Statistical methods for machine learning project report

Laaroussi Mohcen

Abstract

In this project different solutions are analysed, implemented and compared between each other with the aim of finding the best one to the problem of image classification for cats and dogs images. I started from classifying these images with a simple neural network, going up to more complex convolutional neural networks such as VGG16 and ResNet50. I used the TensorFlow API to build and train the models and 5 fold cross validation to evaluate them. I obtained good results from all three models.

1 Introduction

This project analyses and compares different solutions to the problem of classifying images of cats and dogs. In the data processing phase this dataset is analyzed and prepared for training, validation and testing. The proposed solutions are structured with the aim of solving the problem starting from the basics. I started by training simple models, and step by step trying to improve their performance documenting the changes made and their effects on the overall predicting performance.

To further evaluate the algorithms, 5 fold cross validation was done computing risk estimates based on zero one loss.

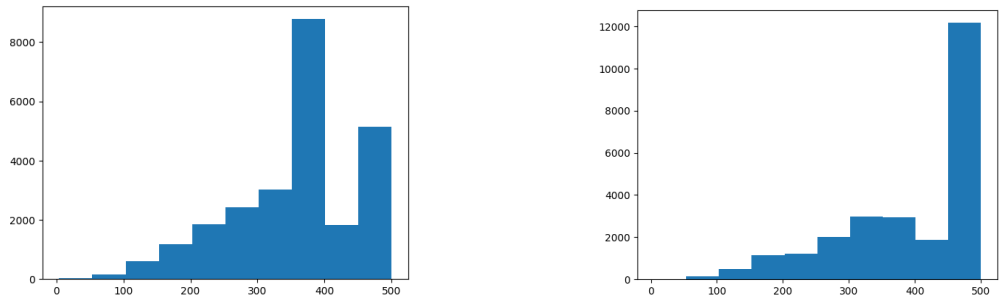
To do so I used TensorFlow APIs both to manage the data preprocessing, and to create and train all the models.

2 Data analysis

The dataset provided consists of 25000 labeled images of cats and dogs, divided exactly in half. In order to import the dataset I uploaded it on Kaggle, to be able to download it into the project using APIs [1] with my credentials. After the dataset had been downloaded, I analyzed it, and found that the images have different color modes and different resolutions: Most images are RGB, which is a common color mode that uses 3 channels to represent color Red-Green-Blue, but there also grayscale images with mode "L", and other images with color with mode "P", in which colors are stored in one channel which takes 256 values that are used to map each value to a color. These images were all converted to "RGB" (distribution of resolutions), and the corrupted ones removed. As shown in pictures (1a) and (1b), more than 75% of images measure more than 300 pixels in height and more than 80% 300 pixels in width.

Finally, the few corrupted images and the ones that i found weren't about neither cats or dogs (example) were removed from the directory. This resulted in a final dataset of 12993 colored images ready to be used for the classification task

[1] https://github.com/Kaggle/kaggle-api



(a) Widths (b) Heights

Figure 1: Distribution of images resolutions

2.1 Image preprocessing

The result of the previous phase is a *pandas* dataframe with a column for the image path and the other one for the label. An additional check was made on the data showed that there is no imbalance between the two classes that could impact the performance of the learning algorithms. In order to split this dataframe of labeled images, I generated three datasets for train, validation and test sets: I used a split ratio of 0.8, 0.1, 0.1 respectively to generate three separate dataframes, that will be used to generate the datasets for training, validate and test. Keras *ImageGenerator* class allows to easly apply tranformations to images during the training phase. The *flow_from_dataframe* function applies these transformations from the images given as input from the three dataframes previously defined. The function allows to choose the size of the images in order to find the best trade-off between result accuracy and training time: in fact this resize operation is necessary since all the images have different resolutions. I tried multiple image sizes, and what was evident, is that there was no particular difference between them. Keeping that in mind, I decided to use an image resolution of 200x200 pixels for the subsequent experiments. In the data generation phase, images were also normalized: each pixel was normalized between 0 and 1: this is useful to make computation more efficient, and because dealing with a binary classification task and the output of the neural network will be between 0 and 1. The last choice to make was about which color mode of the images to use: I chose to grayscale all images since I deemed that color doesn't have much relevance in classifying cats and dogs. Furthermore, using grayscale images can simplify network design and reduce computational requirements during the training phase.

