Cat or Dog

William Biondi

May 21, 2023

Abstract

This project work aims to build a classifier for images, in particular, to distinguish if a given image portrays a cat or a dog in it. This task is challenging for traditional machine learning algorithms such as decision trees or nearest neighbors since they struggle to identify differences and similarities between features, beyond this there is a problem of computational cost. For the reasons mentioned above in this report, I will use Convolutional Neural Networks which is a popular deep-learning method in the image recognition literature along with a data set of 25000 images

1 Introduction

The main goal of this project is to build a binary classifier for images that portrays a cat or a dog. This classification problem assumes that all the images portray a dog or a cat and not any other class of images, for instance, if our classifier receives an image that portrays a human as an input, this classifier will always carry out a probability of being a cat or a dog.

1.1 Why Convolutional Neural Networks

In this project, the choice of using Convolutional Neural Networks comes from the data which consists in images that are known for being a high-dimensionality kind of data. In this framework, traditional machine learning algorithms based on distances such as nearest neighbor and decision trees struggle to handle the enormous quantity of features and distances to compute in the nearest neighbor case, and in the decision trees case is non-trivial to find proper conditions to detect meaningful splittings of the dataset.
On the other hand, Convolutional Neural Networks are designed to manage high-dimensional data like images by learning automatically common patterns using filters.

1.2 Dataset

The dataset for this project is composed of 25000 images in jpg format, which are equally divided into cats and dogs portrayed. Before diving into the first network architecture, I tested if there are images that are corrupted for the Python library TensorFlow and tried to recover some of them using the library Pillow. This operation resulted in 61 unrecoverable images that I deleted, so the final dataset is formed by the two classes illustrated in Table 1

	Cats	Dogs
Images	12473	12466

Table 1: Image classes after cleaning corrupted images

Images were also converted from jpg to RGB therefore our images will be tensors with shapes of (128, 128, 3) and to improve performances the RGB values have been scaled from [0-255] to [0-1] range.

2 Methodology

Inspired by the structure of the human brain's visual cortex, convolutional neural networks have neurons that are arranged in layers. these layers perform operations such as filtering or feature extraction. Finally, the output of these layers is fed into a fully connected layer which performs in our case a binary classification task

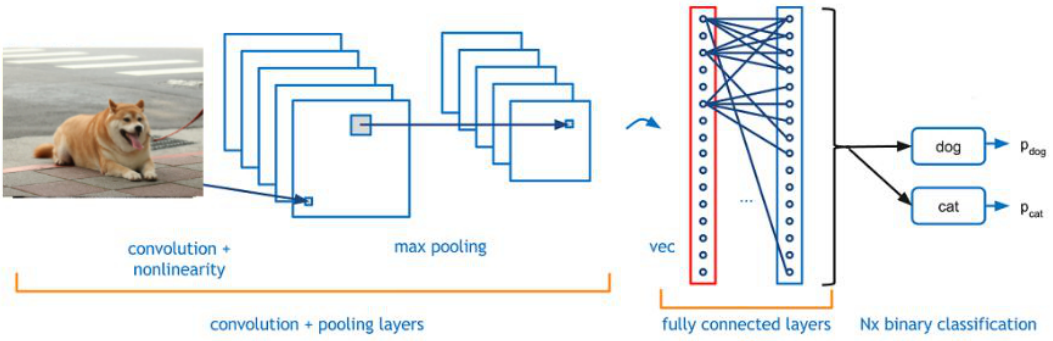

Figure 1: A graphical representation of a Convolutional Neural Network

2.1 Convolutional Layer

The convolutional layer performs a Convolution which is a mathematical operation that carries out how a function g, typically a filter modifies the principal function f. It is based on taking the integral of the two functions with g shifting by τ over the function f at each point t

$$(f * g)(t) := \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$$

This operation, illustrated in Figure 2 is iterated in the layer for an arbitrary number of filters which are also arbitrary in their dimension, typically 3*3 or 5*5.

