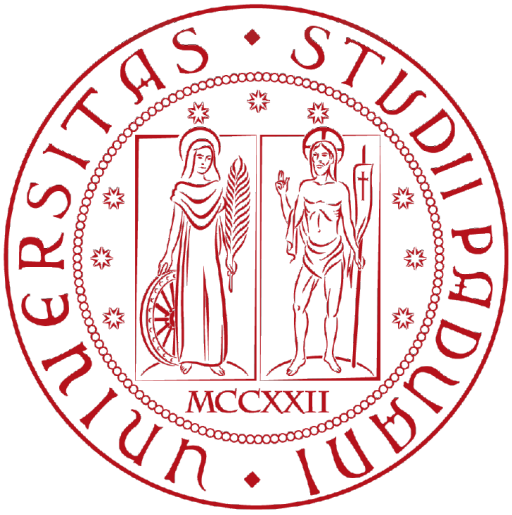


University of Padua

Department of Mathematics

Cognition and Computation Essay

10-02-21

MATLAB simulation to explore a computational model of visual concept learning

Teachers:
Alberto Testolin
Marco Zorzi

Author:
(2019182) Nicola Scremin

Contents

1 Introduction

In my experience I decided to use EMNIST database to train and test my network. It consists in a set of handwritten characters and converted to a 28x28 pixel image. This database match perfectly with MNIST, the unique difference is that the digits are handwritten. In particular EMNIST is subdivided in many databases:

1. EMNIST ByClass: 814,255 characters. 62 unbalanced classes. 731,668 training samples and 82,58 testing samples
2. EMNIST ByMerge: 814,255 characters. 47 unbalanced classes. 731,668 training samples and 82,587 testing samples
3. EMNIST Balanced: 131,600 characters. 47 balanced classes.
4. EMNIST Letters: 145,600 characters. 26 balanced classes.
5. EMNIST Digits: 280,000 characters. 10 balanced classes.
6. EMNIST MNIST: 70,000 characters. 10 balanced classes.

The full complement of the NIST Special Database 19 is available in the ByClass and ByMerge splits. The EMNIST Balanced dataset contains a set of characters with an equal number of samples per class. The EMNIST Letters dataset merges a balanced set of the uppercase and lowercase letters into a single 26-class task. The EMNIST Digits and EMNIST MNIST dataset provide balanced handwritten digit datasets directly compatible with the original MNIST dataset [1].
In particular my work is based on EMNIST Letters database which contains 145600 samples divided as follow: 124800 training samples and 20800 testing samples.

2 Essay

In this section we will discuss about my implementation of neural network and about the results obtained.

2.1 Important things to say

To do my essay, I used the code presented in Lab3 (modified it where needed). In fact I had to convert EMNIST 2D database in 3D. Since Training set is composed by 124800 samples (with 784 feature for each sample 124800x784), I decided to divide it in 150 batch and 832 sample for each batch.
Moreover, in EMNIST database, labels were represented by a number (from 1 to 26). In order to use the code, I convert label in an array which contain "1" in position which correspond to its true label (ex. If a true label is **4**, the new label is an array [1 26] which contains all zeros except in position **4**). In my opinion these are the most important modifications. To see the other changes, look at the project.

2.2 Network

I decided to setup my Deep Neural Network as follow:

- **layersize** = [100 100 200 300 350]
- **nlayers** = length(DN.layersize);
- **DN.maxepochs** = 20;
- **DN.batchsize** = 150;
- **sparsity** = 1;
- **spars_factor** = 0.05;
- **epsilonw** = 0.1;
- **epsilonvb** = 0.1;
- **epsilonhb** = 0.1;
- **weightcost** = 0.0002;
- **init_momentum** = 0.5;
- **final_momentum** = 0.9;

I use 5 layers in order to see better how the network works.

