# Statistical Methods for Machine Learning
# Experimental Project Report
# Neural Networks

Tommaso Locatelli, Martin V. Ranieri
965066, 967158
tommaso.locatelli1@studenti.unimi.it
martinvincente.ranieri@studenti.unimi.it

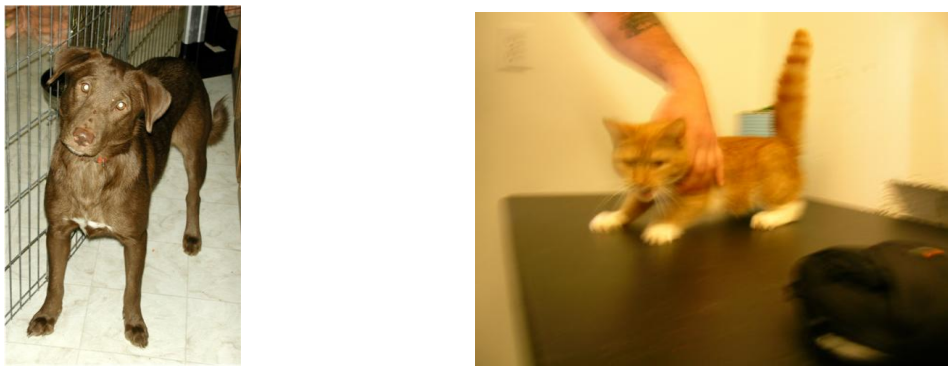
November 30, 2022

**Abstract**

This project address the binary classification task of images: we experimented with different network architectures and we tuned parameters documenting their influence of the final predictive performance. The task is carried out on a dataset of images of dogs and cats. We experimented with three different types of Neural Networks (NNs): the Multi-layer Perceptron, Convolutional Neural Network (CNN) and the Hopfield Network; that give us the chance to tackle the task under different conditions trying to exploiting each architectures pros and cons. [1]

## 0 Preprocessing

The dataset (which is available at Kaggle [2014]) is widely known from its use in Elson et al. [2007]. Since we will make extensive use of Tensorflow 2, it is needed to check which images are corrupted, or incompatible with this library and delete them. After deleting 230 images we are left with 23401 files belonging to 2 classes.



(a) A 327 by 500 image

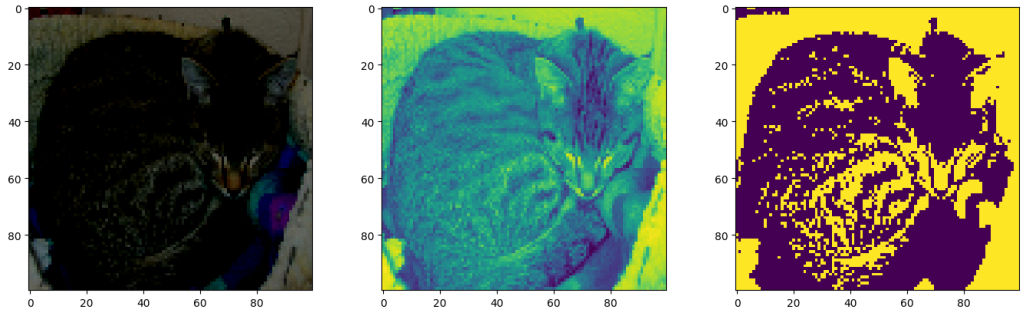
(b) A 500 by 375 image

Figure 1: Two of the original images

Image processing is needed to make easier and faster the ingestion and, therefore, the training of NNs. We experimented with different image sizes and channels of colors. For what concern the image sizes we have chosen to shrink each image into a square resolution, with a number of pixel per side between 64 and 256, trying to tune this parameter for each type of NN. Regarding the channels of colours, on the Multi-layer Perceptron we limited ourself to a single channel (grayscale), we compared

[1] This report is structured in sections which match the notebook structure of the implementation, available at https://github.com/TommasoLocatelli/ml_cats_vs_dogs.

the performance of the same architecture when fed with grayscale or RGB (3 channels) images on the CNN, and finally we experimented binary images on the Hopfield Network.



