Università degli Studi di Milano

Statistical Methods for Machine Learning

Cats vs Dogs image recognition, a Convolutional Neural Network application

Author:
David Fernandez *(988346)*

Lecturer:
prof. dr. Nicolò Cesa Bianchi

November 20, 2022

UNIVERSITÀ
DEGLI STUDI
DI MILANO

Disclaimer

I declare that this material, which I now submit for assessment, is entirely my 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 work. I 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.

1. Introduction

According to Mander and Liu, a Neural Network (NN) is: *"Neural networks are parallel and distributed information processing systems that are inspired and derived from biological learning systems such as human brains. The architecture of neural networks consists of a network of nonlinear information processing elements that are normally arranged in layers and executed in parallel [...] A learning algorithm must be used to train a neural network so that it can process information in a useful and meaningful way."* (Mander and Liu, 2010)

At the same time, NN algorithms can be built in several different ways. A specific type of NN that has become of popular use in certain type of image recognition tasks is Convolutional Neural Networks (CNN). These type of networks have certain architecture and specific parameters to tune, making them a class on their own.

In this paper I will apply different CNN algorithms to a dataset composed by images of cats and dogs, with the objective of fitting a model with the lowest risk estimate. Risk estimates will be computed using 5-fold Cross Validation (CV) and the zero-one loss loss function. The remainder of this paper is organized as follows: Section 2 briefly describes the theoretical background of CNN, Section 3 describes the data pre-processing stage, Section 4 describes the models and their results and Section 5 states conclusions and suggestions for future improvement.

2. Convolutional Neural Networks

CNN as we know them now a days can be traced back as far as the 80's, when Fukushima published the paper *Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position* (Fukushima and Miyake, 1982). Further research has been done since then, but the main idea prevails: add Convolutional layers to a traditional NN architecture to improve performance on computer vision tasks. The notable difference is that CNN allow us to encode image-specific features into the architecture whilst further reducing the parameters required to set up the model (O'Shea and Nash, 2015).

Simple CNN may look something like the display in **Figure 1**. Of course, in reality the number of convolution and hidden layers may vary depending on the task at hand, as well as their parameters. This opens up a vast and large universe of possibilities to build CNN models (no without increasing it's complexity and challenges).

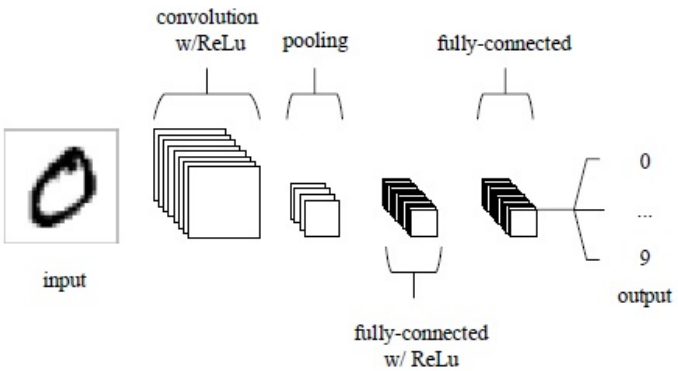

Figure 1: A simple CNN architecture, comprised of five layers (O'Shea and Nash, 2015)

It is worth mentioning that CNN architectures have won several different image recognition contests such as ImageNet, ICPR (Mitosis Detection in Breast Cancer), MICCAI (Medical Image Computing and Computer Assisted Interventions), IJCNN Traffic Sign Recognition Competition, among others, paving the road for further improvement as more and more real case applications are developed using these algorithms.

3. Data pre-processing

The dataset is composed of 12,500 images of cats and 12,500 of dogs of different resolutions. An example can be seen in **Figure 2**. Data pre-processing consists on transforming each RGB channel information into a matrix, i.e. each image has three color channels, therefore three different matrices. Each image will be reshaped to have a size of 100x100 pixels to ease the computation process ahead, so we will end up with a 150x150x3 array for each image. Finally, cats images will be labeled with 0 and dogs images will be labeled with 1.