3 Development process

To classify these images I have trained multiple models and different architectures trying to find the optimal ones one and documenting the changes made and the effects they had. These models are:

- Single and multi layer Perceptron artificial neural network
- Convolutional neural network
- VGG16
- ResNet50

3.1 Single and multi layer Perceptron artificial neural network

The first approach was to start from training a single layer neural network, and then adding few more dense layers.

In the first experiment the newtork has 200x200=40000 pixels as inputs of the 1 layer network, and 1 output neuron. Since it's a binary classification problem, the activation function used is the *sigmoid* function: it takes any real value as input and outputs values in the range 0 to 1. The greater

the input, the closer the output value will be to 1, and it will 0 the smaller the input is.

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

The loss function I chose is the binary cross entropy loss, that works well with this classification problem; it measures the dissimilarity between the predicted probability distribution and the true distribution of the binary labels and penalizes the model a lot more when it predicts a high probability for the incorrect class. The object of the model is to minimize this loss by using optimizer algorithms. For all my experiments i used Adam optimization algorithm, which is an easy to implement and efficient extension to stochastic gradient descent.

$$-(y_{true} * log(y_{pred}) + (1 - y_{true}) + log(1 - y_{pred})) \tag{2}$$

I trained the model on the training set previously obtained, using a batch size of 32. I started with a high learning rate that decays during training. This is done because an initial large learning rate helps the network escape any spurious local minima, and the decaying helps the network converge to a local minimum and avoid oscillation. Finally I evaluated it using accuracy and loss as metrics.

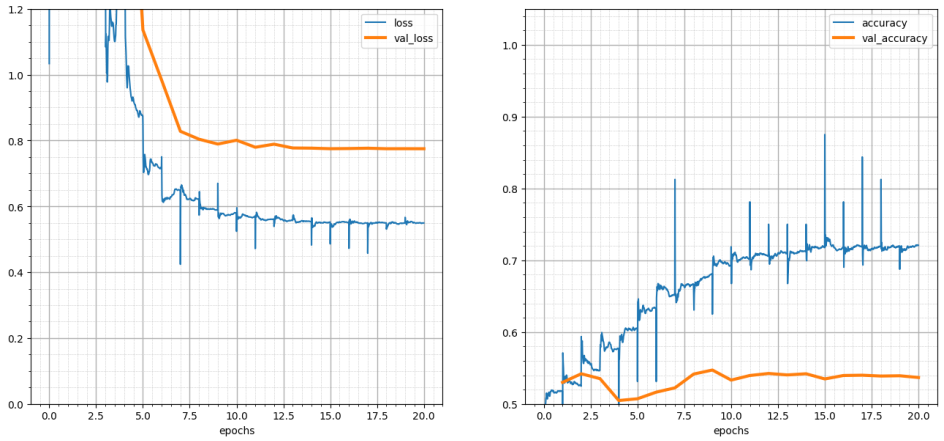


Figure 2: Training and validation loss, training and validation accuracy during the training of a single layer neural network

As shown in Figure 2, the network with the first thing noticeable is the great amount of overfitting, with a value of 0.70 and 0.56 of training accuracy and loss respectively, compared to 0.54 and 0.76 of validation accuracy and loss. The network seems to struggle to learn features from these complex images with little generalization. The first step to solve both issues is to add layers to the network: This helps the network to learn more complex representations, thus improving the model's ability to generalize and potentially reduce overfitting. I tried with adding 2 hidden layers of 200 and 60 neurons respectively, with *Relu* functions as activation of the neurons. The results (Figure 3) show a slight improvement both in terms of validation accuracy and loss and overfitting. However the predicting performance is still bad, with a high value of loss. The problema is that neural networks built like that can't learn information about shapes in images, thus making the transition to Convolutional neural networks essential.