(a) RGB (b) grayscale (c) binary

Figure 2: The same image preprocessed at 100 by 100 pixel size with different colours channels

# 1 Multi-layer Perceptron

## 1.1 Dataset

Due to the simplicity of the Multi-layer Perceptron model and the difficulty of the learning problem on this dataset, we choose to experiment just with grayscale images and to keep fix the image size in this section. In particular we used 100 per side of the squared images.

To avoid any dependency between the hyperparameters tuning and the performance evaluation we first split the data in equally sized training and test sets, made of batches of 42 images each.

The 80% of the training set is kept as training set for hyperparameter tuning while the rest 20% is used as validation set. Instead, the test set will be used for the five fold cross-validation to estimate the risk, ones the hyperparameters will be tune.

## 1.2 Naive model

As first model, we choose to use a two hidden layer Multi-layer Perceptron. So the image is first flattened and the pass to two dense layer with sigmoid activation function and finally pass to the output layer with softmax activation function. We kept the number of neurons per layer decrease with a more or less constant ratio of 1 over 10, as you can see in Figure 3. We also set as optimizer Adam and as training loss the sparse categorical cross entropy.

As first experiment we train the network over the whole training set and then test it over the validation set for 20 epochs. From figure number 4 is easy to see that the model is stuck in an underfitting scenario.

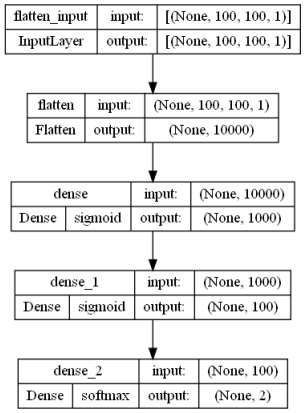



Figure 3: Model structure

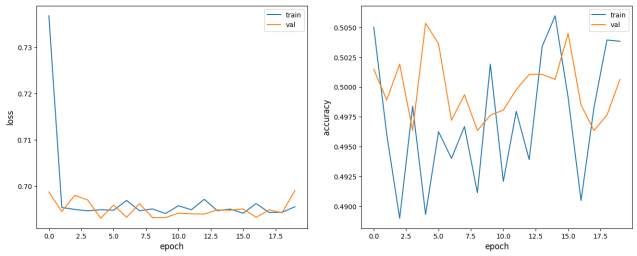



Figure 4: First experiment with sigmoid activation function

We first try to improve the model by changing the activation function in the hidden layers with a ReLU activation function to fight against the vanishing gradient problem.

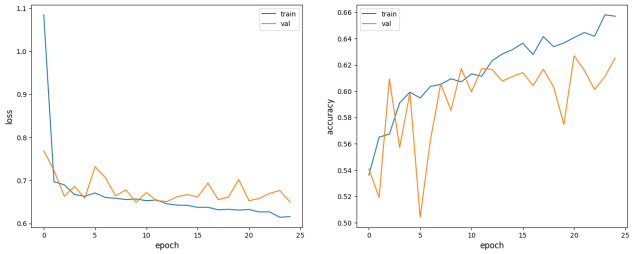



Figure 5: Experiment with ReLU activation function

Both training and validation performance improve until epoch number 10 and it seems to arise a overfitting scenario. But there are a lot of perturbations so we try with a smaller learning rate. Instead of the default value of 0.001 we set it to 0.000001.

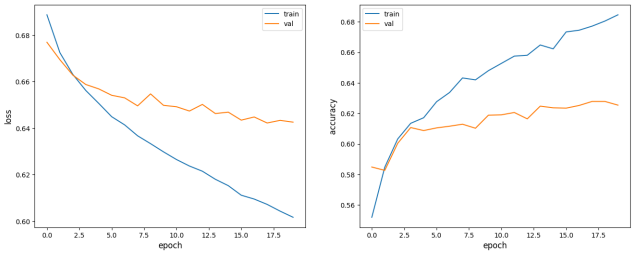



Figure 6: Experiment with lower learning rate

The neural network reach the optimal performance faster and the learning curve is more smooth. We can deduce that with the default learning rate there were a lot of oscillations, so with the new value we obtain a better exploitation. It is also simpler to see the overfitting region after the epoch number 3 and the underfitting before it. Now, that we determined the best hyperparameters, actually: activation function, learning rate and the number of epochs we can estimate the risk with five fold cross-validation on the test set according to zero-one loss.

