Chihuahuas vs Muffins

Matteo Tarenzi

April 16, 2024

Abstract

This project explores the application of machine learning in image classification. The goal was to develop and examine a classifier that could accurately distinguish between classes. For this purpose three neural network architectures have been considered. For each of them, the expected minimal risk has been estimated through a variation of the nested cross validation and, lastly, the resulting three best models have been commented.

Introduction

The project originated from an Internet meme. The joke revolves around the fact that photos of chihuahuas are, humorously, very similar to those of muffins. Shrinking the images of the little dogs could indeed create a risk of confusion when mixed with images of chocolate sweets. This similarity becomes a great opportunity for assessing the capabilities of machine learning models. The question is how well a neural network model can classify images of chihuahuas and muffins without errors. The following section will present the dataset used for the project, along with a sample of the adopted images. After that, the methodology employed for estimating the expected minimal risk will be explained. The third chapter, instead, will introduce the architectures of the three neural networks considered in the project. Finally, the results obtained from the final models will be summarized and interpreted in detail.

The Dataset

The dataset has been obtained from *Kaggle*[1]. It was already divided into two folders: a training one and a test one. Both contained images of the two classes object of the classification. However, not all of them were in the desired format. The first step has been finding those with the wrong color model and convert them to RGB.

The figure below shows a sample extracted from the training folder. Images are pretty different one by another. Some display the chihuahua or the muffin as the main subject. Others instead, involve a more complex environment like a dog printed over an object or a photo taken from a recipe. This variability has played a significant impact on the design, the training and the performance of the models.

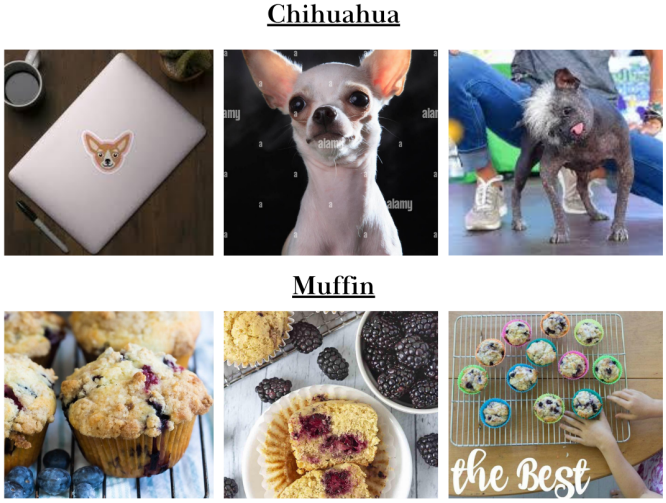

Figure 1: Dataset sample

[1] https://www.kaggle.com/datasets/samuelcortinhas/muffin-vs-chihuahua-image-classification

Methodology

The study involves the development and the evaluation of different models. Four architectures have been considered:

1. Neural Network using image resolution of 32x32;
2. Convolutional Neural Network using image resolution of 32x32;
3. Convolutional Neural Network using image resolution of 64x64;
4. VGG Deep Neural Network using image resolution of 224x224.

The first three models have been created from scratch using *Keras* and then trained over a portion of the dataset. VGG, instead, is a pre-trained model. It uses a very deep Convolutional Neural Network with a lot of layers. Training a model like that would have been too demanding for the project. It has been included in the study only for comparison with the performance of the other models.

This is the reason why the first three were the real subjects of the study. In particular, the objective was to estimate their relative expected risk and compare the performance of the best possible models. These models were obtained using hyperparameters tuning. Therefore, since the estimation required also this tuning process, a Nested Cross Validation would have been the best option to determine the risk. However, this choice seemed too computationally expensive. Consequently, the project has implemented a variation to that process. From the original dataset, a section of the data inside the test folder has been merged to the one inside the training folder. This has created a bigger set for the estimation of the expected minimal risk. The remaining part of the test folder has been used instead for the performance evaluation of the final best models. The variation to NCV is presented now.
The outer loop utilizes a five-folds partition the same way as the Nested Cross Validation. The inner loop, instead, performs a train-validation procedure rather than running a Cross Validation for each combination of hyperparameters. Another change has been introduced in the inner loop. While the original version is performing a grid-search of all combinations, the variation optimizer exploits a Bayesian approach. Once the user has set the number of trials, the optimizer evaluates only those combinations that it predicts will perform better, thereby increasing the efficiency.

