Binary Classification of images with Neural Networks

Margherita Menegazzi
ID: 974494
margherita.menegazzi@studenti.unimi.it

May 31, 2023

1 Introduction

This project's goal is to perform binary classification of cats' and dogs' images using Machine Learning techniques. The best-suited family of algorithms for this task is Neural Networks. Three Neural Network models were constructed and trained, and the final and best model was tested using 5-fold Cross-Validation.

2 Data

The original dataset is composed of 25000 images, of which 12500 are images of dogs and 12500 are images of cats. This is a very simple dataset where the datapoints are vectors of numbers representing the pixels that compose each image, and the only feature of these datapoints is the label assigned to each image which can be 'Dog' or 'Cat'.
Unfortunately, some of these images were corrupted and had to be eliminated in order to proceed with the analysis. The clean dataset can count 24920 images, of which 12454 are dogs and 12466 are cats, so the clean dataset is still balanced.

Some images from the dataset can be observed below to get an idea:

The dataset was divided into a Training Set and a Validation Set using 60% of the original data for training and 40% of the original data for validation. Then, 84% of the Validation Set was taken to make the Test Set. During this process, the color mode of the images was changed to 'RGB', meaning that each pixel was re-coded as an 'RGB' value. This was done to make it easier for the Neural Network to map the input space to a discrete space for classification. Moreover, in the creation of the three datasets the data was shuffled to keep the balance between cats' and dogs' pictures, and to have randomness in the datasets.

3 Neural Networks

As already said, the family of Machine Learning algorithms used to perform an analysis on this dataset are Neural Networks. Neural Networks are a class of predictors inspired by the human nervous system, which is an aggregation of neurons. The individual entities of a Neural Network, called neurons, are combined to make up a complex predictor. A neuron has this simple form:

$$g(v) = \sigma(w^T x)$$

where:

- `sigma`: activation function
- `wt`: weight associated to the edge
- `x`: vector of values

Each input to a neuron is encoded as a real number, and the neuron takes the weighted sum of the inputs it receives. It is important to note that the weights are the only trainable parameters of the network.
The activation function is applied to this input to determine the output of the neuron. The activation function will typically be nonlinear and the most popular are:

- `Sigmoidal`:

$$\sigma(z) = \frac{1 - e^{-z}}{1 + e^{-z}} \quad \in [-1, 1]$$

- `ReLU`:

$$\sigma(z) = (\alpha I(z < 0) + I(z \geq 0))z \quad 0 < \alpha \ll 1$$

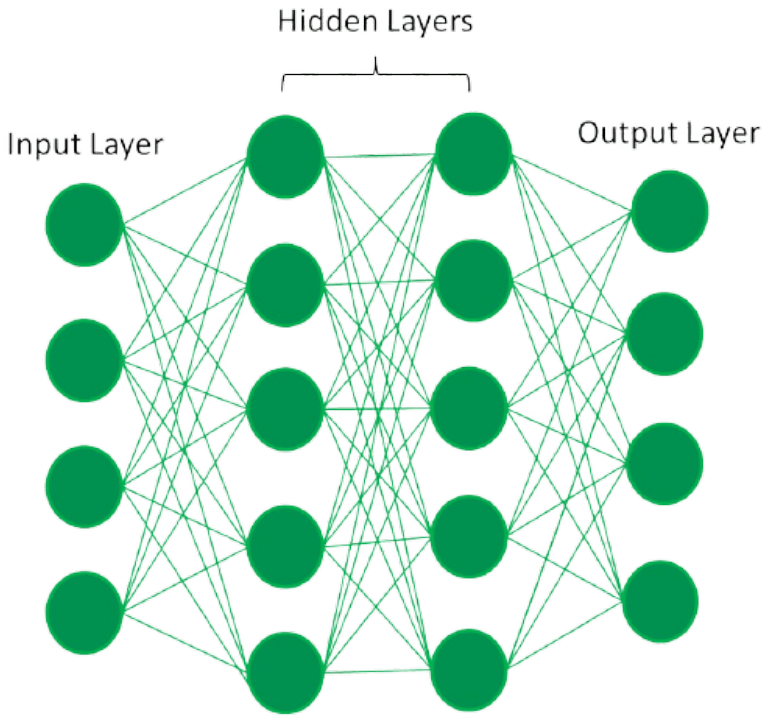
The Sigmoidal function is especially used to predict the probability of an output, and it has an S shape.
The ReLU function is a piecewise linear function and it outputs the input directly if it is positive, otherwise, it will output zero.

