Binary classification of cats and dogs images using Tensorflow 2.0

Aleksandra Łepek

1 Introduction

1.1 Problem statement

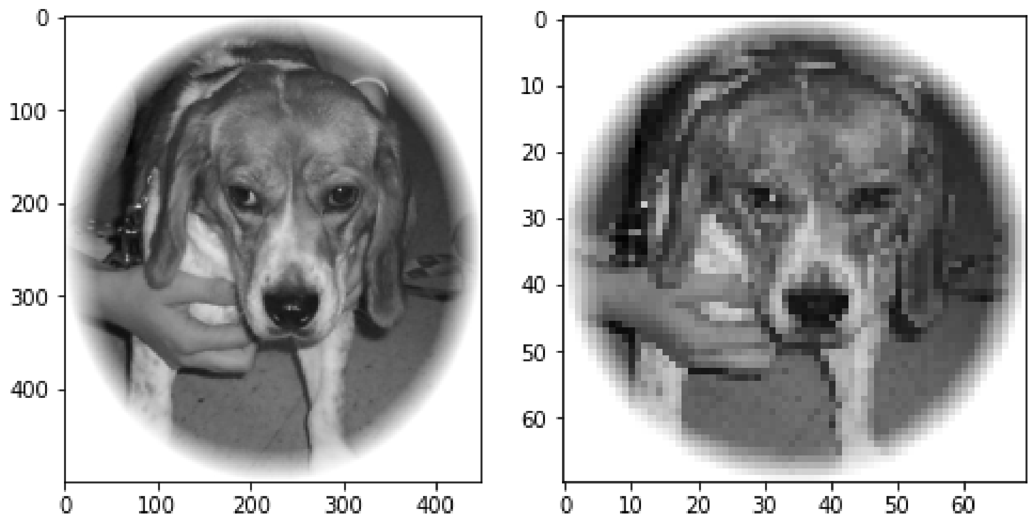
The purpose of this project is to accurately classify images of cats and dogs into their respective classes. This task can pose challenges due to following factors: subtle differences between those types of animals, variety of the backgrounds, poses and environments, irrelevant objects appearing in the images.
The goal is to develop a model using Tensorflow 2.0 that can accurately recognise cats and dogs in a variety of contexts. Additionally this model should be robust to recognise unseen before data and be optimised in terms of performance.

1.2 Data description

The dataset used for this project consists of 25,000 images of cats and dogs, divided into two categories. The images were collected from a variety of sources, including the ImageNet dataset and Kaggle competitions.

1.3 Data preprocessing

Images were converted into array, normalised to size 70x70 pixels and turned into greyscale. Many machine learning models, including neural networks prefer the values they work with to be between 0 and 1. Using this knowledge the image arrays were scaled by division of 255.



2 Methodology

2.1 Convoluted Neural Networks

Convolutional Neural Networks (CNNs) are a powerful tool for image recognition tasks. CNNs are a type of deep learning algorithm that take an input image, pass it through a series of convolutional layers, and output a feature map. Typical CNN for image processing model architecture consist of: input layer, convolutional layer with Rectified Linear activation function (ReLu), pooling layer, connected layer and output layer.
Unlike a regular Neural Network, the layers of a ConvNet have neurons arranged in 3 dimensions: width, height, depth. Their capacity can be controlled by varying their depth and breadth, and they also make strong and mostly correct assumptions about the nature of images (namely, stationarity of statistics and locality of pixel dependencies). Therefore, compared to standard feedforward neural networks with similarly sized layers, CNNs have much fewer connections and parameters and so they are easier to train, while their theoretically best performance is likely to be only slightly worse.[1]

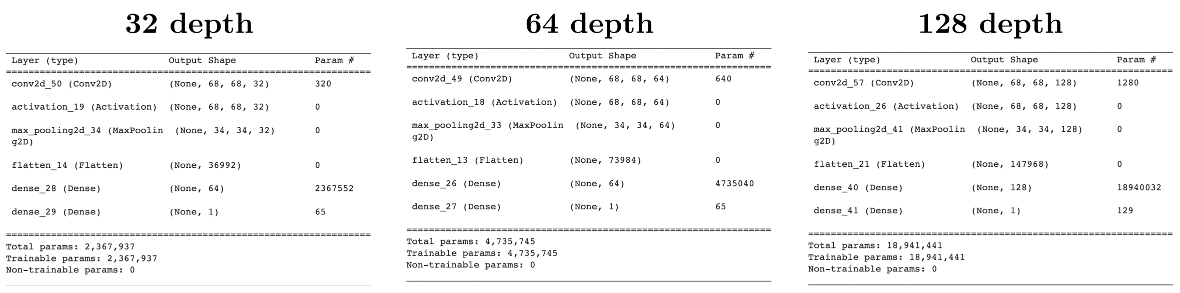
2.2 Models architecture

The project compared different models architecture starting with a single convoluted layer and adding up to 3 convoluted layers. The models also vary in terms of neurons in each of the convoluted layers. Hyperparameters like batch size and number of epochs were also tested. Final comparison of the models will be using batch size of 32 and number of epochs = 6, however if the model performance looks promising the epochs would be increased to 10. Adam was chosen as the optimiser for all models.