These two changes serve the same purpose: reduce the time required by the tuning process, while trying to maintain the search space as big as possible. Unfortunately, there are some downsides. The elimination of a cross validation in the inner loop affects the confidence over the performance during the optimization, since now the tuning is more dependent on the sample. The second problem is about the comparison between different architectures. Since the Bayesian optimizer is used, if the number of trials is not sufficiently high, different folds and consequently different models may not experience the same combination of hyperparameters. This decrease again the robustness of the method. The downsides, however, were both necessary for shortening the time necessary for the estimation of the expected minimal risk. The computational power and the resources given by the nature of the project were, sadly, limited . The Nested Cross Validation estimate would have been a more robust method but would have required shrinking the search space, with a considerable reduction of models performance.

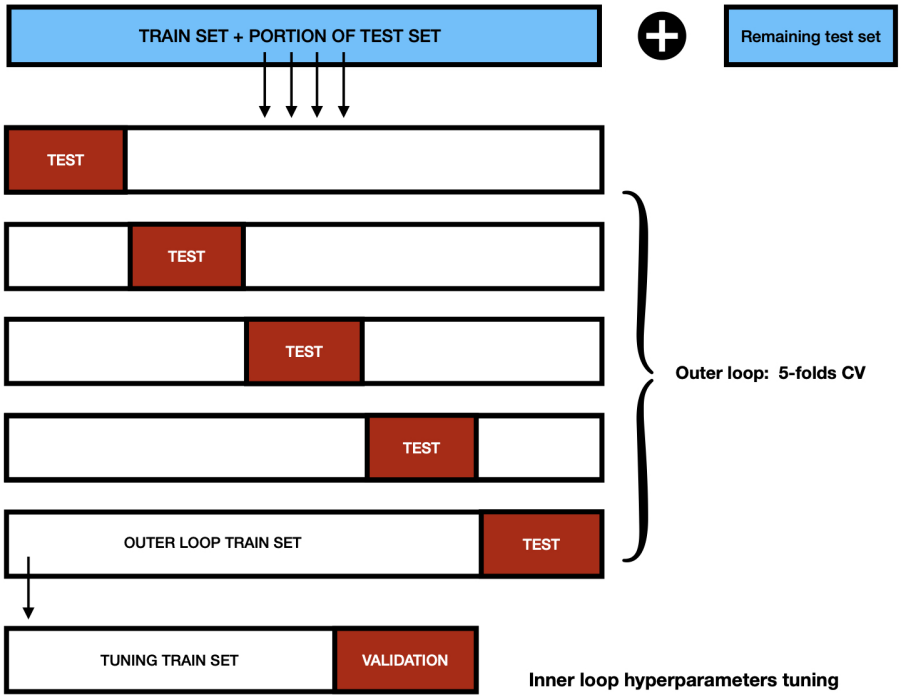

Figure 2: Estimation process

The estimation workflow can be described by the **Figure 2**.
First, data used for the estimation is split into five folds for the outer loop. Taking one fold beside at each time, data is divided into train and validation sets. After that, using the Bayesian optimizer, a hyperparameters tuning process is performed; the number of trials is set by the user based on the architecture. Once that the optimization finishes, using the best hyperparameters recorded, a new model is trained over the union of train and validation sets. Its performance is evaluated using the test fold in the outer loop, and the rescaled zero-one loss is stored. Once that the outer loop is completed, the expected minimal risk is calculated as the mean of the zero-one losses.

One additional step is performed to obtain the desired statistics for the comparison of the three architectures. Since the performance of models trained in the outer loop has been recorded, it is possible to select the hyperparameters associated with the lowest zero-one loss. A final model is then trained over the whole dataset used for the estimation process. These final models can be evaluated using the remaining data inside the test folder that hasn't been already used. Differently to the first three, the VGG has been directly fine-tuned over the set that was used before for the estimation but its performance was evaluated as the other final models. Once that accuracies and others statistics have been stored, the comparison can finally take place.

