Training of a Convolutional Neural Network for the MNIST classification problem

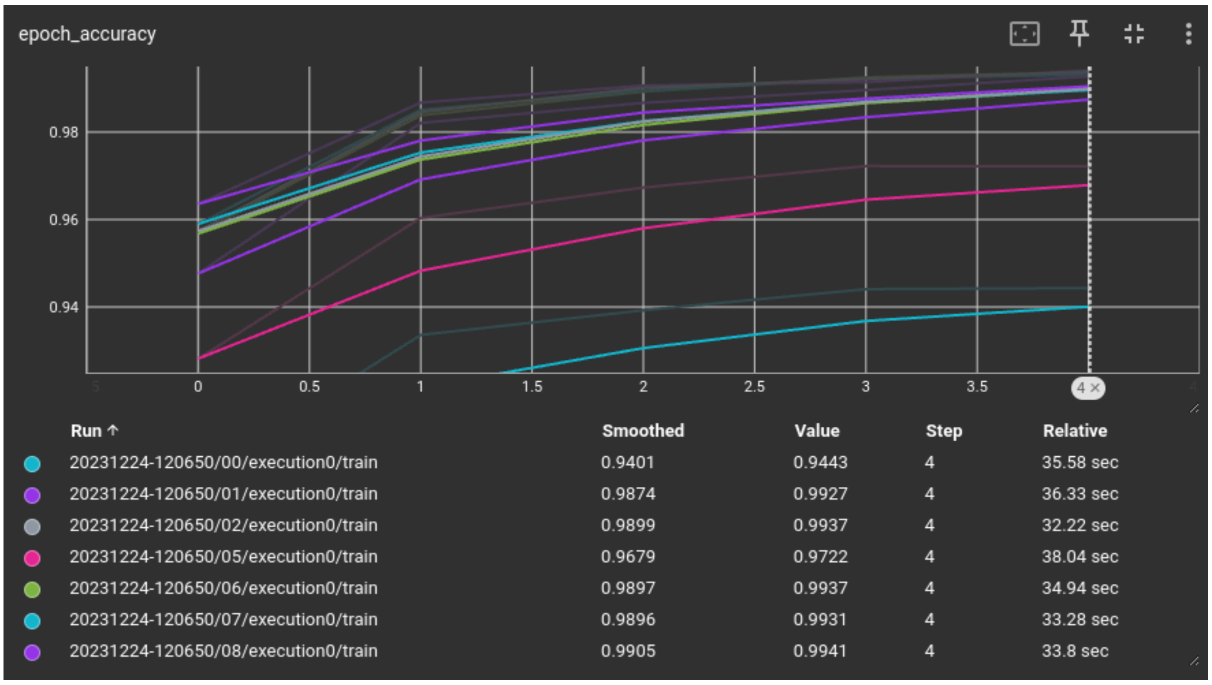
How the model is obtained

With the function “build_model”, we can create a CNN, depending on various hyperparameters.
These hyperparameters are:

- The number of filters used in the Convolutional layers (32 to 128)
- The number of Convolutional layers (1 to 3)
- The number of Dense layers (0 to 2)
- The number of units per dense layer (32 to 256)

We have performed the search using 5 epochs in each CNN.
Let’s check the evolution of metrics during the search:

