MSc in Data Science and Economics
Department of Economics, Management and Quantitative Methods
Università degli Studi di Milano

Cats and Dogs Image Classification

Università degli Studi di Milano
Machine learning, statisitical learning, deep learning and artificial intelligence
JAMAL HIND – 989432

a.y. 2021/2022

Index

1 Abstract……………………………………………………………………………………………4

2 Introduction…………………………………………………………………………………………4

2.1 Data pre-processing…………………………………………………………………………………4

3 Artificial Neural Network………………………………………………………………………6

3.1 Convolutional Neural Networks……………………………………………………………………7

3.2 MobileNetV2…………………………………………………………………………………………9

4 Models implementation…………………………………………………………………………10

4.1 VGG…………………………………………………………………………………………………11

4.1.1 VGG: one block…………………………………………………………………………………11

4.1.2 VGG: two blocks…………………………………………………………………………………11

4.1.3 VGG: three blocks………………………………………………………………………………12

4.1.4 VGG: three blocks + Dropout layer and Batch Normalization……………………………13

4.2 MobileNetV2…………………………………………………………………………………………14

5 Conclusion…………………………………………………………………………………………15

6 Bibliography…………………………………………………………………………………………15

1 Abstract

The main aim of this paper consists on the development of a model that can be used to classify, with the highest accuracy possible, using different models, pictures between two main categories: Dogs and Cats.
The Dogs vs. Cats dataset is a computer vision dataset that involves classifying photos as either containing a dog or a cat, and our basic task consists on the definition and resolution of the problems we will meet during the development of an algorithm classifying such images with the highest accuracy possible.

2 Introduction

The dataset can be downloaded from the address pointed in the bibliography, and it contains 25.000 images, with 12.500 images containing Cats and 12.500 images containing Dogs, in different formats, so not only real ones, but also drawn ones.
In this paper we attempt to create a network that can classify among these two categories, and in fact, this dataset was used as the basis for learning and practicing how to develop, evaluate, and use convolutional deep learning neural networks for image classification from scratch.
In the area of image recognition and classification, the most successful results were obtained using Artificial Neural Network, a technology that was introduced in 1943, that allowed us to classify billions of images by assigning them a label from a finite set. These networks form the basis for most deep learning models, a class of machine learning algorithms that use multiple layers containing nonlinear processing units.
Therefore, we can say that the main objective of this project consists on a binary classification task, with the possibility to automatically identify, by considering the smallest characteristics and discerning features, the image as a cat or a dog.

2.1 Data pre-processing

Once we have downloaded the Zip File and extracted the JPEG files, we simply need to upload the libraries that we will use to satisfy the main tasks of this project. Each image represents either a cat or a dog in a different situation or context.
The last step in preparing the input data was to create a pandas DataFrame in which each image is labeled according to whether it depicts a cat or a dog. By creating a DataFrame, we can easily keep track of the labels (which is 0 if the image is representing a cat and 1 if the image represented is a dog) and the corresponding file names for each image in the dataset. This also allows to easily split the data into training, validation, and testing sets.

Before anything else, what we did is to remove broken images, which are images in which neither of the two classes are being represented, and they are two. We are therefore left with 24.998 images, where 12.499 are images of cats and 12.499 are images of dogs.

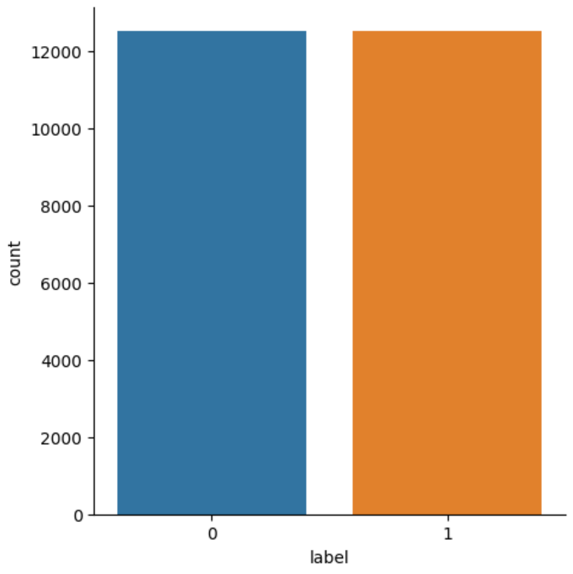


Figure 2.1. 12.499 images representing cats and identified with label 0;
and 12.499 images representing dogs and identified with label 1.