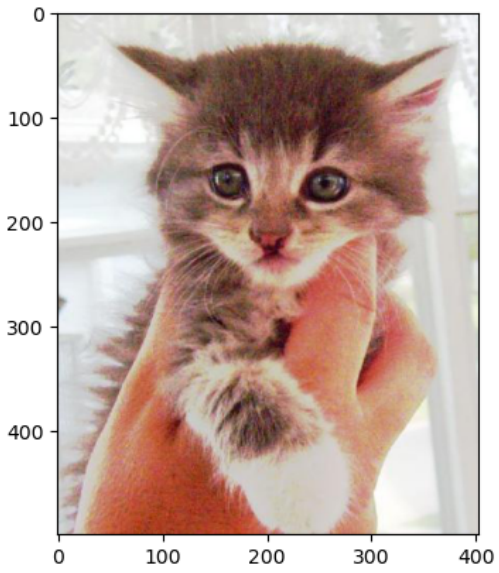

Figure 2: Sample image of a cat from the dataset

Of the total dataset, 54 images were empty and presented an error, so they were dropped. We ended up with a total of 24,946 images. Next step is to split the data into a training and test set. This was done assigning 30% of the dataset to test, and the rest to be trained using 5-fold Cross Validation. Stratification and shuffling is used when splitting the data to guarantee randomness and same balance of cats and dogs images during training and testing.

The input arrays are denoted by X_i and the labels by y_i, where $i \in (train, test)$. Train and test sets shapes are described in **Table 1**.

Dataset	Shape
X_{train}	(17,462, 150, 150, 3)
y_{train}	(17,462, 1)
X_{test}	(7,484, 150, 150, 3)
y_{test}	(7,484, 1)

Table 1: Training and Testing datasets shape

4. Models and Results

Five different models will be trained and tested on the data. General strategy is to start with a baseline model, i.e. **Model 1**, and from there implement regularization techniques, different architectures and hyperparameters to improve risk estimates. Finally, a quick ensemble will be used to test if averaging all the predictions of the five models yield a better results than the individual predictions.

Common ground is that all models will be trained using stratified 5-fold Cross Validation with a zero-one loss objective function and a 20% - 80% validation split. The size of the training batches is 32 images per batch. Number of epochs is 30 and training will stop whenever the validation accuracy stops improving after 2 epochs. These adaptations are done with the objective of reducing computation expenses. Finally, a Sigmoid activation function is used in the last layer of each model.

Model 1

First model is an architecture inspired by the VGG models designed by Simonyan and Zisserman (Simonyan and Zisserman, 2014) for the ImageNet challenge of 2014, and some code from (Brownlee, 2019). Hyperparameters are displayed in **Figure 2** and architecture can be seen in **Figure 3**.

Hyperparameters	Value
Conv1 kernels	32
Conv2 kernels	64
Conv3 kernels	128
Kernel size	3x3
Activation	ReLU
Kernel Initializer	He uniform
Padding	Zero padding
Max Pooling	2x2
Dropout (Conv)	0.2
FC1 neurons	128
Dropout (FC)	0.5
Optimizer	SGD*