A Neural Network combines these neurons into a directed graph G=(V,E), where V=(g1, ..., gn) is the set of neurons and E is the set of edges of the graph. Each node i of the graph computes a function g(v) whose argument v is the value of the nodes j that have an outgoing edge towards node i.

A single neuron could be sufficient to solve a very simple problem, but in most cases, Multilayer Neural Networks are built.
Multilayer Neural Networks link the neurons together making them share inputs and outputs. The basic architecture consists of 3 layers: an input layer, a hidden layer, and an output layer.

- `Input layer`: takes as input raw data and passes it to the rest of the network

- `Hidden layer(s)`: intermediate layers between the input and output layers that process the data and enable the neural network to learn complex tasks
- `Output layer`: takes as input the processed data from the last hidden layer and outputs the final result



Depending on the kinds of hidden layers implemented in a Neural Network, different kinds of algorithms can be shaped to solve the Machine Learning task needed to be solved.
In this project, two types of Neural Network models were implemented: a Feed Forward Neural Network and two Convolutional Neural Networks.

4 The Models

4.1 Model 1: Feed Forward Neural Network

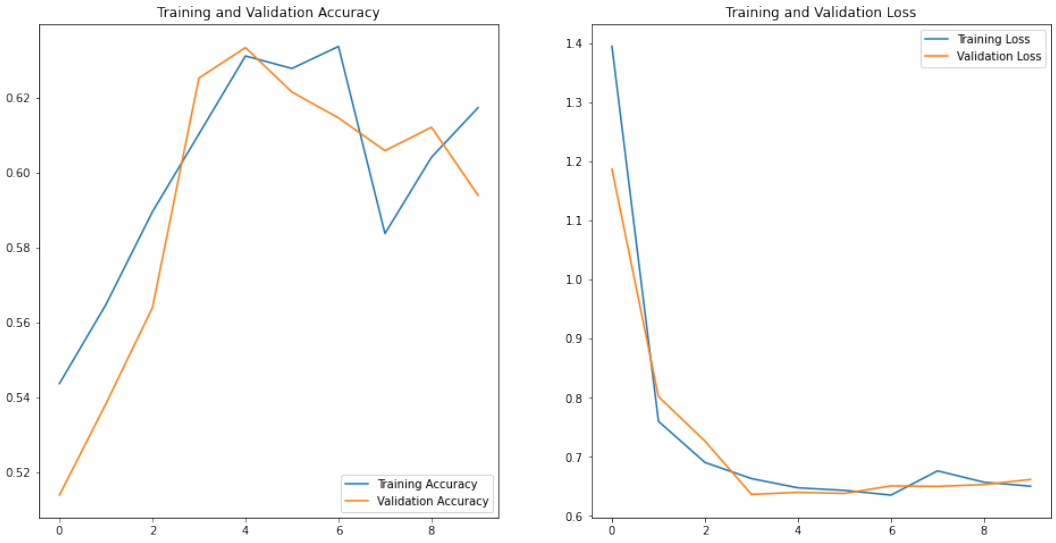
The Feed Forward Neural Network is the simplest from of Neural Network. It can be represented as a Directed Acyclic Graph because the connection between the nodes do not form a cycle. Information is processed in one direction: when data moves through the hidden nodes it always moves forward. This is the structure of the Feed Forward Neural Network implemented here:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling (Rescaling)       (None, 100, 100, 3)       0

