Binary classification of cats and dogs images using Convolutional Neural Networks

Filippo Uslenghi

July 2022

Introduction

Image recognition is an important field of computer vision and machine learning that has a large variety of applications. This task makes large use of neural networks (NN) and convolutional neural networks (CNN) to accomplish it's goal; usually a large amount of data is needed in order to obtain a satisfactory result. In this report I'll work on the task of binary classification, consisting in the recognition of cats and dogs from images. For training and testing the models, I had access to a dataset comprised of 25,000 images of the two classes above. Tensorflow 2[1] is going to be used as frameworks for the creation of the networks.

1 Data preprocessing and loading pipeline

The dataset consists of 25,000 images half-splitted in the two classes (12,500 about cats and 12,500 about dogs), in this way the classes are completely balanced. The images are stored as JPEG files, with 3 channels (RGB) and different shapes.
Unfortunately, the dataset have some issues as 3 images are corrupted, thus not recoverable; 50 are actually encoded as GIF files, 173 as BMP, 4 as PNG and one have a PSD format. For this reason I had to decode and re-encode those images as JPEG files.

In addition, I found some images that were not relevant with the class they belonged to[1]; as shown in figure 1.

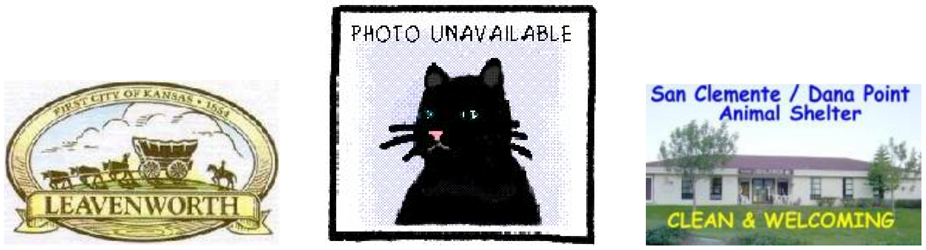
FIRST CITY OF KANSAS 1854
LEAVENWORTH
PHOTO UNAVAILABLE
San Clemente / Dana Point Animal Shelter
CLEAN & WELCOMING
(a)
(b)
(c)

Figure 1: Three images found in the dataset that are not pertinent with the task; note that the image in Figure 1b is present in the dog class.

Next, I had to deal with the loading of the data into memory. As a fact, storing the entire dataset, using 32 bit floating point representation for the values of the pixels of the images, would have taken around 19 GB of memory, making the training of the model not feasible on low memory machines.

For this reason I had to build an input pipeline using the tensorflow API `tf.data.Dataset`[2] that allowed me to load the images directly from disk.

First, I created a list containing the paths to the images of the dataset; then, I shuffled this list in order to remove any bias of memorization of the dataset (in fact, the images in the dataset were sorted by class).

With this list I was able to load the dataset from disk into a buffer of 1000 items; from this buffer the images, together with their own labels, were randomly sampled and loaded in batches of 64 items. Once an element was selected, its space in the buffer was replaced by the next element until emptying the dataset. In addition, I added a *prefetch* operation that allowed later elements to be prepared while the previous elements were being processed; in this way I avoided possible I/O bottleneck from the disk.

2 Experimental setup

In this work I developed five different architectures that achieved different performance. Each of these architectures have a dedicated notebook.

As metric for the evaluation of the models the *zero-one loss* is going to be used, which is defined as:

$$\ell(y,\hat{y}) = \begin{cases} 0 & \text{if } y = \hat{y}, \\ 1 & \text{otherwise.} \end{cases} \tag{1}$$

[1]I was not able to examine the entire dataset and find all the other possible images that where not relevant.

[2]`https://www.tensorflow.org/api_docs/python/tf/data/Dataset`

with y being the label predicted by the model, and $\hat{y}$ being the true label. Each model is going to be evaluated on a test set of 5,000 images.

2.1 An underfitting model

The first model, described in `too_simple_cnn.ipynb`, is a very simple model. As shown by table 1 it is comprised of a single convolutional layer with 8 filters, a single hidden layer with 8 nodes and a single node with sigmoid activation as output layer for the binary class prediction.