Given all the images, we randomly split our data between a training set, a test set and a validation set. The 80% of the dataset is kept as a training set for hyperparameter tuning, while the remaining 20% is the test set. In fact, while the validation set will be used to fit the model, the test set is going to be used for the 5-fold cross validation to estimate the risk and predict the label of the images, ones the hyperparameters will be tuned.
One important step, before the development of a model that can be used to classify with high accuracy the images, is Data Augmentation (or in our case Image Augmentation). Image augmentation consists in a method based on the application of different kinds of transformations techniques to the original images, resulting in multiple transformed copies of the same image, which are considered to be the one different from the other in certain aspects because of shifting, rotating and flipping techniques. Image augmentation reduces the chances of overfitting and it also helps improve generalization.
ImageDataGenerator is used to turn our images into batches of data arrays in memory, so that they can be fed to the network during training.
Moreover, to improve models' performances the pixels have been scaled down from their original scale [0, 255] to a [0, 1] range (dividing each value by 255).

The images reported in the file present different shapes, as we can see from figure 2.2, and this will greatly impact negatively our analysis. So, what we need to do is to convert all the images from JPEG to RGB and scale them down to 128x128 pixels, in order to obtain a dataset of images with the same dimensions.

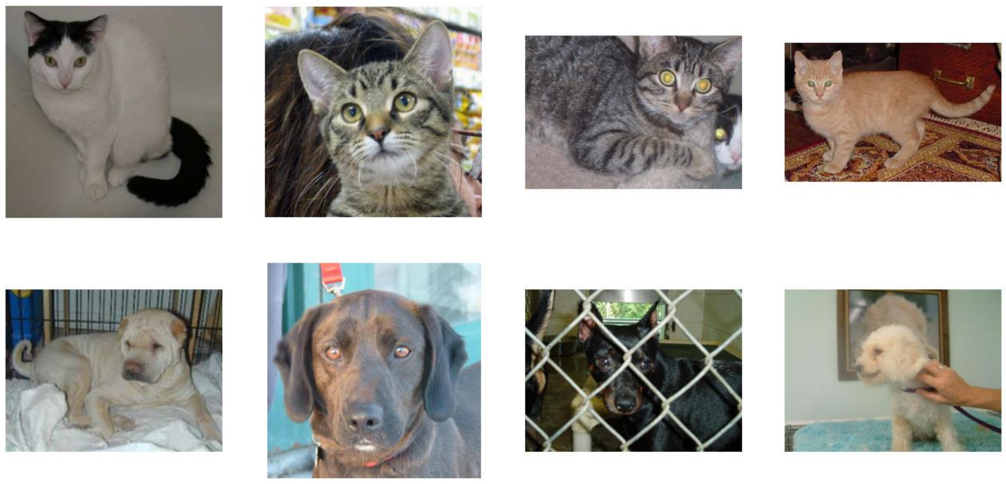


Figure 2.2. Cats and Dogs images randomly extracted from our dataset.

3 Artificial Neural Network

In the sector related to image classification and recognition, the most successful results were obtained by adopting Artificial Neural Networks.
In particular, Neural Networks are a large class of predictors based on the structure of the brain, which consists (oversimplifying) of some great and complex network of interconnected computing devices, the neurons, through which the brain can perform extremely complex computations.

For the purpose of implementing, training and testing the network described in this paper, we used the TensorFlow library, which is an open-source framework created by Google and which give us the possibility to perform numerical computations using data flow graphs. In particular, in a graph the nodes represent mathematical operations, while the graph edges represent the multidimensional data arrays, which are called tensors.
For this project, we start off with a small network and gradually increase the number of layers, so that we can evaluate and compare the changes in terms of performances and analyze the eventual overfitting that we get in the different models we present. Note that, to prevent overfitting a series of techniques, such as Dropout and Batch normalization, are used.
We will also develop and analyze the results obtained considering another famous architectures in the machine learning literature, which is MobileNetV2.

