RoboCup@Home Object Classification

Homework 2
Machine Learning course A.A. 2020/2021

23/12/2020

1 About

This document is about the report of the Homework 2 for the Machine Learning course at Sapienza University of Rome (A.A. 2020/2021).

In this homework we were invited to provide a solution for the classification of images regarding objects in a home environment.

To solve the problem I took into account the given exercise about **Ensebles of Convolutional Neural Networks**, adapting that to the problem and explaining my decisions for the proposed solution.

2 The Dataset

The provided dataset is a subset of pictures developed within the RoboCup@Home competition.
This subset has been randomically choosen and resulted in a total of **8032 images of various sizes** distributed in **8 different classes** such as `juice_carton`, `Buckets` , `Cherries`, `noodle_soup_cup`, `Sporks`, `accent_plate`, `Jelly_Beans_bag`, `plastic_food_container`.
No training nor validation sets were given.

2.1 Dataset Modeling

Since the dataset was etherogeneous I wrote a simple python script in order to divide it into **test set** and **training set**, with a relative proportion of $\frac{1}{3}$ and $\frac{2}{3}$ of the total.
During the dataset division I also had to **resize** pictures. To do so I previously calculated the average values for width and height, respectively resulting in 233.51 and 222.00, letting an average ratio $\frac{width}{height}$ of 1.05.

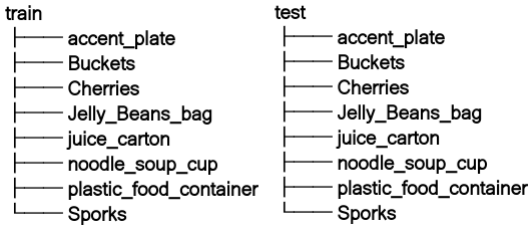
Given that the average ratio is near to one, I decided to **resize** pictures to size 150×150, **padding** when neeeded and **excluding** all those pictures with a $0.5 < ratio < 1.5$ to avoid to give too much distorted picturesfor training.

The new dataset, i.e. the one I used for training and validation, is so composed by a total of 7180 pictures of size 150×150.

How does padding pictures with black pixels while resizing affects the behaviour of a Neural Network? While taking this decision I took into account that modifying the size of an image and adding extra black pixels could change **how the Neural Network learns**: in a case like this the CNN have to learn **that the added pixels are not a relevant part of the picture** and do not help in distinguishing between the classes, as there is no correlation between the black pixels and the belonging class! To cope with this problem might be useful to spend more time training. That is why **I tried to minimize the impact of the black padding pixels** by resizing in a ratio that is not far from the average one and by exlcuding the most 'problematic' pictures.

Another important consideration about the dataset is that it wass full of **noisy data**: some pictures may not be realistically in the proper class, a SUV clearly is not a spork. However nothing can be done except removing those noisy pictures by hand, but since they are part of the game I did not remove them.

At the end, the directory tree in my workspace resulted like:



3 Implemented Solution

As specified above, to solve the problem I used an **Ensemble of three Convolutional Neural Networks**. Ensemble models are used to combine the computations of multiple agents into one final prediction.

I will be honest: the code I used is very similar to the one provided in the specified exercise. The work I have done consists mostly in adapting that code to my problem, trying to understand what was essentially perfect and what needed to be changed and why, verifying if my assumptions were correct or wrong by testing and analysis results with different given inputs. This is described in the testing section.

3.1 CNN Overview

Without being too much verbose I am going to summarize what Convolutional Neural Networks are and how they can be used to solve the given image classification problem.

CNNs are commonly used to analyze visual data, such as pictures. It is important to state that an image can be represented as a *3-D tensor*, where the three dimensions indicate the width, the height and the RGB values of each pixel.
The architecture of a CNN consists in the concatenation of different **layers** between the input layer and the output layer. These inner layers are called *hidden layers*.
In this implementation are present **convolutional** layers and **max-pooling** layers.
In convolutional layers convolutional operations between portions of the image (that is as said a 3D matrix) and some filters are performed, while in max-pooling layers a pooling operation is done. These layers alternate themselves in depth of the Neural Network, providing automatic feature extraction for the classification.