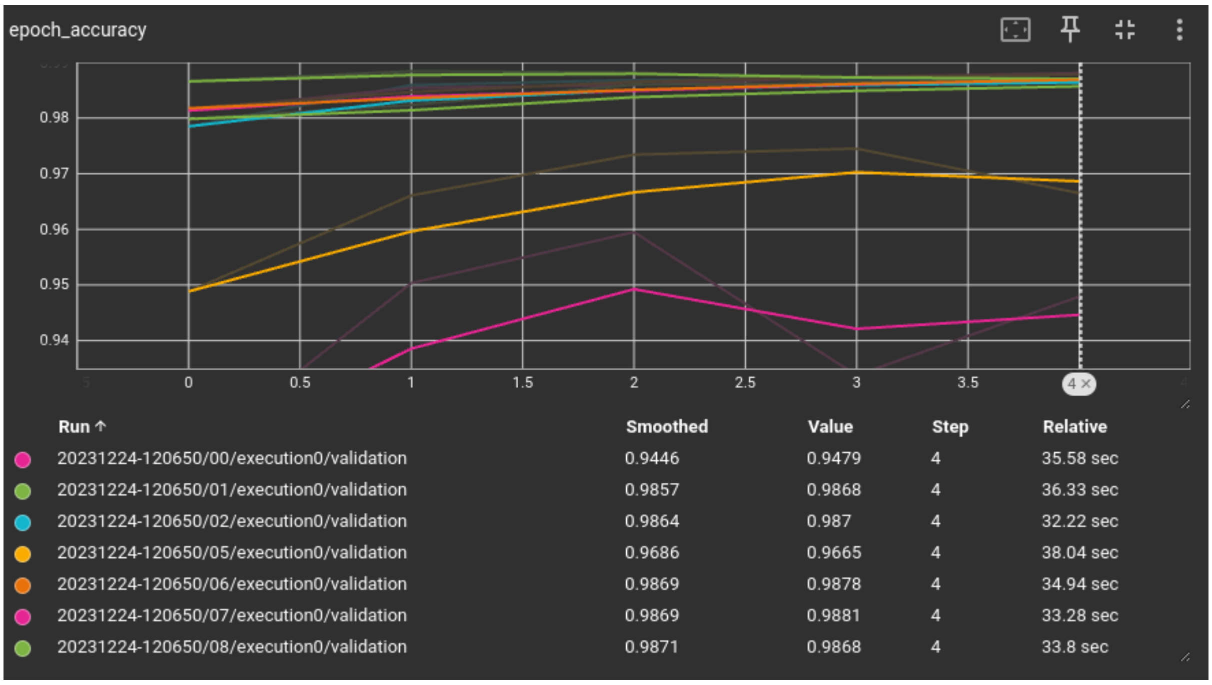
Training accuracy



	Run ↑	Smoothed	Value	Step	Relative
●	20231224-120650/00/execution0/train	0.9401	0.9443	4	35.58 sec
●	20231224-120650/01/execution0/train	0.9874	0.9927	4	36.33 sec
●	20231224-120650/02/execution0/train	0.9899	0.9937	4	32.22 sec
●	20231224-120650/05/execution0/train	0.9679	0.9722	4	38.04 sec
●	20231224-120650/06/execution0/train	0.9897	0.9937	4	34.94 sec
●	20231224-120650/07/execution0/train	0.9896	0.9931	4	33.28 sec
●	20231224-120650/08/execution0/train	0.9905	0.9941	4	33.8 sec

As it is expected, all of the models get a higher training accuracy on each epoch. This should always happen, since the model is adjusting itself to this set.