Estimation

In this section, the four different architectures will be presented in more detail. As stated before, the first three have been created and trained from scratch. Only for them, they have been performed a hyperparameters optimization and the expected minimal risk estimation. The last model is a pre-trained classifier called VGG. It has been created and trained by Oxford University researchers over a big dataset. The study of this model has not involved the estimation of the expected minimal risk. For all them, the main structure will be commented and the main differences pointed out. Furthermore, the hyperparameters search results and the characteristics of the final models will be presented. All the training procedures utilize the same learning rate decay. The rate decrease constantly, defined by the function:

$$Stlr - epoch*(Stlr/60)$$

where '*Stlr*' is the starting learning rate and '*epoch*' is the current epoch.

First model: NN with (32x32)

The input uses an image resolution of 32x32. After the rescaling layer, they have been introduced two layers for data augmentation: one horizontal flip and one 0.1 rotation. Then, the model includes five dense layers with a number of neurons that increases with the depth of the network. Between each of these, there are Dropouts layers with a 0.2 value. Finally, the output is made up by two neurons along with the *softmax* activation function.

The hyperparameters search space for the first structure is:

- **neurons** (first dense layer) : 160, 220, 280, 340, 400;
- **neurons'** (second to last dense layer): 12, 16, 20;
- **activation** (activation function): 'ReLU', 'tanh';
- **Stlr** (starting learning rate): 0.001, 0.0005.

Number of trials per tuning: 20.

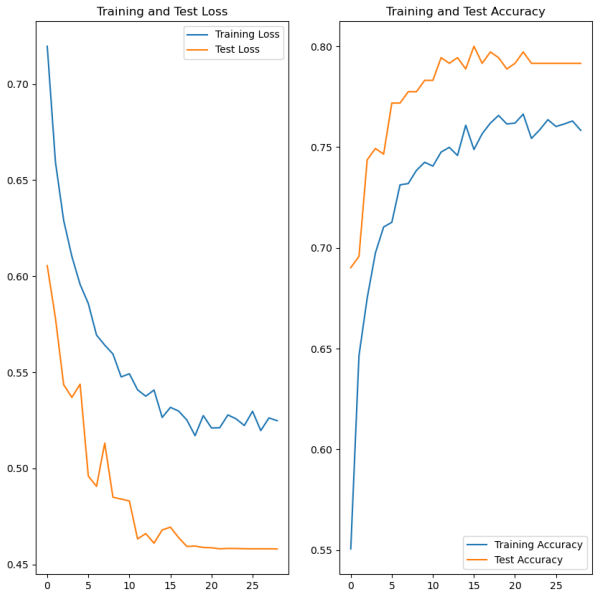

Figure 3: First final model

Once that the expected minimal risk has been estimated, it was possible to extract the hyperparameters of the model associated with the lowest zero-one loss:

neurons: 220, **neurons'**: 12, **activation**: 'tanh', **Stlr**: 0.0001

A final model has been created and then trained on the whole dataset used for the estimation. The remaining data from the test directory has been used for evaluating its performance. **Figure 3** shows that after around twenty epochs the classifier stops improving. Another aspect is that the model seems to perform better on the test set rather than the training one. Even if the difference is not too high, especially for the accuracy, this pattern may require more in-depth evaluations.

Second model: CNN with (32x32)

The model maintains the same architecture of the previous one but it adds some convolutional layers on top of the dense layers. The number of filters increases with the depth of the network. In particular, three convolutional layers with as many dropout layers have been added to the structure of the first model. Moreover, a two dimensional average pooling layer is placed after the first of these three. The resolution of the images remains the same. The introduction of these new layers could potentially enhance the amount of information that the model can extrapolate. In particular it may exploit the proximity of the pixels to obtaining a better performance.

