Cats and Dogs Images Classification

Giulia Franchi-967080

giulia.franchi1@studenti.unimi.it

May 30, 2023

Abstract

The aim of this project is to build different models using the architecture of Convolutional Neural Network (CNN), that are able to recognize and classify images belonging to two different classes of pets: cats and dogs. The task requires to use TensorFlow 2 to train the neural networks, apply some transformations on images and use Cross-Validation to compute risk estimate with zero-one loss of a model previously selected by the hyperparameter tuning and model selection.

1 Introduction

CNNs are type of deep neural networks very useful for processing image data, thus they are widely used for tasks such as image classification. Neural network takes inspiration from the structure and function of biological neurons in the human brain. It is typically composed by layers, with each layer consisting of multiple nodes, the input receives data and passes it on to hidden layers, which apply mathematical operations to transform the input and transfers it to the output layer, and then produces the final output to make predictions or decisions. CNNs, designed for processing images, are composed by a series of convolutional layers followed by pooling layers, connected to one or more fully connected layers. What makes interesting CNN rather then simple Neural Network is its ability to pick up or detect patterns and make sense of it.

Next sections I am going to explain how the dataset is composed and how I make the pre-processing, thereafter I will show the neural networks that I created choosing the "best" hyperparameters. Indeed, when you build a model for neural network there are a lot of hyperparameters that need to be tuned, for this reason I use the library *Keras Tuner*, a framework perfectly combines with Tensorflow, to automate optimization of hyperparameters.

2 DATA AND METHODS

2.1 DATA

The dataset contains 25000 images, mostly in jpg format, that have been classified among two classes: cats and dogs. The labels assigned to the images are: 0 for Cats and 1 for Dogs. I've noticed that 1578 files were corrupted, so I decide to filter them out to guarantee a successful analysis, thus the remaining images are 23410.

The dataset is balanced, meaning that number of samples for each class is roughly equal: 11748 images for Cats and 11674 images for Dogs. Having a balanced dataset is important in machine learning because it ensures that the learning algorithm is not biased towards any particular class, there's some little discard of 72 images, but I think this should not have a great influence on the final result of classification.

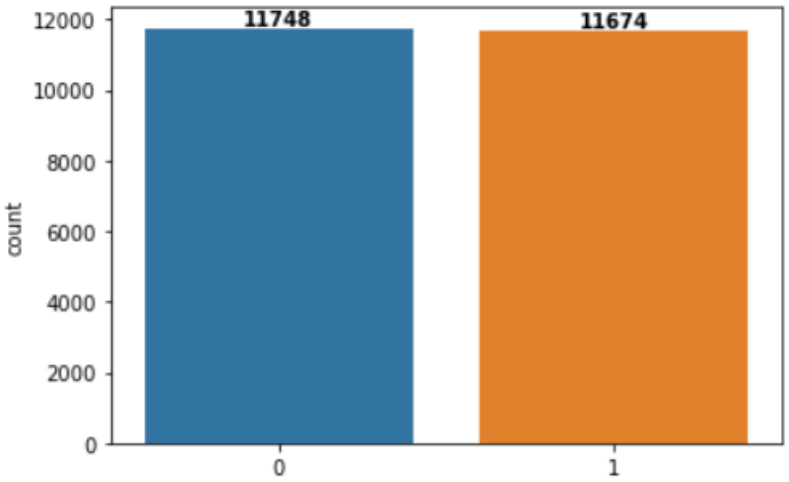

Figure 1: Number of images per class: Cats and Dogs.

2.2 Pre-processing

CNNs work with tensors which are matrices in three dimensions, so in addition to the width and the length, a tensor has got also the depth. First two dimensions are represented by the pixels of the image, while the depth represents its color. In fact, color images have a value for depth equal to 3, this means that one image can be depicted by a three-dimensional matrix composed by three matrices in two dimensions. The first matrix has pixel values for the color RED, the second one for color GREEN and the third one for color BLUE (RGB).

Considering these 3 different aspects I take some manipulation on image inputs:

- *Normalising the values of pixels.* I decide to maintain color for images because I believe that color could be a significant factor in distinguishing between cats and dogs. Normalisation of the images during the pre-processing stage is essential for CNNs to improve training convergence and prevent challenges during modeling, so I scale pixel values to the range 0-1.
- *Resizing.* The images in this dataset have different resolution, so I decide to give all them equal size: *80x80* pixels. Maybe a higher size could be better but due to limited resources capabilities of my computer I prefer to resize a little bit down.

It's possible to see an example of re-sized data below.