The main goal of the project is therefore to develop a system that can identify images of cats and dogs with the highest possible accuracy.

3.1 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) are a type of deep learning algorithm based on the concept of convolution, which is a mathematical operation used to extract features from an image. CNNs actually are similar to other neural networks, but since they have different convolutional layers, its complexity is greater than the one that we have of other neural networks.

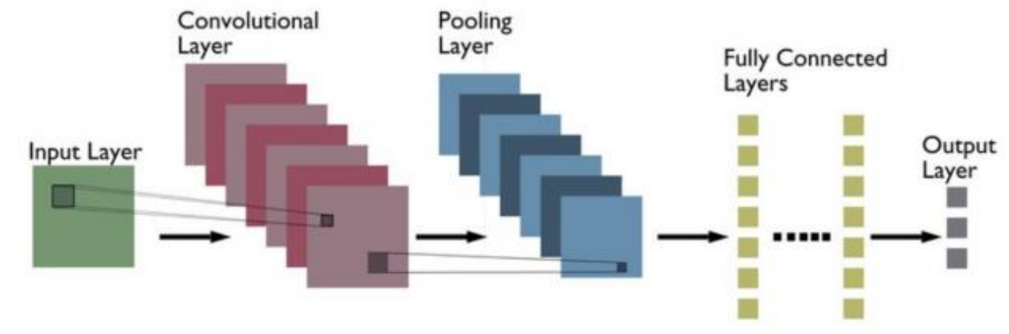


Figure 3.1. CNN typical architecture

As we can see from the figure above, the typical CNN architecture is characterized by the following different layers:

Convolutional layer: Convolutional layers are the first layer that we encounter when we use a CNN algorithm. This layer is extremely important and it perform the convolution operation on the input image, extracting relevant features.
In particular, this layer is based on the dot product between a set of filters (also called kernels) and the corresponding input region at each position, giving back as an output a feature map summarizing the activations of each filter across the input image. This process allows CNN to detect different types of patterns in the input image, such as edges, corners, and textures. Note that convolutional layers can be stacked to create more complex models, and as a result this can increase the number of intricate features that we can detect from images.

Pooling layer: Pooling layers are used after convolutional layers to reduce the size of the input, making it easier to process and use less memory. In other words, by reducing the spatial resolution of the feature maps, pooling layers make the network more robust to small variations in the input image while preserving at the same time the most important features. There are two main types of pooling: max pooling, which takes the maximum value from each feature map and average pooling, which takes the average value from each feature map. In this project I used Max pooling.

Flatten layer: An additional layer is the flatten layer, which is used in conjunction with a fully connected layers. In particular, this layer takes the output from the last convolutional or pooling layer, which is a 3D tensor of height, width and depth, and flattens it into a 1D vector. This allows the output to be reshaped and processed by a fully connected layer for final classification or prediction.

Fully connected layer: Fully connected layers are used at the end of a CNN, when the aim is to consider the features learned by the previous layers and use them to make some predictions or perform some classifications. Fully connected layers are also known as dense layers and in particular we have that each neuron in a dense layer is completely connected to every other neuron in the previous layer.
The number of neurons in a fully connected layer is typically chosen based on the complexity of the task and the size of the input data. In fact, in deep neural networks, fully connected layers can have millions or even billions of parameters, making them computationally expensive to train.

Dropout Layer: Dropout layers are a regularization technique placed after the Pooling layer and used to prevent overfitting. The basic idea behind dropout is to randomly ”drop out” (set to zero) some of the neurons in a layer, nullifying their contribution and leaving unmodified all the others. This forces the network to learn more robust features and reduces the likelihood of overfitting to the training data.

Batch normalization: Batch normalization layers are placed after the Convolutional layer and before the Pooling layer. This layer adopts a technique used to help in the coordination of the update of multiple layers in the model, by scaling the output of the layer, specifically by standardizing the activations of each input variable per mini-batch, such as the activations of a node from the previous layer (standardization refers to rescaling data to have a mean of zero and a standard deviation of one).

Activation Function. The sigmoid function is a smooth S-shaped curve that maps any input to a value between 0 and 1, making it useful for binary classification tasks. ReLU is a simple and widely used activation function that sets any negative input to zero, allowing the network to learn faster and avoid the vanishing gradient problem.

