens them and then converts the output layer to predict four labels only. Figure 15 shows the modified model.

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 resnet50 (Functional)        (None, 2048)              23587712

 flatten (Flatten)            (None, 2048)              0

 dense (Dense)                (None, 4)                 8196

=================================================================
Total params: 23,595,908
Trainable params: 8,196
Non-trainable params: 23,587,712
_________________________________________________________________
```

Figure 15: ResNet50 Summary

Again, we are only training the output layer. Let us now look at the training process. When training the networking we get the following accuracy and loss plots:

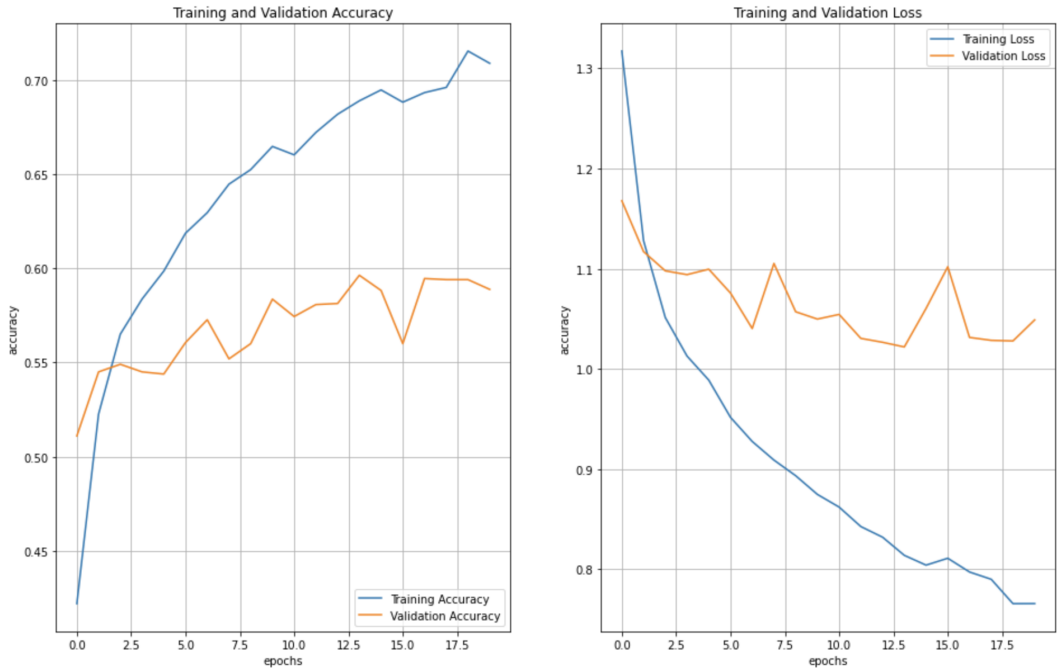

Figure 16: Loss and Accuracy Plots (ResNet50)

Figure 17 shows the confusion matrix associated with the ResNet model. We observe that the classification is somewhat evenly distributed among the four classes.

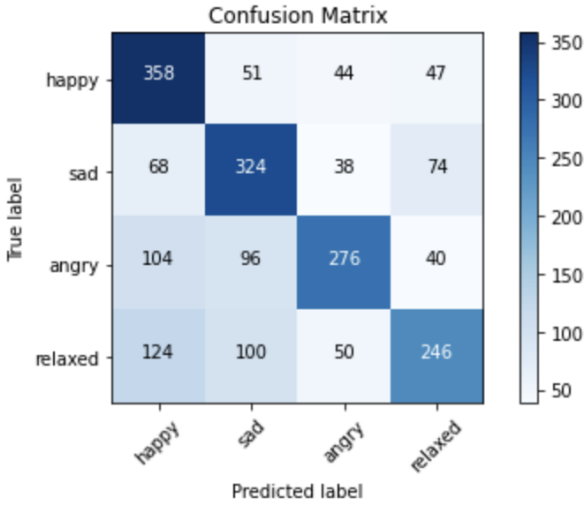

Figure 17: Confusion Matrix ResNet50

In the last section, we briefly looked at how VGG16 differs from LetNet5, but did not expound the differences. In the next section, we compare all the models we had so far in detail, including ResNet-50 and how it differs form the other models.

3 Comparison Study

In the previous section we presented three algorithms and measured their classification performance. In this section we will take a look at the results from each algorithm and compare them against each other. The first algorithm we tried was a simple LeNet-like model with a few modifications to enhance its performance. This model achieved c.a. 62%. The learning process– as can be seen in Figure 4 – was fairly stable. It is important to keep in mind that the problem we are trying to solve is complex. Our data set does not only contain dogs, some pictures have humans besides dogs and there are a few cat pictures in the mix as well. Thus learning becomes complicated, which inevitably affects the accuracy of the model. As we noted in previous sections, a validation accuracy of around 60% is quite high for this particular data set (see this notebook for a comparison). Moreover, we also observed that complex algorithms like VGG16 and ResNet50 do not perform very well on this data set. There are several reasons that could explain this phenomenon. According to an article from the Journal Chemical Information and Computer Sciences, "A model overfits if it is more complex than another model that fits equally well." (Hawkins, 2004). This provides us with a *prima facie* explanation to why the simple model out-performs the sophisticated one. But why did the VGG16 perform poorly in comparison with ResNet50? There are several possible answers to this question. The first is that the preprocessing used in this report is not suited for the VGG16 model. The second answer is that ResNet50 simply performs better than VGG16 on classification. This is corroborated by the Keras website, where the top-5 accuracy of the VGG16 is lower than that of ResNet50[7]. Nevertheless, we were able to achieve an accuracy of 62% on the test data, which is already an improvement over the notebooks on Kaggle. This is likely due to the use meta-learning, which was not utilised in any of the available open-source code on Kaggle.

3.1 Reflection and Challenges