3 Results

3.1 Models with 1 CNN layers

Models with the single convoluted layer did not show iprovement in validation accuracy as well as validation loss after training for more than 2 epochs. The accuracy on the level of 0.75 and loss around 0.55 were not satisfactory.



32 depth

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_50 (Conv2D)           (None, 68, 68, 32)        320

activation_19 (Activation)   (None, 68, 68, 32)        0

max_pooling2d_34 (MaxPoolin   (None, 34, 34, 32)       0
g2D)

flatten_14 (Flatten)         (None, 36992)             0

dense_28 (Dense)             (None, 64)                2367552

dense_29 (Dense)             (None, 1)                 65

=================================================================
Total params: 2,367,937
Trainable params: 2,367,937
Non-trainable params: 0
```

64 depth

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_49 (Conv2D)           (None, 68, 68, 64)        640

activation_18 (Activation)   (None, 68, 68, 64)        0

max_pooling2d_33 (MaxPoolin   (None, 34, 34, 64)       0
g2D)

flatten_13 (Flatten)         (None, 73984)             0

dense_26 (Dense)             (None, 64)                4735040

dense_27 (Dense)             (None, 1)                 65

=================================================================
Total params: 4,735,745
Trainable params: 4,735,745
Non-trainable params: 0
```

128 depth

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_57 (Conv2D)           (None, 68, 68, 128)       1280

activation_26 (Activation)   (None, 68, 68, 128)       0

max_pooling2d_41 (MaxPoolin   (None, 34, 34, 128)      0
g2D)

flatten_21 (Flatten)         (None, 147968)            0

dense_40 (Dense)             (None, 128)               18940032

dense_41 (Dense)             (None, 1)                 129

=================================================================
Total params: 18,941,441
Trainable params: 18,941,441
Non-trainable params: 0
```

Figure 1: Single convoluted layer modes summary

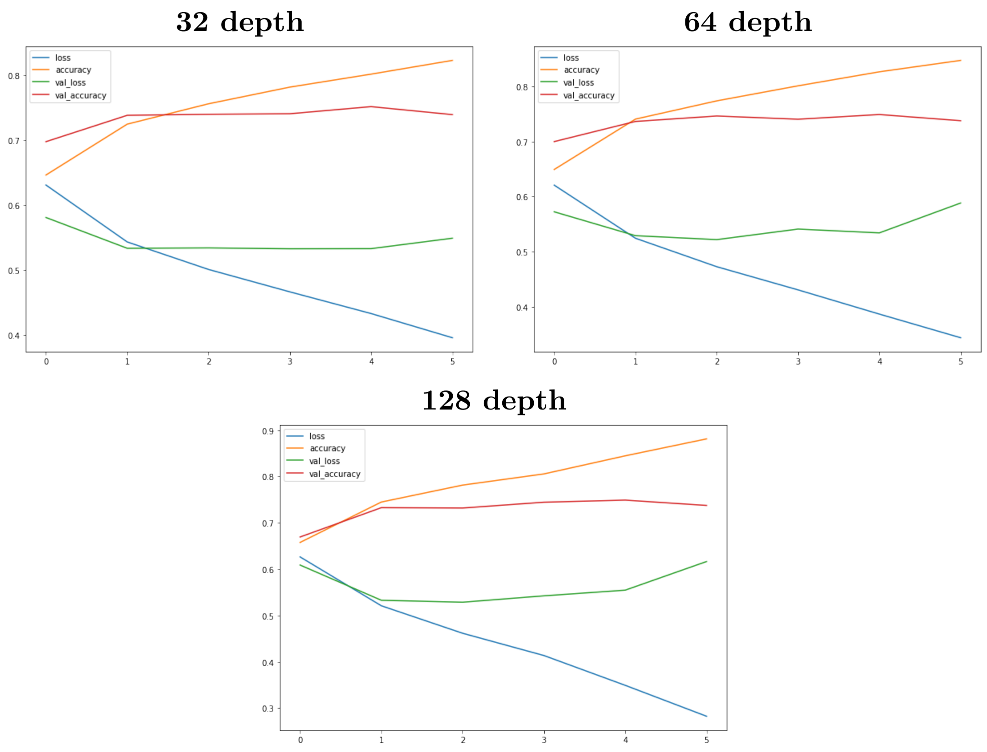
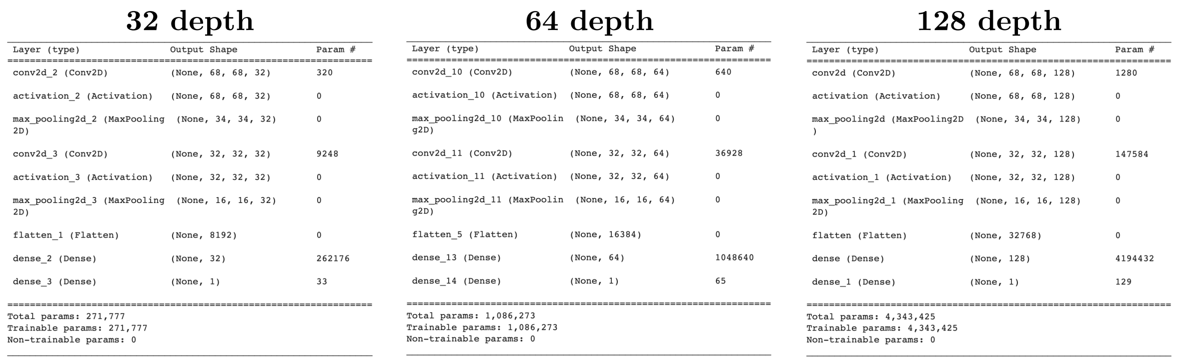


Figure 2: Single convoluted layer modes accuracy and loss across epochs

3.2 Models with 2 CNN layers

Models with the double convoluted layer show improvement to those with only single layer. The accuracy on the level of around 0.77 and loss below 0.5 were still not satisfactory. The best performimg model of those 3 was the one with 32 depth. After 3rd epoch we are starting to see that the model is overfitting.



32 depth