Validation accuracy

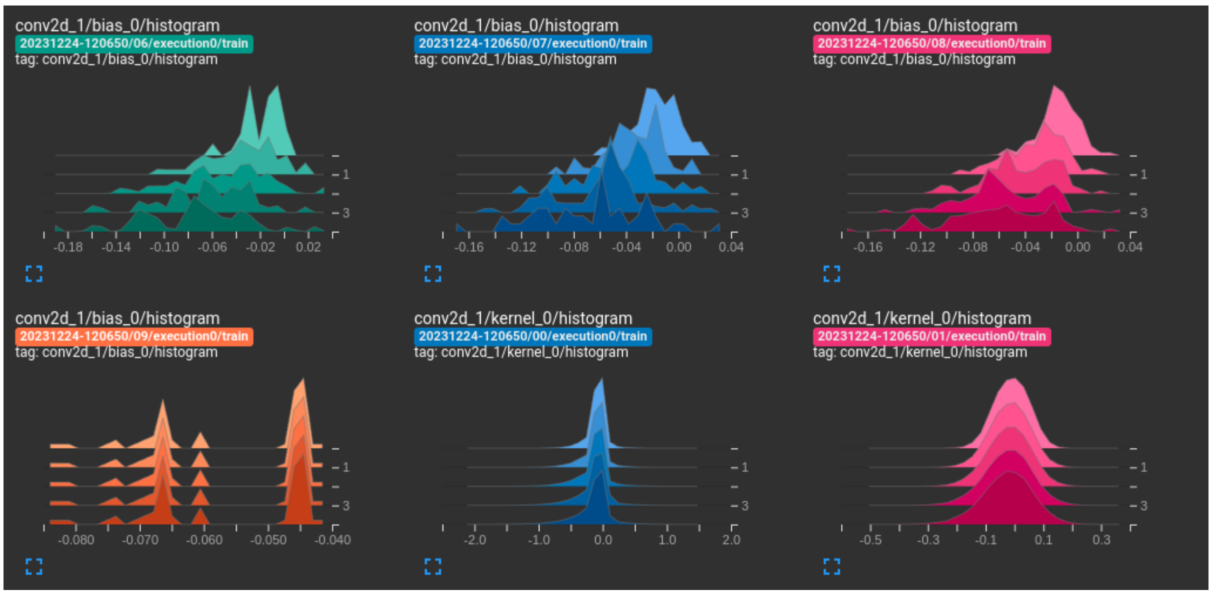


	Run ↑	Smoothed	Value	Step	Relative
●	20231224-120650/00/execution0/validation	0.9446	0.9479	4	35.58 sec
●	20231224-120650/01/execution0/validation	0.9857	0.9868	4	36.33 sec
●	20231224-120650/02/execution0/validation	0.9864	0.987	4	32.22 sec
●	20231224-120650/05/execution0/validation	0.9686	0.9665	4	38.04 sec
●	20231224-120650/06/execution0/validation	0.9869	0.9878	4	34.94 sec
●	20231224-120650/07/execution0/validation	0.9869	0.9881	4	33.28 sec
●	20231224-120650/08/execution0/validation	0.9871	0.9868	4	33.8 sec

Now, due to the model is not getting adjusted to this data, it is normal that the accuracy decreases sometimes.

For that reason, we want to use the model that does not have this decrease, because it probably is having a better generalization. Though, we have to assume the risk that this may imply a very high overfitting, not just to the training set, but also to the validation one.

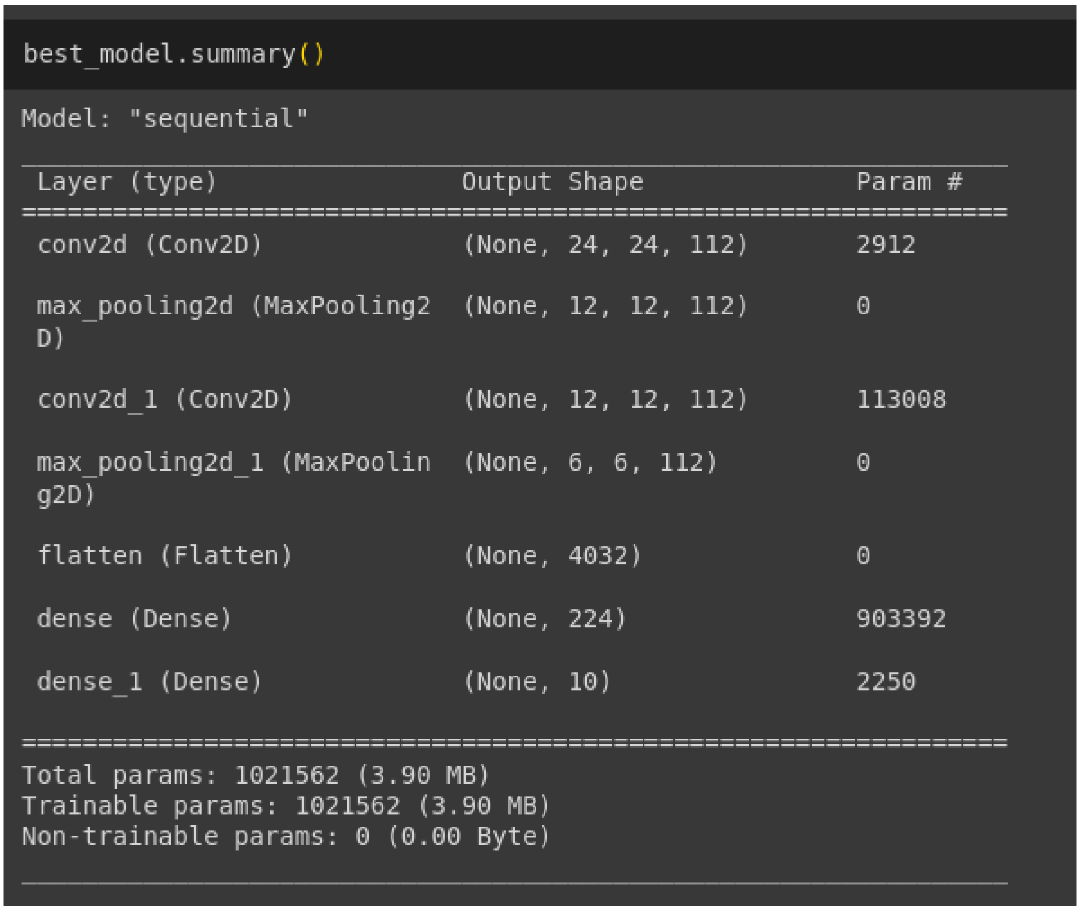
Another insight that can be obtained via TensorBoard is the histogram of hyperparameters' "quality" (acc):



The fact that most of them have a, relatively, normal distribution, is a good indicator.
This means that the Law of Great Numbers is applying to our hyperparameters, and that the best result that we obtain is probably not a particular, outlier case, but a general, mean case, that is prone to generalization.