 flatten (Flatten)           (None, 30000)             0

 dense (Dense)               (None, 128)               3840128

 dense_1 (Dense)             (None, 32)                4128

 dense_2 (Dense)             (None, 2)                 66

=================================================================
Total params: 3,844,322
Trainable params: 3,844,322
Non-trainable params: 0
_________________________________________________________________
```

This architecture consists of an input layer, two hidden layers with the ReLU activation function, and an output layer.
The first layer is a pre-processing layer that scales the input pixel values between 0 and 1 by dividing them by 255. The shape of the input data is specified as a 100x100 image with 3 color channels (RGB).
The second layer is used to flatten the image in input into a 1D vector. This is necessary to connect it to the subsequent fully connected layers.
The following two layers are fully connected layers with respectively 128 and 32 neurons, and they use the ReLU activation function to learn complex patterns in the data and introduce non-linearity to the network.
The final layer outputs the predicted probabilities of an image being classified correctly.

This model is then trained using the Training set and Validation set that were created at the beginning, 10 training epochs are used.
During each epoch, the model goes through the entire Training set, calculates the loss function (Sparse Categorical Crossentropy was used), and adjusts the weights and biases to minimize the loss. The model's performance on the validation data is evaluated after each epoch.

The performance of this model can be visualized in the plot below:



The model reaches the best validation accuracy at epoch 5 at 63%, after that point some overfitting behavior can be observed.
Since a Feed Forward Neural Network is the simplest algorithm that could be implemented, it is probably not well suited for this classification problem. For this reason, another type of Neural Network is used.

4.2 Model 2: Convolutional Neural Network

Convolutional Neural Networks are a more complex type of algorithm that, in the case of image classification, can do a better job of separating the object of interest in an image from the noisy background of the photo.

How are they able to do this? Instead of treating the entire image as a whole, CNNs apply filters to the input data. These filters slide over the image to extract relevant features and capture patterns, such as edges, textures, and shapes, by performing mathematical operations called convolutions.
By using multiple filters in each layer, CNNs can learn to detect various types of features simultaneously. The result is a set of feature maps that highlight important aspects of the input image.

These feature maps are obtained by combining the responses of the filters across different regions of the image.

The second key component of CNNs are pooling layers. Pooling layers reduce the spatial dimensions of the feature maps while retaining the most significant information. The most widely used type of pooling layer is max-pooling, which selects the maximum value within each pooling window.

After several convolutional and pooling layers, the feature maps are flattened into a 1D vector. This vector is then fed into fully connected layers, and from this point, the structure of the Neural Network is the same as a Feed Forward Neural Network with fully connected layers and an output layer.

The second model constructed for this analysis is a Convolutional Neural Network with the following structure:

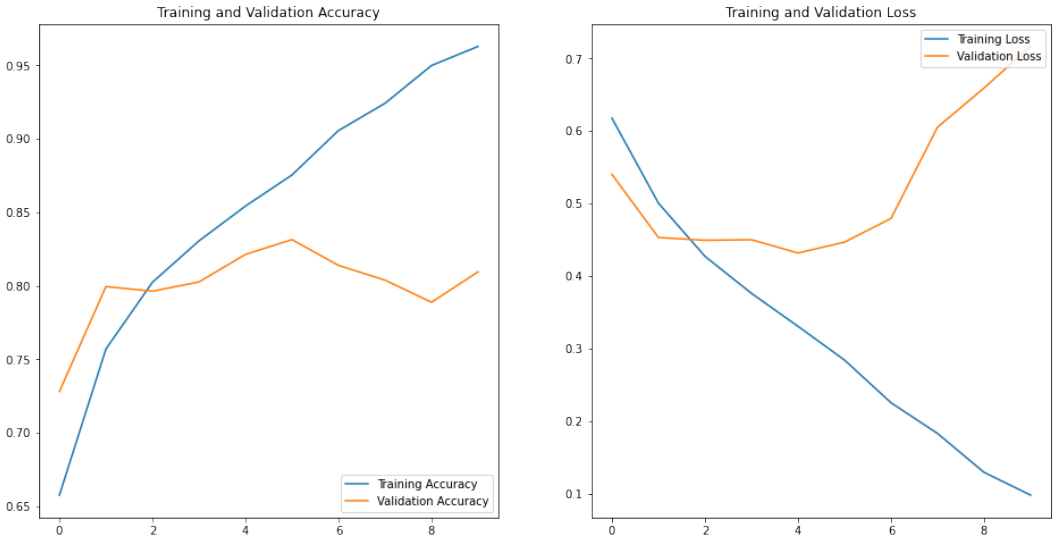
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_1 (Rescaling)     (None, 100, 100, 3)       0

 conv2d (Conv2D)             (None, 100, 100, 16)      448

 max_pooling2d (MaxPooling2D  (None, 50, 50, 16)       0
 )

 conv2d_1 (Conv2D)           (None, 50, 50, 32)        4640

 max_pooling2d_1 (MaxPooling  (None, 25, 25, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 25, 25, 64)        18496

 max_pooling2d_2 (MaxPooling  (None, 12, 12, 64)       0
 2D)

 flatten_1 (Flatten)         (None, 9216)              0

 dense_3 (Dense)             (None, 128)               1179776

 dense_4 (Dense)             (None, 2)                 258

=================================================================
Total params: 1,203,618
Trainable params: 1,203,618
Non-trainable params: 0
_________________________________________________________________
```