Layers	Filters	Nodes	Kernel size	Activation
Conv2D	8	-	3	relu
MaxPooling	-	-	2	-
Dense	-	8	-	relu
Dense	-	1	-	sigmoid

Table 1: Model summary of the first architecture.

Clearly this network isn't big enough, with respect to the complexity of the task, to achieve any adequate result. In fact, this model shows how a small network is going to underfit when it is used to solve a complex task.

As shown in figure 2 the accuracy of the model, either in the validation and training set, doesn't increase during the training phase, meaning that the network is not capable of learning useful features.

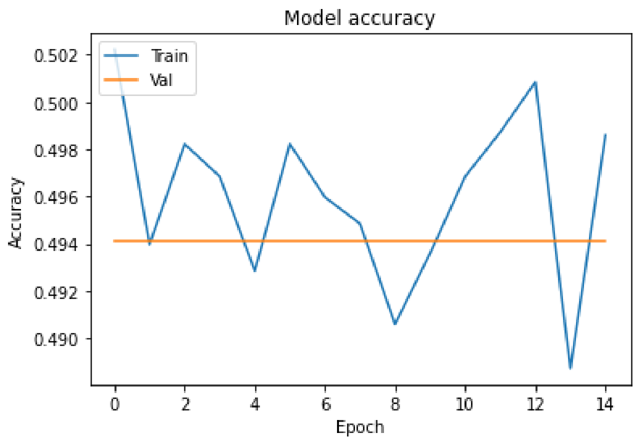

Figure 2: The accuracy of the underfitting model during a training of 15 epochs. In blue is represented the score on the training set, in orange the score on a validation set.

On a single hold-out, with a test set of 5,000 images, this model scored a zero-one loss of 2489, meaning that the model miss-matched almost half of the samples in the test set. Evaluating the model through a 5-fold cross validation, which gives better statistical results, it achieved a mean zero-one loss of 3864, with a standard deviation of 1406, meaning that this model performed worse than a random classifier.

2.2 An overfitting model

The second model, described in `simple_cnn.ipynb`, is a slightly improved version of the previous model. As shown in table 2 this networks is composed of two convolutional layers, a single dense layer and the single output node.

Layers	Filters	Nodes	Kernel size	Activation
Conv2D	16	-	3	relu
MaxPooling	-	-	2	-
Conv2D	32	-	3	relu
MaxPooling	-	-	2	-
Dense	-	64	-	relu
Dense	-	1	-	sigmoid

Table 2: Model summary of the second architecture.

This architecture has an additional convolutional layer, allowing for the extraction of more complex features from the images. The larger hidden layer makes the learning process more fiesable. This architecture performed, in a single holdout, a zero-one loss score of 1464, and in a 5-fold cross validation a mean score of 1600 with standard deviation of 209. This meant an absolute increase of 40% in performance compared to the previous model. As shown in figure 3 this model overfitted the training data, as the training score increased while the validation score remained the same.

2.3 Hyperparameter tuning via nested cross validation

The next architecture that I tested was designed in a similar way to the previous one, with an additional convolutional layer, accounting for a total of three convolutional layers, and a bigger dense layer, now with 256 nodes.

In order to obtain the best model out of this architecture I ran a *nested cross validation*, testing different combination of hyper-parameters.

Unfortunately, the computational resources I had access to were limited and I was able to test only a few hyper-parameters. For the same reason I also had to randomly subsample the dataset and run the internal cross validation on the subsampled version of the dataset. For the evaluation of the model returned by the internal cross validation, I ran a 5-fold cross validation on the entire dataset, allowing direct comparison of the results of this architecture with the others I previously tested.