The issue of overfitting will be a recurrent theme in the next chapters, and more solutions will be proposed to solve it.

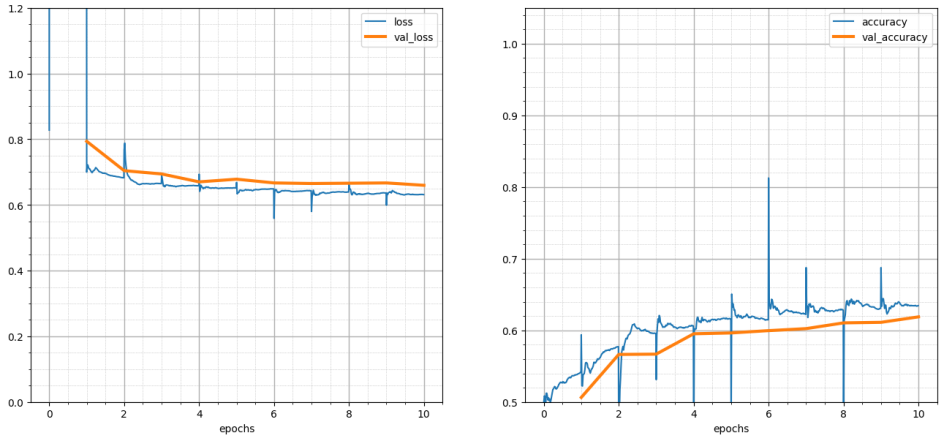


Figure 3: Training and validation loss, training and validation accuracy during the training of a multi layer neural network

3.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a type of neural network architecture for processing grid-like data, such as images [1]. They are very popular in solving various computer vision tasks, including image classification, object detection and image segmentation. They are designed to exploit the spatial structure of images and they do it by incorporating different types of layers: convolutional layers, pooling layers, and fully connected layers.

In the first trial with this type of neural network I implemented the model with 2 convolutional layers and a dense layer. The convolutional layers always have a kernel size of 3x3, and ReLU activation function, whereas they might have different number of filters. In this case 16 for the first and 32 for the second. Between each convolutional layer there is a max pooling layer, with a sliding windows that applied a *max* operation to reduce the size. Finally a flatten layer changes the shape of data into an array suitable for the last dense layer with 128 neurons and an output layer of 1 neuron, with the hyperparameters being the same as the neural network described in the previous section.

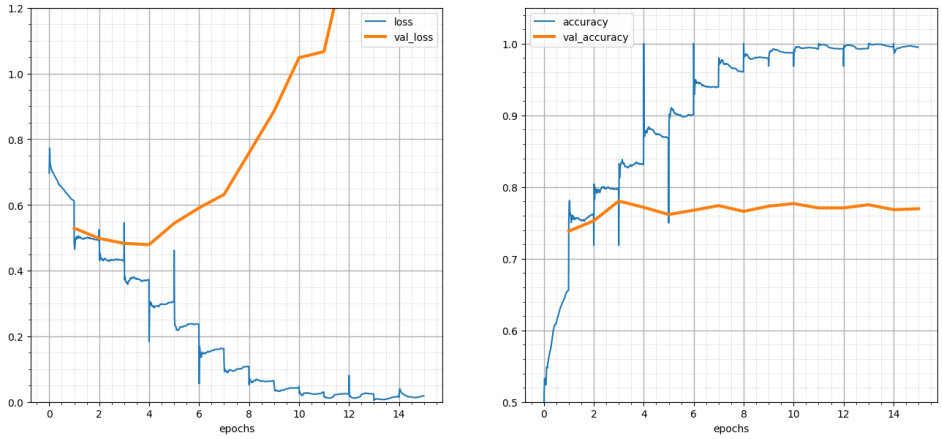


Figure 4: Training and validation loss, training and validation accuracy during the training of simple CNN

This configuration of CNN shows how this type of networks has the potential to learn more about images, but still needs more adjustments because it can easily be seen that overfitting is still a big factor.