*lr=0.001, momentum=0.9

Table 2: Hyperparameters, Model 1

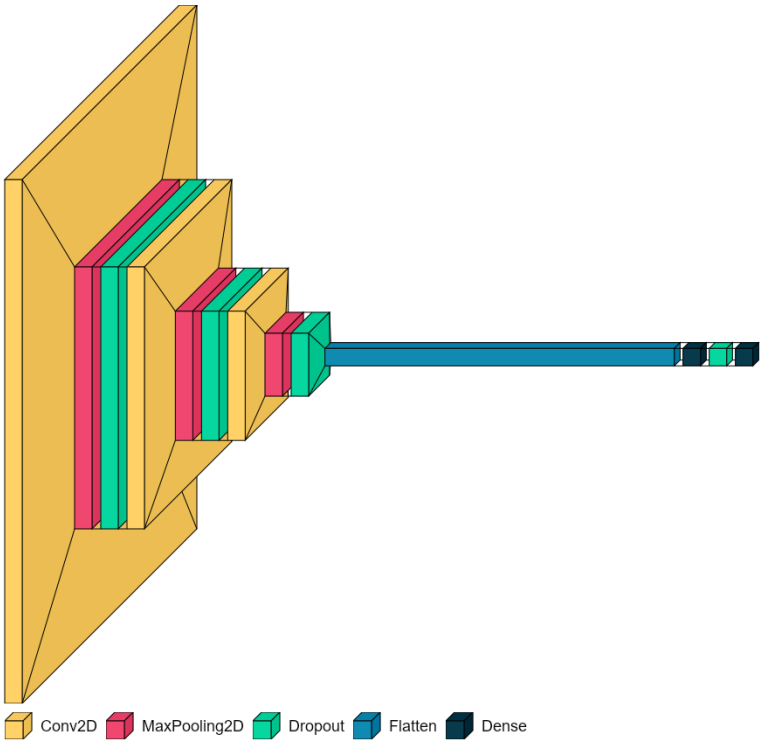

Figure 3: Architecture, Model 1

Training and validation loss are plotted in **Figure 4**. Fold 1 and 5 showed a strange behavior against the rest of the folds, converging faster and displaying lower losses. The total number of epochs seem to be right when combined with a patience parameter of 2, as most of the folds had stopped earlier and the model doesn't seem to overfit the data. **Table 3** displays the CV training accuracy, CV validation result and testing results.

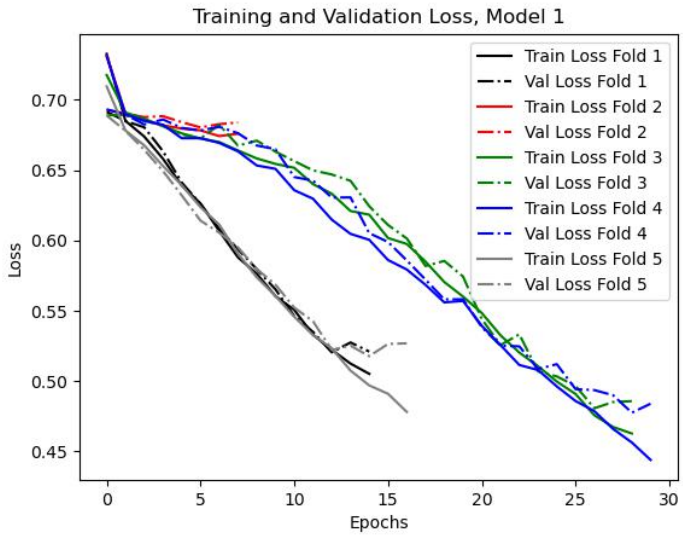

Figure 4: Model 1 CV results

Data	Loss
Training	0.6104
Validation	0.6145
Test	0.4859

Table 3: Risk estimates, Model 1

Model 2

In **Model 2**, the exact same architecture is trained, but some hyperparameters are modified. First, to try a different kernel initializer, the Glorot normal is used, padding is set to no padding and the optimizer is changed to Adam. Motivation behind these minor changes is basically to test if something slightly different would achieve better results, specially with the optimizer, for which its parameters were chosen based on Keras website implementation. Hyperparameters are summarized in **Table 4** and architecture is plotted in **Figure 5**.