Figure 2: Original VS reduced data

Before building the structure of my neural network model I do two more steps for preparing data:

- All images are stored as a Numpy array.
- The entire dataset is first divided into train (80 percentage of data) and test data (20 percentage), randomly shuffled in order to avoid the model learning patterns based on the order of images in the dataset. Then, I obtain validation data (20 percentage)) carrying out on a part of training data.

3 Model for Convolutional Neural Network

CNNs have hidden layers called convolutional layers and also non-convolutional layers.

- **Convolutional layers** are able to detect patterns, they use a set of kernels, small matrix of weights with the function of filter, that are applied to the input image to extract features. These filters are actually what detects edges, circles, squares, corners, etc.,
- **Max Pooling Layer**, which is a kind of pooling layers, scans a region of the matrix and take the maximum value inside that region. Its scope is to reduce the spatial size of the feature maps, making the network more efficient.
- **Activation functions** are mathematical functions applied after each output layer. There are different types, for my model I use ReLu, the most common, that introduces non-linearity to the model and allowing it to learn more complex patterns in the data. Only after the final layer I apply *Sigmoid function* that takes values between 0 and 1, for this reason could be interpreted as probability distribution, giving the probability that an observation belongs to a class.
- **Flatten layer**: this layer flattens a multi-dimensional input, so the output of the previous MaxPooling2D layer, into a one dimensional tensor, so it can be fed into the dense layer.
- **Dense layers** are fully connected with the inputs. They can be one or more and take the flattened output of the previous layer, the aim is to consider the features learned by the previous layers and use them to make some predictions.

What I describe above is the fundamental structure of a convolutional neural network, moreover it's possible to use additional techniques or hyperparameters to improve the performance of the model and to avoid overfitting. Therefore, the selection process will be focused on finding the optimal combination of CNN layers, hyperparameters and regularization techniques.

3.1 Model 1

The *model 1* represents the baseline for the analysis as it is a very simple convolutional neural network. It's composed by:

- **Conv2Dlayers**: the first has 16 filters, while the following layers have 32 and 64 filters. The filter size is (3,3) for all layers, and the activation function used is ReLu.
- **MaxPooling2D layers** after each convolutional layer.
- **Flatten layer**
- **Dense layers:**.The first dense layer has 64 units with ReLu function and the last has only 1 unit with Sigmoid function, which outputs the probability of the image being a dog (dog=1).
- **Model compilation**: the model is compiled using *Adam* optimizer with a learning rate equal to 0.005, loss is set as *Binary cross-entropy* since it's a problem of binary classification and metrics is set as "*accuracy*".

In addition, I set the hyperparameters of *model 1*: *batch size=32* and *number of epochs=10*.

The total number of trainable parameters are 285857, which is not high compared to the rest of the models of the notebook. These parameters are optimized during the training process through the gradient descent. Gradient descent allows to minimize the difference between the predictions of the model and the actual ground truth. In a neural network, trainable parameters typically include the weights and biases associated with the connections between neurons. These parameters are adjusted iteratively during training to improve the model's ability to make accurate predictions.