The hyper-parameters involved in the internal cross validation were:

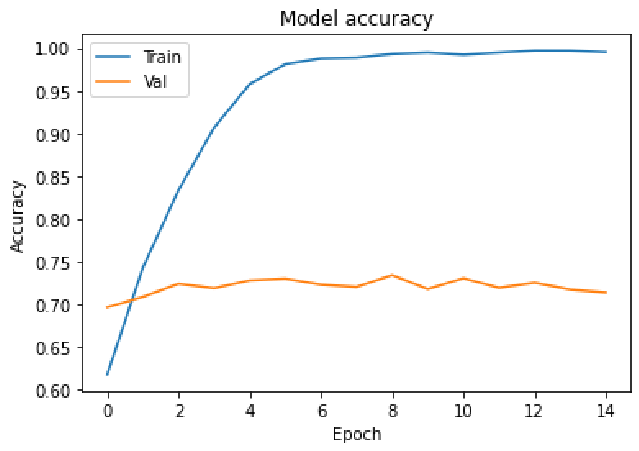

Figure 3: The accuracy of the overfitting model during a training of 15 epochs. In blue is represented the score on the training set, in orange the score on a validation set.

- Number of **epochs** used for the training of the model, with value being one of 10, 15 or 20
- The size, in pixel, of the edge of the square shaped **kernel** used in the convolutional operation, with value being one of 3, 5 or 7.
- The number of **filters** that the convolutional layers had to generate, with value being 16, 32 or 64
- Whether to have an **increasing number of filters** in the convolutional layers or not. If an increasing number of filters was selected the first convolutional layer would have had $filters$ number of filters, the second convolutional layer would have had $filters * 2$ number of filters, while the third one would have had $filters * 4$ number of filters.

Table 3 shows a snapshot of the grid used during the hyperparameter tuning.

Looking at the results of the internal cross validation[3] by fixing one hyperparameter at a time, we can see that there are some that do not influence the results in a precise manner. Others, instead, have a direct impact on the performance of the classifier that will be returned by the algorithm.

Figure 4 shows the results of the scores obtained by the models tested in the internal cross validation having fixed the number of epochs, in figure 5 the number of filters extracted by the convolutional layers is fixed, and in figure 6 the hyperparameter fixed is the size of the kernel used in the convolutional operation.

[3]As a result of subsampling, the zero-one loss scores are related to a test set of 2,000 images.

Filters coefficient	Filters	Kernel size	Epochs
same	16	3	10
same	16	3	15
same	16	3	20
same	16	5	10
same	16	5	15
same	16	5	20
...	...	...	...
incremental	64	5	10
incremental	64	5	15
incremental	64	5	20
incremental	64	7	10
incremental	64	7	15
incremental	64	7	20

Table 3: The grid used to perform the nested cross validation

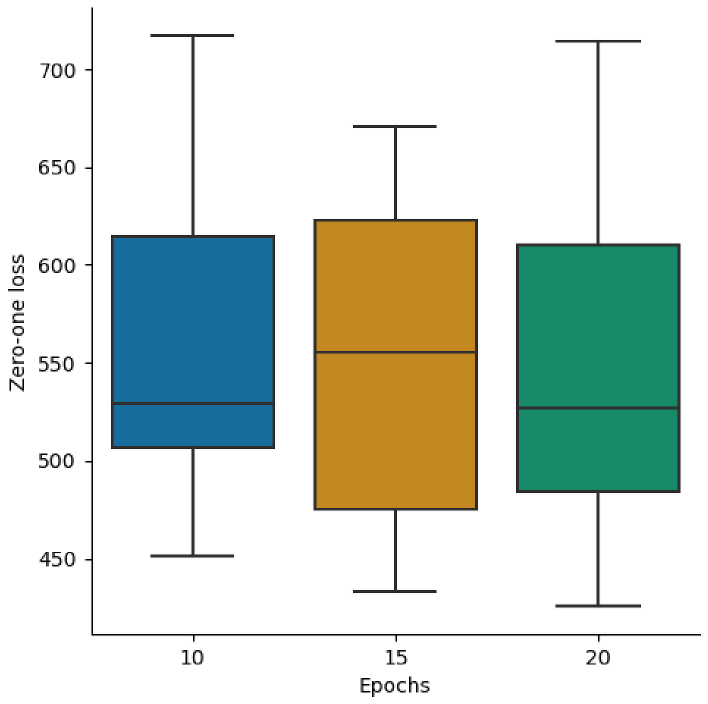

Figure 4: The distribution of the zero-one loss on the test set of the models tested in the internal cross validation, having fixed the number of epochs: it can be seen that the number of epochs used for training did not affect the results notably. This can be explained as if, right after few epochs, the models were not able to generalize any further.