Hyperparameters	Value
Conv1 kernels	32
Conv2 kernels	64
Conv3 kernels	128
Kernel size	3x3
Activation	ReLU
Kernel Initializer	Glorot normal
Padding	No padding
Max Pooling	2x2
Dropout (Conv)	0.2
FC1 neurons	128
Dropout (FC)	0.5
Optimizer	Adam*

*lr=0.001, $\beta 1$=0.9, $\beta 2$=0.999, ε=1x10^{-7}

Table 4: Hyperparameters, Model 2

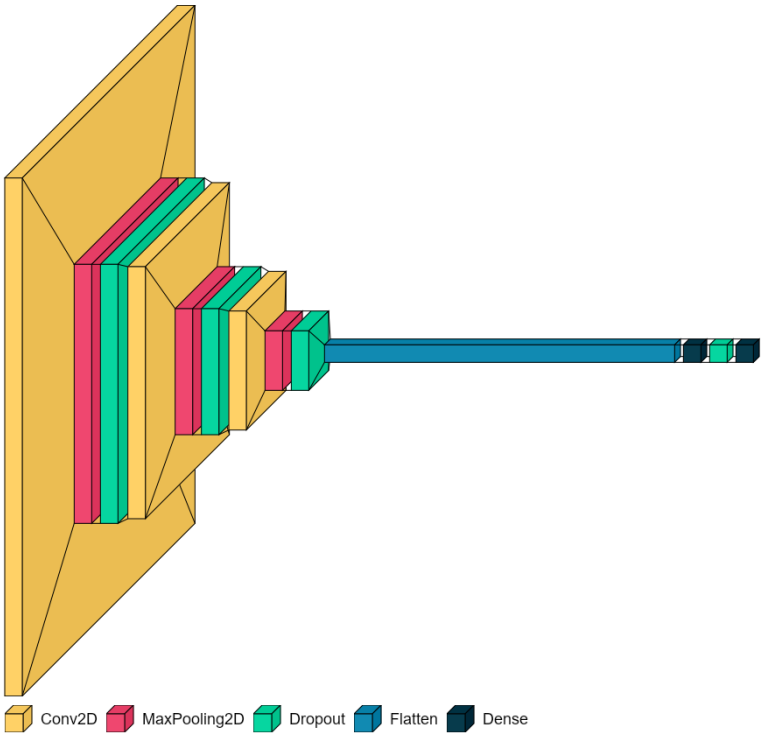

Figure 5: Architecture, Model 2

Results for **Model 2** suggest an improvement both in the speed of training and in the risk estimates with respect to **Model 1**. Something that is still happening is that the test set has a lower loss than the CV estimates. Must be noted that this is also seen in the validation sets during the training for the first epochs, after which the validation loss starts to increase over the training loss.

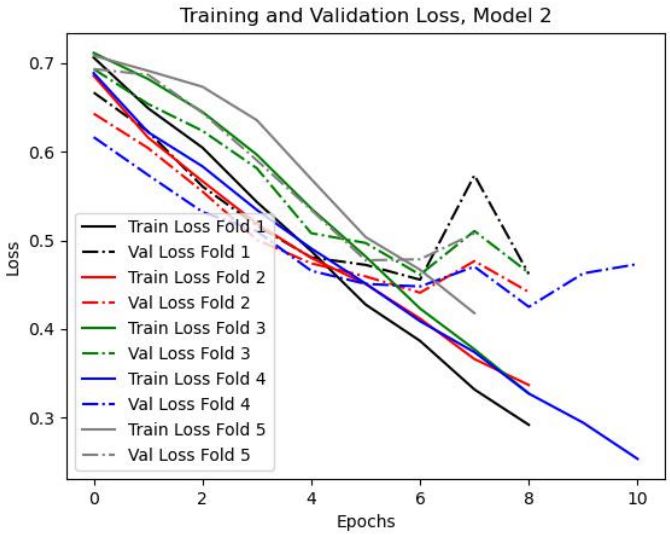

Figure 6: Model 2 CV results

Data	Loss
Training	0.5111
Validation	0.5334
Test	0.4309

Table 5: Risk estimates, Model 2

Model 3

Given the improvement in the hyperparameters adjustments in **Model 2** against **Model 1**, **Model 3** will stand with the same hyperparameters, but with a slightly different architecture. An additional fully connected layer will be added on top of the existing layer, the new layer is added with its respective dropout layer.