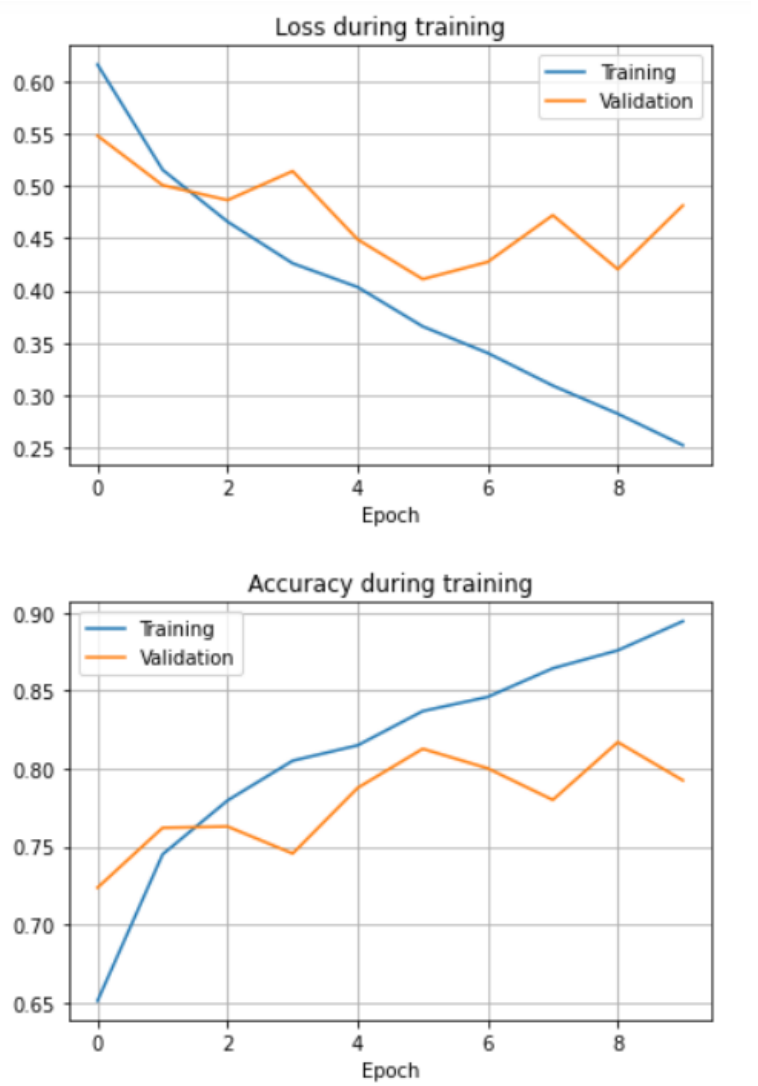

Figure 3: Model1 Training and Validation Metrics.

As we can see *model 1* shows some overfitting as it achieves a high training accuracy of 89.46% but much lower validation accuracy of 79.24% . The validation loss curve starts to increase after the 2nd epoch, while training loss continues to decrease. This indicates that the model starts capturing noise and inaccurate data from the dataset, which degrades the performance of the model, as consequence it's not able to generalize to new unseen data. Addressing overfitting is fundamental to improve model's performance on real-world data.

3.2 Model 2

The *model 2* has a similar structure to the baseline model, but I change the filters of the first convolutional layer (now are 32). In addition, it introduces:

- **Dropout layers**, which consist into a regularization technique that works by randomly dropping out, i.e. setting to zero, a fraction of the neurons in a layer during the training. This forces the network to learn more robust representations of the data by preventing the neurons from relying too heavily on any one input feature. In this case the rate is set to *0.25* after the first convolutional layer and *0.50* in second-to-last dense layer.

- **Padding** refers to the technique of adding extra rows and/or columns of zeros around the edges of an input image. Convolutional and pooling operations can reduce the spatial dimensions of the output, which can lead to a loss of important spatial information. In this case, the value for padding is set to "*same*" that results in padding with zeros evenly to the left/right or up/down of the input, so the output has the same size as the input.

The total number of trainable parameters is 360545. The compile of the model is equal to the previous model.