The use of a convolutional network requires to increase the space of the hyperparameters optimization. Consequently, also the number of trials per search is increased form twenty to forty. This time, the search performs also the choice between using or not the dropout for the training. These hyperparameters have been added to the search:

- **kernels** (convolutional layers) : 3,5;
- **filters** (first convolutional layer): 10,20;
- **activation-c** (act. function for convolutional layers): 'ReLU', 'tanh';
- **Use of dropout**: True, False;

  N° of trials per tuning: 40.

The filters increase at each of the convolutional layers. At first, they are selected by the search, then they are doubled and lastly become three times the number in the first layer. As before, the starting learning rate, the neurons in the dense layers and the activation function are also object of the hyperparameters tuning.

Finished with the estimation of the expected minimal risk, it is possible to extract the model with the lowest risk among those of the outer loop. Its parameters can be used to build the second final model.

kernels: 3, **filters**: 20, **activation-c**: ReLU,
neurons: 340, **neurons'**: 16, **activation**: ReLU,
Dropout: False, **Stlr**: 0.0005.

It is easy to observe that the model uses more neurons in the dense layers with respect to the first model and the starting learning rate is higher than before. Moreover, the activation functions are both *ReLU* instead of *tanh*. It is important to point out that dropout has not been employed. This, along with a higher learning rate, makes the test loss more unstable as we can see from the figure below. The instability ends when the training reaches the fifteenth epoch. After that, the test performance stabilizes, and the classifier starts to overfit.

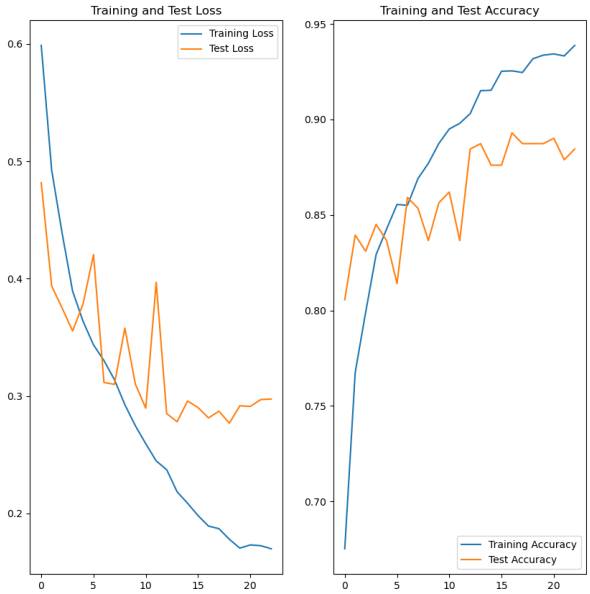

Figure 4: Second final model

Third model: CNN with (64x64)

Third architecture is yet again a CNN. This time, however, the structure utilizes a bigger image resolution as input. The idea is to test if a greater resolution can provide a better performance in classifying the two classes and how great is the difference between the models capabilities. The architecture has not been modified from the previous model. What changes is that the input layer is now exploiting 64x64 pixels instead of 32x32. Hyperparameters search space remains the same.

As before, once that the outer loop has ended, it is possible to extract the set of hyperparameters associated with the best classifier:

kernels: 5, **filters**: 20, **activation-c**: relu,
neurons: 220, **neurons'**: 12, **activation**: tanh,
Stlr: 0.0001. **dropout**: False.

This time, the dimensions of the filters have been increased, even though their number remains the same. Another change is that, instead of using the same activation function for both convolutional and dense layers, the best performance has been achieved using a combination of $ReLU$ and *tanh*. Once again, dropout has not been implemented, but the starting learning rate has been set to a smaller value. The last difference is in the number of neurons; all the layers in this model now utilize fewer nodes than the second model.