Optimizer. The optimizer is a key component of training neural networks that determines how the model’s parameters (weights and biases) are updated during the training process. The optimizer uses the gradient of the loss function with respect to the model’s parameters to update them in the direction that minimizes the loss. Adam (Adaptive Moment Estimation) is an adaptive optimization algorithm that is widely used for training deep neural networks. This adaptive learning rate ensures that the parameters are updated more effectively, leading to faster convergence and better performance. Adam is also robust to noisy or sparse gradients, making it an effective optimizer for a wide range of deep learning tasks. However, it may require more memory than other optimizers and can sometimes result in overfitting if the learning rate is not set appropriately.

Loss Function. The goal of the loss function is to measure the difference between the predicted output of the network and the true output, which is typically represented as a scalar value. In CNNs, the most commonly used loss functions include cross-entropy loss, mean squared error (MSE) loss, and binary cross-entropy loss. Binary cross-entropy loss is a widely used loss function for binary classification tasks in convolutional neural networks. It is a variation of cross-entropy loss and is designed to measure the difference between the predicted output and the true binary label, which is typically represented as either 0 or 1.

We can therefore conclude that Convolutional Neural Network (CNN) can recognize distinctive features or patterns from any position of an image like, in our case, cat ears or a dog nose. We experimented with different evolution of our base model to assess how different changes in a CNN architecture could affect our performance.

3.2 MobileNetV2

MobileNetV2 was introduced by Google in 2019 and is an evolution of a previous architecture called MobileNetV1. It consists of a convolutional neural network architecture that performs well on mobile devices.
As we can see from figure 3.2, its architecture is based on an inverted residual structure, where the residual connections are between the bottleneck layers. MobileNetV2 presents the following blocks: one residual block with stride of 1 and another block with stride of 2 for downsizing. For each block there are three layers: the first layer is 1×1 convolution with ReLU6; the second layer is the depth wise convolution, and the third layer is another 1×1 convolution but without any non-linearity.

The main purpose for adopting this type of architecture consists of the fact that this one reduces the memory usage and the computational expenses. To improve the accuracy, a customized head containing five different layers was defined to the base model of MobileNetV2, instead of the classification layer. This customized head contains an average pooling layer set to (7 × 7), a flatten layer, a dense layer, a dropout layer, and a softmax layer. Note that the flattened neurons were fed to a dense layer with the activation function being ReLU.

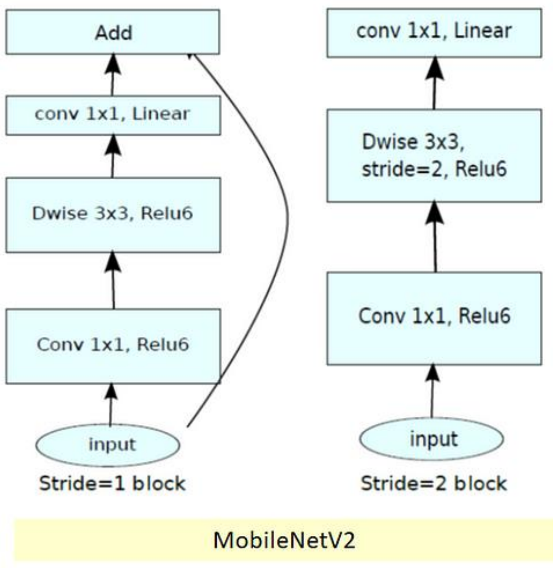


Figure 3.2. MobileNetV2 typical architecture

4 Models implementation

We tested multiple models, changing the number of layers and filters, in order to improve the accuracy of the model and decrease the loss that the model encountered. Note that during the fitting of each model, to keep the computational costs under control, we limited the number of training epochs to 10, which consists on the count of one forward and backward propagation of the training samples (based on the number of batch size that I selected during the Image Generation phase I derived the number of training examples that will be processes during each epoch, which in our case are 563).

Precisely, as we already said in the previous chapter, we set the activation function of every layer (except for the dense layer) to rectified linear unit (ReLU), while the dense layer, which outputs the prediction, was configured with a sigmoid activation function. We used the Adam optimizer.
Note that the hyperparameters of this layer are the number of nodes and the activation function.