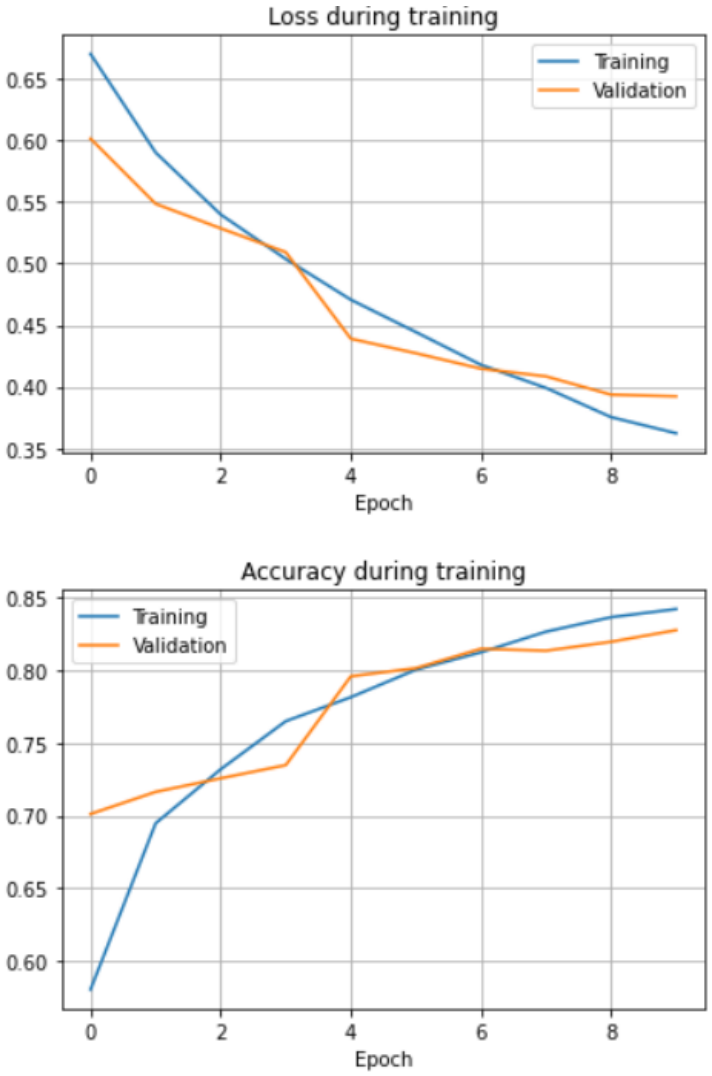

Figure 4: Model2 Training and Validation Metrics.

As we can see the train and validation loss curves are much closer to each other as opposed to the *model 1*. This also true for accuracy curves. In fact, we can observe that both validation loss and accuracy have positive trend, with the accuracy of train of 84.22% and accuracy of valid set of 82.76%. This satisfactory result would probably improve if the epochs were increased or the batch size changed.

3.3 Model 3

The architecture of this *model 3* is similar to the second one, but I change the units of my convolutional layers (32, 64, 128) and the units of second-to-last dense layer in 32. Moreover I add Batch Normalization layer after each activation layer and also two callbacks when training the model.

- **Batch normalization** is a technique used for normalizing the inputs of each layer, both convolutional and fully connected, to have zero mean and unit variance across batch samples. Thus, the distribution of the inputs to a layer changes over training process. Empirical benefits are faster convergence speed and improved accuracy, for this reason I try to use a higher learning rate, making learning easy.
- **Early Stopping** callback monitors the loss of the validation set and it stops the training after a *patience* of 5 epochs when the monitored metric stops improving.
- **Reduce on Plateau** callback reduces the learning rate by a factor of 2-10 (default) when the validation loss stops improving.

The total number of parameters of this model is 503937.

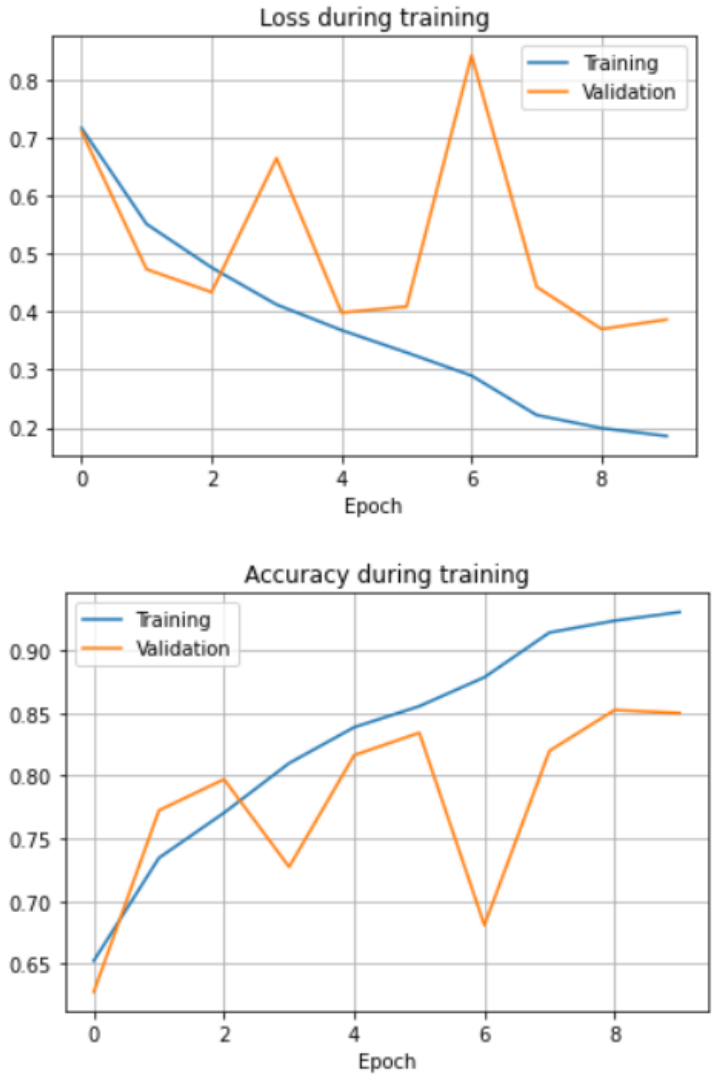

Figure 5: Model3 Training and Validation Metrics.