Two more layers are needed to ensure the CNN works: a **flattening** layer and a **dropout** layer.
Since at the end of the last max-pooling layer we still have a three dimensional output while we need only a two dimensional vector (that is a list of the classes in where the input matches with a certain confidence), the flattening layer consists in re-arranging the 3D tensor into a 2D one and then dropping out some neurons with a certain probability with the purpose of 'turning off' some neurons to avoid overfitting.

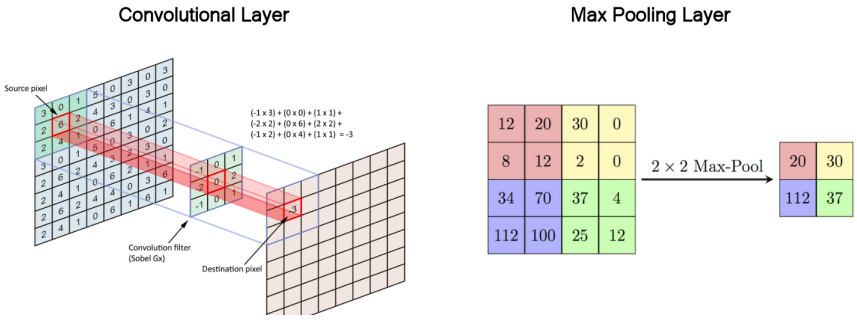


Figure 1: A Convolution operation and a Max Pool operation

All the layers are concatenated to create the Convolutional Neural Network. These operations are stated in the `CNN()` and `Ensemble()` functions in the code.

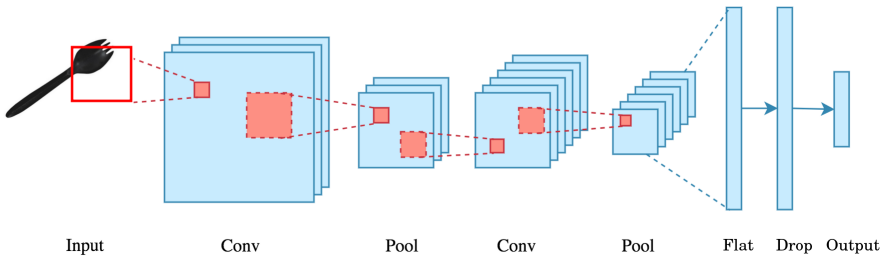


Figure 2: A Convolutional Neural Network

3.2 Key Hyperparameters

After generating the model what we obtain is the skeleton of a Neural Network. In order to obtain an agent that can classify an input, the CNN must be trained, so that connection between the nodes (neurons) can be filled with information.
The training process in a non-deterministic process and in this moment different balancs of some hyperparameters may change the output considerably.

In my tests I focused my attention on changing these hyperparams, training the Ensemble and then changing the parameters again considering the

received feedback in terms of accuracy. Here are some hyperparameters I wanted to change to find an acceptable solution.

base_width Is the number of filters, that doubles at each level of depth. Each filter has a dimentionality of **kernel_size**, that in this case is $3 \times 3 \times 3$. Filters are used to compute the convolution operations. More practically, filters are small tensors initialized with random values. When applying the filter on a convolutional layer, the filter slides over each 3×3 set of pixels (the other dimension represents the RGB values of each pixel) for the entire image, producing as output a smaller image made of scalars that are the products of the single steps of the convolutiond.

Activation Function Every time a convolution is performed, instead of instantly putting the value inside the neuron, that value is passed to the activation function. In this specific case, **ReLU** has been choosen as activation function. This function is a kind of $f(x) = x^+ = max(0, x)$, where x is the input of a neuron. This function is used as rectifier, to normalize at 0 all negative values, leaving unchanged poisitive ones.

