Cats and dogs images recognition

Chiara Saini

July 6, 2022

Abstract

The following analysis aims to explore Convolutional Neural Networks for image recognition. The images used in this analysis are those of cats and dogs. Firstly, the dataset, the image preprocessing and the train-test split will be explained. Secondly, the theory of Convolutional Neural Networks will be introduct. After that, five models will be explained. The first model was created as a benchmark. In the second model, slight complexity is added with respect to the first one. The third model was built with a complex hyperparameter tuning technique, and the fourth model has a different learning rate than the others. The final model was developed through 5-fold Cross-Validation.

1 The dataset

The dataset was initially composed of 25000 images (jpg format) belonging to two classes: Cats and Dogs. Of these 25000 files, 1590 were corrupted and filtered out to guarantee a successful analysis. The overall analysis was conducted on 23410 images, 11741 belonging to the Cats class and 11669 belonging to the Dogs class. The classes are slightly unbalanced: the Dogs class presents 72 fewer images than the Cats class. In Figure 1, it is possible to see the number of images belonging to the Cats class in orange and the Dogs class in blue.

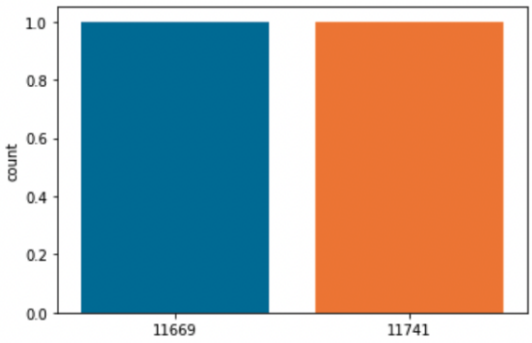

Figure 1: Number of images per class: Cats and Dogs.

1.1 Image prerocessing

Initially, the images were in RGB color mode. RGB images are composed of three independent color channels: red, green, and blue. Together these reproduce a set of arrays of colors that will give a colored image as output. An RGB image will be three of the two-dimensional matrix concatenated to each other, one for each channel of the color. In figure 2 it is possible to see an RGB image representation[1].

Gray-scale images result from measuring light intensity at each pixel and representing it exclusively using shades of gray. The intensity of pixels is expressed in a range between zero (minimum presence of light, black) and one (maximum presence of light, white). Gray-scale images can be represented as

[1]Reference: "Principal Component Analysis For Image Data in Python", Ask Python, https://www.askpython.com/python/examples/principal-component-analysis-for-image-data

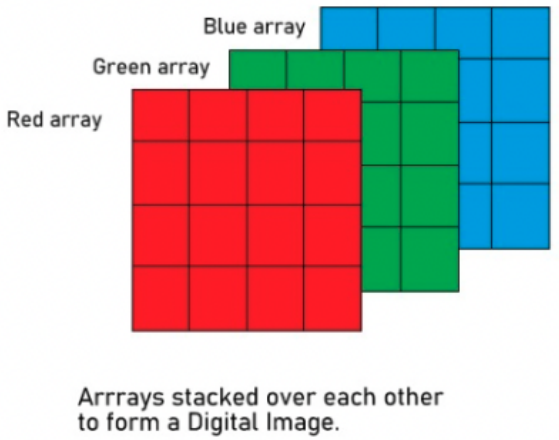

Figure 2: Example of RGB image representation.

a two-dimensional matrix of numbers (each number referring to a pixel of the image). Gray-scale is often used to simplify the calculations and remove eventual noise. In figure 3 it is possible to see a Gray-scale image representation [2]. In this analysis, images are processed in Gray-scale to reduce the time of calculations.

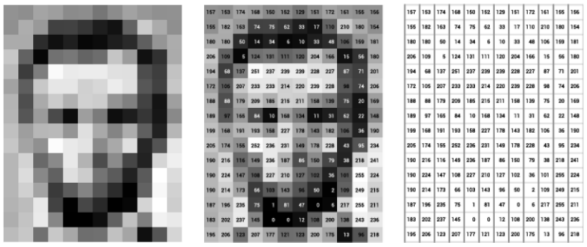

Figure 3: Example of Gray-scale image representation.

The images were stored as an array, using the command

```
\cv2.imread(os.path.join(path, img), cv2.IMREAD_GRAYSCALE)"}
```

.

Through this command, it was possible to join the directory to the images and convert them immediately into Gray-scale. It is possible to see an example of Gray-scaled image in figure 4.

Finally, given that the images are all in different sizes, it is mandatory to reshape them prior modeling. The images are converted to a 180x180 pixel size. It is possible to see an example of re-sized image data in figure 5.

Labels were assigned to the images: 0 for Cats and 1 for Dogs and then randomly shuffled, and images were normalized.

1.2 Train and test split

The dataset was divided using sklearn train-test split, with test being 20

2 Convolutional Neural Network Theory

Neural Networks (NNs) are a type of directed graph whose nodes correspond to neurons and edges correspond to links between them. Each neuron receives as input a weighted sum of the outputs of

[2]Reference: "Tutorial 1: Image Filtering", Stanford education, https://ai.stanford.edu/ syye-ung/cvweb/tutorial1.html

Figure 4: Gray-scale image representation of data.