The *Model 3* shows improvement in learning from the training data with a training accuracy of 93.05% . However, it experiences again overfitting as evidence by the significant fluctuations in valida-

tion accuracy and loss, indicating potential benefits from further fine-tuning or additional regularization techniques from more stable performance.

3.4 Model 4

In this stage I try to use a common technique in machine learning called **data augmentation** to reduce overfitting when training a model. It consists in increasing the diversity of training set by applying random (but realistic) transformations on images: flipping, resizing, cropping, brightness and contrast.

Below a sample of an image.

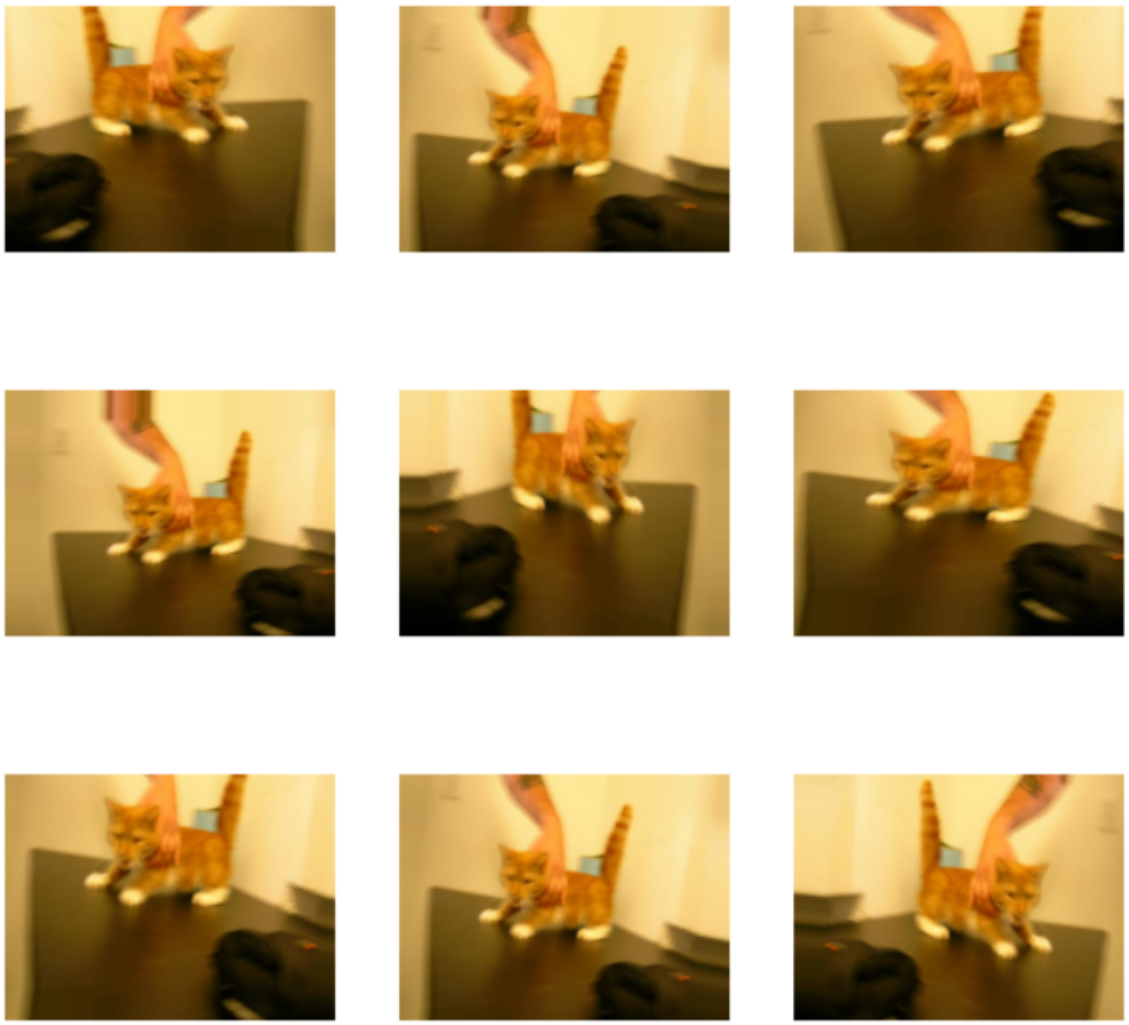

Figure 6: Example of augmentation on an image

Model 4 has a structure a little bit different from the previous ones, indeed in this model there are 4 convolutional layers, instead of 3 like above, with different units of filters for each layer (32, 64, 128, 256) and then the second-to-last dense layer has 512 units. Batch Norm doesn't assure better performance always, sometimes it greatly affect on validation result but other times it doesn't work well. Considering the worst performance of the previous model I decide to remove Batch Normalization and to introduce another **regularization factor** called **L2**, which is a penalty equal to the square of the magnitude of coefficients. It is added to the convolutional and dense layer with a value of 0.001.