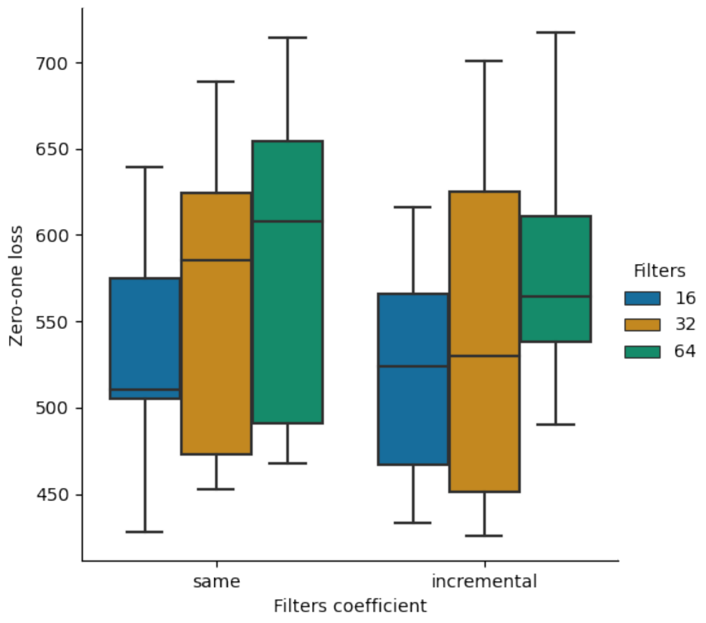

Figure 5: The distribution of the zero-one loss on the test set of the models tested in the internal cross validation, having fixed the number of filters: on the x axis is the value of the *filters coefficient* hyperparameter while color represent different value of the *filters* hyperparameter; looking at the mean of each distribution shows that choosing an incremental number of filters from one convolutional layer to the next, achieves a slightly better result overall. Instead, it seems that choosing a starting number of filters equals to 64 makes for the worst predictions; a value of 16 makes for more consistent results while a value of 32 is going to get the most variant results. Overall, the overlapping nature of these distributions shows that changing these hyperparameters is not going to lead to a precise improvement.

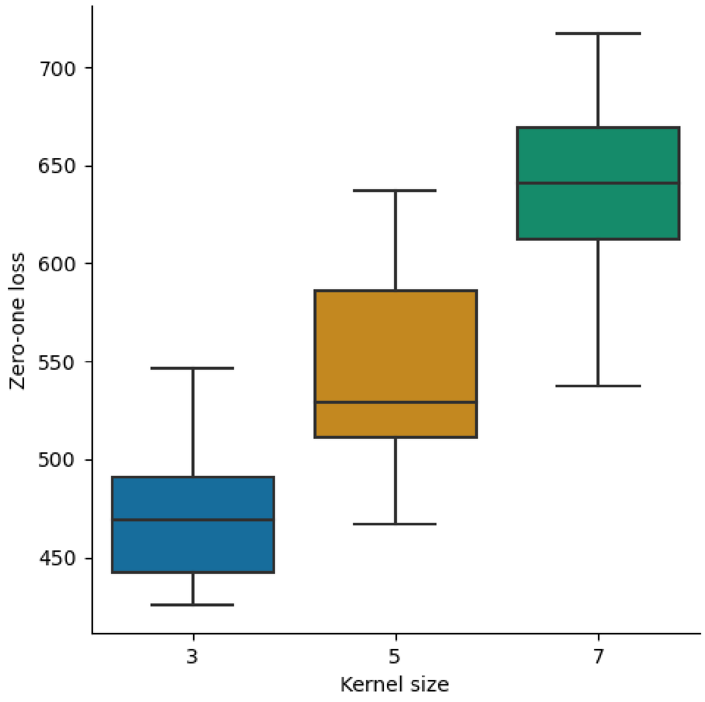

Figure 6: The distribution of the zero-one loss on the test set of the models tested in the internal cross validation, having fixed the size of the kernel: in this case, the difference in performance between the values of the size of the kernel are evident. As the kernel size increase the performance drops considerably; making the size of 3 the best one.