**Risk estimation result: 0.404**

## 1.3 Many neurons model

Imagine to have the resource to double the number of neurons of a Neural Network, if it is a better idea to simply multiply by two the number of neurons per layer or double the number on layers and redistribute the neurons is exactly the goal of this two last Multi-layer Perceptron models. As point out in the lecture notes we expect to confirm that "networks with many hidden layers, as models able to strike a good balance between bias and variance. In particular, some theoretical results show that the most effective way of decreasing the bias error of is to create new layers in G as opposed to increase the size of layers that already exist."Cesa-Bianchi [2022]

So this Neural Network is obtained just by multiplying by two the neurons in the hidden layers, actually 2000 and 200. As activation function is kept ReLU and as learning rate 0.000001. We can again tune the number of epoch on the training and validation set and as we can see in Figure 7 it seems to go in overfitting after epoch number 2. Moreover, the performance doesn't seem to be improved significantly. By running again a five fold cross validation on the test set we can estimate the risk according to the zero one loss obtaining the following result.

**Risk estimation result: 0.382**

The expected loss is lowered just by two percent point by using the double of the neurons in each hidden layer.

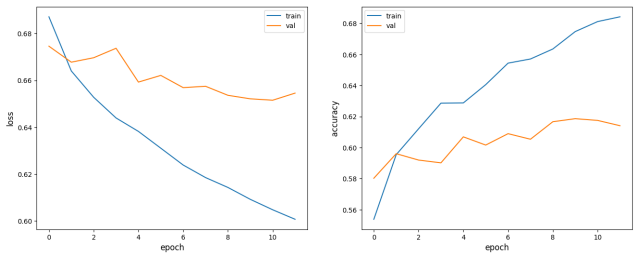



Figure 7: Experiment with double neurons per hidden layer

### 1.4 Many layers model

Finally, instead of doubling the number of neurons per layer we double the number of layer and then redistribute the neurons. We came up with a four hidden layers with respectively: 750, 250, 75 and 25 neurons each. With a total number of hidden neurons equal to 1100 as in the first Naive model. Again we keep hyperparameters fixed and tune the number of epochs with the training and validation set. The network seems to not need a lot of epochs and goes in overfitting quickly, so we choose a number of epochs equal to 3 for the last five fold cross-validation estimate.

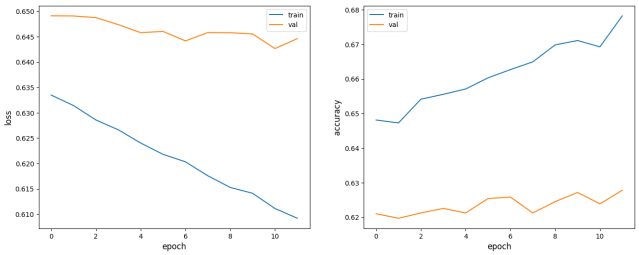



Figure 8: Experiment with double hidden layers

**Risk estimation result: 0.317**

So, as expected, adding more layers improve the performance vastly better than adding neurons per layers.

## 2 Convolutional Neural Network

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. CNN architectures consist in two section: the head is built by pairing multiple times a convolutional layer together with a pooling layer (usually a maximum or average pooling). The convolutional layer will provide a set of kernel filters to extract specific patterns from all the segment of an image, then the pooling layer will emphasize the presence (or not) of that pattern. The tail of a CNN is basically a dense FF NN that will weigh the patterns for each specific outcome.

**Baseline** Before going through different evaluations, we provide a base model that reflect a plain architecture of a CNN. Implemented architecture could be seen in Figure 16. As before, data have been split in half: the second part have been used later for risk evaluation, and the first part of the dataset was devoted to hyperparameter tuning from which losses and accuracies below have been computed. As you can see in Figure 9, our base model achieved an accuracy of 0.79 after 12 epochs.