3.3 Overfitting

To mitigate overfitting i applied different techniques: early stopping, image augmentation, dropout regularization and batch normalization.

With early stopping, the network monitors the performance on the validation set and stops the training process before it deteriorates or reaches a plateau. I started with a high number of epochs, and the network stops the training process based on early stopping rules monitoring the validation accuracy.

The second technique used in this project to mitigate overfitting is *batch normalization*. Through a layer, it normalizes the activation of each layer within the network, introducing a form of regularization, and allowing for a more stable training process. I used this technique by adding a batch normalization layer after every convolutional layer.

The other possibility is to simplify the model by introducing dropout: in this technique random fractions of neurons are switched off during training. This helps prevent the network from relying too much on specific neurons and improves generalization. It can be applied to both fully connected and convolutional layers. In the latter, the dropout layers sets entire channels or feature maps to zero during training. The number of neuron to shut down is chosen based on a percentage given to the layer. For instance, 0.5 indicates that half of the neurons are to be shut down. I tried many different CNN architectures, combined with dropout layers of different rates. In fact, choosing the best dropout rates proved to be crucial because dropout rate set too high may lead to underfitting, while setting them too low may not provide sufficient regularization. For example I started with adding a dropout layer of 0.5 after every convolutional and dense layers, and the results show an improvement in terms of overfitting, but with a decrease in validation accuracy and loss, compared to what it was without dropout layers. This suggests that this level of dropout is quite aggressive, and if applied to every layer, it results in the network losing a lot of information, and that it struggles to lean meaningful representations. The next steps consists of gradually lowering the dropout rates and observe the impact on the training and validation metrics.

I did this experiments on a CNN with 5 convolutional layers with different number of filters (32,64,64,128,256), a dense layer, and the hyparameters beingh the same as the example before. I arrived at this structure starting from the base one described in the previous section,and step by step making it more and more complex.

The last technique implemented is image augmentation: it aims to increase the size and diversity of the training dataset, by applying transformations to the images, such as rotation, horizontal flip, zoom range, shear range, width and height shift. It helps improving the performance of the model on unseen data and improves generalization, thus helping in reducing overfitting. The drawback of it is the increased training time and additional computational resources required, due to the increased variability in the training data.

By training the model previously set up with augmented images, I noticed a reduction in training and validation accuracy, as well as the loss. To try to solve this I readjusted the dropout rates to adapt the network and make it able to learn meaningful representations of images.

The results (Figure 5) show a training accuracy and loss of 0.93 and 0.16 and a validation accuracy and loss of 0.93 and 0.16. However a better evaluation needs to be made. By using cross validation a better evaluation of these changes can be done.