```
Layer (type)                Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)           (None, 68, 68, 32)        320

activation_2 (Activation)   (None, 68, 68, 32)        0

max_pooling2d_2 (MaxPooling  (None, 34, 34, 32)       0
2D)

conv2d_3 (Conv2D)           (None, 32, 32, 32)        9248

activation_3 (Activation)   (None, 32, 32, 32)        0

max_pooling2d_3 (MaxPooling  (None, 16, 16, 32)       0
2D)

flatten_1 (Flatten)         (None, 8192)              0

dense_2 (Dense)             (None, 32)                262176

dense_3 (Dense)             (None, 1)                 33

=================================================================
Total params: 271,777
Trainable params: 271,777
Non-trainable params: 0
_________________________________________________________________
```

64 depth

```
Layer (type)                Output Shape              Param #
=================================================================
conv2d_10 (Conv2D)          (None, 68, 68, 64)        640

activation_10 (Activation)  (None, 68, 68, 64)        0

max_pooling2d_10 (MaxPoolin  (None, 34, 34, 64)       0
g2D)

conv2d_11 (Conv2D)          (None, 32, 32, 64)        36928

activation_11 (Activation)  (None, 32, 32, 64)        0

max_pooling2d_11 (MaxPoolin  (None, 16, 16, 64)       0
g2D)

flatten_5 (Flatten)         (None, 16384)             0

dense_13 (Dense)            (None, 64)                1048640

dense_14 (Dense)            (None, 1)                 65

=================================================================
Total params: 1,086,273
Trainable params: 1,086,273
Non-trainable params: 0
_________________________________________________________________
```

128 depth

```
Layer (type)                Output Shape              Param #
=================================================================
conv2d (Conv2D)             (None, 68, 68, 128)       1280

activation (Activation)     (None, 68, 68, 128)       0

max_pooling2d (MaxPooling2D  (None, 34, 34, 128)      0
)

conv2d_1 (Conv2D)           (None, 32, 32, 128)       147584

activation_1 (Activation)   (None, 32, 32, 128)       0

max_pooling2d_1 (MaxPooling  (None, 16, 16, 128)      0
2D)