2.3.1 The best model

The model that performed best in the internal cross validation is described in table 4. It has an incremental number of filters in the convolutional layers, starting from 32 filters, then 64 and 128 in the last layer. The kernel performing the convolution has a shape of 3x3 pixel and the training lasted for 20 epochs. I then ran this model in a 5-fold cross validation on the entire dataset in order to make the comparison with the other model reliable. The model scored an average zero-one loss of 1044 with a standard deviation of 41, making for a 11% increase in performance from the previous model.

As shown by the figure 7 also this model overfits the training data. Too reduce overfitting I applied dropout to each layer of the network, expecting to see a slight improvement of the model. Furthermore I wanted to add batch normalization in order to improve performance and training speed. The resulting architecture is summarized in table 5.

Testing the regularized model on a 5-fold cross validation resulted in mean

Layers	Filters	Nodes	Kernel size	Activation
Conv2D	32	-	3	relu
MaxPooling	-	-	2	-
Conv2D	64	-	3	relu
MaxPooling	-	-	2	-
Conv2D	128	-	3	relu
MaxPooling	-	-	2	-
Dense	-	256	-	relu
Dense	-	1	-	sigmoid

Table 4: Model summary of the architecture of the best model returned by the internal cross validation.

Layers	Filters	Nodes	Kernel size	Activation
Conv2D	32	-	3	relu
MaxPooling	-	-	2	-
BatchNorm	-	-	-	-
Dropout (0.2)	-	-	-	-
Conv2D	64	-	3	relu
MaxPooling	-	-	2	-
BatchNorm	-	-	-	-
Dropout (0.2)	-	-	-	-
Conv2D	128	-	3	relu
MaxPooling	-	-	2	-
BatchNorm	-	-	-	-
Dropout (0.2)	-	-	-	-
Dense	-	256	-	relu
BatchNorm	-	-	-	-
Dropout (0.3)	-	-	-	-
Dense	-	1	-	sigmoid

Table 5: Model summary of the architecture of the best model returned by the internal cross validation with regularization.

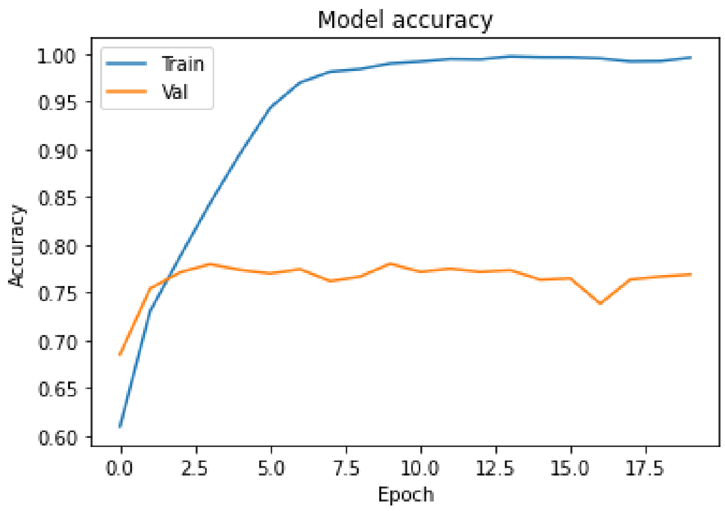

Figure 7: The accuracy of the best model throughout training in 20 epochs. In blue is represented the score on the training set, in orange the score on a validation set.

zero-one loss score of 841 with a standard deviation of 84, obtaining an increase in performance of 4%, making it the best model overall, correctly classifying 83% of the samples it was tested on.

The figure 8 shows that the model still overfits the training data, indicating that adding dropout was not sufficient to overcome this behaviour, even though it slightly improved the model.