Hyperparameters	Value
Conv1 kernels	32
Conv2 kernels	64
Conv3 kernels	128
Kernel size	3x3
Activation	ReLU
Kernel Initializer	Glorot normal
Padding	No padding
Max Pooling	2x2
Dropout (Conv)	0.2
FC1 neurons	128
FC2 neurons	64
Dropout (FC)	0.5
Optimizer	Adam*

*lr=0.001, $\beta 1$=0.9, $\beta 2$=0.999, ε=1x10^{-7}

Table 6: Hyperparameters, Model 3

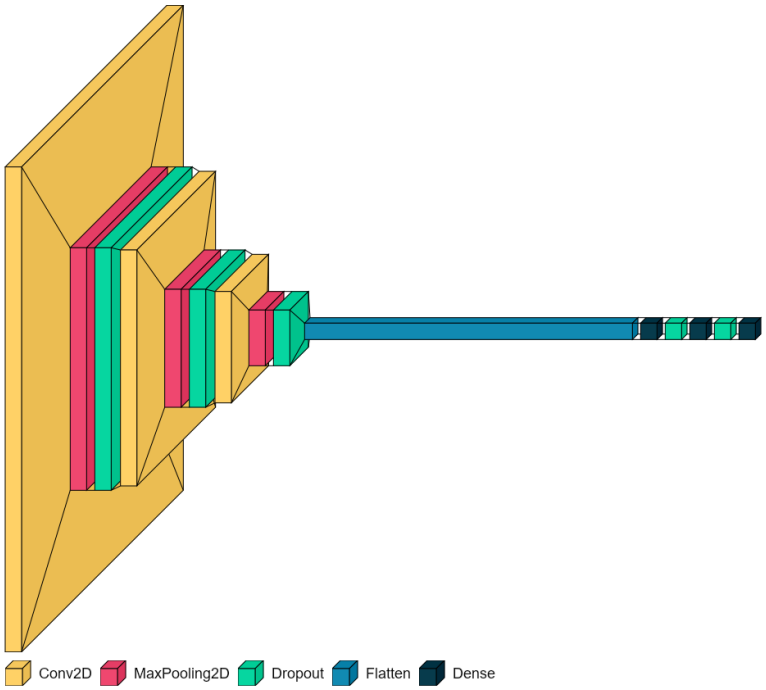

Figure 7: Architecture, Model 3

Results regarding training speed remain more or less the same, while the risk slighlt increased in each dataset. Nevertheless, it seems from the validation graph that training and validation risk behavior is more consistent across epochs.

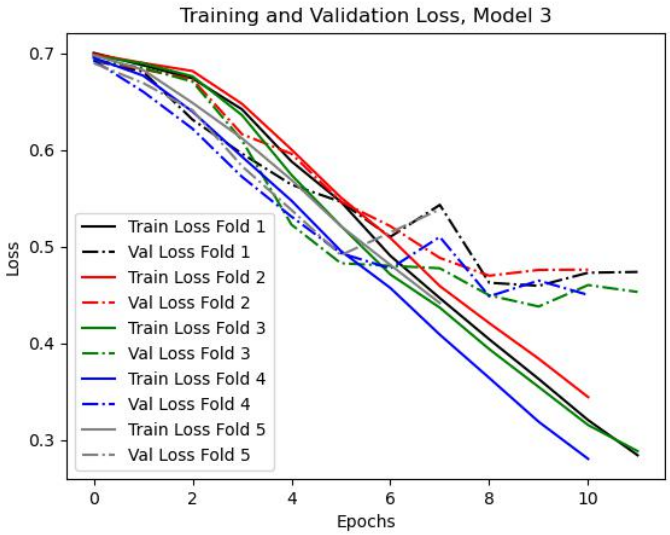

Figure 8: Model 3 CV results

Data	**Loss**
Training	0.5284
Validation	0.5556
Test	0.4401

Table 7: Risk estimates, Model 3

Model 4

For **Model 4** changes in the architecture and in the regularizer are made. First, an additional fully connected layer is added to the two that already had **Model 3**, and a convolution layer is removed, dropout layers are changed for batch normalization layers with the expectation of reducing training time (Ioffe and Szegedy, 2015), also a L2 kernel regularizer is implemented with an $\alpha$of 0.01 on each activation layer.