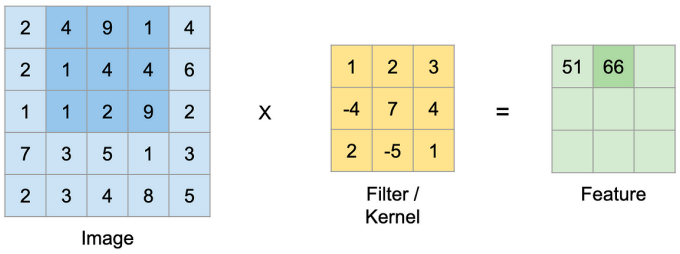

Figure 2: An example of a convolution

2.2 Max Pooling layer

Once we receive the results of the convolution we use Max Pooling layers to downsize the feature map, this is useful to reduce the coefficients to compute and it also allows deeper convolutional layers to learn broader patterns. Similarly to the convolution, Max Pooling uses a sliding window onto the input feature matrix and takes the maximum value inside the considered window. A practical example is illustrated in Figure 3 if we have a 4*4 input matrix and a 2*2 window, our operation will result in a 2*2 matrix with the maximum value of each considered quarter of the matrix.

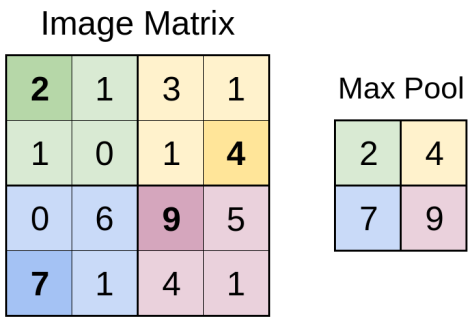

Figure 3: An example of Max Pooling operation

2.3 Activation Function

Going back to the human brain's concept we have our feature matrix which corresponds to the input that a neuron receives: an activation function is a mathematical function applied to the input and it determines whether the neuron has to be activated based on the received input. With the activation function, we introduce non-linearity which is needed because in a case having neurons that perform only linear transformation, we would have an equivalent behavior of a linear regression model.
In this project, I used two kinds of activation functions:

1. **Rectified Linear Unit (ReLU):** it returns the maximum between 0 and the input x

$$f(x) = max(0, x)$$

2. **Sigmoid:** this function maps every input x in the [0-1] interval, which is in our case the probability of being one of the two classes, that is why is widely used in binary classification problems

$$f(x) = \frac{1}{1 + e^{-}x}$$

2.4 Flatten and Fully connected layer

Following the convolutional and max pooling layers of our neural network we find a Flatten layer that transforms the input into a 1-dimensional vector. This vector is forwarded to the dense layer which are the layers that are learning complex relationships between features learned by previous layers. The dense layer is the one that performs our classification task, carrying out for each image the probability of belonging to each class

2.5 Dropout Layer

This layer has a regularisation role in neural networks: it consists in setting to zero a given rate of output features, this operation will add noise which helps reduce the variance error of the network. This feature is widely used to prevent overfitting.

2.6 Data Augmentation

Another method to prevent our model from having overfitting is to add a Data Augmentation layer to the training set: this layer performs random edits to images such as:

1. Horizontal flip
2. Rotation
3. Zoom in or out
4. Shifting left or right by a random number of pixels

The results of these operations, illustrated in Figure 4 will increase the input size and randomness which can train our model better to recognize images where subjects involved are harder to detect and classify.