Depth Represents how deep must build the Neural Network in terms of layers: in this implementation, each level of depth consists in the concatenation of a convolutional layer and a pooling layer. The deeper the Neural Network goes, the more sophisticated the filters become, and so what is expected by increasing depth is to have more precision extracting features from each layer. So in later layers rather than simple shapes, the filters may be able to detect more complicated objects and patterns.

Batch The `batch_size` is the number of samples given to the algorithm from the training set to train the network. In few words, it is the number of pictures that are propagated through the Neural Network in a single step of training. The smaller this number is, the faster is the Network to be trained, but will a smaller batch we penalize precision. Batches are taken at random, and this contributes to the non-determinism of the Neural Network.

Loss Function As loss function we use the **cathegorical_crossentropy**, mainly used for multi-class classification, where a single object belongs to only one of the classes, like in our case.

Dropout The dropout operation is what makes a CNN unpredictable: this is the probability to cut off a neuron from the Network, and it is used to avoid overfitting.

More hyperparameters exist, of course, but these are those I took in consideration during my tests.

4 Tests

Even knowing the specific function of each parameter, the result of a training can only be hypotized, so the most reliable way to configure these parameters for a specific predictive modeling problem is via systematic experimentation, that is what I did.
For the tests I took in consideration:

- Dropout Position: between convolutional and max-pool layers or as last layer.
- Batch number: starting with 32, then increasing or decreasing.
- Depth: starting with 4, then increasing or decreasing.
- Number of Filters: starting with 16, then halfing or doubling. (base_width).
- Dropout Probability: starting with 0.4, then increasing or decreasing.

All the tests are done using **30 epochs** for training.

Tests were performed on Google Colab platform, with limited specifications and usable memory.

4.1 Test 1 - Dropout Position

Two tests, changing the dropout position and observing a big increase of accuracy.

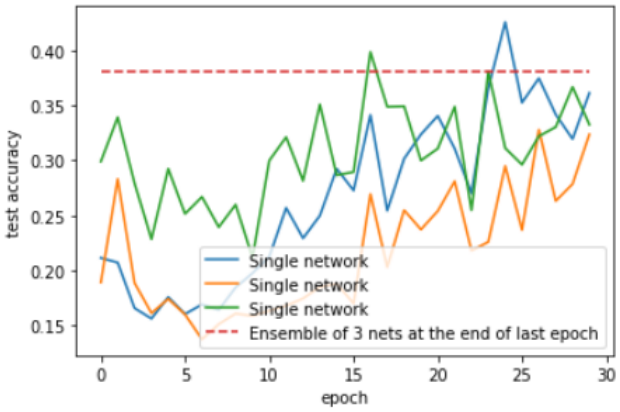
4.1.1 Test 1.1

In the first test I started putting the dropout layer between each convolutional and max-pooling layer. The test resulted in very low accuracy, maybe because a lot of information has been loss from each layer.

Dropout Position	Batch Number	Depth	Number of Filters	Dropout Prob
Between	32	4	16	0.4

Dropout Position	Batch Number	Depth	Number of Filters	Dropout Prob
End	32	4	16	0.4

```
Single models test accuracy:  [0.36131995916366577, 0.32372596859931946, 0.3324979245662689]
Ensemble test accuracy:  0.3805346700083542
```

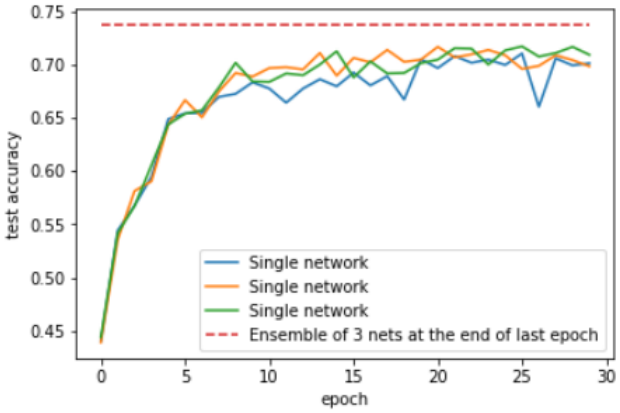


This model is clearly instable.