Hyperparameters	**Value**
Conv1 kernels	32
Conv2 kernels	64
Kernel size	3x3
Activation	ReLU
Kernel Initializer	Glorot normal
Kernel Regularizer	L2 (α=0.01)
Padding	No padding
Max Pooling	2x2
FC1 neurons	128
FC2 neurons	64
FC3 neurons	32
Optimizer	Adam*

*lr=0.001, β1=0.9, β2=0.999, ε=1x10^{-7}

Table 8: Hyperparameters, Model 4

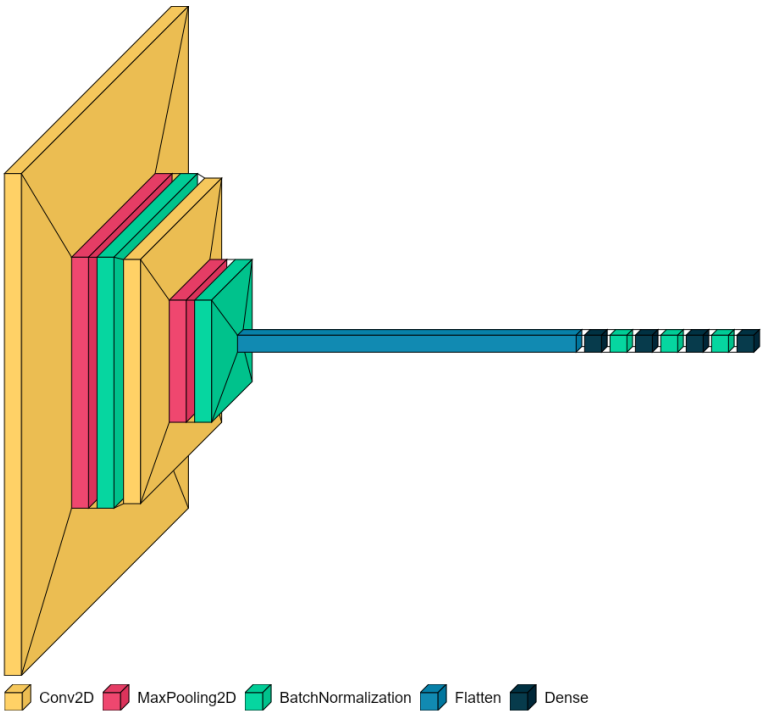

Figure 9: Architecture, Model 4

Risk estimates are much worse than in previous models, moving around 1 and 2. Training time was more or less the same as in previous models. Nevertheless, test risk is still lower than training and validation risk