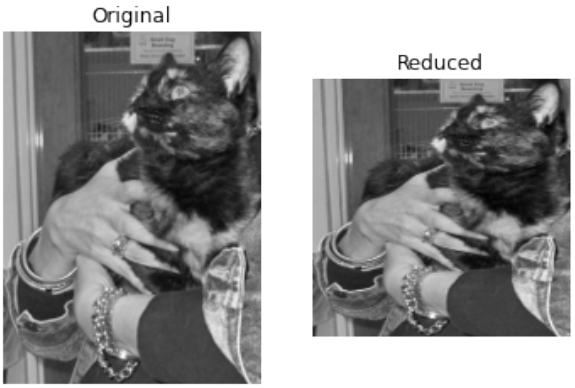

Figure 5: Original VS reduced data.

the neurons connected to its incoming edges. Convolutional Neural Networks (CNNs) are one of the most used NNs architectures. CNNs are used to recognize visual patterns and features directly from pixel images with a certain degree of variability. The identification of patterns and features is possible thanks to the ability of CNNs to keep important information about the data analyzed. This is done through its special architecture, which will be explained in the following sections. The complexity of CNN increases with each layer. The most superficial layers focus on simple features like colors and edges. As the image data progresses through deeper layers of the CNN, the algorithm starts to recognize more significant elements or shapes of the object until it finally identifies the intended object.

2.1 The Convolutional layer

A Convolution is an integral on two functions f and g, producing a third function c. The third function expresses the amount of overlapping of one function as it is shifted to the other. It combines one function with another.

$$c = (f * g)(t) = \int_t^0 f(T)g(t - T)\, dT$$

Where c is the convolution output function, f is the original input function, g is the function that is shifted over the input function, t is the variable representing the range of the shift and T is the shifting against t [3]. The convolutional layer is the core of CNNs, requiring input data, a filter, and a feature map.

[3]Reference:"Convolution", The Science of Machine Learning, https://www.ml-science.com/convolution

Convolution uses filters to perform a convolutional operation as it analyzes the input with respect to its dimensions. In doing so, the features belonging to different classes will be detected and extracted from the image using the Kernel. A convolution layer consists of several convolution channels (depth or filters). Convolution channels are represented by numbers (such as 16, 32, 64, 128, etc.) that are equal to the number of channels in the output of a convolutional layer. The Kernel is the size of the convolution filters. They are square matrices that can take sizes 1x1, 3x3, or 5x5. Convolution filters remove unnecessary data and pull features from the input image. The Kernel is a matrix (in this case, a 3x3 matrix) that moves over the input data by a stride (in this case, it is a pixel unit), performing the dot product with a sub-region of input data. In this way, the network will learn through the filters that activate when detecting a specific feature in the spatial position of the input. The output of the convolutional layer is a feature map. To add complexity, more than one convolutional layer can be piled.

Padding is the addiction of pixels needed for the convolutional Kernel onto the edge of the image. In this way, it is possible to solve the border effect. Convolution can use three different forms of padding in the form of zeros around a matrix:

- **No Padding**: only the original input is used (Figure 6). This is called a "valid" operation. Without padding, the filter is applied to the borders of an image just one time. In this way, it is possible to give more relevance to the information stored in the borders of an image [4];

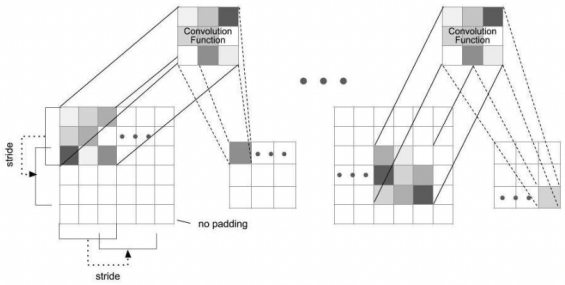

Figure 6: No padding.

- **Half Padding**: the padding is around part of the input (Figure 7). This is called the "same" operation. This is the padding used in the model;

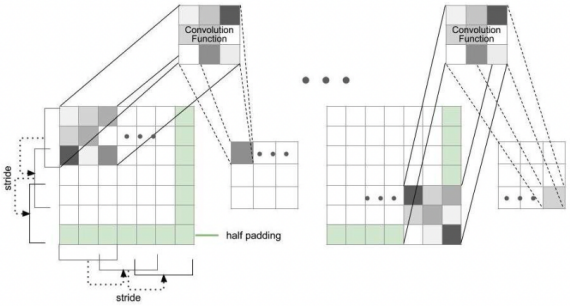

Figure 7: Half padding process.

- **Full Padding**: the padding is all around input (Figure 8). This is called the "full" operation.

Note that in the examples of pictures 7 and 8use a stride of 2, while in the model developed, the stride is of one unit pixel.