I modify also the model compile:

- **AdamW,** as alternative of Adam optimizer, is a stochastic optimization method that modifies the typical implementation of weight decay in Adam, by decoupling weight decay from the gradient update.
- **decrease learning rate** to 0.0001.
- **increase epochs** to 15.

Figure 7: Model4 Training and Validation Metrics.

The *Model 4* shows a promising overall performance concerning accuracy and loss. It reaches a peak validation accuracy of 82.50% at epoch 10, indicating that extended training led to better results. In comparison to *model 3*, this last model consistently displayed an increasing validation accuracy throughout the epochs with small fluctuations. Its accuracy progressively improved over time without any evidence of overfitting.

4 Hyperparameter Tuning

4.1 Model 5

In general the performance of a machine learning model depends on its configuration. Finding an optimal structure, both for the model and for the training algorithm, is a big challenge. Model configuration can be defined as a set of hyperparameters which influences model architecture and selecting the right set of them is called hyperparameter tuning. Even though tuning might be time-and CPU-consuming, the end result sometimes unlocks the highest potential capacity for a model. In this section I use $KerasTuner$, a brilliant tool that helps with hyperparameter tuning in a smart and convenient way. In fact, it uses a sampling algorithm for selecting the best hyperparameters combination from a list of all possible combinations.

I decide to take the best model in terms of performance and the simplest one in terms of cost computing, so I choose the *model 2*.

The hyperparameters chosen are:

- *number of units for filters* of the first and the third convolutional layer. A filter, which is a set of weights, is systematically applied to the input data to create a feature map.
- *dropout rate* for the two layers, this is a value between 0 and 1 and represents the probability of individual neurons being turned off during the training process.
- *number of units of the first dense layer.*
- *learning rate* for Adam Optimizer.

At the end, I define the tuner (*RandomSearch*), specify the objective argument, in this case I want to maximize the validation accuracy considering the first 10 epochs, and then set the maximum number of trials, that is the number of times I want to repeat my study.

From this study it turns out that best parameters are:

- *number of units for the first and the third convolutional layers*: 16 and 96.
- *first and second dropout rate*: 0.1 and 0.4.
- *number of units of the first dense layer*: 128.
- *optimal learning rate*: 0.001.

So, the final structure is summarized below.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 80, 80, 16)        448

max_pooling2d_3 (MaxPooling  (None, 40, 40, 16)        0
2D)

dropout_2 (Dropout)          (None, 40, 40, 16)        0

conv2d_4 (Conv2D)            (None, 40, 40, 32)        4640

max_pooling2d_4 (MaxPooling  (None, 20, 20, 32)        0
2D)

conv2d_5 (Conv2D)            (None, 20, 20, 96)        27744

max_pooling2d_5 (MaxPooling  (None, 10, 10, 96)        0
2D)