4.1.2 Test 1.2

By changing the Dropout position I obtaind a considerable increasing of accuracy.

```
Single models test accuracy:  [0.7017543911933899, 0.6988304257392883, 0.7096908688545227]
Ensemble test accuracy:  0.7376775271512114
```



4.2 Test 2 - Changing number of batches

Maintaining the successful results from Test 1.2, I tried both to half the batch number and also to increase it by its half: decreasing the number of batches caused a small drop in accuracy, while increasing it caused a small increase. This gap is suprisingly symmetric.

4.2.1 Test 2.1

Decreased the batch number from 32 to 16.

Dropout Position	Batch Number	Depth	Number of Filters	Dropout Prob
End	16	4	16	0.4

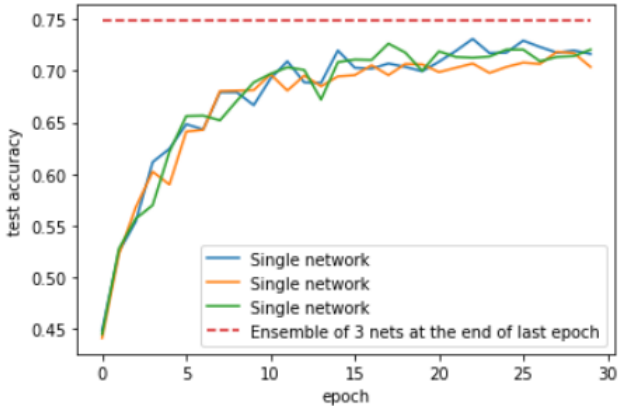
```
Single models test accuracy:  [0.6850459575653076, 0.7030075192451477, 0.6908938884735107]
Ensemble test accuracy:  0.7276524644945698
```

4.2.2 Test 2.2

Batch number increased from 32 to 48: small accuracy increase noticed.

Dropout Position	Batch Number	Depth	Number of Filters	Dropout Prob
End	48	4	16	0.4

```
Single models test accuracy:  [0.7163742780685425, 0.7038429379463196, 0.7205513715744019]
Ensemble test accuracy:  0.7485380116959064
```



4.3 Test 3 - Changing depth

Since depth is important for the filters development, I expected that augmenting depth would have increase accuracy, and so happened. For the tests I tried both by decreasing and increasing depth of one unit. Except for depth, I maintained the hyperparameters from Test 2.2.

4.3.1 Test 3.1

Depth decreased from 4 to 3. Accuracy dropped!

Dropout Position	Batch Number	Depth	Number of Filters	Dropout Prob
End	48	3	16	0.4

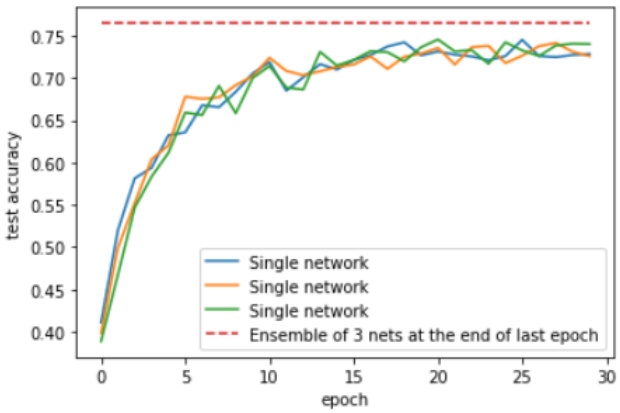
```
Single models test accuracy:  [0.6691729426383972, 0.670426070690155, 0.6875522136688232]
Ensemble test accuracy:  0.7017543859649122
```