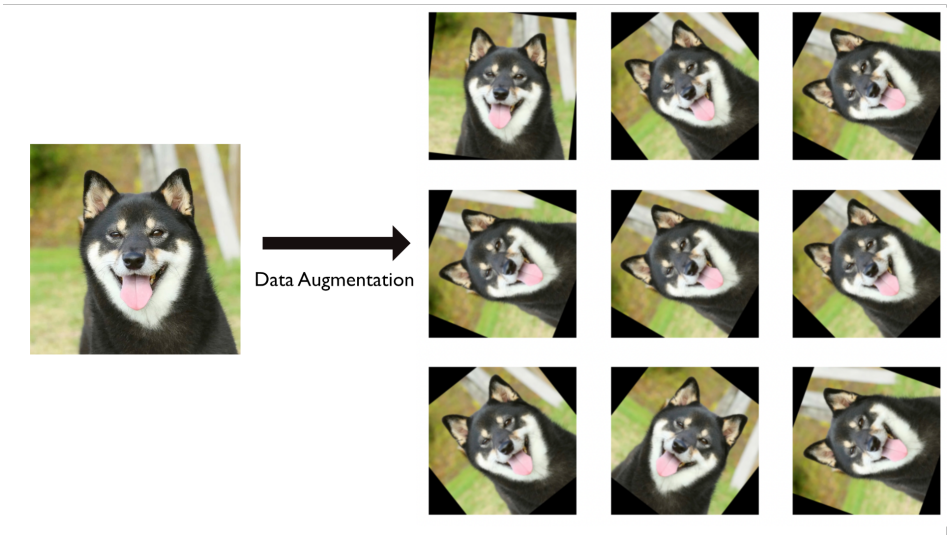

Figure 4: Results of Data Augmentation

2.7 Loss Function

The task of this project is to make a binary classification of images that portrays a dog or a cat, so the loss function used in this project will be the binary cross entropy which is defined as follows

$$l_{BCE} = -\frac{1}{N}\sum_{i=1}^{N} y_i ln(p(y_i)) + (1 - y_i) ln(1 - p(y_i))$$

This loss function has to penalize bad predictions and in our case, it does by adding to the loss, for each image labeled as dog $y_i = 1$ the log probability of being a dog image of $ln(p(y_i))$ and in the cat image case, $y_i = 0$ it adds the log the probability of being a cat image $ln(1 - p(y_i))$.

2.8 Optimizer

The loss function described above is considered an objective function that has to be minimised. To reach this goal of minimisation, it is required to establish the learning rate: it is a hyperparameter that determines the step size of how weights needed to be adjusted during training. The bigger this rate the higher the risk of the loss function diverging and on the other side, if this rate is too small our loss function will converge more slowly which is not recommended since our time and computational resources are limited. In this project, we will use an optimization technique called Adam. Adam is an algorithm that combines the benefits of two techniques: Adaptive Gradient Algorithm and Root Mean Square Propagation. Intuitively at each iteration, Adam updates the weights using a weighted sum of the first and second moments of the gradients. Then weights are updated.

3 Model description and partial results

In this section, I will describe four neural network models that are sequentially evolving from a baseline model to a model which guarantees higher accuracy

3.1 Baseline model

The baseline model is composed of an input layer with images in tensor shape (128,128,3) and two convolutional blocks, each one composed by:

1. Convolutional layer with 32 filters of size 3*3
2. Activation layer with ReLU as an activation function

3. Max Pooling layer with pool sized 2*2

Finally, the fully connected block is composed of a flattened dense layer with 64 nodes and an activation layer with ReLU then finally a single node dense layer with an activation layer that uses sigmoid as an activation function

3.1.1 Results

Compiling this model with Adam as a learning rate optimizer and binary cross entropy as a loss function, we perform training and validation of the model in 30 epochs, with an early stop of 5 epochs of patience on the validation loss (i.e. Fitting will stop automatically if the loss function does not improve in 5 epochs). Fitting stopped at the 8th epoch and resulted respectively 98% accuracy in the training set and 77% accuracy in the validation set: the gap shown in Figure 5 suggests that the baseline model is overfitting and it is enhanced by the loss values in the validation set in Figure 6 which is increasing as we go forward in the epochs instead of decreasing like the loss computed in the training set which is below 0.2.

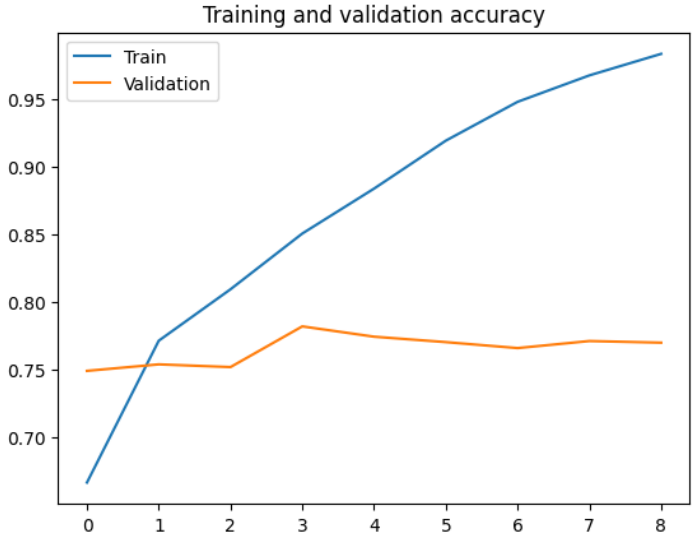

Figure 5: Base Model accuracy

3.2 Model 1

To prevent overfitting we modify our base model with an additional convolutional block and dropout layers, so now each convolutional block is composed by:

1. Convolutional layer with 32 filters of size 3*3
2. Activation layer with ReLU as an activation function
3. Max Pooling layer with pool sized 2*2
4. Dropout layer with 20% rate

This additional convolutional block diminished the number of trainable parameters: from 1.8 million to 420 thousand. The output block with the fully connected layer remains the same as the base model

3.2.1 Results

Compiling this model with Adam as a learning rate optimizer and binary cross entropy as a loss function, I performed training and validation of the model in 50 epochs with an early stop of 5 epochs of patience on the validation loss. Fitting stopped at the 15th epoch and resulted in an improvement

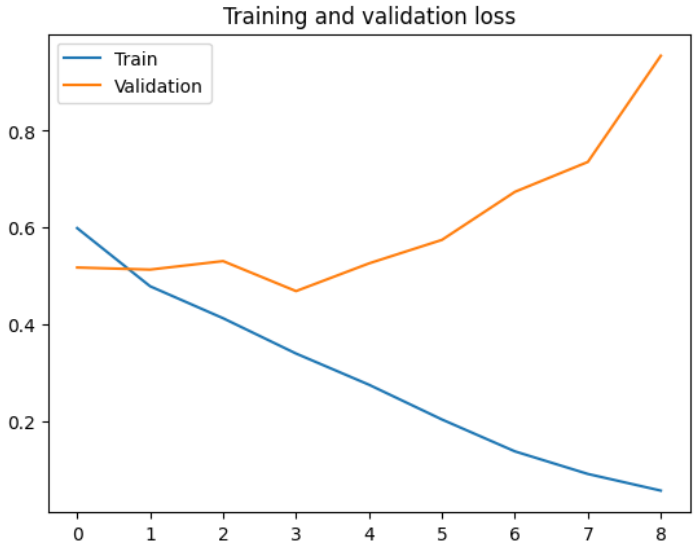

Figure 6: Base Model loss

of the overfitting problem with 90% accuracy in the training set and 85% accuracy in the validation set. However, the overfitting problem is still unsorted as shown in figure 7 and also the loss values in the validation set in figure 8 which trend is decreasing along the first epochs but halfway through tends to be stable around 0.4 unlikely the loss computed in the training set which is constantly decreasing until 0.25 as the minimum.

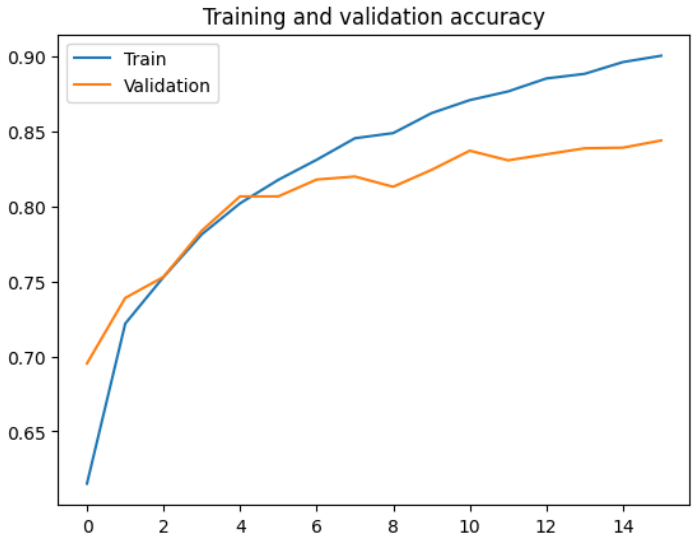

Figure 7: Model 1 accuracy

3.3 Model 2

Despite the additional layer is partially preventing the previous model from enormous overfitting, I still had overfitting so another possible solution to overcome this problem could be Data Augmentation illustrated in Section 2.6: I applied the following transformations exclusively in the training set, while I kept the validation set as before:

1. Horizontal flip

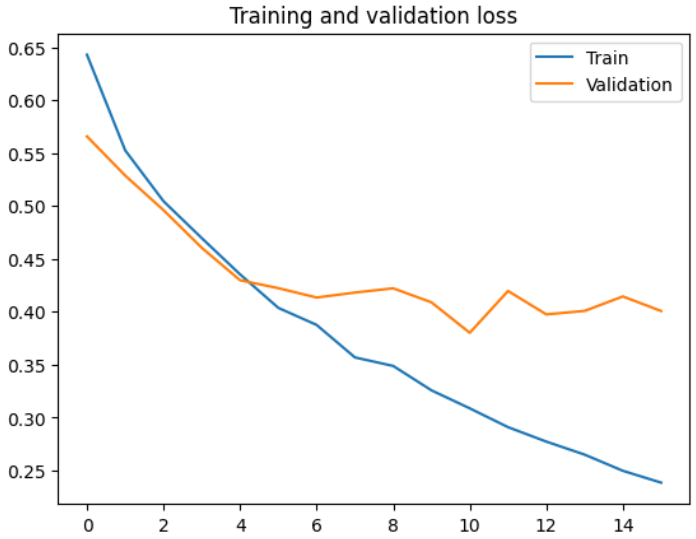

Figure 8: Model 1 loss

2. Rotation: 45° range
3. Shifting left by 25% range
4. Shifting right by 25% range
5. Shifting up by 25% range
6. Shifting down by 25% range

we train the model using the same architecture as the previous model, the only parameter changed is the number of filters of the third convolutional layer, from 32 to 64

3.3.1 Results

Compiling this model with Adam as a learning rate optimizer and binary cross entropy as a loss function, I performed training and validation of the model in 50 epochs with an early stop of 4 epochs of patience on the validation loss. and this resulted in 73% accuracy in the training set and 76% accuracy in the validation set: the overfitting problem is now sorted as the validation accuracy performs better than the noised training accuracy: this is also shown in the accuracy plot in figure 9 and the loss values in the validation set in figure 10 which trend is decreasing over the epochs until 0.3 as the minimum.

3.4 Model 3

The previous model provides a solution that prevents overfitting but more accuracy is still needed: in this model, the architecture will have four convolutional blocks, each one composed by:

1. Convolutional layer with 32 filters of size 3*3 (third and fourth layer with 64 filters)
2. Activation layer with ReLU as an activation function
3. Max Pooling layer with pool sized 2*2
4. Dropout layer with 20% rate

Finally, the fully connected block is composed of a flattened dense layer with 64 nodes and an activation layer with ReLU followed by a Dropout layer with 50% rate, then finally a single node dense layer with an activation layer which uses sigmoid as an activation function