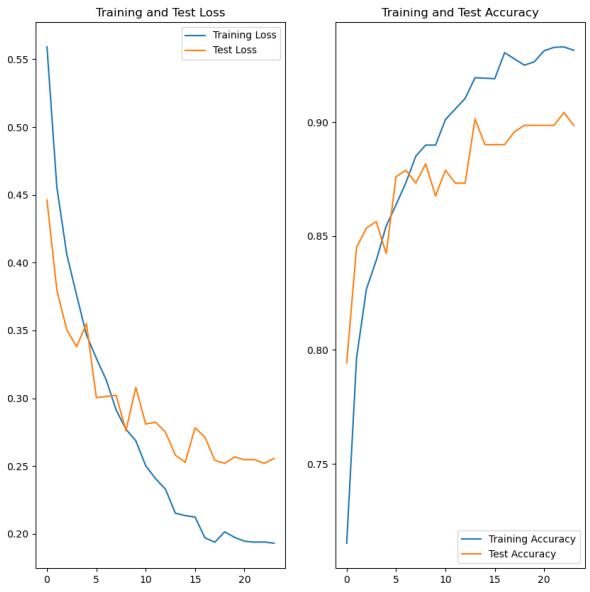

Figure 5: Third final model

From the **Figure 5** it is possible to observe that after twenty epochs the final model stops learning and the losses stabilizes. Although the training loss continues to decrease, post this point, the classifier begins to exhibit signs of overfitting, as indicated by the increasing test loss. Compared to the previous learning, this one seems more stable, probably because of the smaller learning rate and the higher dimension of the filters.

Fourth model: Deep CNN

Differently from the others, this model has not been constructed from scratch. It is a pre-trained model created by Oxford University researchers[2]. The model is called Visual Geometry Group and has been trained over the ImageNet dataset. It is a very deep Convolutional Neural Network that utilize a lot of layers and an high image resolution of 224x224. Its structure is presented in the **Figure 6**.

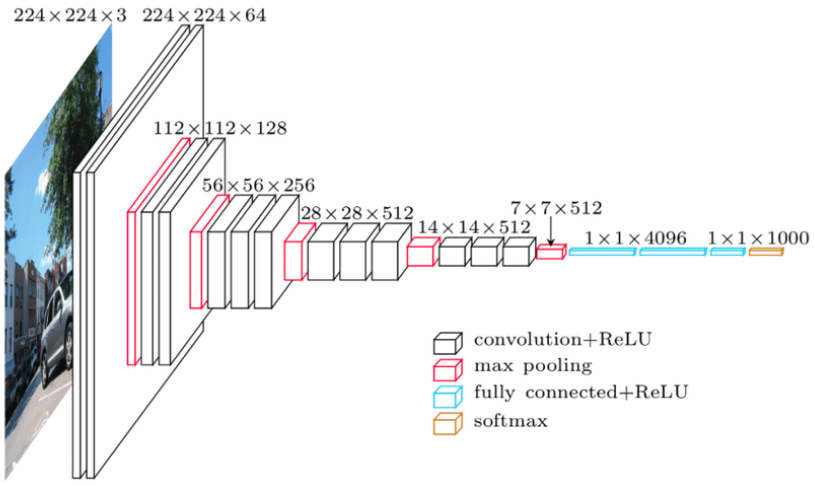

Figure 6: VGG structure

Once this model has been imported, some slightly adjustments have been made to refine it for the project's purpose. Firstly, additional layers have been added to its structure: three dense layers, three dropout layers, and one final output layer with two nodes and the *softmax* activation function. Afterward, the model has been fine-tuned using the dataset that was previously adopted for risk estimation.

[2]"Very Deep Convolutional Networks for Large-Scale Image Recognition", K. Simonyan, A. Zisserman

Results

This chapter presents the results of the project and discusses the differences in capabilities between the architectures, as well as the performance of the final classifiers. **Table 1** provides an overview of the key statistics intended to discussion. As outlined in the methodology chapter, the expected minimal risk has been estimated only for the first three architectures. VGG model has been introduced only for a comparison of its performance with the three final classifiers.

Table 1 displays five main statistics. Firstly, the performance of the final classifiers is assessed using the *zero-one loss* metric alongside *accuracy* and *f-one* for both of the classes. Secondly, the estimation of the expected minimal risk for the three architectures is presented in the form of *zero-one loss*. It is crucial to note that the selection of the three architectures examined in this project follows a deliberate progression. It begins with a simpler network and incrementally increases complexity. This complexity includes introducing convolutional layers, expanding the hyperparameters search space and ultimately augmenting the image resolution used by models.