4.3.2 Test 3.2

As hypotized, augmenting the depth caused a rais of the accuracy.

```
Single models test accuracy:  [0.7284879088401794, 0.725981593132019, 0.7401837706565857]
Ensemble test accuracy:  0.7648287385129491
```

Dropout Position	Batch Number	Depth	Number of Filters	Dropout Prob
End	48	5	16	0.4



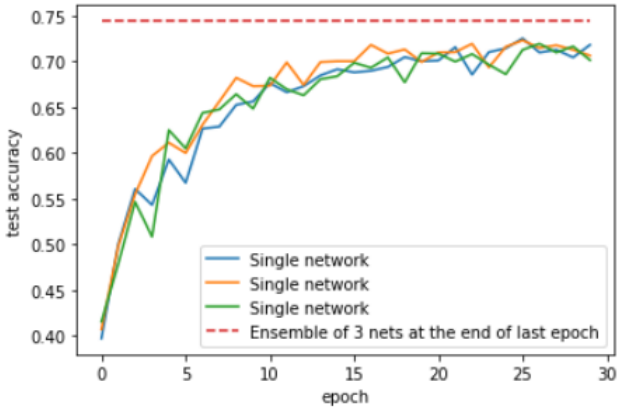
4.4 Test 4 - Different Filter Number

Maintaining the settings from the successful Test 3.2, I observed a big drop of accuracy by reducing the number of filters from 16 to 8, so I took the decision to increase them to 32, having a very small accuracy boost (it is impossible to say that this increasing in accuracy is due to the filter augmentation, but sureley by removing some filters I also lose accuracy).

4.4.1 Test 4.1

Dropout Position	Batch Number	Depth	Number of Filters	Dropout Prob
End	48	5	8	0.4

```
Single models test accuracy:  [0.7180451154708862, 0.7063491940498352, 0.701336681842804]
Ensemble test accuracy:  0.7439431913116124
```

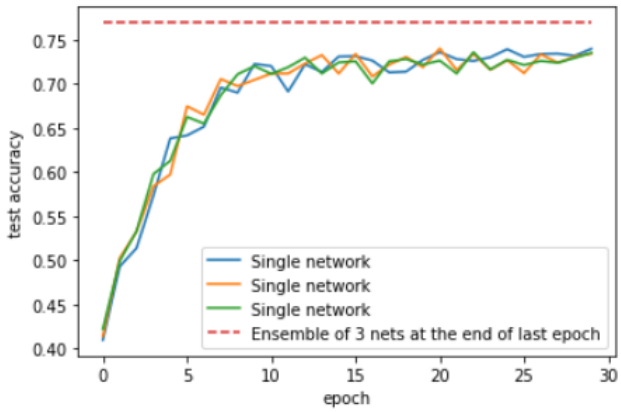


4.4.2 Test 4.2

This is the successful test.

Dropout Position	Batch Number	Depth	Number of Filters	Dropout Prob
End	48	5	32	0.4

```
Single models test accuracy:  [0.7397660613059998, 0.7343358397483826, 0.7351712584495544]
Ensemble test accuracy:  0.7694235588972431
```



4.5 Test 5 - Differences in Dropout Probability

With the params of the Test 4.2, I changed now the dropout probability. I was expecting to see an increasing of the accuracy when giving a lower

probability of dropout, however what turned out is that with an higher dropout probability the accuracy increases. This may be caused by the high presence of noisy data in the dataset.

4.5.1 Test 5.1

Dropout Position	Batch Number	Depth	Number of Filters	Dropout Prob
End	48	5	32	0.2