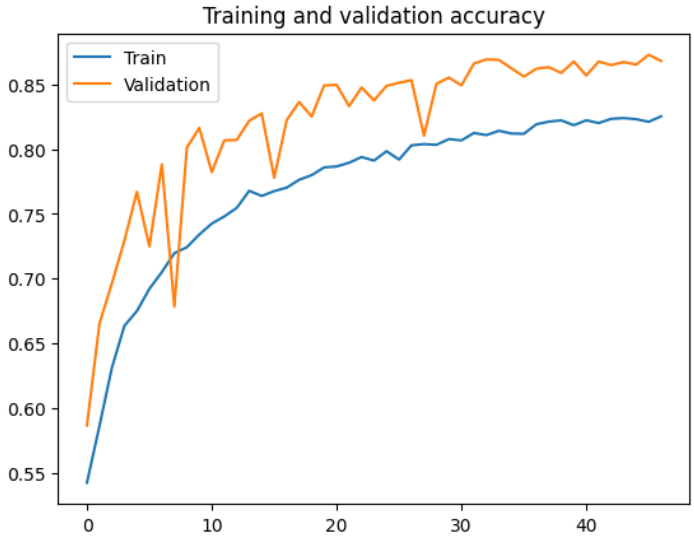

Figure 9: Model 2 accuracy

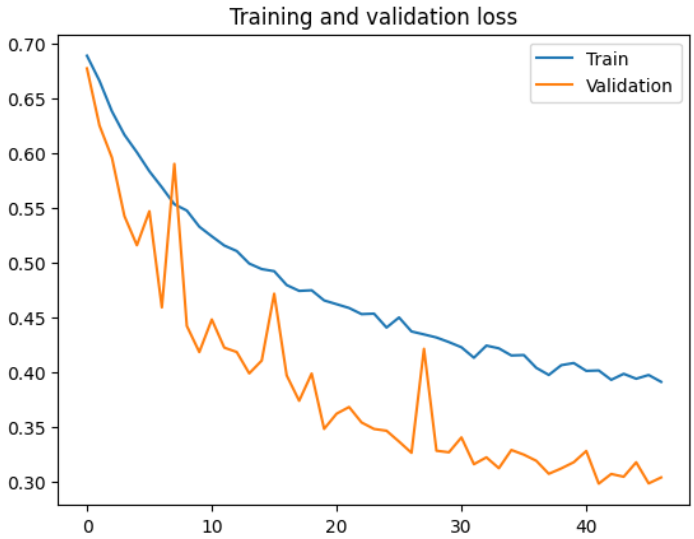

Figure 10: Model 2 loss

3.4.1 Results

Compiling the third model with Adam as a learning rate optimizer and binary cross entropy as a loss function, we perform training and validation of the model in 50 epochs and this resulted in 80% accuracy in the training set and 84% accuracy in the validation set: the overfitting problem is now sorted as the validation accuracy performs better than the noised training accuracy: this is also shown in the accuracy plot in figure 11. However, we still have problems with the loss values with 0.35 as the minimum for the validation set in figure 12 which trend is not constantly decreasing as we go forward in the epochs.

4 Final model and results

Once I achieved good results in the third model, I used hyperparameter tuning to get the best hyperparameter settings in the model's architecture in particular for the following hyperparameters

1. Number of filters in the convolutional layers [16-512]

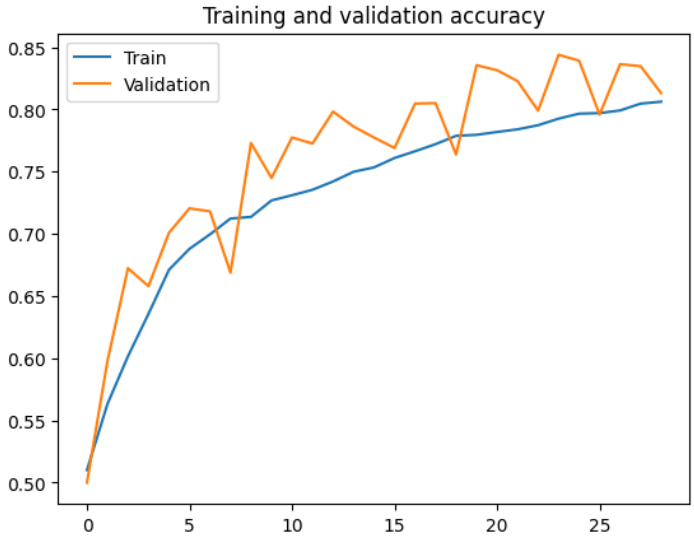

Figure 11: Model 3 accuracy

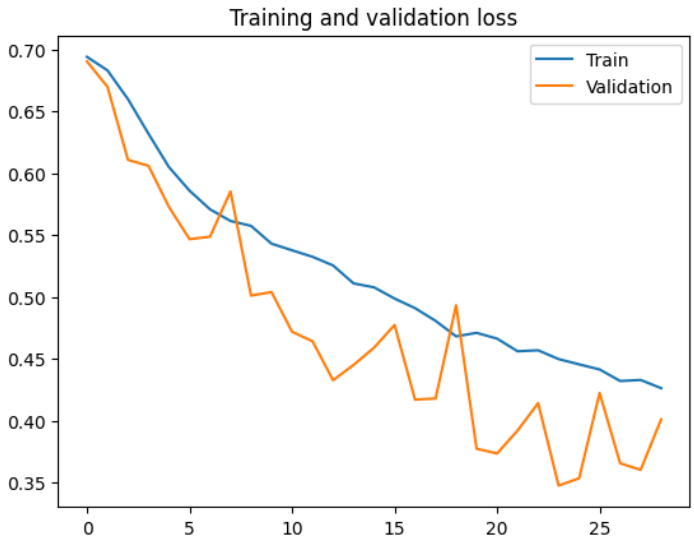

Figure 12: Model 3 loss

2. Dropout rate in the hidden layer[5% - 30%]
3. Number of units in the fully connected layer [64-256]
4. Dropout rate in the dense layer [20% - 50%]