### 2.1 Data

Image pre-processing can affect the pattern extraction in meaningful way. We experiment two main pre-processing to evaluate their influence on performance.

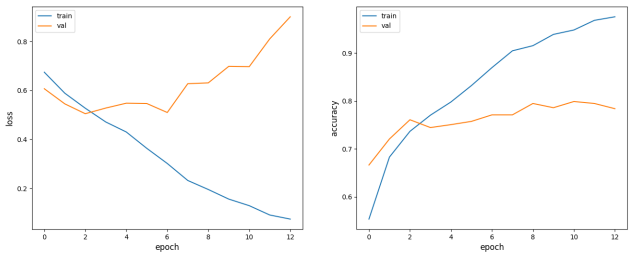



Figure 9: Base Model: training loss and accuracy

**Grayscale** We reduced image channels from 3 to 1 to evaluate if colours are effectively useful to detect features that can discriminate from our two target classes (dog or cat). From Figure 10 we can see that colours are not so useful after all. Our base model reached nearly identical performance (looking at numbers of epochs needed and achieved accuracy) when fitted onto gray-scaled images. Thus peroformance, and specifically memory footprint, could be improved by just ignoring colours.

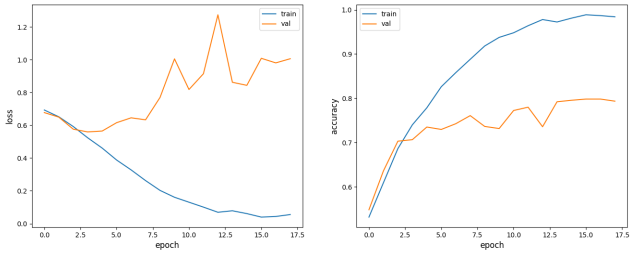



Figure 10: Grayscale images: training loss and accuracy

**Smaller size** We reduced image resolution from `128x128` to `64x64` pixels to evaluate how can lower quality images make harder for a CNN to still detect patterns inside of them. From Figure 11 we can see that our base model cannot improve its forecast from something different than a random guess. We can clearly state that images at higher resolution are strongly helpful to train a CNN to recognize specific patterns.

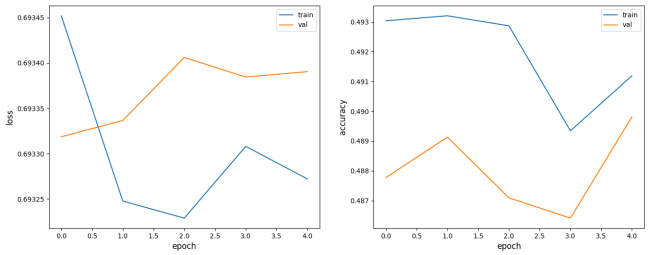



Figure 11: Smaller images: training loss and accuracy

## 2.2 Model

We experimented different evolution of our base model to assess how different changes in a CNN architecture could affect performance.

**Adding more layers** Adding more layers result in ad worsening of the performance, keeping a fixed number of maximum epochs. This is probably explainable due to the slower convergence.

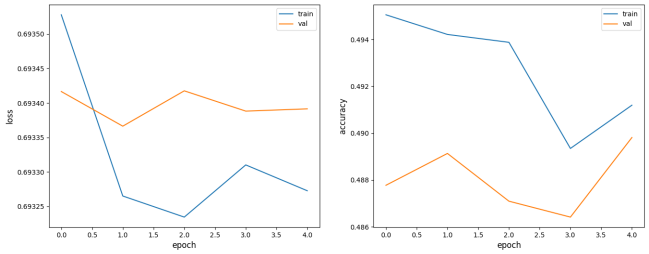



Figure 12: Model with more layers: training loss and accuracy

**Adding more filters** The addition of filters, instead, improved the performance as lead to networks to capture more patterns.