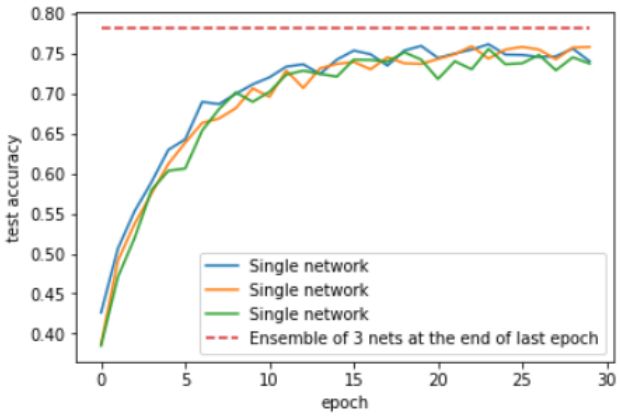
```
Single models test accuracy:  [0.6988304257392883, 0.7188805341720581, 0.7268170714378357]
Ensemble test accuracy:  0.752297410192147
```

4.5.2 Test 5.2

By increasing the dropout probability I noticed a big leap ofof accuracy.

Dropout Position	Batch Number	Depth	Number of Filters	Dropout Prob
End	48	5	32	0.6

```
Single models test accuracy:  [0.7397660613059998, 0.7577276229858398, 0.7368420958518982]
Ensemble test accuracy:  0.7807017543859649
```



4.6 Final Result

Considering that the rising the dropout probability caused an high increasing in accuracy, I tried to bring this value at 0.8 and the result was that I reached 0.8 accuracy, also by augmenting the batch number to 64. This is the test I used for the final result.

Dropout Position	Batch Number	Depth	Number of Filters	Dropout Prob
End	64	5	32	0.8

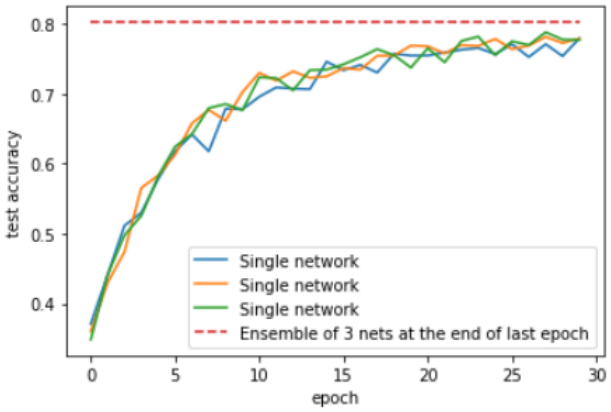
Epochs from 1 to 5:

```
Epoch 1/30
 2/75 [..............................] - ETA: 7s - loss: 6.4394 - funct
75/75 [==============================] - 15s 201ms/step - loss: 5.7404
Epoch 2/30
75/75 [==============================] - 14s 187ms/step - loss: 5.0043
Epoch 3/30
75/75 [==============================] - 14s 189ms/step - loss: 4.5110
Epoch 4/30
75/75 [==============================] - 14s 188ms/step - loss: 4.0935
Epoch 5/30
75/75 [==============================] - 14s 187ms/step - loss: 3.7997
```