Model obtained

After executing the random search, the best model obtained is the following one:



```
best_model.summary()

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 24, 24, 112)       2912

 max_pooling2d (MaxPooling2  (None, 12, 12, 112)       0
 D)

 conv2d_1 (Conv2D)           (None, 12, 12, 112)       113008

 max_pooling2d_1 (MaxPoolin  (None, 6, 6, 112)         0
 g2D)

 flatten (Flatten)           (None, 4032)              0

 dense (Dense)               (None, 224)               903392

 dense_1 (Dense)             (None, 10)                2250

=================================================================
Total params: 1021562 (3.90 MB)
Trainable params: 1021562 (3.90 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In this particular case, we obtained as the best model one that uses one internal neural layer, apart of the final dense layer for the output.

This suggests an important finding; this particular dataset behaves very well when being treated with filters, as it can be inferred that it should be common among images, since they do not behave like traditional data.

Obviously, this doesn't mean that neural perceptrons are useless, at all.
This just means that the best model this search could find relies on convolution.

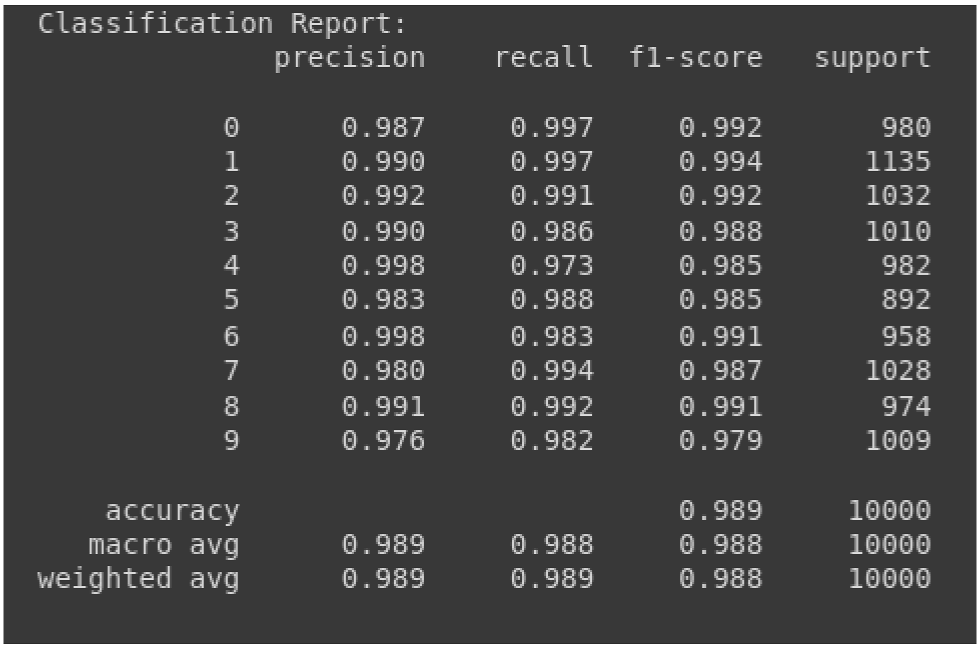
When computing the accuracy on the test set, we obtain the following result: 0.989. This is a very good result, ensuring us that there isn't much overfitting nor any other big problem that could have been originated.

Confusion Matrix

The confusion matrix obtained from the best model previously mentioned would be:

	0	1	2	3	4	5	6	7	8	9
0	997	0	0	0	0	0	1	1	0	1
1	0	1132	2	1	0	0	0	0	0	0
2	1	1	1023	1	0	0	0	5	1	0
3	0	0	2	996	0	6	0	2	2	2
4	1	4	0	0	955	0	1	2	2	17
5	2	0	0	7	0	881	0	1	0	1
6	8	2	0	0	0	4	942	0	2	0
7	0	1	2	1	0	0	0	1022	1	1
8	1	0	2	0	0	2	0	1	966	2
9	0	3	0	0	2	3	0	9	1	991

We can see that, apart from the main diagonal, there are very low numbers. This is a good indicator, but let's get in detail to see more information:



```
Classification Report:
              precision    recall  f1-score   support

           0      0.987     0.997     0.992       980
           1      0.990     0.997     0.994      1135
           2      0.992     0.991     0.992      1032
           3      0.990     0.986     0.988      1010
           4      0.998     0.973     0.985       982
           5      0.983     0.988     0.985       892
           6      0.998     0.983     0.991       958
           7      0.980     0.994     0.987      1028
           8      0.991     0.992     0.991       974
           9      0.976     0.982     0.979      1009

    accuracy                          0.989     10000
   macro avg      0.989     0.988     0.988     10000
weighted avg      0.989     0.989     0.988     10000
```

If we wanted, we could get into more detail to analyze what each of these metrics is indicating.

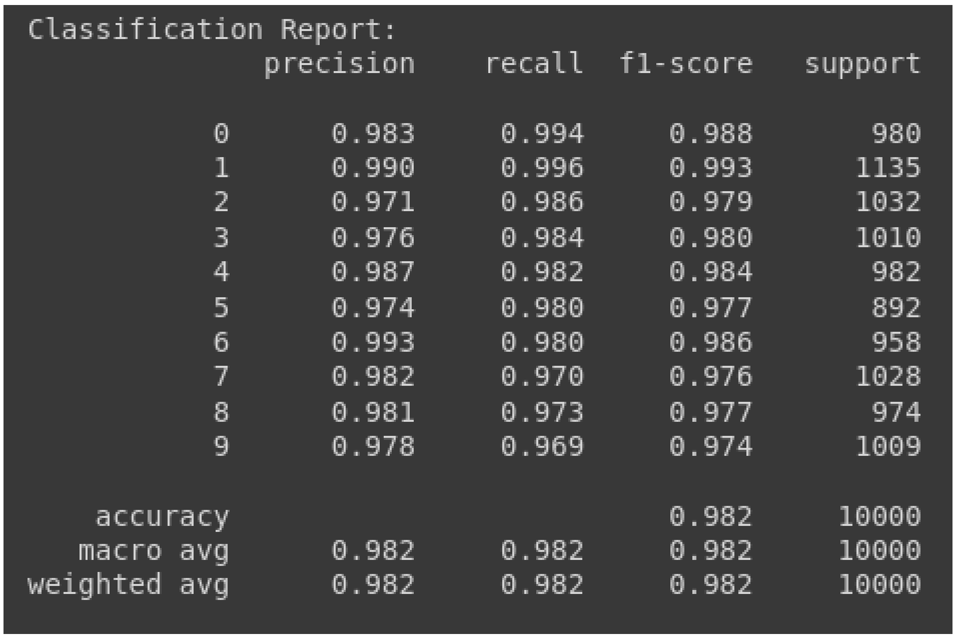
Though, the main conclusion we can obtain from these scores is that our model is having good behavior on our test set, and that it has probably learned relevant information about our training set.

Using features from SVM

If we extract the information being obtained after the convolution, in the Flatten layer, as we can see in the model's summary, we obtain features with 4032 dimensions.

This is a very high dimensionality, so it might be correct to perform a PCA in order to extract some of the most relevant information, and not feed the SVM with too many dimensions that it can not handle correctly.

Also, since SVM is based on distances, it might be a good idea to standardize the data.
These are the results obtained using the SVM:



```
Classification Report:
              precision    recall  f1-score   support

           0      0.983     0.994     0.988       980
           1      0.990     0.996     0.993      1135
           2      0.971     0.986     0.979      1032
           3      0.976     0.984     0.980      1010
           4      0.987     0.982     0.984       982
           5      0.974     0.980     0.977       892
           6      0.993     0.980     0.986       958
           7      0.982     0.970     0.976      1028
           8      0.981     0.973     0.977       974
           9      0.978     0.969     0.974      1009

    accuracy                          0.982     10000
   macro avg      0.982     0.982     0.982     10000
weighted avg      0.982     0.982     0.982     10000
```

This score would have been very difficult to obtain if we had developed the features ourselves, so it is a big advance.

Despite the fact that we still get better results with the trained CNN, the use of these features could be very interesting, specially for feature comprehension, or to use this as a pretrained model in order to solve another similar, but not exactly equal, classification task.

Also, despite not being the purpose of this notebook, fine-tuning the SVM might get even better results.