flatten_1 (Flatten)          (None, 9600)              0

dense_2 (Dense)              (None, 128)               1228928

dropout_3 (Dropout)          (None, 128)               0

dense_3 (Dense)              (None, 1)                 129

=================================================================
Total params: 1,261,889
Trainable params: 1,261,889
Non-trainable params: 0
```

Figure 8: Caption

I train the final model for 10 epochs.

The *model 5* doesn't seem to be overfitting, but its overall performance is not good as I expected, due to the high number of fluctuations of validation curves. It achieves 84.85% of validation accuracy in epoch 7 but after that it tends to decrease. It's also worth noting that validation accuracy is sometimes higher than training accuracy because of data augmentation. In addition, of course, some hyperparameters like the number of epochs and regularization techniques could be implemented to try to improve the overall performance.

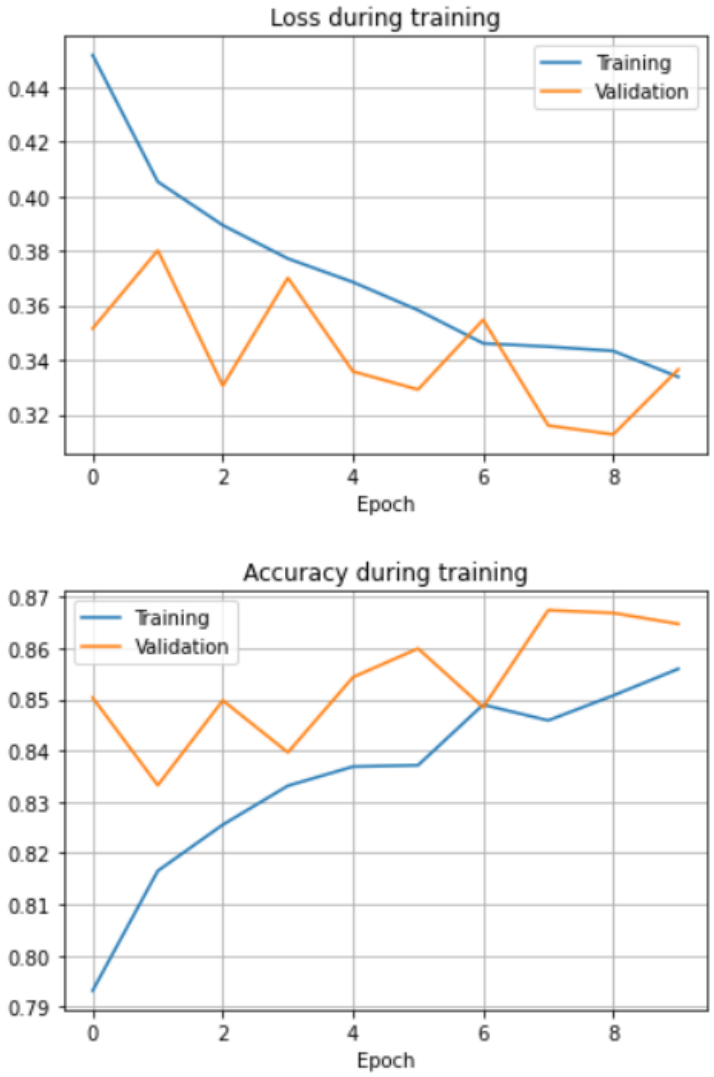

Figure 9: Model5 Training and Validation Metrics.

5 Final model evaluation

In this section I will evaluate *model 5* performances on the test set. In fact, this is the model that obtains the best validation accuracy at the first 10 epochs. It turns out to output a test accuracy of 83.76% and a loss of 0.4267.

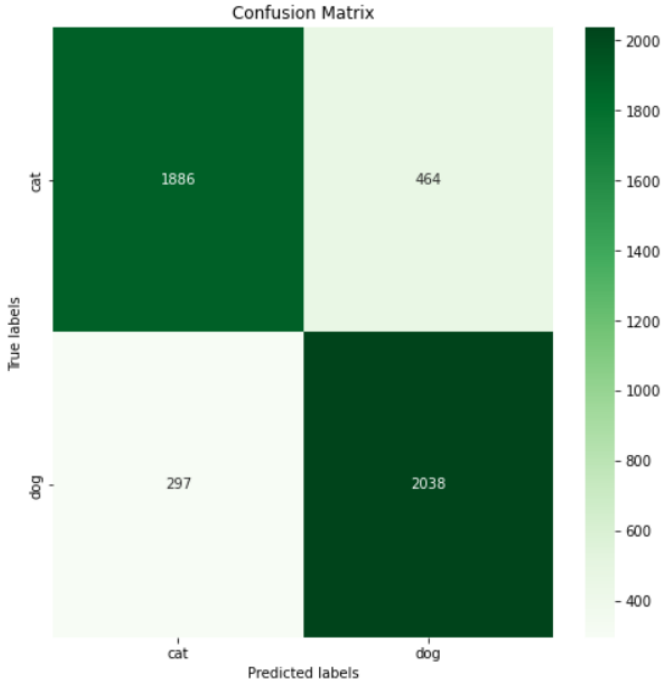

Figure 10: Model 5 confusion matrix

The confusion matrix shows that the performance is a bit unbalanced: the proportion of mistakes on dog images and on cat images differs for 167 mistaken images . While some mistakes may be due to difficult images with poor quality or unusual positioning, it's also apparent that there are some relatively straightforward images. This suggests that there is potential for improvement and that with more time and computational resources, it may be possible to train a better model.

Figure 11: New image taken from Internet

To test this model with new data, an image of a cat was taken from the internet and in this case the model predicts the true.

6 CROSS-VALIDATION

In the previous sections I use a simple approach of splitting the data into 3 parts, *Train, Validation* and *Test* sets, but using this technique we are not sure that the model can perform well on unseen data. In fact, dividing into *Train* and *Validation* sets may induce a bias since there's no guarantee of a randomness even if the whole dataset is considered a random sample. Therefore some data points with useful information may to be excluded from the training procedure, and the model fails to learn the data distribution properly. In order to mitigate this bias we can average the test error stemming from different test samples. This is precisely what **K-fold Cross Validation** does, it rotates the test