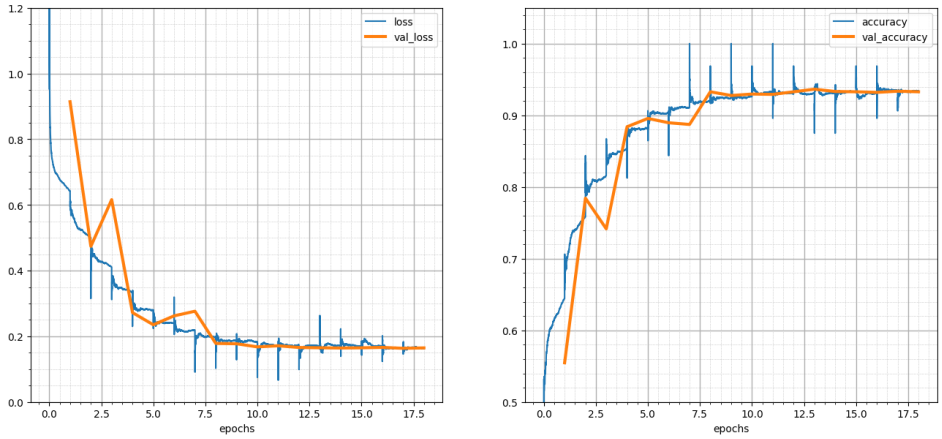


Figure 5: pp

3.3.1 Evaluation

A first evaluation of the model was done on the test set previously generated: I obtained and accuracy of 0.94 and loss of 0.13. Therefore, the model has good generalization performance and performs well on new unseen data.

I also evaluated the model using cross validation, specifically 5 fold cross validation: it's a technique used to assess the performance and the generalization ability of a model. It involves splitting the dataset into 5 different folds, training the model on a combination of 4 folds, and evaluating the performance using the remaining one. This process is repeated 5 times, with each fold serving as validation set in each iteration. For evaluating the risk estimates I used accuracy and zero-one loss, which measures the error, or misclassification rate, commonly used in binary classification tasks.

$$l(y, \widehat{y}) = \begin{cases} 0, & \text{if } y{=}\widehat{y} \\ 1, & \text{otherwise} \end{cases} \tag{3}$$

Therefore, the sum of the loss over the validation set is the number of miscalssified images.

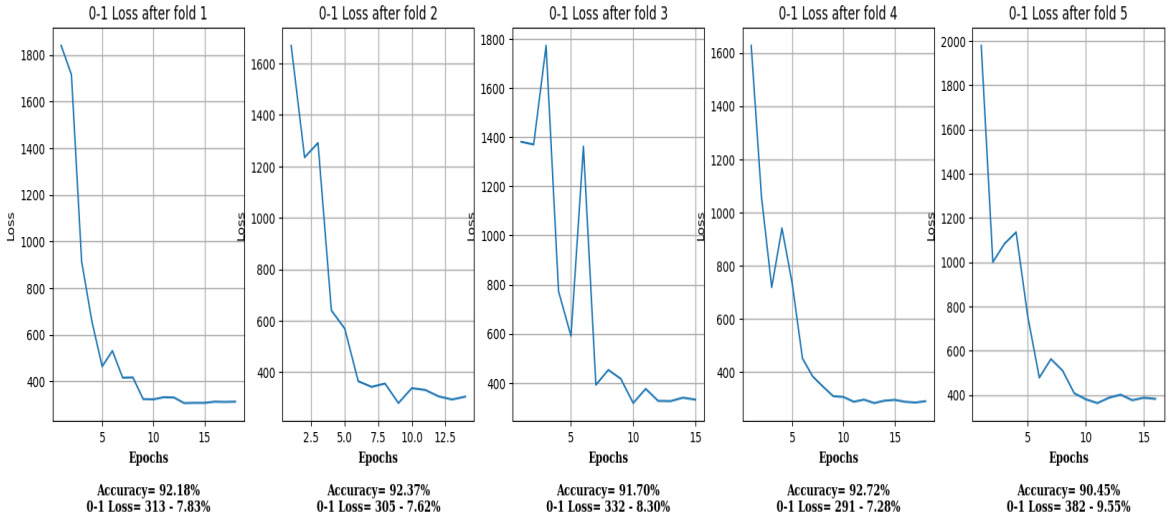


Figure 6: 5 fold cross validation results

The average estimates are: 91.88% of accuracy and 8.11% of zero one loss.