The prediction is given by an estimation of the probability of an instance belonging to a certain class, where 1 stands for "this example is of class Dogs" and 0 for "this example is not of class Dogs ", which is a perfect binary classifier. If the probability falls below 0.5, sigmoid will assign it to one class (‘Cats), rather than the other (‘Dogs). Its linearity decreases the complexity of the network, resulting in faster computations. For these reasons, ReLU has become the standard activation function for the hidden layers of most neural networks.

4.1 VGG

The models we will implement take inspiration from the VGG architecture, proposed by Andrew Zisserman and Karen Simonyan in 2014. We tested different types of VGG networks, progressively increasing the number of layers, in order to defeat overfitting and increase accuracy.
The VGG (Visual Geometry Group) presents a structure made of blocks of two and three convolutional layers. Each pair or triplet of layers is then followed by a pooling layer, ending in some fully connected layers after flattening.
Although the exact same number of layers and parameters have not been replicated for computational costs, the models presented in this paper have a similar structure, based on blocks of convolutional layers. Four models have been developed and tested, each with a different composition of blocks and number of layers.

4.1.1 VGG: one block

In this first model we have just one block made by a pair of convolutional layers with 32 nodes each, followed by a maxpooling layer, a flatten layer and two Dense layers.

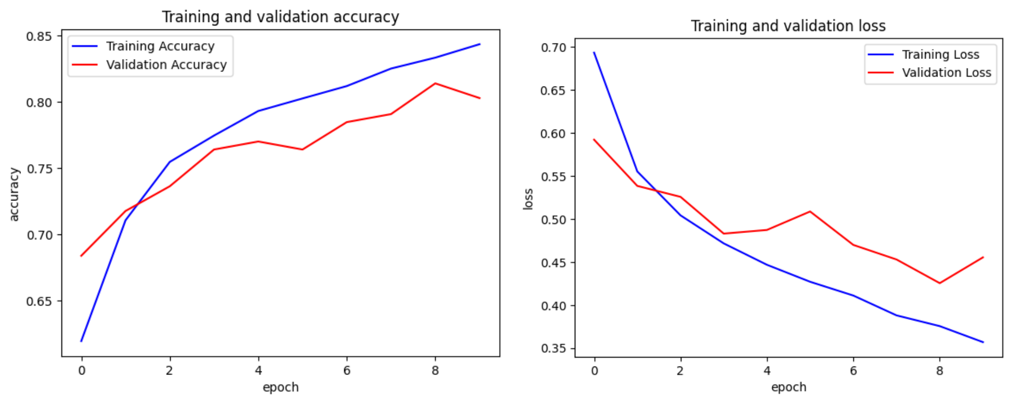


Figure 4.1. Training and validation accuracy and loss for one block VGG.

As we can see from the two graphs of this first model, the accuracy for the training and the validation set present a huge gap, leaving a higher degree of loss. In particular, the accuracy of this first model for the test set is just 81.26 %, with a percentage of loss of 0.44483. The zero-one loss computed is 0.1874.

4.1.2 VGG: two blocks

In this model we have two blocks, each one of them made by a pair of convolutional layers (with 32 nodes and 64 nodes) and each one of them followed by a max pooling layer; a flatten layer and two Dense layers. What we can see from the figure is that, by increasing the number of layers, as we expected, the accuracy is increasing and at the same time the loss is decreasing.