flatten (Flatten)           (None, 32768)             0

dense (Dense)               (None, 128)               4194432

dense_1 (Dense)             (None, 1)                 129

=================================================================
Total params: 4,343,425
Trainable params: 4,343,425
Non-trainable params: 0
_________________________________________________________________
```

Figure 3: Double convoluted layer modes summary

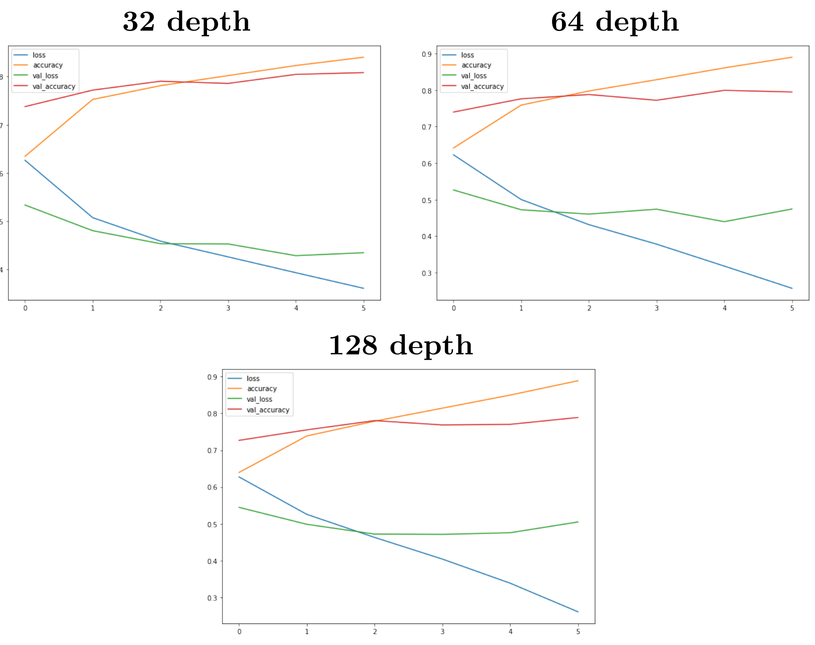
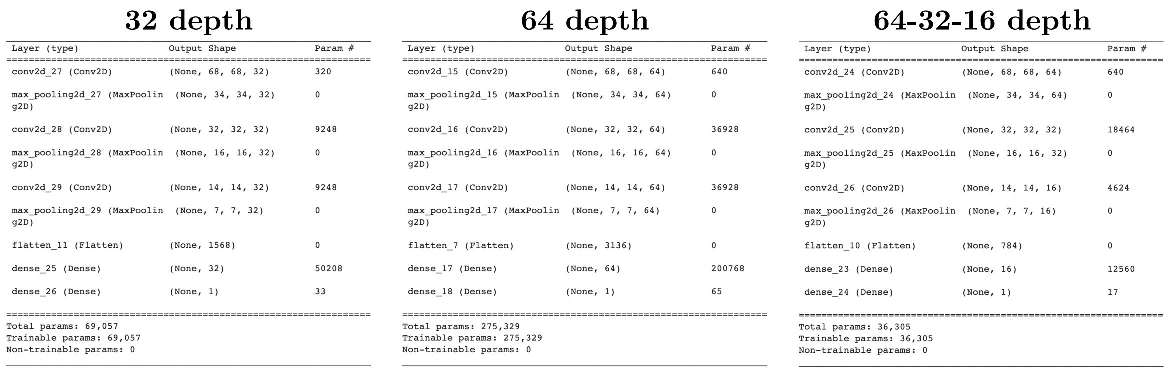


Figure 4: Double convoluted layer modes accuracy and loss across epochs

3.3 Models with 3 CNN layers

Models with the triple convoluted layer show significant improvement to those with only single layer. The number of epochs was increased to 10 as models showed continuous improvement in the first 6 epochs.With validation accuracy as well as validation loss after training for more than 2 epochs. The accuracy on the level of around 0.77 and loss below 0.5 were still not satisfactory.
The best performing model of those 3 was the one with triple 64 depth - validation loss: 0.3600 and validation accuracy: 0.8386, however for almost all the models, after 3rd epoch, we are start to see not improving performance on validation set, which shows increasing is overfitting.



32 depth

```
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_27 (Conv2D)          (None, 68, 68, 32)        320

 max_pooling2d_27 (MaxPoolin  (None, 34, 34, 32)       0
 g2D)

 conv2d_28 (Conv2D)          (None, 32, 32, 32)        9248

 max_pooling2d_28 (MaxPoolin  (None, 16, 16, 32)       0
 g2D)

 conv2d_29 (Conv2D)          (None, 14, 14, 32)        9248

 max_pooling2d_29 (MaxPoolin  (None, 7, 7, 32)         0
 g2D)

 flatten_11 (Flatten)        (None, 1568)              0

 dense_25 (Dense)            (None, 32)                50208

 dense_26 (Dense)            (None, 1)                 33

=================================================================
Total params: 69,057
Trainable params: 69,057
Non-trainable params: 0
```

64 depth