3.4 VGG16

I wanted to compare multiple types of architechtures [2], and the first one i tried other than classic CNN is VGG16: it's an architecture of 13 convolutional layers to a fixed filter size of 3x3, 5 max-pooling layers, and 3 fully connected layers. It's a large network with about 138 million parameters, and its aim is to create a very deep neural network capable of improving the ability to learn hidden features. The original VGG16 model was trained on the ImageNet [2] dataset, where images have a fized size of (224x224x3).

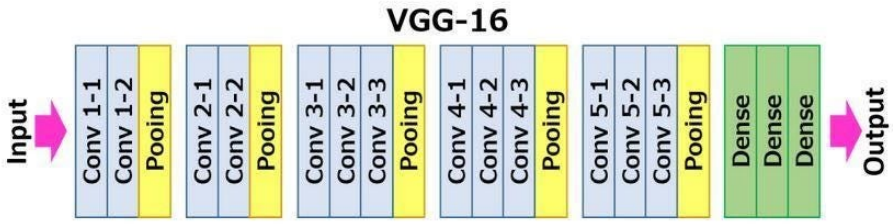


Figure 7: VGG16 structure

I chose a pre-trained VGG16 convolutional neural network [3], and substituted the last dense layer with a trainable one with 512 neurons, with ReLu activation function, and a single output neuron with a sigmoid activation function. I trained the model with new training and validation datasets, composed by augmented RGB images, given that VGG16 is designed to work with colored images with three color channels (red, green, blue).

The results (Figure 8) show an overall performance improvement with 0.08 of loss and 0.96 of accuracy.

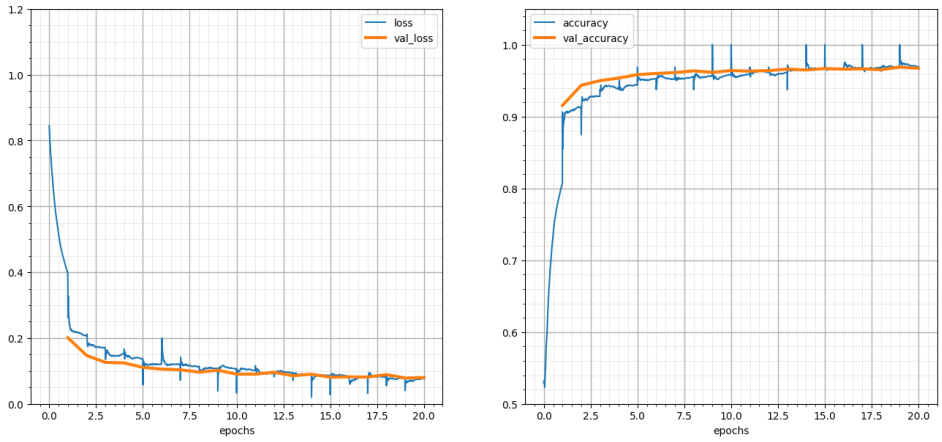


Figure 8: Training and validation loss, training and validation accuracy during the training of VGG-16

3.4.1 Evaluation

I used the set composed of RGB images to evaluate the model, and I obtained a validation accuracy of 0.97 and loss of 0.06, and a training accuracy of 0.97 and 0.07 of loss.

I also performed cross validation with 5 folds, and calculated the risk estimates:

[2] https://www.image-net.org/

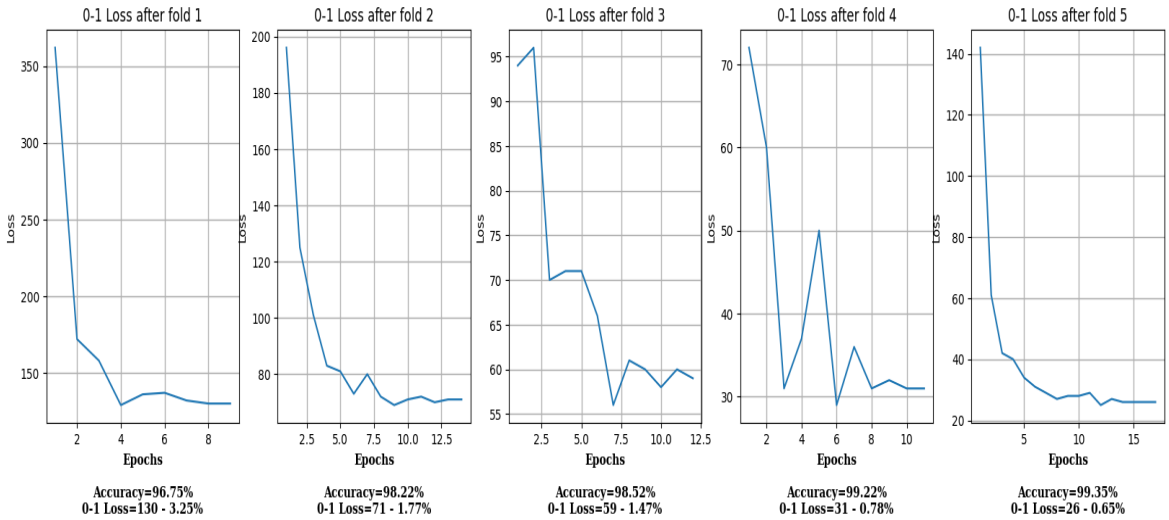


Figure 9: 5 fold cross validation results

The average estimates are: 98.41% of accuracy and 1.58% of zero one loss.

3.5 ResNet50

ResNet50 is a convolutional neural network [4] with 50 layers, consisting of 48 convolutional layers, 1 max pooling layers and 1 average pooling layer. It has around 26 millions parameters, and like for VGG-16, it was trained on the imagenet dataset, and works with RGB images of size 224x244 pixels. I trained the model with the same training and validation datasets used for training VGG16.

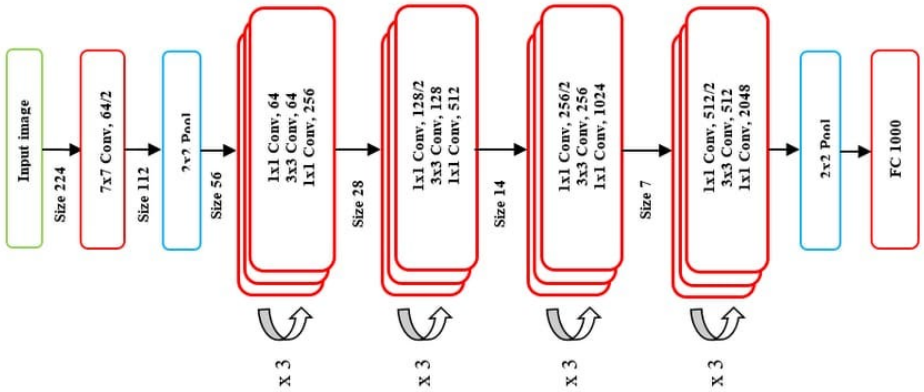


Figure 10: ResNet50 structure

I trained and validated the network with RGB images and the results (Figure 11) show a training loss and accuracy of 0.58% and 99.82%, and a validation loss and accuracy of 3,5% and 99,20%.