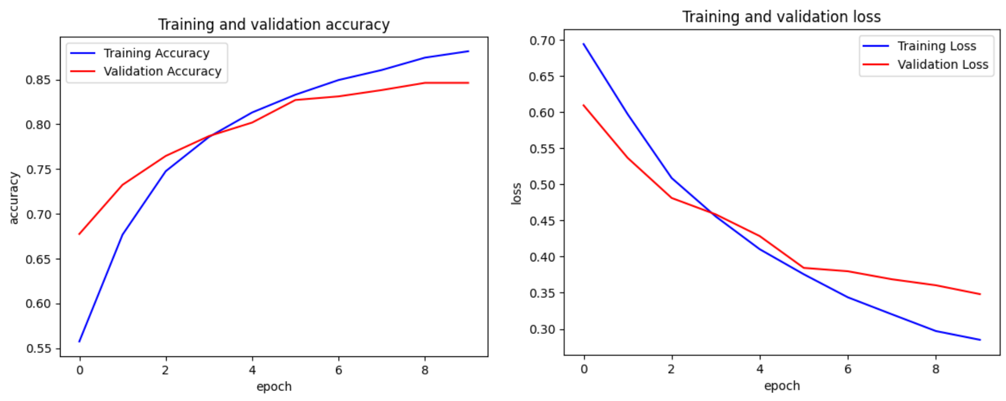


Figure 4.2. Training and validation accuracy and loss for two blocks VGG.

The accuracy computed for the test set is higher for this model, with respect to the previous one, with a value of 86.58 %. The zero-one loss decreased to 0.1342.

4.1.3 VGG: three blocks

This time we are considering three blocks, increasing the filters from 32 to 64 to 128, each one of them with a maxpooling layer, a flatten layer and two dense layer. Once again the accuracy is increasing and the loss is decreasing, and also the gap between the two training and validation accuracy and loss is decreasing (especially in epoch 6), avoiding overfitting.

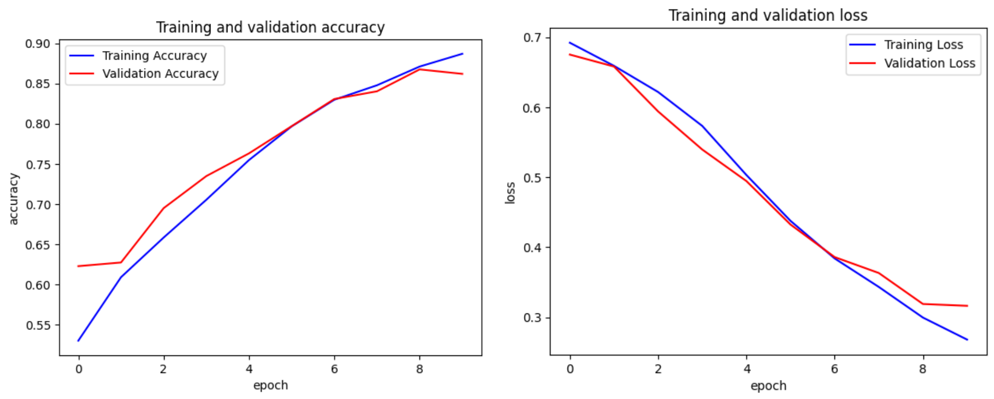


Figure 4.3. Training and validation accuracy and loss for three blocks VGG.

The accuracy of the test set is 86.74 %, with a zero-one of 0.1326. Even if the percentage of the accuracy is improving and gradually increasing, we are still in a situation in which we do not have an ideal value, and in fact, as we can see from figure 4.4, the number of images that are mistakenly put in the other class is still high. We will therefore add a Dropout layer and a Batch Normalization layer, that, as we said before, are used to avoid overfitting and to increase the predicting accuracy of our model on the test set.

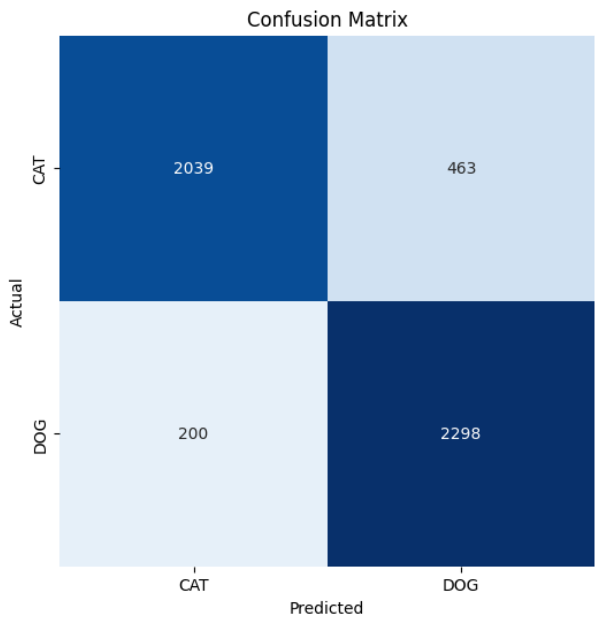


Figure 4.4. Confusion Matrix of model 3.