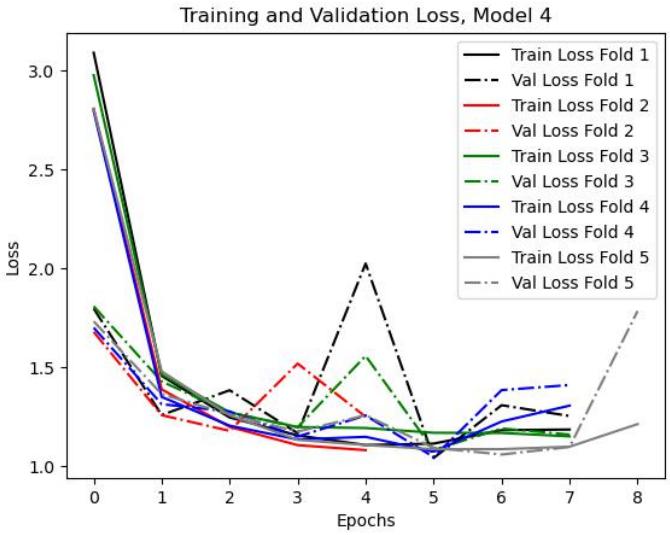

Figure 10: Model 4 CV results

Data	Loss
Training	1.4357
Validation	1.3505
Test	1.0488

Table 9: Risk estimates, Model 4

Model 5

The final model will combine what worked from each past model and will modify the architecture of the fully connected layers for a deeper one. The resulting architecture is one with three convolutional layers, and two fully connected layers. Dropout for the convolutions is slightly higher than previous models, optimizer is again Adam.

Hyperparameters	Value
Conv1 kernels	32
Conv2 kernels	64
Conv2 kernels	128
Kernel size	3x3
Activation	ReLU
Kernel Initializer	Glorot normal
Padding	No padding
Max Pooling	2x2
Dropout (Conv)	0.25
FC1 neurons	512
FC2 neurons	256
Dropout (FC)	0.5
Optimizer	Adam*

*lr=0.001, $\beta 1$=0.9, $\beta 2$=0.999, ε=1x10^{-7}

Table 10: Hyperparameters, Model 5

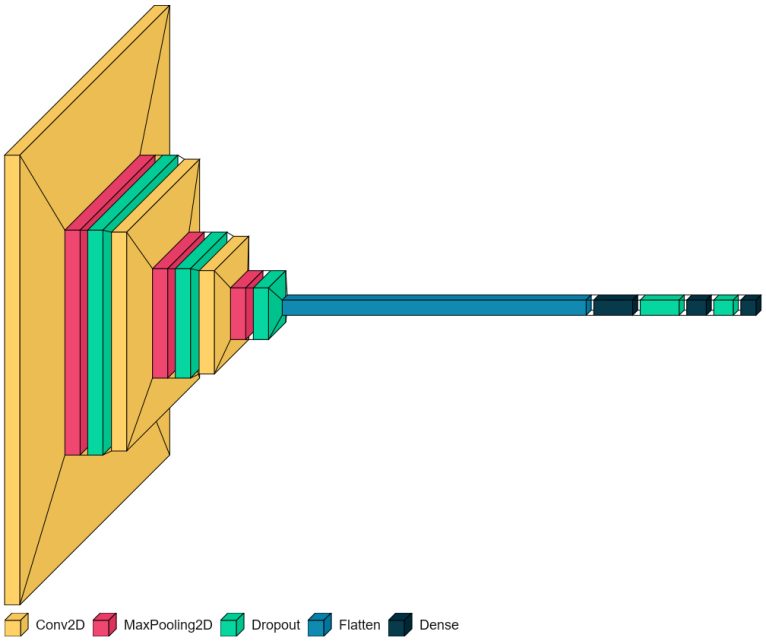

Figure 11: Architecture, Model 5

Results are in line with those in **Model 2** and **Model 3**. Behaviour regarding training speed over epochs is similar, as well as CV and test risk estimates, suggesting that increasing the number of fully connected layers and neurons did not changed the predictive power of the model substantially.