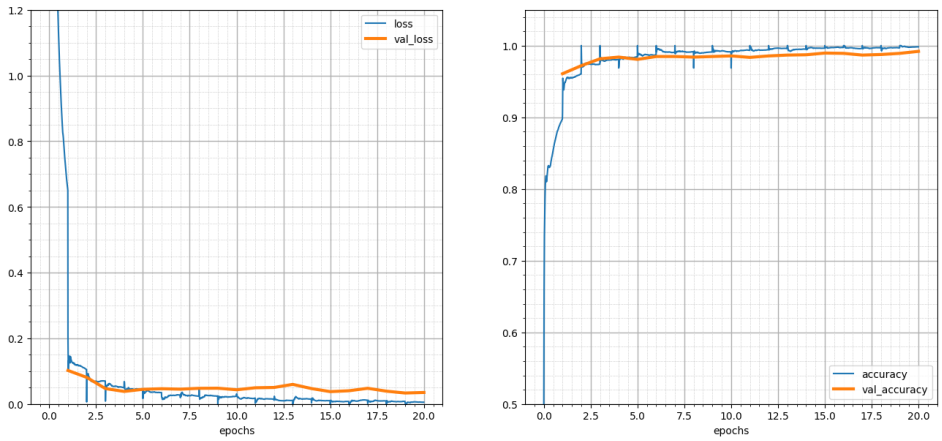


Figure 11: Training and validation loss, training and validation accuracy during the training of ResNet50

3.5.1 Evaluation

I evaluated this model with the same methods explained before: I tested the network on the test set and I obtained a loss of 4.70% and accuracy of 98.83%.

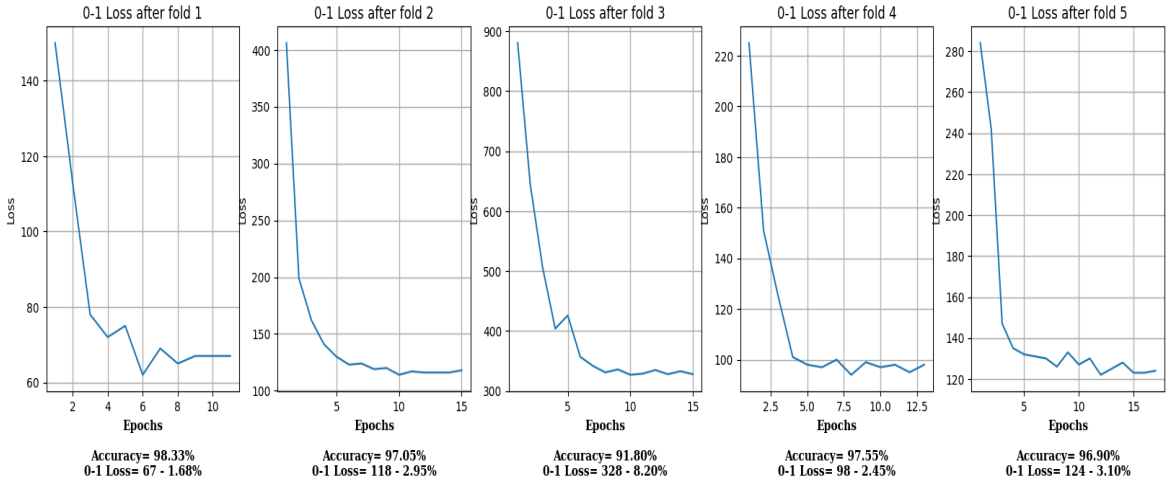


Figure 12: 5 fold cross validation results

Finally, performing cross validation showed that with 5 folds the average estimates obtained are: accuracy of 96.32% and zero one loss of 3,67%.

4 Conclusion

In conclusion, I have shown that there are various models to classify cats and dogs images with very good performance. Especially using convolutional neural network in different forms and variations.

Model	Train	Validation	Test
CNN	93%	93%	94%
VGG16	96%	96%	97%
ResNet50	99%	99%	98%

References

[1] Jianxin Wu. Introduction to convolutional neural networks. 2017.

[2] Kitsuchart Pasupa Supawit Vatathanavaro, Suchat Tungjitnob. White blood cell classification: A comparison between vgg-16 and resnet-50 models. 2018.

[3] Andrew Zisserman Karen Simonyan. Very deep convolutional networks for large-scale image recognition. 2014.

[4] Shaoqing Ren Jian Sun Kaiming He, Xiangyu Zhang. Deep residual learning for image recognition. 2016.

A Online resources

The sources for this project are available via GitHub