4.1.4 VGG: three blocks + Dropout layer + Batch Normalization

In this last model we are keeping the same number of layers that we had before, adding a dropout layer and a Batch Normalization layer.

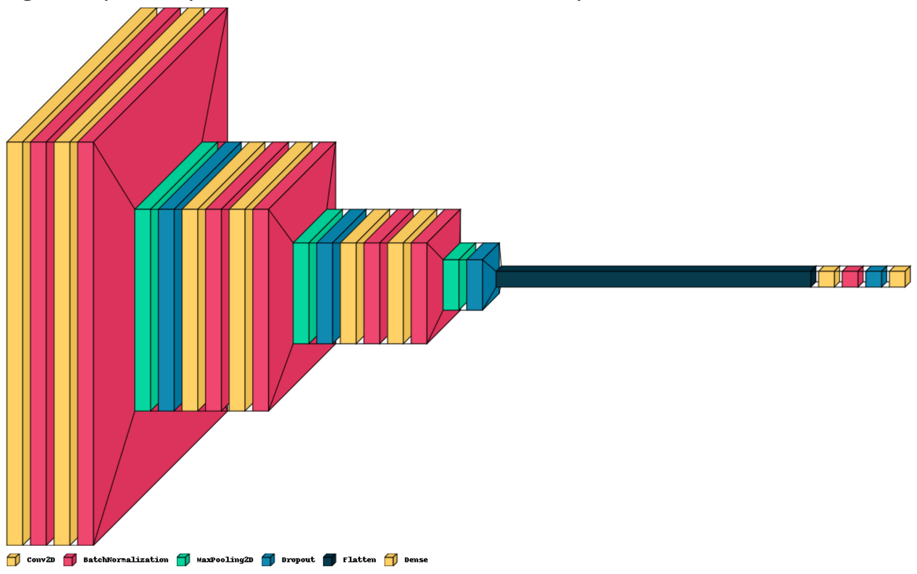


Figure 4.5. model’s architecture.

This model presents a higher accuracy for the test set (90.14%) and a lower zero-one loss (0.0986), but even in such a situation, observing the graphs, we can easily see that there is a certain gap between the training and the validation accuracy and loss, especially in epoch 8.

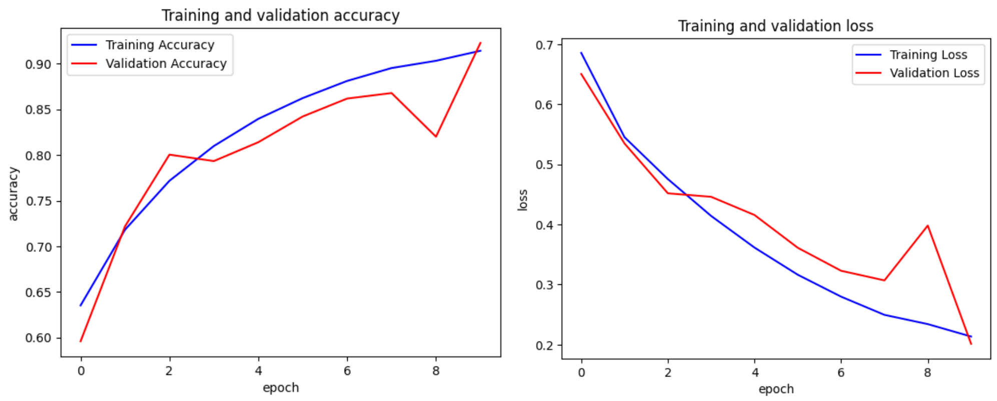


Figure 4.5. Training and validation accuracy and loss for three blocks VGG + Dropout layer and Batch Normalization.

4.2 MobileVNet2

We implemented a model based on MobileVNet2. As we can see we have a large number of layers, and as a result the training and validation accuracy is high, as the number of epochs is increasing, defeating overfitting.