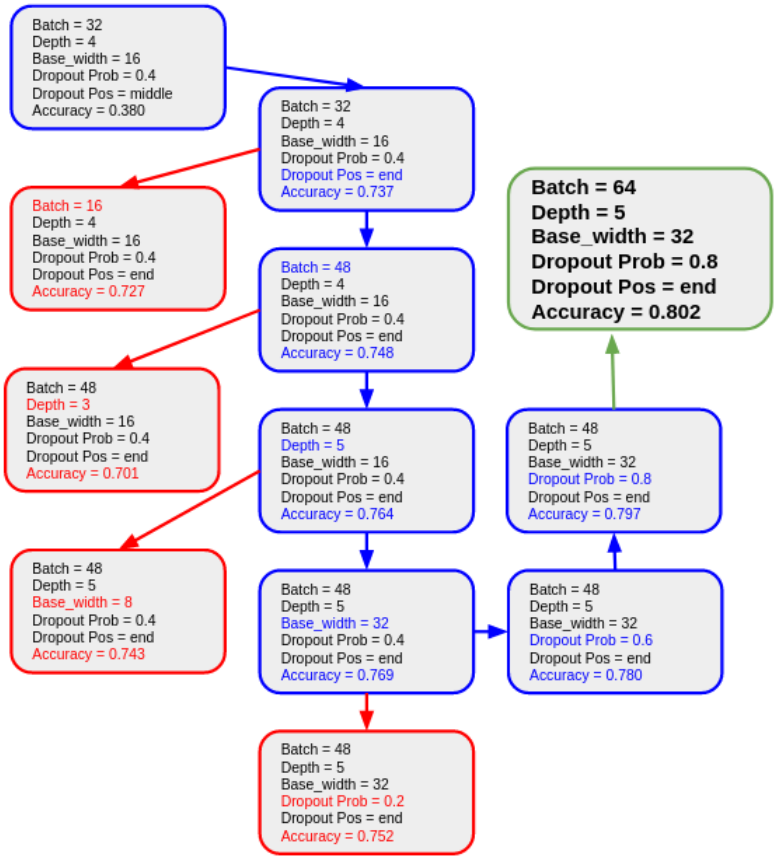
Epochs from 25 to 30:

```
Epoch 25/30
75/75 [==============================] - 14s 186ms/step - loss: 0.7223
Epoch 26/30
75/75 [==============================] - 14s 187ms/step - loss: 0.6861
Epoch 27/30
75/75 [==============================] - 14s 187ms/step - loss: 0.6445
Epoch 28/30
75/75 [==============================] - 14s 187ms/step - loss: 0.6198
Epoch 29/30
75/75 [==============================] - 14s 187ms/step - loss: 0.5291
Epoch 30/30
75/75 [==============================] - 14s 187ms/step - loss: 0.5070
```

```
Single models test accuracy:  [0.7794486284255981, 0.7790309190750122, 0.7769423723220825]
Ensemble test accuracy:  0.802422723475355
```


0.8
0.7
0.6
0.5
0.4
test accuracy
0
5
10
15
20
25
30
epoch
Single network
Single network
Single network
Ensemble of 3 nets at the end of last epoch

4.7 Test Tree


Batch = 32
Depth = 4
Base_width = 16
Dropout Prob = 0.4
Dropout Pos = middle
Accuracy = 0.380
Batch = 32
Depth = 4
Base_width = 16
Dropout Prob = 0.4
Dropout Pos = end
Accuracy = 0.737
Batch = 16
Depth = 4
Base_width = 16
Dropout Prob = 0.4
Dropout Pos = end
Accuracy = 0.727
Batch = 48
Depth = 4
Base_width = 16
Dropout Prob = 0.4
Dropout Pos = end
Accuracy = 0.748
Batch = 64
Depth = 5
Base_width = 32
Dropout Prob = 0.8
Dropout Pos = end
Accuracy = 0.802
Batch = 48
Depth = 3
Base_width = 16
Dropout Prob = 0.4
Dropout Pos = end
Accuracy = 0.701
Batch = 48
Depth = 5
Base_width = 16
Dropout Prob = 0.4
Dropout Pos = end
Accuracy = 0.764
Batch = 48
Depth = 5
Base_width = 32
Dropout Prob = 0.8
Dropout Pos = end
Accuracy = 0.797
Batch = 48
Depth = 5
Base_width = 8
Dropout Prob = 0.4
Dropout Pos = end
Accuracy = 0.743
Batch = 48
Depth = 5
Base_width = 32
Dropout Prob = 0.4
Dropout Pos = end
Accuracy = 0.769
Batch = 48
Depth = 5
Base_width = 32
Dropout Prob = 0.6
Dropout Pos = end
Accuracy = 0.780
Batch = 48
Depth = 5
Base_width = 32
Dropout Prob = 0.2
Dropout Pos = end
Accuracy = 0.752

5 Conclusion

Even if it seems that increasing always brings to better performances, it is also true that a compromise has to be reached: a result of 0.8 in accuracy can not be perfect, but it is important to notice that this has been achieved with only 30 epochs of training.
Of course it is possible to max all the hyperparameters, but this would cost too much for training the Neural Network.

It is always about tradeoff.