Due to computational limitations, I stopped the configuration search at the 13th trial, obtaining 93% of validation accuracy with the hyperparameters setting illustrated in Table 2:

4.1 Training results

After these trials, I retrained the model with the found best hyperparameters and obtained the following results in accuracy and loss in both the train and validation set. Focusing on Figure 13, the accuracy curve of the validation set is above the training accuracy curve and reaches a plateau of 93% of accuracy. Also, the loss curve for the validation is below the training set curve which indicates the absence of the overfitting problem with minimum around 0.15

Hyperparameters	
Filters in Conv Layer 1	16
Filters in Conv Layer 2	128
Filters in Conv Layer 3	192
Filters in Conv Layer 4	256
Hidden dropout rate	5%
Dense Layer units	256
Dense dropout rate	30%

Table 2: Best values for accuracy found in hyperparameter tuning search

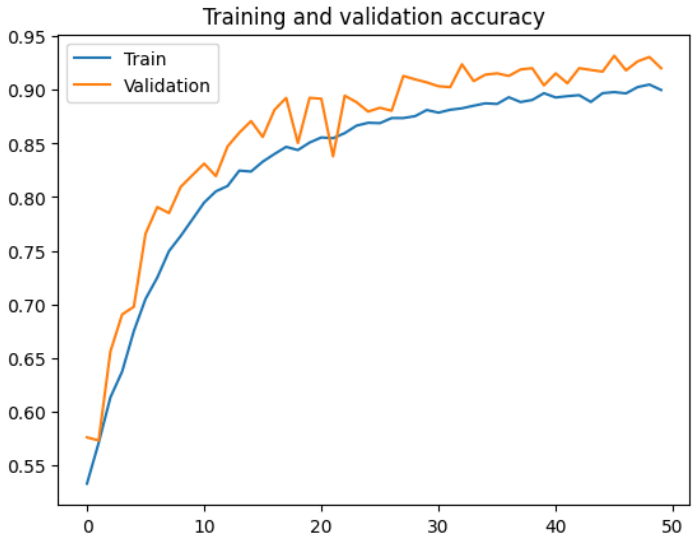

Figure 13: Hypermodel accuracy

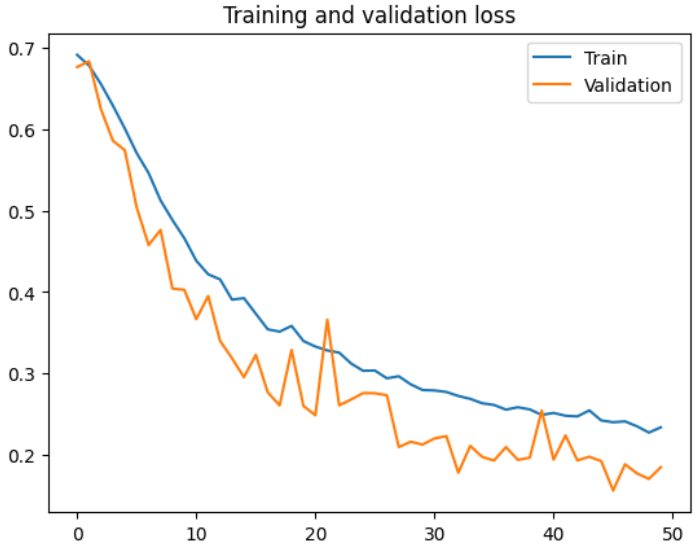

Figure 14: Hypermodel loss

4.2 Testing results

All the previous section's results were based only on Training and validation data: in the data pre-processing I kept 4987 images apart from this as a test set. Using our latest model it results in 91% accuracy and 0.2 in loss. In the following Table 3, I reported the confusion matrix of the classifier to

sum up the performances.

	Precision	Recall	F1 - score	Support
Cats	0.88	0.96	0.92	2479
Dogs	0.95	0.88	0.91	2508

Table 3: Confusion matrix of the classifier

Precision measures how the model is accurate in predicting a label. It is the ratio of the number of actual true labeled images divided by all the true predictions including false positives. For instance, it measures the real number of cat images in the dataset divided by the number of cat images predicted by the model.