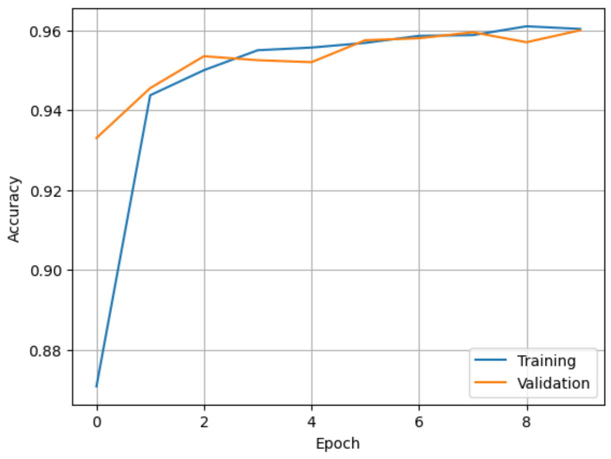


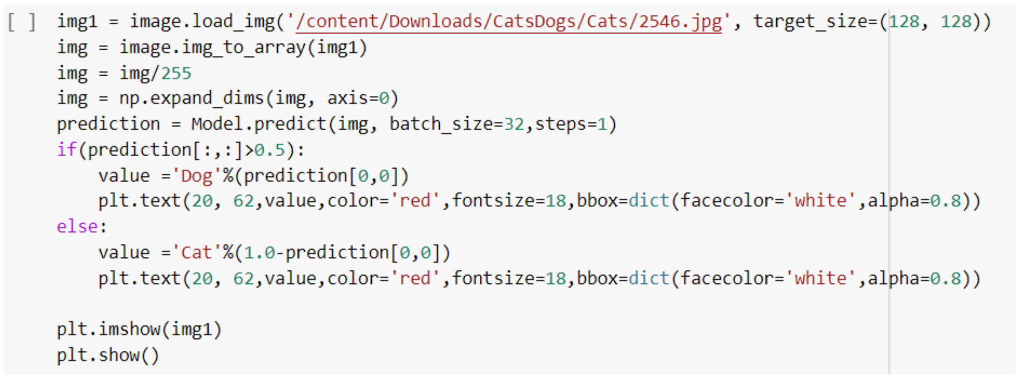
Figure 4.5. Training and validation accuracy for a MobileVNet2 model.

This is the best optimal model, that gives us back a training accuracy of 96 %. This model, which presents a zero-one loss value of 0.0322 (the lowest), is being trained one last time (with the best hyperparameters) using 5-Fold Cross Validation. Once we divide our data into 5 fold, we will perform 5 iteration, and for each one of them we will treat one fold as a testing dataset and the remaining one as training dataset. At the end we obtained that the average value for accuracy was 96.71 % while the average value for the zero-one loss was 3.288 %.

5 Conclusion

Several different CNN architectures were tested to determine which one performed the best on the input data. The best one, which is the one with the highest accuracy and the lowest loss, is MobileNetV2, and we performed on it a cross-validation test to compute the risk estimate.

We then verified that our model would correctly classify the images of our dataset by simply changing the path of the image that need to be loaded, and as we expected, since we have a high accuracy, the images were correctly classified in the majority of the cases.



```
[ ] img1 = image.load_img('/content/Downloads/CatsDogs/Cats/2546.jpg', target_size=(128, 128))
    img = image.img_to_array(img1)
    img = img/255
    img = np.expand_dims(img, axis=0)
    prediction = Model.predict(img, batch_size=32,steps=1)
    if(prediction[:,:]>0.5):
        value ='Dog'%(prediction[0,0])
        plt.text(20, 62,value,color='red',fontsize=18,bbox=dict(facecolor='white',alpha=0.8))
    else:
        value ='Cat'%(1.0-prediction[0,0])
        plt.text(20, 62,value,color='red',fontsize=18,bbox=dict(facecolor='white',alpha=0.8))

    plt.imshow(img1)
    plt.show()
```

Figure 5.1.Classification of the images using the implemented MobileNetV2 model.

6 Bibliography

Dogs.vs.cats. https://unimibox.unimi.it/index.php/s/eNGYGSYmqynNMqF/download

Different types of CNN explanations https://vitalflux.com/different-types-of-cnn-architectures-explained-examples/

MobileNetV2 model https://towardsdatascience.com/review-mobilenetv2-light-weight-model-image-classification-8febb490e61c