```
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_15 (Conv2D)          (None, 68, 68, 64)        640

 max_pooling2d_15 (MaxPoolin  (None, 34, 34, 64)       0
 g2D)

 conv2d_16 (Conv2D)          (None, 32, 32, 64)        36928

 max_pooling2d_16 (MaxPoolin  (None, 16, 16, 64)       0
 g2D)

 conv2d_17 (Conv2D)          (None, 14, 14, 64)        36928

 max_pooling2d_17 (MaxPoolin  (None, 7, 7, 64)         0
 g2D)

 flatten_7 (Flatten)         (None, 3136)              0

 dense_17 (Dense)            (None, 64)                200768

 dense_18 (Dense)            (None, 1)                 65

=================================================================
Total params: 275,329
Trainable params: 275,329
Non-trainable params: 0
```

64-32-16 depth

```
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_24 (Conv2D)          (None, 68, 68, 64)        640

 max_pooling2d_24 (MaxPoolin  (None, 34, 34, 64)       0
 g2D)

 conv2d_25 (Conv2D)          (None, 32, 32, 32)        18464

 max_pooling2d_25 (MaxPoolin  (None, 16, 16, 32)       0
 g2D)

 conv2d_26 (Conv2D)          (None, 14, 14, 16)        4624

 max_pooling2d_26 (MaxPoolin  (None, 7, 7, 16)         0
 g2D)

 flatten_10 (Flatten)        (None, 784)               0

 dense_23 (Dense)            (None, 16)                12560

 dense_24 (Dense)            (None, 1)                 17

=================================================================
Total params: 36,305
Trainable params: 36,305