One of prominent issues was the complexity of the data set. Dog emotions are subtle and in some cases, two emotions could be labelled differently despite looking very similar. Even with regularisation some models did not go above 45% accuracy (e.g., VGG16). Apart from that, hardware limitations imposed further limitations on how much of the data set can be used. For the most part, the code was tested on a 2019 MacBook Pro with an Intel Iris GPU, which is not compatible with TensorFlow. To be able to run the code efficiently I used Goolge Colab, and Colab comes with its own limitations, such as restrictions on computing units, *etc.* As is usually the case, overfitting was an issue, too. However, increasing regularisation and adding more data have reduced the overfitting.

4 Discussion

The various algorithms we presented here have shown that the problem we are attempting to solve is relatively complex[8]. A possible improvement to the data set could include more emotions, since angry and sad are often similar. Relaxed and sad could be easily conflated as well. In other words, adding emotions that are not among the four emotions we have discussed here could potentially yield better training results. As we mentioned in the second section, the data was (probably) collected by asking humans to classify images. This issue could be resolved by studying the facial expressions of dogs and trying to come up with a better way to determine the emotions that involves less human bias. The results obtained here could further be refined and incorporated into an app that classifies images in real time. Furthermore, the data set itself does not only contain dog images, which hinders the training process. For example, while working with the data, I have encountered a few images where the human dominated the image and the dog was barely visible and in the corner, which means that the algorithm will not be able to learn the dog's features properly. Cleaning the data set and ensuring that each picture contains only one visible dog–large enough so the algorithm could learn its features– could (and should) improve the overall performance. Finally, with hyper-parameter optimisation we might be able to achieve a much higher accuracy. This report also shows the possibility– albeit only empirically–of achieving a better accuracy using meta-learning on different models.

[7] see https://keras.io/api/applications/

[8] which is further corroborated by the results obtained in the open-source examples.

References

N. Dolensek, D. A. Gehrlach, A. S. Klein, and N. Gogolla. Facial expressions of emotion states and their neuronal correlates in mice, 2020. URL https://pubmed.ncbi.nlm.nih.gov/32241948/.

S. Gonzalez. Neural networks for image classification: Tensorflow and keras, 2021. URL https://www.modeldifferently.com/en/2021/10/image_classification/.

D. M. Hawkins. The problem of overfitting. *Journal of Chemical Information and Computer Sciences*, 44(1):1–12, 2004. doi: 10.1021/ci0342472. URL https://doi.org/10.1021/ci0342472. PMID: 14741005.

K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, 2015. URL https://arxiv.org/pdf/1502.01852v1.pdf.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2016. URL https://arxiv.org/abs/1512.03385.

Y. LeCun, L. Bottou, Y. Benigo, and P. Haffner. Gradient-based learning applied to document recognition, 1998. URL http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf.

D. Mota-Rojas, M. Marcet-Rius, A. Og, I. Hernández-Ávalos, C. Mariti, J. Martínez-Burnes, P. Mora-Medina, A. Casas, A. Domínguez, B. Reyes, and A. Gazzano. Current advances in assessment of dog's emotions, facial expressions, and their use for clinical recognition of pain, 2021. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8614696/.

A. Rosebrock. Lenet – convolutional neural network in python, 2016. URL https://pyimagesearch.com/2016/08/01/lenet-convolutional-neural-network-in-python/.

K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition, 2014. URL https://arxiv.org/abs/1409.1556.