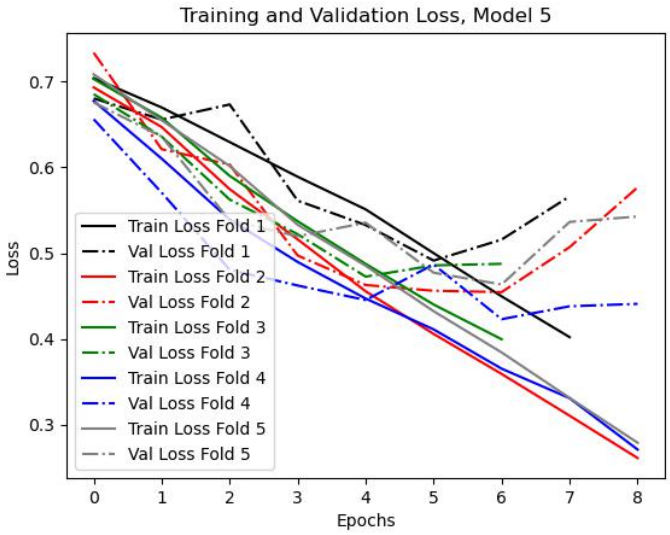

Figure 12: Model 5 CV results

Data	**Loss**
Training	0.5054
Validation	0.5435
Test	0.4254

Table 11: Risk estimates, Model 5

Ensemble prediction

To improve predictive capacity of the models, an ensemble prediction is produced for all of them. The ensemble is simple: it averages the five predictions made for a single test image, and repeats for each image on the dataset, resulting in a simple average prediction. Accuracy of the ensemble is higher than that of individual models, as shown in **Table 12**.

Model	**Accuracy**
Model 1	0.7696
Model 2	0.8103
Model 3	0.8021
Model 4	0.6940
Model 5	0.8088
Ensemble	0.8260

Table 12: Ensemble prediction

5. Conclusions

As it usually happens, we end up with more questions than answers after testing different methods in search for results. We have fitted five different CNN models to a dataset composed of images of cats and dogs, with the objective of building an algorithm with the lowest risk estimate, in accordance with computational capacity and reproducibility of the results. All models were trained for 30 epochs, but with a callback for early stopping with patience of 2 over decreasing validation loss. The lowest CV risk estimate (zero-one loss) achieved was that of **Model 2** with 0.5334, even though it was not very different from that of **Model 3** or **Model 5**. When evaluating accuracy, **Model 2** takes also the first place, with a 81,03% accuracy score, but a simple average ensemble of all the models yielded a higher accuracy with a 82,60% score.

As it is commonly said, machine learning is sometimes more art than science, and tunning hyperparameters can become a daunting task, mainly because 1) training times can easily become very long and 2) it is difficult to isolate the effect of a single change on an hyperparameter, holding the rest of them constant. Nevertheless, this project shed some light about:

1. SGD optimizer and He uniform kernel initializer seem to make the learning process more consistent and less sparse, i.e., slow but steady.
2. Adam optimizer and Glorot normal kernel initializer did improve training speed, but they activated the callback way earlier as validation score would explode in some epochs.
3. Zero padding or no padding didn't seem to have a strong effect during this exercise.
4. Increasing the number of fully connected layers and/or the number of neurons on each layer didn't improve CV loss as well.
5. Batch Normalization by itself didn't seem to help a lot. It is also true that it was implemented with an L2 kernel regularizer, yielding the worst results among all models tested. It is not clear which of both contributed more to the worsening of previous results.
6. It seems that the convolutional layers are the ones that determine the performance of the model, more than the fully connected layers.

Further experiments can be done to test the performance of the optimizers and their interactions with different kernel initializers, specially implementing a higher patience on the callbacks. On the same line, test other architectures that learn slower but with more consistency, as most of the models tested here were fitted in ten or less epochs. Also, batch size was held constant for all the experiments, providing a window for future testing.

Finally, if the final objective of a project of this kind is to release an algorithm for production, an ensemble method could be used to improve accuracy. On this paper the ensemble method was a simple average, but if a weighted average is implemented it could improve results and its weights can even be optimized using Grid Search or another algorithm (Bhattiprolu, 2021).

Bibliography

References

Fukushima, K., & Miyake, S. (1982). Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets* (pp. 267–285). Springer.

Mander, L., & Liu, H.-w. (2010). *Comprehensive natural products ii: Chemistry and biology* (Vol. 1). Elsevier.

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *International conference on machine learning*, 448–456.

O'Shea, K., & Nash, R. (2015). An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*.

Brownlee, J. (2019). How to classify photos of dogs and cat.

Bhattiprolu, S. (2021). Python for microscopists.