Non-trainable params: 0
```

Figure 5: Triple convoluted layer modes summary

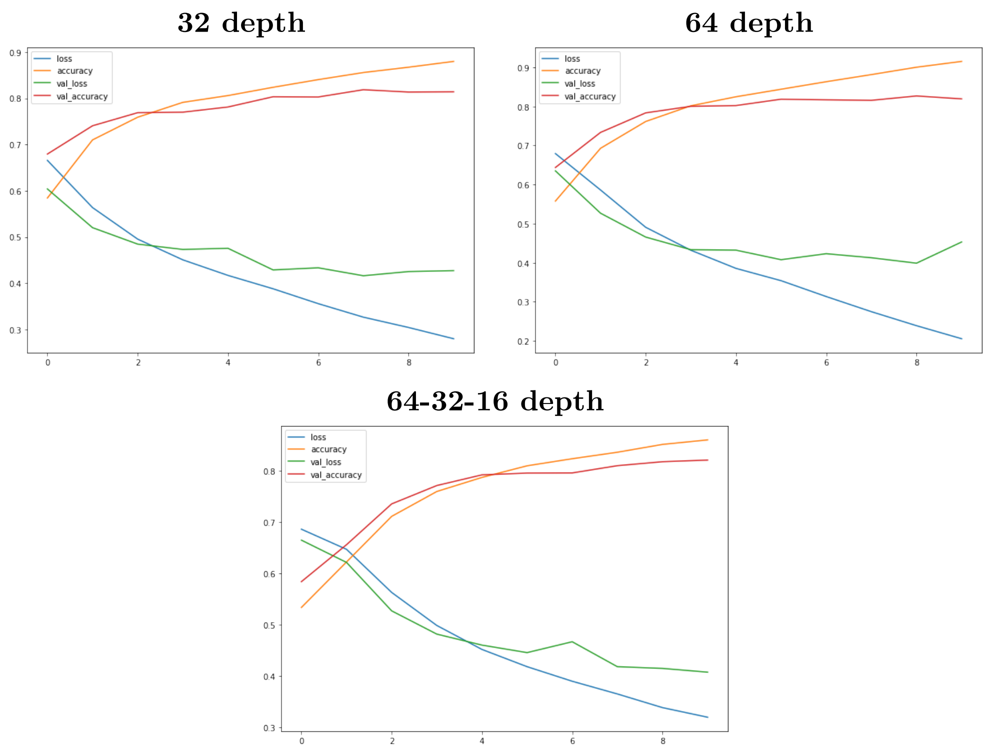


Figure 6: Triple convoluted layer modes accuracy and loss across epochs

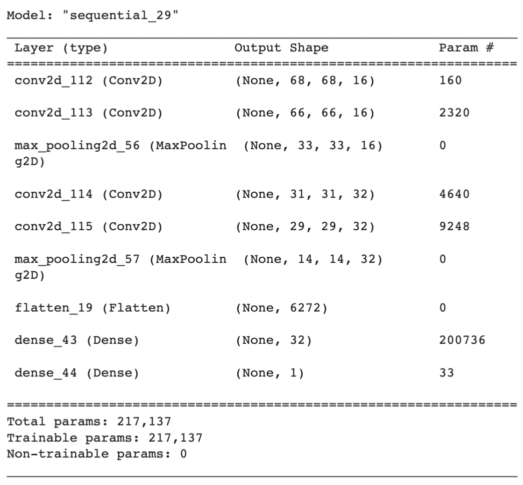
3.4 VGG models

Models with the triple convoluted layer were performing better than any previous solution however still not reaching satisfactory level. The VGG models were tried next as based on literature and internet resources those work well in image classification tasks. Those models developed by Visual Geometry Group (VGG) at the University of Oxford.

The model is composed of a series of convolutional layers, each followed by a max pooling layer and a ReLU activation function. The convolutional layers are used to extract features from the images, while the max pooling layers are used to reduce the spatial size of the feature maps. At the end of the model fully connected layers are used for classification.

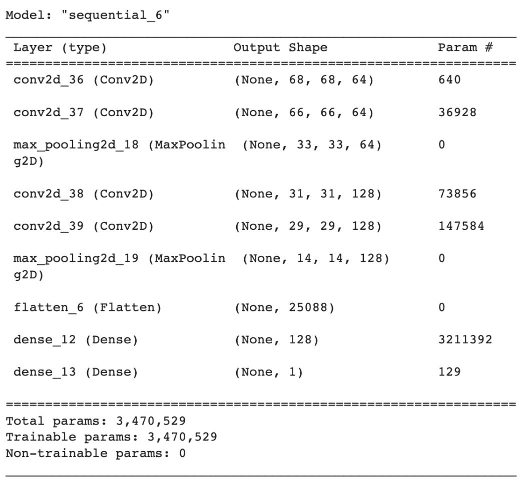
Number of epochs was increased to 12 as continuous improvement was shown when running models for the first 8 epochs. Three layer VGG model with 32, 64, 128 depth in each layer respectively was performing the best reaching validation loss: 0.3397 and validation accuracy: 0.8546.

16x2, 32x2



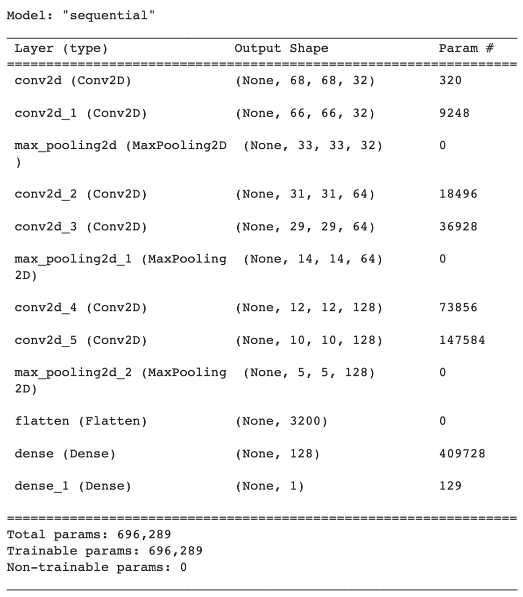
```
Model: "sequential_29"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_112 (Conv2D)         (None, 68, 68, 16)        160