	NN	**CNN-32**	**CNN-64**	**VGG**
zero-one loss	0.208	0.110	0.079	0.039
accuracy	0.792	0.890	0.921	0.961
'C' f-one	0.82	0.90	0.93	0.97
'M' f-one	0.75	0.87	0.91	0.95
exp. min. risk	0.233	0.141	0.113	-

Table 1

The comparison highlights a noticeable trend: as the complexity of the architectures increases, there is a corresponding decrease in the expected minimal risk. This reduction higher when transitioning from the initial model to the second one, indicating the beneficial impact of introducing convolutional layers. Furthermore, the observation that increasing the image resolution leads to improved classifier performance supports the notion that higher resolution images provide more information for the model to learn from.

A discrepancy between the estimated minimal risk and the final models risk can be detected. This can be attributed to several factors. Firstly, the final classifiers has been trained on a larger dataset compared to the sets used for risk estimation enhancing the final classifiers performance. Additionally and perhaps more significantly, the evaluations have been conducted on different images. As presented in the 'Methodology' chapter, the data has been taken from the part of test folder that has not been used before for estimation. Although the expected and the final risks differ, it's important to note that the overestimation doesn't necessarily invalidate the comparison between architectures. Since there is a consistent overestimation across the three models, the relative ranking of the architectures remains unchanged. The last thing that can be extrapolated for from the table is that all of the models have more difficulties in classifying muffins rather than chihuahuas. Even VGG that is classifying very well the first class, experiences a slight difference when looking at the f1-statistics. One possible reason to that is the different looks that muffins can manifest themselves, especially when decorated with colored frosting.

Overall, the data suggests that the VGG architecture generally outperforms the other architectures in terms of classification accuracy, zero-one loss, and f-one. This could be attributed to its deeper architecture and utilization of pre-trained weights, which allows it to capture more complex features in the data. However, CNN-64 seems to perform nicely when taking into consideration the amount of data that has been used to train it and the overall simple architecture adopted. When comparing the two models, is also essential to consider factors such as computational complexity and resource requirements that have been exploited to train the VGG classifier. For this reason, although the performance of the CNN-64 does not seem mind blowing, given the simple task that the project is trying to carry on, the results are quite satisfactory.

Conclusions

The goal of classifying dogs and muffins was just a excuse to assess the capabilities of neural networks and obtain expertise with these type of models. Although the classifiers have obtained a performance that is not high enough to be used with confidence, the CNN-64 shows satisfactory prediction capabilities considering the nature of the project. When looking at some of its misclassified photos indeed, it is possible to observe that some of them do not actually belong to either of the two classes. For instance, there are images of other dog breeds when looking at chihuahuas, and even uncooked cake mix when looking at muffins. Consequently, it is natural that using a limited dataset, the models are struggling with predictions.

This does not mean that the models are perfectly optimized. There are a lot of further improvements that could be implemented to obtain more robust and effective results. Firstly, as it has already been said, the method used for estimating the expected minimal risk has its downsides. Introducing nested cross validation will give more confidence in the estimation and could possibly results in the choice of more performing classifiers. Secondly, optimization search space can be extended introducing more values and hyperparameters. Potentially, this could further increase the predicting capabilities. Another option could be to improve the architectures. For example, regularizers like Lasso or Ridge can be introduced, more filters and neurons added to the structure and the network could be made deeper. All these improvements, however, require a significant increment in the resources and in the computational power. In the end, the most advantageous progress would come from increasing the size of the dataset. More data should leads to more accurate estimations as the final models learn from a wider range of images.

Overall, considering the lighthearted tone of the task, the project successfully achieved its goal of showcasing machine learning model capabilities and evaluating differences between various architectures. To conclude, although there are imperfections, it can be considered that the work done so far is a good starting point for creating more diverse and complex models.

I/We declare that this material, which I/We now submit for assessment, is entirely my/our 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/our work. I/We 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/us or any other person for assessment on this or any other course of study.