[4]Figure 6,7,8 came from this source: "Convolution", The Science of Machine Learning, https://www.ml-science.com/convolution

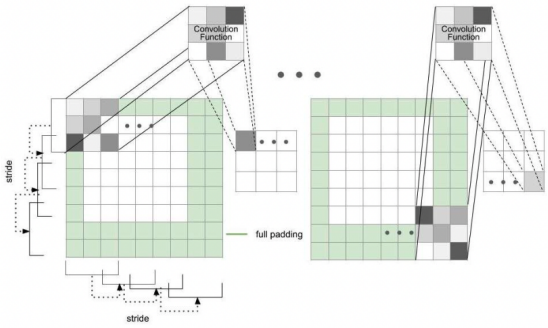

Figure 8: Full padding process.

2.2 Activation function ReLU

In Neural Networks, the activation function is used to transform the input values of neurons. A weighted sum of inputs is passed through an activation function, and this output serves as an input to the next layer. It introduces non-linearity so that the network can learn the relationship between the input and output values. If no activation is specified, it will be automatically assigned a linear function. In this analysis, the activation function used in convolutional layers is the *Rectified Linear Unit* (ReLU) activation function. This type of function returns 0 if it receives a negative input, and for any positive value x, it returns that value back[1]. Mathematically is is expressed as:

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

It can also be written as:

$$f(x) = \begin{cases} 0, & \text{if } x < 0 \\ x, & \text{if } x \geq 0 \end{cases}$$

Where X is an input value.

2.3 Pooling layers

Pooling layers are required to down-sample the feature maps. This is done by summarizing the features in patches of the feature map. The pooling function moves across the feature map every 2x2 patch (Figure 9[2]).

Two common methods of pooling are max-pooling and average-pooling:

- Max pooling takes out the maximum value between the features within the patch;
- Average pooling takes out the average of features within the patch.

Pooling is used to reduce dimensionality but preserve all the important features of the feature map. In the model, the default pooling is used (2x2).

2.4 Flatten layer

Flatten layer flattens a multi-dimensional input into a single dimension. This allows the modeling of the input layer and the final building of a neural network model[3]. In Figure 10 [4] it is possible to see the process of flattening, the outcome of flattening is represented by the yellow dots.

[1]Reference: "ReLU (Rectified Linear Unit) Activation Function", https://iq.opengenus.org/relu-activation/

[2]Source of figure 9: "Pooling", The Science of Machine Learning, https://www.ml-science.com/pooling

[3]Reference: "CNN: Step 4 — Connection", Medium, Panadda Kongsilp, https://medium.com/@PK$_K$*wanG*/*cnn* − *step* − 4 − *connection* − *e*67*d*3*a*49*baa*8

[4]Source of figure 10 and 11: "CNN: Step 4 — Connection", Medium, Panadda Kongsilp, https://medium.com/@PK$_K$*wanG*/*cnn* − *step* − 4 − *connection* − *e*67*d*3*a*49*baa*8

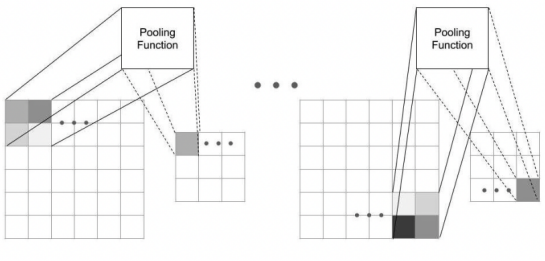

Figure 9: Pooling process.

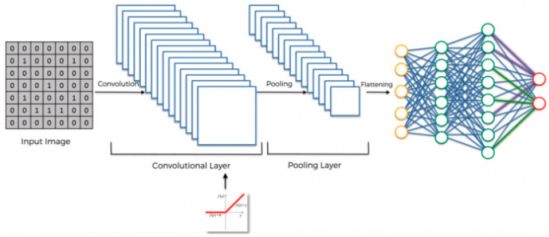

Figure 10: Process of CNN model, having a grayscale image as input.

2.5 Dense layer

The dense layer is a layer that is deeply connected with its preceding layer because it receives its output. The neurons of the dense layer perform matrix-vector multiplication: the column vector of the dense layer is equal to the row vector of the output from the preceding layers. The final output will be a vector.

amsmath

$$\mathrm{Ax} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \dots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \end{bmatrix} = \begin{bmatrix} a_{11}x_1 + a_{12}x_2 \cdots + a_{1n}x_n \\ a_{21}x_1 + a_{22}x_2 \cdots + a_{2n}x_n \\ \vdots + \vdots \cdots + \vdots \\ a_{m1}x_1 + a_{m2}x_2 \cdots + a_{mn}x_n \end{bmatrix}$$

There is a set of hyper-parameters that must be specified, such as the units and the activation function. Units are the most basic and necessary parameters of the Keras dense layer. They define the size of the output from the dense layer. It represents the dimensionality of the result vector; therefore, it must be a positive integer [5]. In this analysis, the activation functions used for the dense layers are the ReLU activation function and the *Sigmoid activation functio*. The Sigmoid function in machine learning usually refers to the logistic function; it is also called the logistic Sigmoid function. It is used as an activation function. The Sigmoid function is monotonic and guarantees that the output will be between 0 and 1. The Sigmoid is also a non-linear function; therefore, its output will be a non-linear function of the weighted sum of inputs. Thanks to the non-linearity, CNNs can easily learn non-linear problems[6]. A mathematical representation of the Sigmoid function is represented below.