 conv2d_113 (Conv2D)         (None, 66, 66, 16)        2320

 max_pooling2d_56 (MaxPoolin  (None, 33, 33, 16)       0
 g2D)

 conv2d_114 (Conv2D)         (None, 31, 31, 32)        4640

 conv2d_115 (Conv2D)         (None, 29, 29, 32)        9248

 max_pooling2d_57 (MaxPoolin  (None, 14, 14, 32)       0
 g2D)

 flatten_19 (Flatten)        (None, 6272)              0

 dense_43 (Dense)            (None, 32)                200736

 dense_44 (Dense)            (None, 1)                 33

=================================================================
Total params: 217,137
Trainable params: 217,137
Non-trainable params: 0
_________________________________________________________________
```

64x2, 128x2



```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_36 (Conv2D)          (None, 68, 68, 64)        640

 conv2d_37 (Conv2D)          (None, 66, 66, 64)        36928

 max_pooling2d_18 (MaxPoolin  (None, 33, 33, 64)       0
 g2D)

 conv2d_38 (Conv2D)          (None, 31, 31, 128)       73856

 conv2d_39 (Conv2D)          (None, 29, 29, 128)       147584

 max_pooling2d_19 (MaxPoolin  (None, 14, 14, 128)      0
 g2D)

 flatten_6 (Flatten)         (None, 25088)             0

 dense_12 (Dense)            (None, 128)               3211392

 dense_13 (Dense)            (None, 1)                 129

=================================================================
Total params: 3,470,529
Trainable params: 3,470,529
Non-trainable params: 0
_________________________________________________________________
```

32x2, 64x2, 128x2



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 68, 68, 32)        320

 conv2d_1 (Conv2D)           (None, 66, 66, 32)        9248

 max_pooling2d (MaxPooling2D  (None, 33, 33, 32)       0
 )

 conv2d_2 (Conv2D)           (None, 31, 31, 64)        18496

 conv2d_3 (Conv2D)           (None, 29, 29, 64)        36928

 max_pooling2d_1 (MaxPooling  (None, 14, 14, 64)       0
 2D)

 conv2d_4 (Conv2D)           (None, 12, 12, 128)       73856

 conv2d_5 (Conv2D)           (None, 10, 10, 128)       147584

 max_pooling2d_2 (MaxPooling  (None, 5, 5, 128)        0
 2D)

 flatten (Flatten)           (None, 3200)              0

 dense (Dense)               (None, 128)               409728

 dense_1 (Dense)             (None, 1)                 129

=================================================================
Total params: 696,289
Trainable params: 696,289
Non-trainable params: 0
_________________________________________________________________
```

Figure 7: VGG modes summary

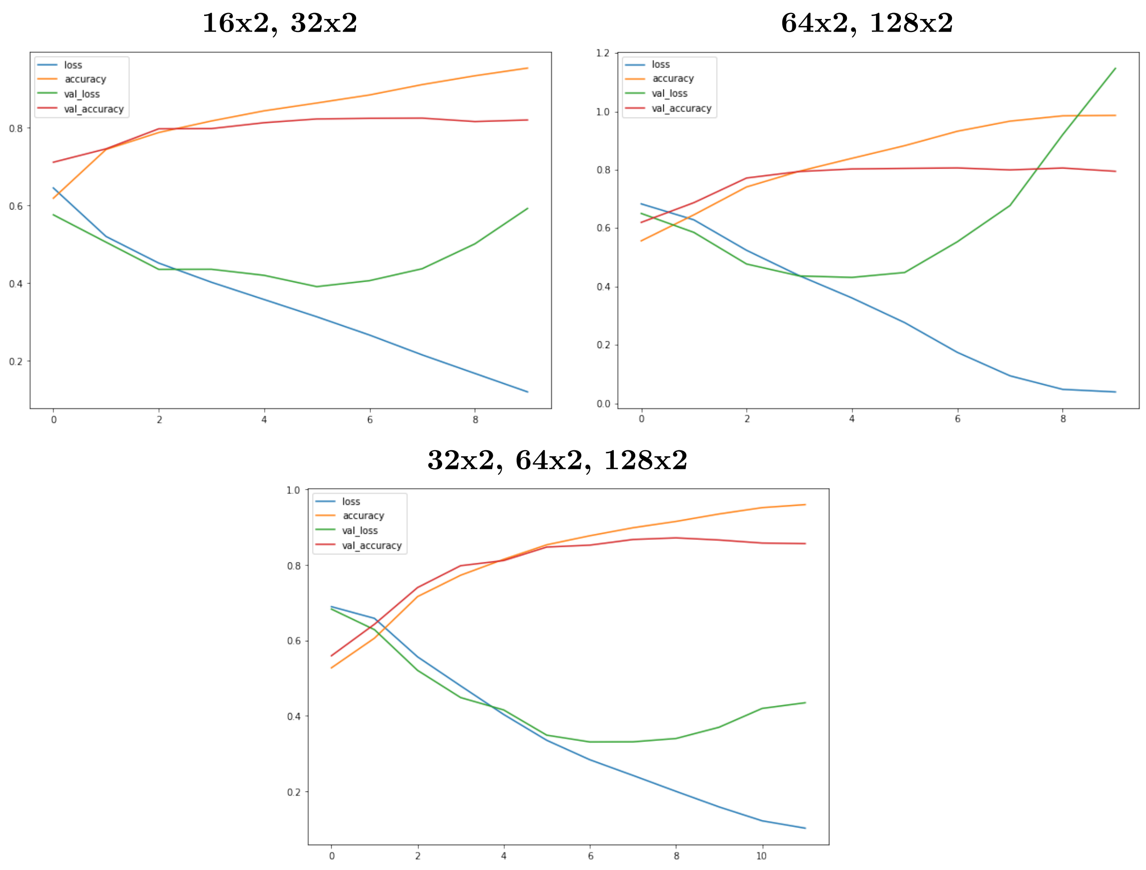


Figure 8: VGG modes accuracy and loss across epochs

3.5 Overfitting

To reduce overfitting of the best performing model from above-mentioned data argumentation technique was applied. VGG model with 32x2,64x2,128x2 was tested. Initially the images were only randomly rotated vertically, in the next step the slight random rotation was applied and in the last step a random zoom. The epochs for the flip+zoom and flip+zoom+rotation were increased to 20. However still the validation error was not satisfactory, not reaching below 0.4.