sample across the whole dataset and for every test sample the remaining dataset becomes the training sample. For each split, the test error is computed after fitting the model over the corresponding training sample. The test errors from each split are averaged to obtain the average test error, or the cross-validated error. So, **K-fold CV** gives a model with less bias compared to other methods. The ***parameter k*** , in this case equal to 5, establishes how may folds the dataset is going to be divided. Every fold gets the chance to appears in the training set (k-1) times, which ensures that every observation in the dataset appears, thus enabling the model to learn the underlying data distribution better. The goal of this evaluation is to obtain an estimate of the generalization error (test error). The architecture of the model used in this section is the one corresponding to the *model 5*. The results from CV for 7 epochs are:

- **Mean Accuracy**: 0.5233
- **Mean Zero-One-Loss**: 0.4767

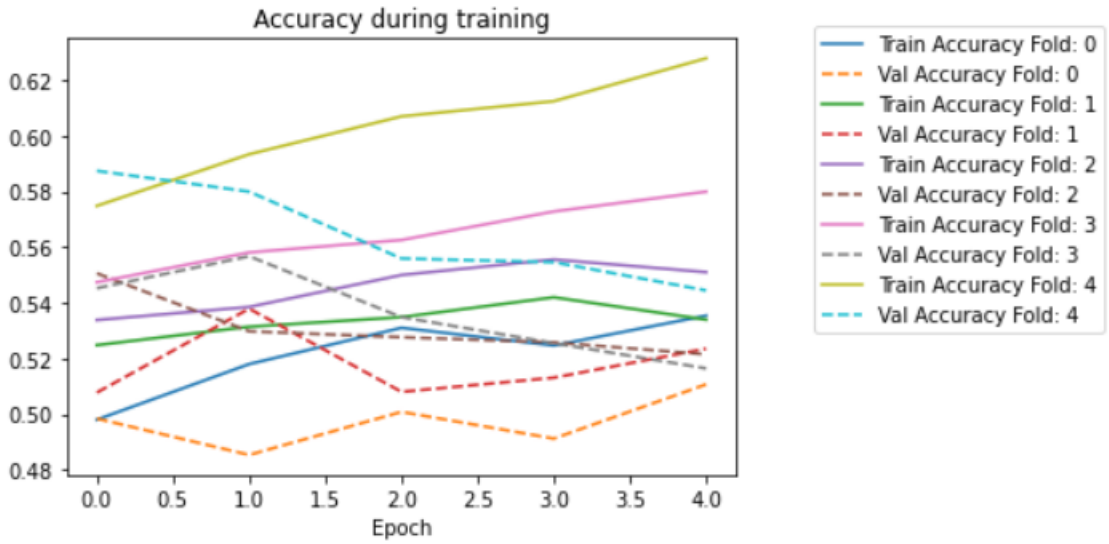

Figure 12: Accuracy during training and validation

With an accuracy of only 52.33% the model is not reliable to fulfill its task of classifying images of cats and dogs. Since the result is close to 50% , the experiment has the same reliability like tossing a coin. If the accuracy from the cross-validation method is less than the accuracy from the holdout method, it indicates that the model is overfitting, thus the model memorizes the training examples and captures noise than actually learns or identifies the true pattern/relationship from the training examples.

7 Conclusion

In this project, I have explored the development of a deep learning model for the classification of cats and dogs images.

The different models have their pros and cons. Having a large number of trainable parameters can lead to increased overfitting and poorer generalization of the model, while having too few trainable parameters may result in the model not learning important features from the available images. Complex networks also require higher computational resources. I start from a very baseline architecture and gradually introducing improvement to its architecture. I have applied several regularization techniques such as batch normalization, kernel regularizer, and dropout, and tried different callbacks during the training process. However, the results vary (referring to the obtained graphs), and satisfactory accuracy is not achieved in the validation set. Future developments could involve experimenting with CNN with different resizing techniques (e.g., 200x200) and various types of data augmentation can be explored.

References

Lectures from course Statistical Methods for Machine Learning 2021-2022

https://unimibox.unimi.it/index.php/s/eNGYGSYmqynNMqF