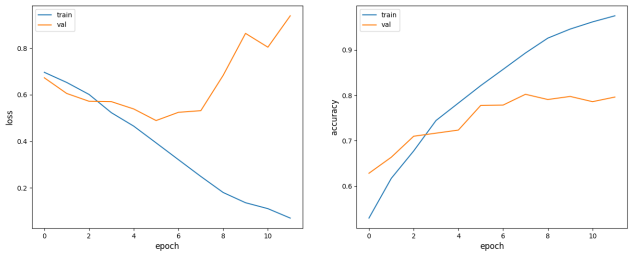



Figure 13: Model with more kernels: training loss and accuracy

### 2.3 Risks Estimate

Finally, we compute the five fold cross-validation risk estimate of the model with more filters and grayscale images, as best model.

**Risk estimation result: 0.195**

## 3 Hopfield Network

Both the Multilayer Perceptron and CNN are example of FeedForward (FF) NN, that are networks with an acyclic graph. In this final part of the project we decide to experiment with an example of recurrent network. We choose one of the simplest forms, the so-called Hopfield networks. An Hopfield Network is a NN without hidden layers. Each neuron is both an input and output layer and each neuron has a connection to any other neuron, with expect to itself. The connection weights are symmetric, the input function of each neurons is the weighted sum of the outputs of all the other neurons, the activation function is a threshold function, and the output function is the identity function.

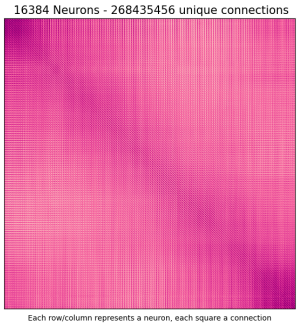



Figure 14: Weights matrix

The basic functioning of an Hopfield network consists in having a number of neurons equal to the size of the vector that have to process, initialize each neuron with the respective vector value, just binary values are allowed, and then update asynchronously all the neurons until a stable state is reached. Thanks to the *Convergence Theorem for Hopfield Networks* we know that if are updated asynchronously, a stable state is reached in a finite number of steps. If the neurons are traversed in an arbitrary, but fixed cyclic fashion, at most $n2^n$ steps.

This make Hopfield Networks very well suited to implement so-called associative memory. If a pattern is presented to an associative memory, the memory returns whether this pattern coincides with one of the stored patterns. Hopfield networks are employed as associative memory by exploiting

their stable states, of which one is eventually reached in the work phase. If we determine the weights and the thresholds of a Hopfield network in such a way that the patterns to store are exactly the stable states, the normal update procedure of a Hopfield network finds for any input pattern a similar stored pattern. In this way "noisy" patterns may be corrected or disturbed patterns may still be recognized. Details of the proof of the theorem and the procedure, with which the weights are chosen to ensure that the memorized patterns are actually stable states, can be found in Steinbrecher [2016]. Convergence is strictly related to the physical interpretation of this model, which associate to each state a level of energy, as the energy decrease during the computation and the number of possible states are finite it can't go on forever.

As far as our project is concerned, some observations are significant. First of all, stored patterns are not the only stable states that the network can reach. In fact, Binary Hopfield Networks (BHNs) are prone to "spurious" minima. If memories learned by a BHN are too similar, or if too many pattern vectors are learned, the network risks converging to an in-between memory, some combination of learned patterns. In other words, the network will fail to discriminate between patterns and becomes useless (Science [2022]). Beside this we have to convert the associative memory task in a binary image classification task. In order to do that we proceed as follows: we store in the network some images as training example ensuring that at least one image per label is stored, then, to predict the label of a new image, we feed the network with its vector representation and, once it has converged to a stored pattern, we use this latter label to do the prediction. The idea is that an image will probably converge to the more similar stored pattern that will hopefully have the same label.

## 3.1 Hopfield class & Dataset

As regards the implementation of this network, we did not rely on Tensorflow. Instead we extended a class outlined in the article Science [2022]. We modified and add several functions, but two changes in particular are noteworthy. First, the random order in which neurons update is fixed, in order to exploit the last property of the *Convergence Theorem*. Secondly, a procedure to deal with spurious states is introduced. If the stable state reached doesn't belong to the stored patterns a linear regression on this last is done: the one, which coefficient has the greatest absolute value, is chosen as the closest pattern, and its label is taken as prediction. Moreover, to minimize the risk of running into spurious states, a training pattern selection procedure is defined below to let patterns stored be dissimilar as much as possible.