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives}$$

Recall measures how the model is able to identify all the positive instances. It is the ratio between the number of actual true label images divided by true predictions including false negatives

$$Precision = \frac{TruePositives}{TruePositives + FalseNegatives}$$

F1-score is the harmonic mean between precision and recall. It is a single metric that gives us an overall measure of the model's performances

$$F1 = \frac{TruePositives}{TruePositives + \frac{FalsePositives + FalseNegatives}{2}}$$

4.3 Cross validation

Since the last model listed is the best performer I decided to run a 5-fold cross validation to have a deeper detailed knowledge of the model's performances. I divide the dataset into 5 subsets and at each iteration, I use 4 subsets for training and the remaining one for testing, for each iteration I store the scores of loss and accuracy. In Figure 15 and 16 we observe similar behavior in both training accuracy and loss curves for each fold with training accuracy always below test accuracy and training loss always above test loss shown in Tables 4 and 5. This is due to the noise present in the training folds only.

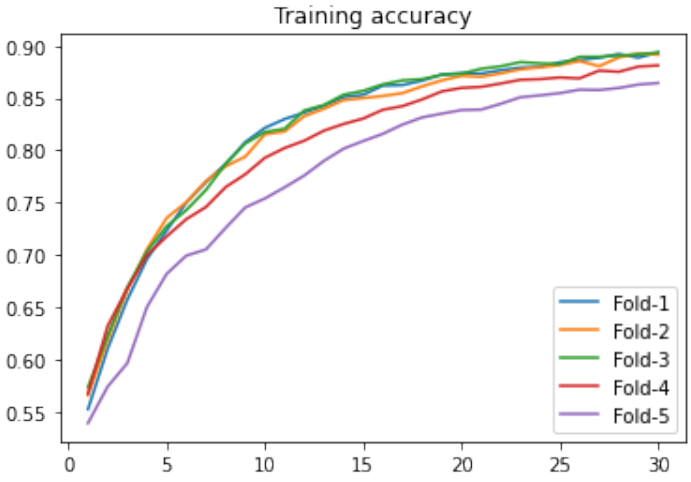

Figure 15: Train accuracy per fold

Finally, I compute a mean of all the scores obtained which results are illustrated in Table 6

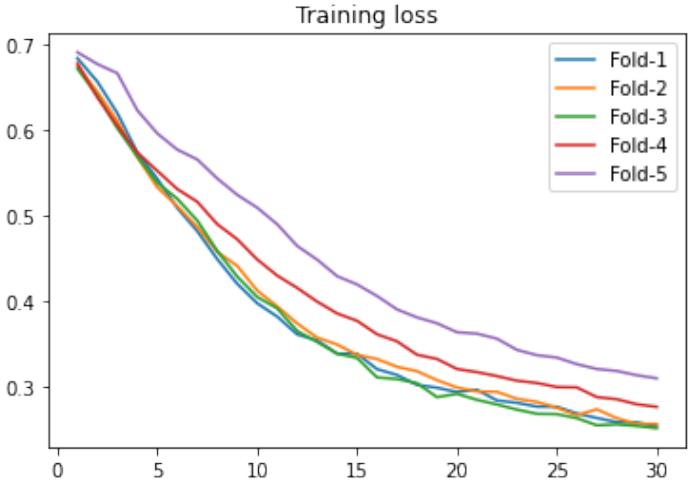

Figure 16: Train loss per fold

Training Accuracy	Test Accuracy
89%	91%
89%	90%
89%	92%
88%	90%
86%	88%

Table 4: Accuracy scores per fold

Training Loss	Test Loss
0.25	0.19
0.25	0.21
0.25	0.18
0.27	0.24
0.30	0.25

Table 5: Loss scores per fold

CV Accuracy (mean)	CV Loss (mean)
90%	0.22

Table 6: Cross validation scores results

5 Conclusions

The results obtained in all the previous sections reveal the following conclusions:

1. Adding convolutional layers to the model increases accuracy, i.e. networks learn and generalize better, however without regularization factors it leads to overfitting
2. Data augmentation and Dropout layers are an effective solution to the overfitting problem
3. Hyperparameter tuning is effective to improve the performances of the model. However, the search operation is expensive in both computation and time sides since finding an optimal setting is an NP-hard problem so alternative solutions are needed to rise accuracy.

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 offenses 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.