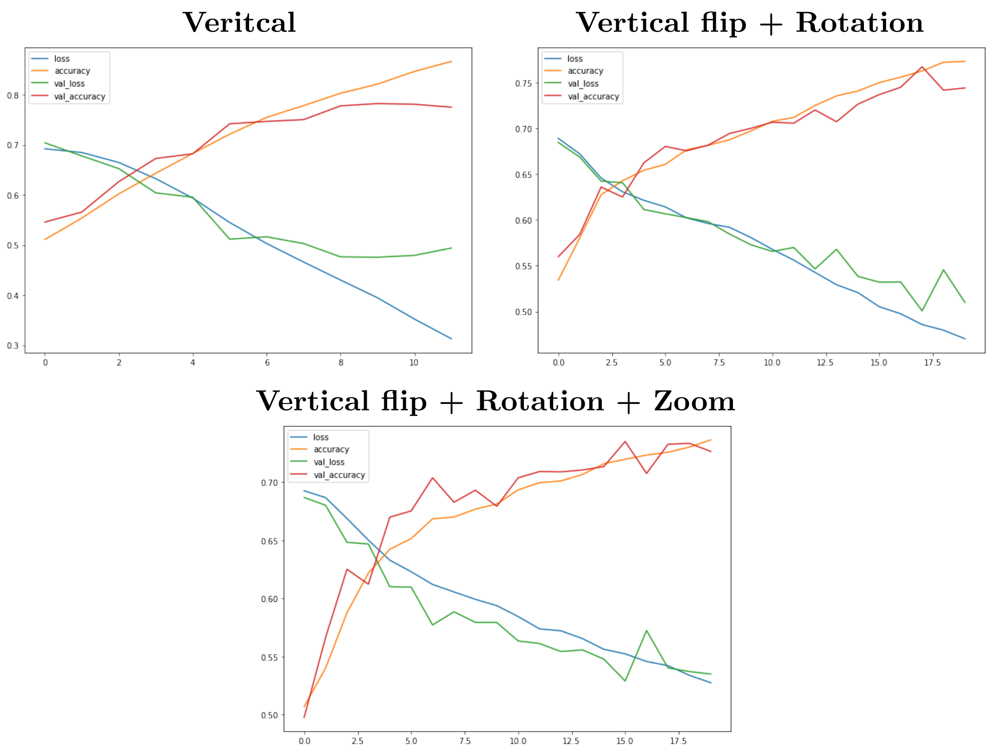


Figure 9: Comparison of different data argumentation parameters

3.6 Conclusions

CNN Models with 3 layers outperformed models with less layers, which was expected result. However not always increasing the depth of the convoluted layers would lead to the significant improvement, this was perfectly visible in the VGG models as well as 2 or 3 layers standard CNN models.

Interestingly performance of the VGG models with 6 layers was close to the model with 3 layers of 64 depth.The best performing model (VGG 32x2,64x2,128x2) reached the validation loss: 0.3397 and validation accuracy: 0.8546. Followed closely by the model consisting of 3 convoluted layers each of depth 64 with validation loss: 0.3600 and validation accuracy: 0.8386

Overfitting of all the models was observed with significant performance difference between test and validation set, which shows that model was learning too detailed information that could not be then reapplied. Worth noting is the fact that once we started to randomly flipping the images the performance significantly dropped, which shown that model was relying on the orientation of the animal (upside down or standard). To improve the performance other more complex architecture could be tested, with even more layers.

4 Disclaimer

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

References

[1] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey Hinton. "ImageNet Classification with Deep Convolutional Neural Networks." *Communications of the ACM* 60.6 (2017): 84–90. Web

5 Resources

1. Computer vision using deep learning neural network architectures with Python and Keras. Verdhan, Vaibhav. 1st ed. 2021. Place of publication not identified: Apress, 2021. Web.

2. Russakovsky, Olga et al. "ImageNet Large Scale Visual Recognition Challenge."

3. Jarrett, Kevin et al. "What Is the Best Multi-Stage Architecture for Object Recognition?" 2009 IEEE 12th International Conference on Computer Vision. IEEE, 2009. 2146–2153. Web.

4. Khan, Salman. A Guide to Convolutional Neural Networks for Computer Vision. Place of publication not identified: Morgan and Claypool Publishers, 2018. Web.

5. CNN Explainer - https://poloclub.github.io/cnn-explainer/

6. Stanford CS231n Deep Learning for Computer Vision https://cs231n.github.io/convolutional-networks/