This model has the same type of pre-processing input layer as the previous model.
It then has three convolutional layers where filters have a 3x3 dimension, and the number of filters gradually increases in each layer to capture more complex patterns.
Each convolutional layer is followed by a max-pooling layer that reduces the spatial dimensions of the filter map by taking the maximum value within each pooling window.
A flatten layer is used to flatten the 3D output from the last convolutional layers into a 1D vector. This prepares the data for input into the fully connected layer.
A fully connected layer with 128 neurons layer takes the flattened input vector and performs a linear transformation. All the convolutional layers and the fully connected layer implement the ReLU activation function.
Finally, the output layer with 2 neurons represents the predicted classes.

Again, the model is trained using the Training set and Validation set in 10 epochs, and the Sparse Categorical Crossentropy loss function is used.

The performance of this model can be observed on the following page.
This model reaches a peak validation accuracy of 83% at epoch 6, after that point some overfitting behavior is still present. This overfitting could be caused by a lack of regularization techniques applied in the model.

The absence of regularization techniques can contribute to overfitting because the model can become too complex and it can easily memorize the training data, including noise and outliers. This can lead to overfitting as the model starts to fit the training data too closely, failing to capture the underlying patterns in the data.
Since the model tries to fit every data point precisely, it may pick up irrelevant variations that are specific to the training set but not representative of the true underlying relationships. As a result, the model may perform poorly on new data that contains different variations or noise levels.



To solve this problem, a third model implementing Dropout layers is created.

4.3 Model 3: Convolutional Neural Network with Dropout layers

Dropout is applied to the hidden layers. At each training iteration, each neuron has a probability of being temporarily dropped out. When a neuron is dropped out, its output is set to zero, and it is removed from the network for that particular iteration. The probability is applied independently to each neuron so that different neurons are dropped out in each training iteration.

This technique reduces overfitting because by randomly dropping out neurons it reduces the network's sensitivity to particular input patterns and encourages the learning of more generalized features.

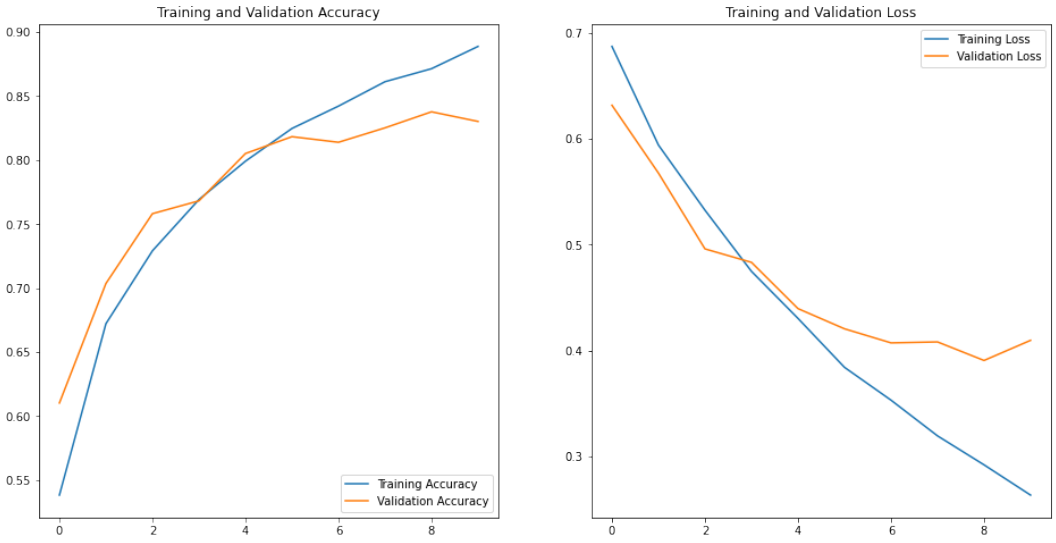
The third and final model of this project is just a slight modification of the previous one. Three main changes were made:

- `Increasing size of filters`: now at each convolutional layer not only the number of filters, but also the size of the filters increases. This allows the convolutional layers to extract increasingly complex features from the input images.
- `Dropout layers`: after each max pooling layer, a Dropout layer is added with a dropout rate of 20%, so 20% of neurons are randomly deactivated during training.
- `Sigmoidal activation function`: the fully connected layer now uses the sigmoidal activation function, because it squashes the input values into the range of 0 to 1 and this makes it convenient for our binary classification task.

The structure of this third model can be observed on the next page.

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_2 (Rescaling)     (None, 100, 100, 3)       0

 conv2d_3 (Conv2D)           (None, 100, 100, 16)      448

 max_pooling2d_3 (MaxPooling  (None, 50, 50, 16)       0
 2D)

 dropout (Dropout)           (None, 50, 50, 16)        0

 conv2d_4 (Conv2D)           (None, 50, 50, 32)        18464

 max_pooling2d_4 (MaxPooling  (None, 25, 25, 32)       0
 2D)

 dropout_1 (Dropout)         (None, 25, 25, 32)        0

 conv2d_5 (Conv2D)           (None, 25, 25, 64)        165952

 max_pooling2d_5 (MaxPooling  (None, 12, 12, 64)       0
 2D)

 dropout_2 (Dropout)         (None, 12, 12, 64)        0

 flatten_2 (Flatten)         (None, 9216)              0

 dense_5 (Dense)             (None, 128)               1179776

 dense_6 (Dense)             (None, 2)                 258

=================================================================
Total params: 1,364,898
Trainable params: 1,364,898
Non-trainable params: 0
_________________________________________________________________
```

Once again, the model is trained using the Training set and Validation set in 10 epochs, and the Sparse Categorical Crossentropy loss function is used. These changes allow the performance of the model to improve, as the image below shows:



The validation accuracy still reaches a peak of 83%, but now this peak happens at epoch 9 and the distance between training accuracy and validation accuracy is a lot smaller. This indicates that the model does not overfit anymore.

Now that a good final model has been found, it can be tested using 5-fold cross-validation.

5 Testing Model 3: 5-Fold Cross Validation

Cross-validation is a technique used to assess the performance of a model in a sound way.
Since this is done to test the model, now the Test set that was created at the beginning is used to see if model 3 performs well outside of the training phase.
Using 5-fold cross-validation for testing works in the following way:

- The Test set is divided in five equal subsets called 'folds';
- For five iterations, the model is trained on four folds and evaluated on the remaining fold. In each iteration, a different fold is used as the validation set, and the other four folds are combined to form the training set. This ensures that each subset acts as a validation set exactly once;
- After each iteration the model's performance is assessed, in this case the metrics of interest are accuracy and loss;
- To obtain an overall assessment of the model's performance, the evaluation metrics from the five iterations are averaged together in the end.

To ensure that the model is given enough time to stabilize its performance, each fold runs 40 epochs. Unfortunately, this is not enough to avoid overfitting. For each fold, from the first to the last epoch the accuracy fluctuates between 48% and 51%. Many things were done to change this result: lower the number of epochs, change the learning rate, modify the parameters of the model, and check that the test set is balanced, but none of these things helped in improving the test performance of model 3. In the end, the average scores for all folds were an accuracy of 49% and a zero-one-loss of 30%.

6 Conclusion

Considering all the things that were done to make sure that the model would perform well on the test set, the most probable cause for the overfitting of the final model in the test phase is that the available data was not enough in quantity and diversity to allow the model to learn generalized patterns.
Augmentation techniques could be implemented to reduce overfitting and improve the performance of the neural network model in the test phase, this could be considered for further research.