*When the BHN have to store* $\boldsymbol{n}$ *patterns a training set with* $\boldsymbol{m}$ *examples is presented, with* $\boldsymbol{m} > \boldsymbol{n}$*. Then the matrix distance between images is computed with respect to the hamming distance. The image with the lowest distance value and most frequent label inside the training set is deleted until we are left with just* $\boldsymbol{n}$ *examples to be stored. This ensure that at least one pattern per label is kept.*

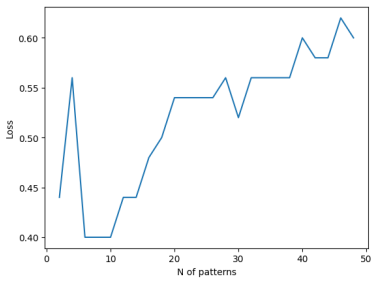



Figure 15: Performance of the validation loss varying the cardinality of the training set

Concerning the dataset, one of the most interesting features of this network is that is it one of the few NN that don't need to be feed with a lot of data. Rather, too many stored patterns can cause problems regarding spurious states. Other than that, the training phase result to be much faster. So, we dealt with a small fraction of the original dataset, just 160 images for instance. Moreover, Hopfield networks deal just with binary vectors [2], so we converted grayscale images by applying a threshold function to each pixel. These two facts make this model need very few resources to store a training set that allows significant performances to be achieved.

## 3.2 Experiments

The first interesting result we obtained, confirm the expectation that this network have always zero training error. In principle this depend on the fact that, if a stored pattern is a stable state and then the network is initialized with this stored pattern, the network has already converged and returns

[2]This refer to BHN but from 2020 Continuos Hopfield Networks are introduced and they have also proven to have incredible performance on a large number of tasks.

exactly the label associated to the stored pattern. Then we tune the only hyperparameter we are left with, that is the number of stored patterns. We test the performance on the validation set for even numbers of stored patterns from 2 to 50. From Figure 15 it seems to appear the classical underfitting-overfitting U shape, with best choice of number of patterns between 8 and 10. Finally, we can run five fold cross-validation to estimates the risk with respect to the zero-one loss.

**Risk estimation result: 0.382**

It has reach a performance comparable to the two firsts models of Multilayer perceptron but with much less memory and time consumption. Furthermore, it is also interesting to outline that, for how we defined our models, the set of trainable parameters for the FFNN and the BHN are at the same order of cardinality.

At the end of the notebook it is also available an animation that show an example of image convergence.

# 4 Conclusion

Addressing this task with such different networks architecture give us the opportunity to exploit each architecture pros and cons and document many training parameters influence on the performance. The Multilayer Perceptron represent the simplest network to deal with this task and we used it to document the influence of the simplest training parameters such as: learning rate, activation function and also to corroborate how adding layers can be a better option instead of adding neurons. Convolutional neural networks are the most suitable models to solve this task as they are the state of the art technologies for image pattern recognition. Their performance outstand all the others models taken into account in this project but at the cost of more hyperparameters and time consumption. The Hopfield network represent an interesting example of how Recurrent NN can be applied at this task. It doesn't reach a particular performance but outstand from all the others models for the simplicity of the training phase and the lower consumption of computational resources compared to models that have similar performances.

# References


Cesa-Bianchi, N. (2022). Neural networks and deep learning. https://cesa-bianchi.di.unimi.it/MSA/Notes/deep.pdf.

Elson, J., Douceur, J. R., Howell, J., and Saul, J. (2007). Asirra: a captcha that exploits interest-aligned manual image categorization. *CCS*, 7:366–374.

Kaggle (2014). Dogs vs. cats. https://www.kaggle.com/competitions/dogs-vs-cats/data.

Science, T. D. (2022). Hopfield networks: Neural memory machines. https://towardsdatascience.com/hopfield-networks-neural-memory-machines-4c94be821073.