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

Where $S(x)$ is the Sigmoid function and e is the euler number.

[5]Reference: "A Complete Understanding of Dense Layers in Neural Networks", Developercorner, Yugesh Verma, 19 September 2021, https://analyticsindiamag.com/a-complete-understanding-of-dense-layers-in-neural-networks/

[6]"A Gentle Introduction To Sigmoid Function", Machine Learning Mastery, Mehreen Saeed, 25 August 2021, https://machinelearningmastery.com/a-gentle-introduction-to-sigmoid-function/

In Figure 11 are represented the flattening and dense hidden layers final and the output of the CNN. The flattening layer is represented with yellow circles, the dense layers are represented in green circles, and the output is represented in red circles.

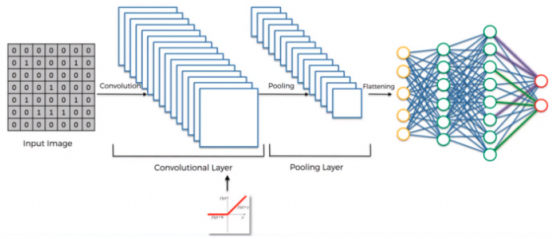

Figure 11: Flattening layer, dense layers and output of a CNN.

2.6 Compiling the neural network

2.6.1 Optimizer and learning rate

Adaptive Moment Estimation (Adam) is an adaptive learning rate optimization algorithm that's been designed specifically for training deep neural networks. It was first published in 2014. The learning rate of this algorithm is a configurable hyper-parameter; it has a range between 0.0 and 1.0. The default learning rate is 0.001. The learning rate rules on the ability of the model to adapt to the problem. While smaller learning rates require a bigger number of training epochs, given the smaller weight changes each update, larger learning rates require fewer training epochs given rapid weight changes. A too-large learning rate can drive the model to a sub-optimal solution, while a too-small learning rate could yield the process to get stuck [7].

2.6.2 Loss function

With *binary cross-entropy*, it is possible to compare each predicted probability with the actual class of output, which can be either 0 or 1. The function calculates how close the result is to the actual value. Binary Cross Entropy is the negative average of the log of corrected predicted probabilities.

$$Loss = \frac{1}{outputsize} \sum_{i=1}^{outputsize} y_i \cdot log\hat{y}_i + (1 - y_i) \cdot log(1 - \hat{y}_i)$$

Where $\hat{y}_i$ is the i-th scalar scalar value in the model output, y_i is the target value and $outputsize$ is is the number of scalar values in the model output[8].

2.6.3 Metrics

The metric used is Accuracy, which calculates how often predictions equal labels. The Accuracy metric creates two variables: total and count. These two local variables are used to compute the frequency of which the prediction of the label was correct ($\hat{y} = y$)[9].

2.6.4 Test results

The test results were evaluated with the following metrics:

- **Accuracy**;

[7] "Gentle Introduction to the Adam Optimization Algorithm for Deep Learning", Machine Learning Mastery, Jason Brownlee, 3 July 2017, https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

[8] Reference: "Binary Cross Entropy/Log Loss for Binary Classification", Analytics Vidhya, Shipra Saxena, 3 March 2021, https://www.analyticsvidhya.com/blog/2021/03/binary-cross-entropy-log-loss-for-binary-classification

[9] Reference: "Accuracy metrics", Keras, https://keras.io/api/metrics/accuracy$_metrics$

- **Precision**:also known as positive predictive value, is the number of true positive results divided by the number of all positive results;
- **Recall**: also known as sensitivity value, is the number of true positive results divided by the number of all the examples that should have been marked as positive;
- **F-1 score**:the F-score is a measure of a test's accuracy. The F-1 score is the harmonic mean of Precision and Recall[10].

3 Models

3.1 Model 1

Model 1 is the first model of the analysis. This model was kept simple in order to be used as a benchmark for further analysis.

3.1.1 Architecture

Model 1 is a sequential model. The number of layers is seven and the parameters in the model are 4055905.

1. **Convolutional layer**: the size of the filter is 32, the Kernel is 3x3. Inside the convolutional layer the rescaling of the input is done (re-scaled by 255) and the input shape is defined (180x180, grayscale image);
2. **Activation layer**: activation function ReLU;
3. **Pooling layer**: standard measures are left for this layer, pooling size is 2x2;
4. **Flatten layer**;
5. **First Dense layer**: the size of the filter is 16;
6. **Final Dense layer**: the size of the filter is 1;
7. **Activation function layer**: Sigmoid.