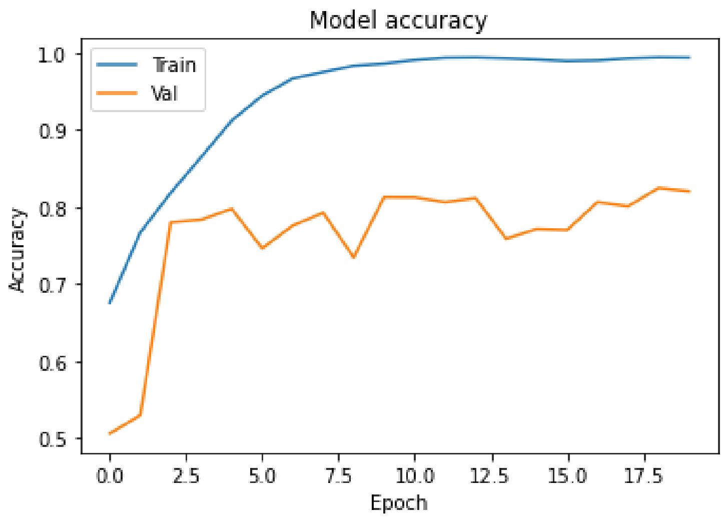

Figure 8: Training and validation score of the regularized model. In blue is represented the accuracy on the training set, in orange the accuracy on a validation set.

2.4 Transfer learning

As a last experiment, I wanted to try transfer learning, adopting a pretrained model. In particular, I decided to use the VGG16[2] network, trained on the ImageNet[3] dataset, a dataset consisting of over a million images of 1000 different objects. From the pretrained model I took only the convolutional base, then I froze the weights, and I substituted the original fully connected layers with a single layer with 256 nodes, using relu activation and adding dropout. Finally I added the single node for binary classification.

The results of this model outperformed the previous ones in the 5-fold cross validation, achieving a mean zero-one loss of 115 with a standard deviation of 9, suggesting a great consistency in the predictions.

Figure 9 shows the results of the 5-fold cross validation of all the models presented in this report; there is also a tabular representation in table 6.

3 Conclusions

In this work I created five models: the first one wasn't able to deal with the complexity if the task, underfitting the data. The second model was complex enough to achieve satisfactory result but, overall, remained quite simple. The third architecture I tried was more complex than the previous ones; additionally I run a nested cross validation in order to find the best performing combination of hyperparameters. The resulting model showed an improvement in the

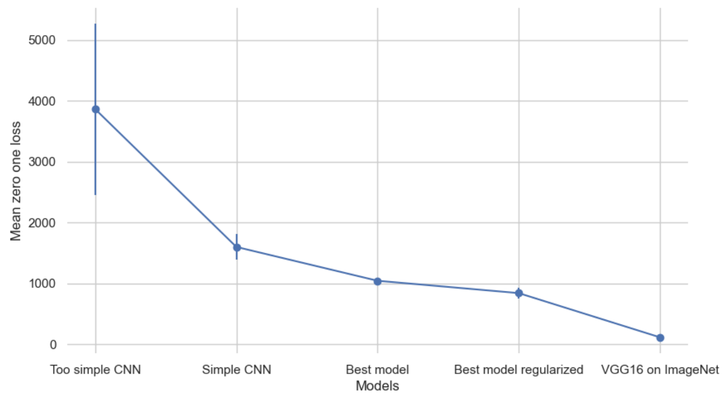

Figure 9: The zero-one loss score of all the model presented in this report in the 5-fold cross validations; the error bars represent the standard deviation.

Models	Mean zero-one loss	std zero-one loss
Too simple CNN	3865	1406
Simple CNN	1600	209
Best model	1044	41
Best model regularized	841	84
VGG16	115	9

Table 6: The results of the models presented in this report in the 5-fold cross validations.

performance with respect to the previous ones. Furthermore, I wanted to apply dropout and batch normalization, obtaining a small increase in performance. Finally, the last model adopted transfer learning, obtaining the best model overall, compared to the previous ones.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks

for large-scale image recognition," 2014. [Online]. Available: https://arxiv.org/abs/1409.1556

[3] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.