Steinbrecher, R. K. C. B. C. B. S. M. M. (2016). *Computational Intelligence*. Springer Nature.


# Appendix

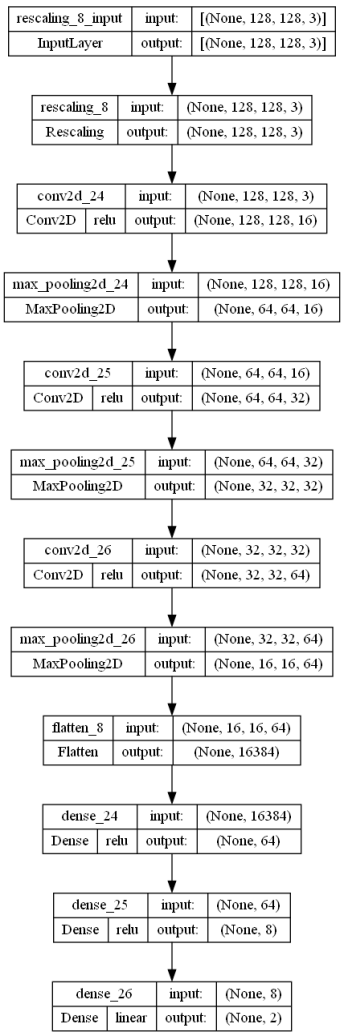



Figure 16: Architecture of the base model

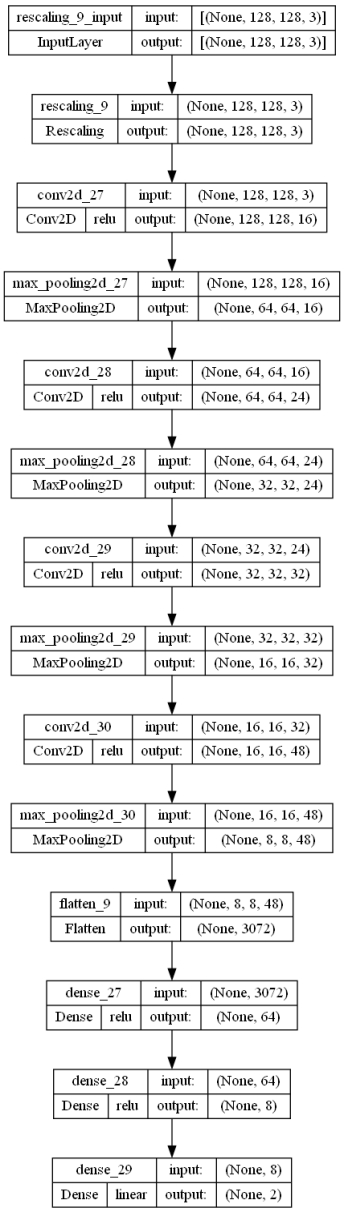



Figure 17: Architecture of the model with more layers

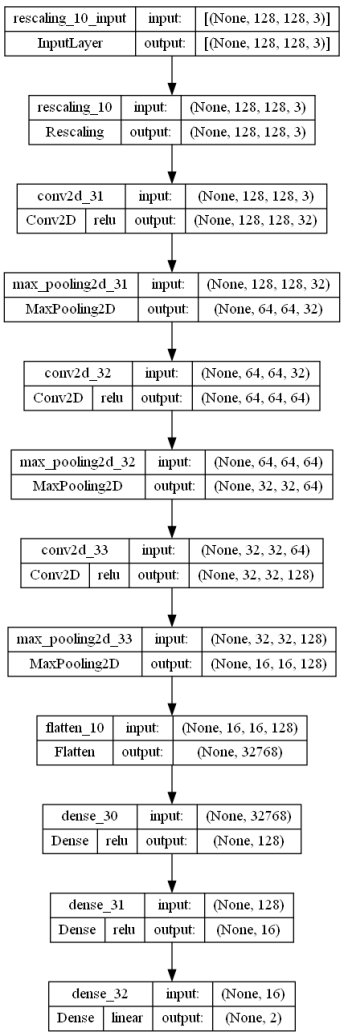



Figure 18: Architecture of the model with more kernels