In figure 12 it is possible to see the summary of *Model 1*.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 178, 178, 32)	320
activation (Activation)	(None, 178, 178, 32)	0
max_pooling2d (MaxPooling2D)	(None, 89, 89, 32)	0
flatten (Flatten)	(None, 253472)	0
dense (Dense)	(None, 16)	4055568
dense_1 (Dense)	(None, 1)	17
activation_1 (Activation)	(None, 1)	0

Total params: 4,055,905
Trainable params: 4,055,905
Non-trainable params: 0

Figure 12: Summary of *Model 1*.

[10]Reference: "The Best Metric to Measure Accuracy of Classification Models", Clevertap, Jacob Joseph, https://clevertap.com/blog/the-best-metric-to-measure-accuracy-of-classification-models/

3.1.2 Compiling and fitting

The neural network was compiled as follows:

- **Optimizer**: Adam;
- **Loss**: Binary cross-entropy;
- **Metrics**: Accuracy.

The model fit was performed on the training set, and the the validation was set as 0.20 of the training set. The epochs evaluated were 10. The number of epochs was chosen because a superior number would have been too computationally intensive. The ideal number of epochs would be such that any further training provides little to no boost in validation accuracy. Another parameter to tune is the batch size. A larger batch size will allow the model to train more rapidly but will also lead to poor generalization. Smaller batch sizes have faster convergence to good solutions, but they require intensive computation and do not guarantee convergence to the global optima. The batch size was set to 32.

3.1.3 Loss curves and Accuracy curves

From figure 13 it is possible to see that the validation loss strongly increases after just two epochs (1.7015 validation loss at the tenth epoch) while the training loss decreases (0.0436 training loss at the tenth epoch). The validation accuracy doesn't seem to perform well (0.6818 validation accuracy at the tenth epoch), while training accuracy is nearly perfect (0.9874 training accuracy at the tenth epoch). Given these results it is safe to say that Model 1 leads to over-fitting.

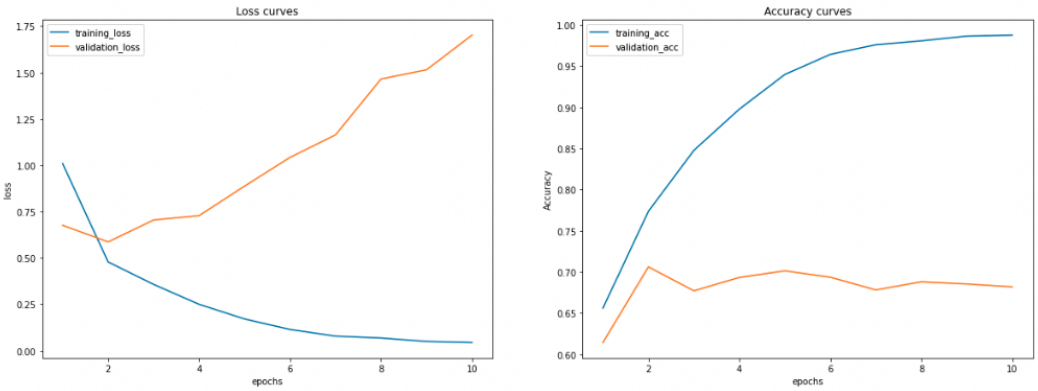

Figure 13: Loss and accuracy curves of *Model 1*.

3.1.4 Prediction skills

The metrics for the evaluation of the prediction skills for *Model 1* gave the following results:

- **Accuracy**:0.699701;
- **Precision**:0.69975;
- **Recall**:0.699721;
- **F-1 score**:0.699694.

These results were expected given the shape of the validation accuracy curve.

3.2 Model 2

Model 2 was done to experiment the effects of the addiction of a second convolutional layer on the loss and accuracy.

3.2.1 Architecture

Model 2 is a sequential model. The number of layers is ten and the parameters in the model are 610805.

1. **First Convolutional layer**: the size of the filter is 16, the Kernel is 3x3.The the input is re-scaled by 255 and the input shape is defined (180x180, grayscale image). Padding "same" is added;
2. **Activation layer**: activation function ReLU;
3. **Pooling layer**: standard measures are left for this layer, pooling size is 2x2;
4. **Second Convolutional layer**: the size of the filter is 32, the Kernel is 3x3, the paggind is set to "same";
5. **Activation layer**: activation function ReLU;
6. **Pooling layer**: pooling size is 2x2;
7. **Flatten layer**;
8. **First Dense layer**: the size of the filter is 10;
9. **Final Dense layer**: the size of the filter is 1;
10. **Activation function layer**: Sigmoid.

In figure 14 it is possible to see the summary of *Model 2*.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 178, 178, 64)      640

activation_8 (Activation)    (None, 178, 178, 64)      0

max_pooling2d_5 (MaxPooling  (None, 89, 89, 64)       0
2D)

conv2d_6 (Conv2D)            (None, 87, 87, 32)        18464

activation_9 (Activation)    (None, 87, 87, 32)        0

max_pooling2d_6 (MaxPooling  (None, 43, 43, 32)       0
2D)

flatten_3 (Flatten)          (None, 59168)             0

dense_6 (Dense)              (None, 10)                591690

dense_7 (Dense)              (None, 1)                 11

activation_10 (Activation)   (None, 1)                 0
...
Total params: 610,805
Trainable params: 610,805
Non-trainable params: 0
_________________________________________________________________
```

Figure 14: Summary of *Model 2*.

3.2.2 Compiling and fitting

The neural network was compiled as done in *section 4.1.2*.

3.2.3 Loss curves and Accuracy curves

Despite the addition of a convolutional layer, the accuracy and loss curves have not improved much, as shown in figure 15. The validation loss starts to increase after six epochs and at the tenth epochs is

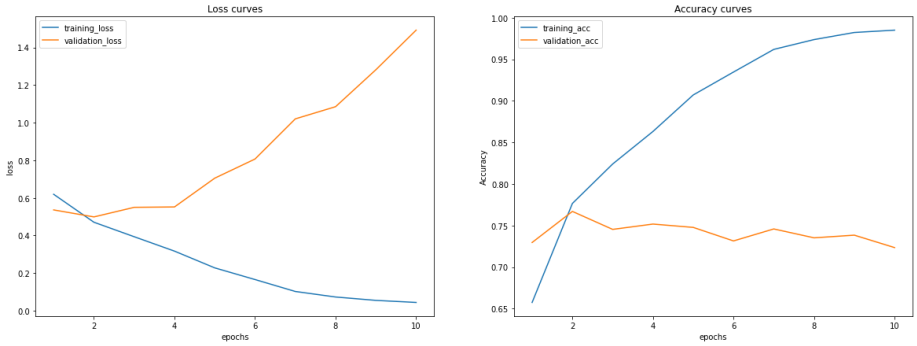

Figure 15: Loss and Accuracy curves of*Model 2*.

1.4920, which is slightly better than the result of *Model 1*.The training loss decreases less rapidly and at the tenth epoch reaches a value of 0.0996 (also this result is slightly better than the previous). The validation accuracy performed better as opposed to the one of *Model 1*: it reaches a value of 0.7670 at the tenth epoch. The training accuracy has a smaller value at epoch ten as opposed to *Model 1*: 0.9796. While the validation accuracy fluctuate around 0.73, the training accuracy skyrockets and reaches 0.9796. Although the values between training and validation have slightly approached, there are clear signs of over-fitting.

3.2.4 Prediction skills

The metrics for the evaluation of the prediction skills for *Model 1* gave the following results:

- **Accuracy**:0.73003;
- **Precision**:0.732181;
- **Recall**:0.729883;
- **F-1 score**:0.73003.

These results were expected given the shape of the validation accuracy curve.

3.3 Model 3

Model 3 was build through an hyperparameter tuning method.

3.3.1 Hyperparmeter Tuning and architecture

The hyperparameter tuning was performed to choose the best amount of convolutional layers, the best convolutional filter and the best choice for the Dense layer.

After an initial investigation through *Tensorboard* extension, it was possible to establish that the best number of convolutional layers is 3. The best models featured the following architectures:

1. 3 convolutional layers, 32 nodes,0 dense layer, 1657025674 parameters;
2. 3 convolutional layers, 64 nodes, 0 dense layer, 1657025674 parameters;
3. 3 convolutional layers, 128 nodes, 0 dense layer, 1657025674 parameters;
4. 3 convolutional layers, 32 nodes, 1 dense layer, 1657025674 parameters;
5. 3 convolutional layers, 64 nodes, 1 dense layer, 1657025674 parameters;

6. 3 convolutional layers, 128 nodes, 1 dense layer, 1657025674 parameters;
7. 3 convolutional layers, 32 nodes, 2 dense layer, 1657025674 parameters;
8. 3 convolutional layers, 64 nodes,2 dense layer, 1657025674 parameters;
9. 3 convolutional layers, 128 nodes, 2 dense layer, 1657025674 parameters.

Tensorboard was used to visualize the curves of loss and accuracy of training and validation. The activation functions for the convolutional layers and the dense layers are the same as for *Model 1* and *2*.

3.3.2 Compiling and fitting

The parameters for the compilation and fitting of the models are the same as for *Model 1* and *2*.

3.3.3 Loss curves and Accuracy curves

Given the scores of each model, number 1 was chosen as the best one. Figure16 shows that the accuracy curves of train and validation tend to stay much closer to each other with respect to the previous models. Also, the loss curves seem to do better. Overall, both accuracy and loss start to diverge at epoch three. At epoch ten, the training loss was 0.2655, while the validation loss was 0.5549. Concerning accuracy values, the train at epoch ten had a value of 0.8853 and the validation 0.7587. This leads to thinking that there is little overfitting, but compared to past models, the performance has improved.

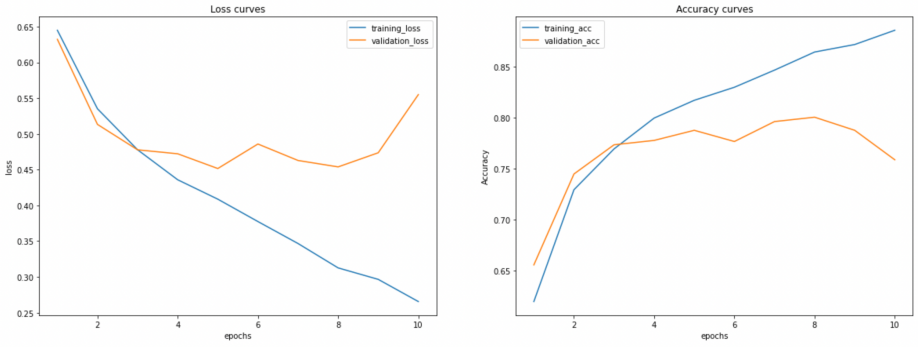

Figure 16: Loss and Accuracy curves of*Model 3*.

3.3.4 Prediction skills

The metrics for the evaluation of the prediction skills for *Model 3* gave the following results:

- **Accuracy**:0.755446;
- **Precision**:0.780984;
- **Recall**:0.754994;
- **F-1 score**:0.74954.

Prediction skills slightly improved from *Model 2*.

3.4 Model 4

In *Model 4*, another architecture was used to see if it was possible to outcome the results of *Model 3*.

3.4.1 Architecture

Model 4 is a sequential model. The number of layers is thirteen and the parameters in the model are 333077.

1. **First Convolutional layer**: the size of the filter is 16, the Kernel is 3x3.The the input is re-scaled by 255 and the input shape is defined (180x180, grayscale image). Padding "same" is added;
2. **Activation layer**: activation function ReLU;
3. **Pooling layer**: pooling size is 2x2;
4. **Second Convolutional layer**: the size of the filter is 32, the Kernel is 3x3, the paggind is set to "same";
5. **Activation layer**: activation function ReLU;
6. **Pooling layer**: pooling size is 2x2;
7. **Third Convolutional layer**: the size of the filter is 64, the Kernel is 3x3, the paggind is set to "same";
8. **Activation layer**: activation function ReLU;
9. **Pooling layer**: pooling size is 2x2;
10. **Flatten layer**;
11. **First Dense layer**: the size of the filter is 10;
12. **Final Dense layer**: the size of the filter is 1;
13. **Activation function layer** with Sigmoid.

In figure 17 it is possible to see the summary of the model.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_10 (Conv2D)           (None, 180, 180, 16)      160

activation_14 (Activation)   (None, 180, 180, 16)      0

max_pooling2d_10 (MaxPoolin  (None, 90, 90, 16)       0
g2D)

conv2d_11 (Conv2D)           (None, 90, 90, 32)        4640

activation_15 (Activation)   (None, 90, 90, 32)        0

max_pooling2d_11 (MaxPoolin  (None, 45, 45, 32)       0
g2D)

conv2d_12 (Conv2D)           (None, 45, 45, 64)        18496

activation_16 (Activation)   (None, 45, 45, 64)        0

max_pooling2d_12 (MaxPoolin  (None, 22, 22, 64)       0
g2D)

...
Total params: 333,077
Trainable params: 333,077
Non-trainable params: 0
_________________________________________________________________
```

Figure 17: Summary of*Model 4*.

3.4.2 Compiling and fitting

The parameters for the compilation and fitting of the models are the same as for *Model 1* and *2*, with the only exception of the learning rate of the optimizer, which is set to 0.0001.

3.4.3 Loss curves and Accuracy curves

The train and validation loss curves are much closer to each other as opposed to other models (see figure 20). This is also true for accuracy curves. The training loss at epoch ten is 0.3738, while the validation loss is 0.4461. The training accuracy is 0.8333 and the validation accuracy is 0.8067. Furthermore, what can be observed from figure 20 is that both validation loss and accuracy have a much stronger positive trend than those of the other models. This satisfactory result would probably improve if the epochs were increased or the batch size changed.

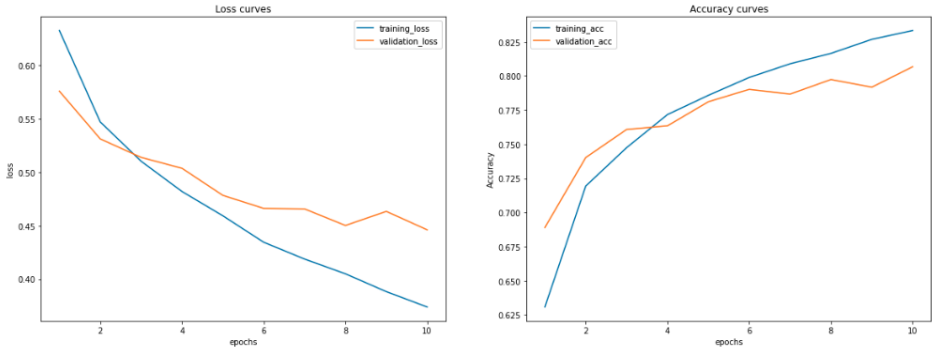

Figure 18: Loss and accuracy curves of *Model 4*.

3.4.4 Prediction skills

The metrics for the evaluation of the prediction skills for *Model 4* gave the following results:

- **Accuracy**:0.789193;
- **Precision**:0.791767;
- **Recall**:0.789332;
- **F-1 score**:0.78878.

Overall, prediction skills improved from past models.

3.5 Cross-Validation

Finally a 5-Fold cross validation was used, its purpose is to improve the accuracy and reduce overfitting. The architecture used is described in *section 3.5.1*. Cross-validation uses all data to make predictions, by dividing the dataset in k-Folds. There are five iterations in which training is always done on four folds and the validation on one fold. Within each iteration the training and the validation folds change. This process is better explained figure 21.

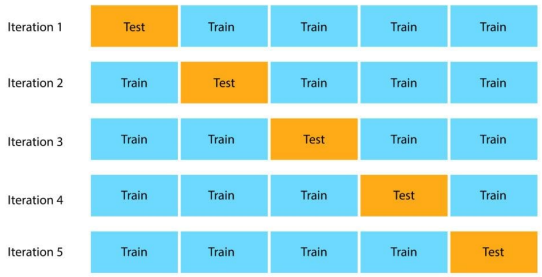

Figure 19: Example of 5-Fold Cross-Validation.

Thanks to the 5-Fold Cross-Validation, the python script computes five different models and returns the average metrics for each. Therefore, the algorithm's performance is more likely realistic because train and test were performed on all data.

3.5.1 Architecture

The Cross-Validation was performed on a model with the following architecture:

1. **First Convolutional layer**: the size of the filter is 32, the Kernel is 3x3.The input shape is defined (180x180, grayscale image), the padding is "same", and the activation function is ReLU;
2. **Second Convolutional layer**: the size of the filter is 64, the Kernel is 3x3, the padding is "same", and the activation function is ReLU;
3. **Pooling layer**: standard measures are left for this layer, pooling size is 2x2;
4. **Flatten layer**;
5. **First Dense layer**: the size of the filter is 512;
6. **Dropout layer**: randomly sets input units to 0 with a frequency of rate at each step during training time, which should helps prevent overfitting;
7. **Final Dense layer**: the size of the filter is 1, the activation function is Sigmoid;

It is possible to see the summary of the model in Figure **??**.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 180, 180, 32)      320

conv2d_3 (Conv2D)            (None, 180, 180, 64)      18496

max_pooling2d_1 (MaxPooling  (None, 90, 90, 64)        0
2D)

flatten_1 (Flatten)          (None, 518400)            0

dense_2 (Dense)              (None, 512)               265421312

dropout_1 (Dropout)          (None, 512)               0

dense_3 (Dense)              (None, 1)                 513

=================================================================
Total params: 265,440,641
Trainable params: 265,440,641
Non-trainable params: 0
_________________________________________________________________
```

Figure 20: 5-Fold Cross-Validation model summary.

For the convolutional and dense layers and the first, the activation function was ReLU; for the last dense layer, the activation function was Sigmoid. The loss used was the binary cross-entropy loss, the optimizer was the Adam optimizer, and the metric was the accuracy. The Cross-Validation was performed on five epochs to ease the computation. The batch size was 32, and the validation split was 0.1. To achieve the best results and ease the computation, early stopping and model checkpoints (save the best only) were added to the compilation of the model.

In figure 21 the result of the Cross-Validation is shown. It is possible to see that the result of validation accuracy fluctuates around 0.7, and as the epochs go toward ten, the more the accuracy stabilizes around this value. It is clear that this model leads to overfitting.

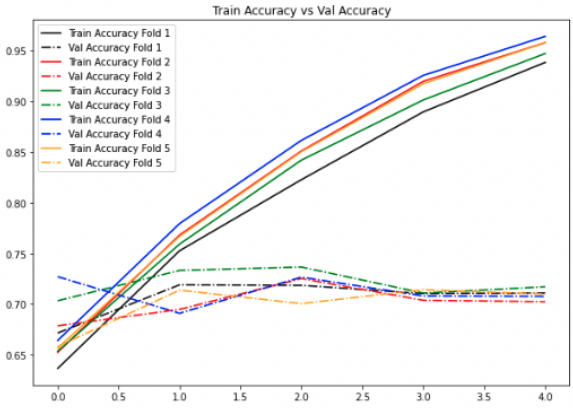

Figure 21: Example of 5-Fold Cross-Validation.

3.6 Conclusions

The best performing model is *Model 4*, which had a lower learning rate than the other models. To test this model with new data, an image of a cat was taken from the internet (Figure 22[11]).

Figure 22: Cat.

Model 4 correctly specified the class of figure 22. This result confirms the predictive ability of *Model 4*.

[11] Source: https://placekitten.com/g/200/300

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.