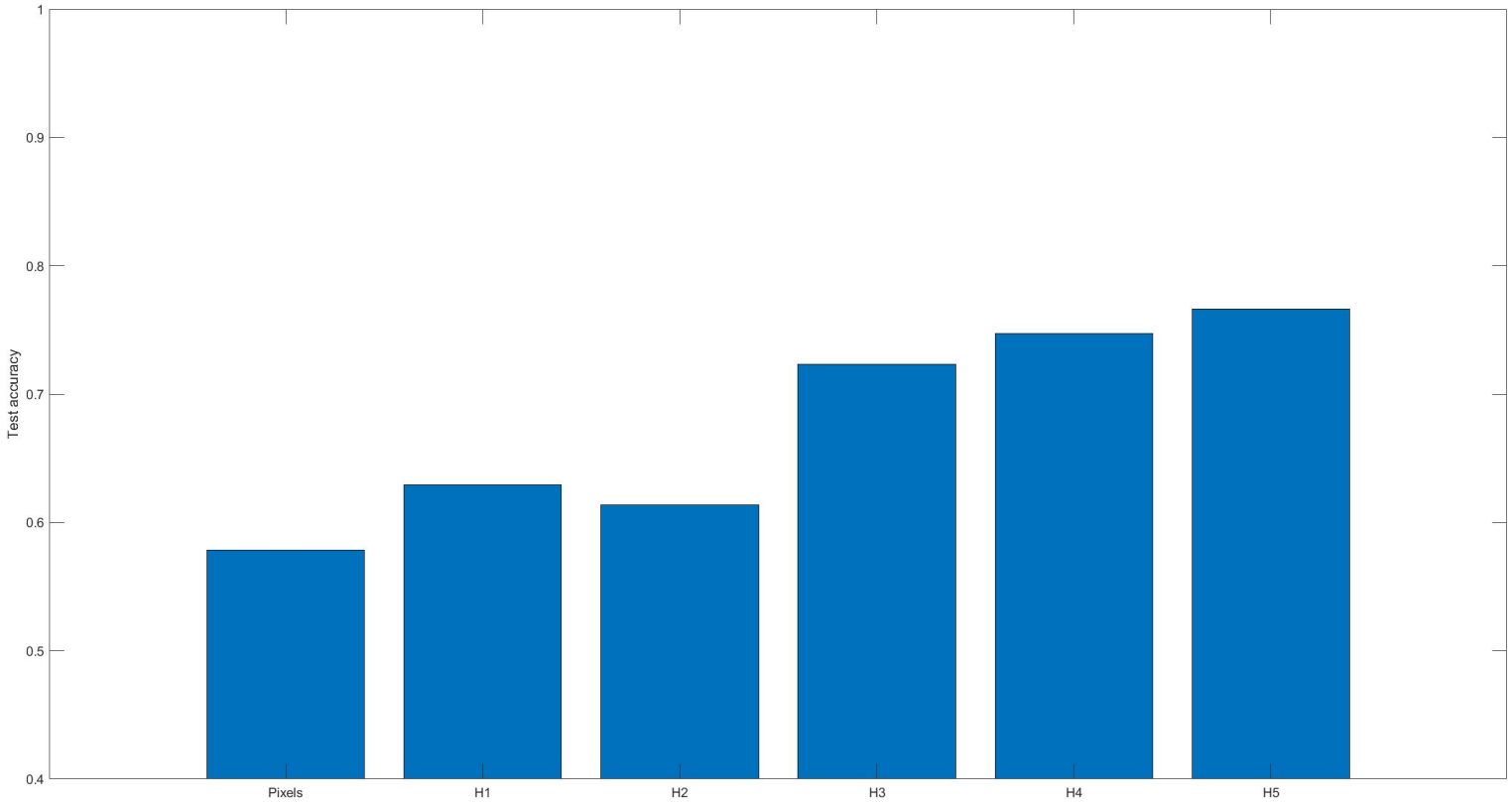


Figure 1: Accuracy based at different levels of the hierarchy, using test set

2.3 Linear read-outs at different levels of the hierarchy

In this section we will analyze the results about Linear read-outs at different levels of the hierarchy, as we can observe in Figure 1. We can easily affirm that the worst model is the one that classify images based on "raw images" (aka. pixels), it cannot generalized very well, its accuracy is 0.579. While, Model trained at level 5 can generalize very well in fact its accuracy rate is 76,6%. As we can see from the fig. 1, the accuracy on test set improve at each step except at layer 2. This could happen because at this layer the model cannot generalize the "correct" feature. Another possible reasons is that the second and third layer has the same number of neurons. I tried other combination of the network and I noticed that, every time network has two consecutive layer with the same size, the accuracy on test set does not improve (I think it is just a coincidence). Nevertheless, looking at Layer 3, the accuracy rate improves significantly and also the generalization of the features improves. Looking at these results seem that more layers mean more accuracy but this is not true. In fact we must be careful to not overfit.

Layer	Train accuracy	Test Accuracy
Pixels	0.593	0.579
H1	0.628	0.629
H2	0.622	0.617
H3	0.727	0.723
H4	0.753	0.750
H5	0.771	0.766

Table 1: Table which contains performance for each level.

2.4 Confusion matrices

In this section I will analyze the confusion matrices, in particular the confusion matrices at "raw Images", layer 1 and layer 5 in order to understand the performances and explain the difference between them. We know from 1 that using pixel to classify the test accuracy is 57,9% (that is the mean about blue samples / red samples) and it is the worst accuracy rate. At this step the model fails to classify correctly the letter "I" with the character "L". In fact, as we can see from fig. 3, the model assigns the letter "I" to the letter "L" 328 times and to character "T" 167 times. Instead, "I" is correctly classify 569 times. One possible reasons is that looking at raw images, the letter "I" can be confused with the others two letters due to their similar form. In fact also the letter "L" is usually confused with the letter "I". Moreover there are other missclassified label, as we can observe in fig. 3. For example the letter is classified "G" with "Q" and the character "P" with letter "F". Surprisingly the model generalize pretty well the difference between "M" with letters "N" and "W", which is not obvious.

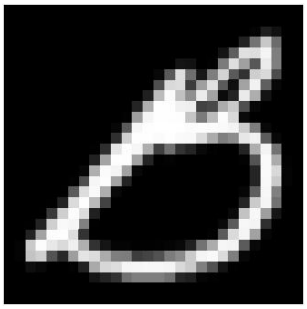


Figure 2: sample of test set which is perfectly classify in L5 and wrongly classify at L1

Fig. 4 shows the performance about layer 1. We can easily observe that, even if the performance are better (according to table 1), the model makes the same error as the last case.

In layer 5, instead, the performance increases considerably. In fact, as we can see from fig. 5, almost all the samples are correctly classified. In particular fig. 2 shows a character which is perfectly classify in L5 and wrongly classify at L1. At layer 1, it is classify with true label "D". A possible reason is that, maybe, the model recognize that there is a primitive curve associated with a primitive bar but it cannot catch the other hump. Another way is that, even if the model catches the second curve at the top of the image, it attributes a higher score to the label "D". While at layer 5 the sample is easily classified. It is correctly classified starting from Layer 3, maybe because the model "sees" there are two primitives curve associated with a bar and classify it as "B". It is important to remark that the model at layer 5 still do some errors. A possible explanation is that Maybe some letters are written in a "tricky way" and it is impossible to guess the true label, even for a human. However we could increase the hidden layers or augment the training set in order to see if the performance increase (be careful to do not overfit).

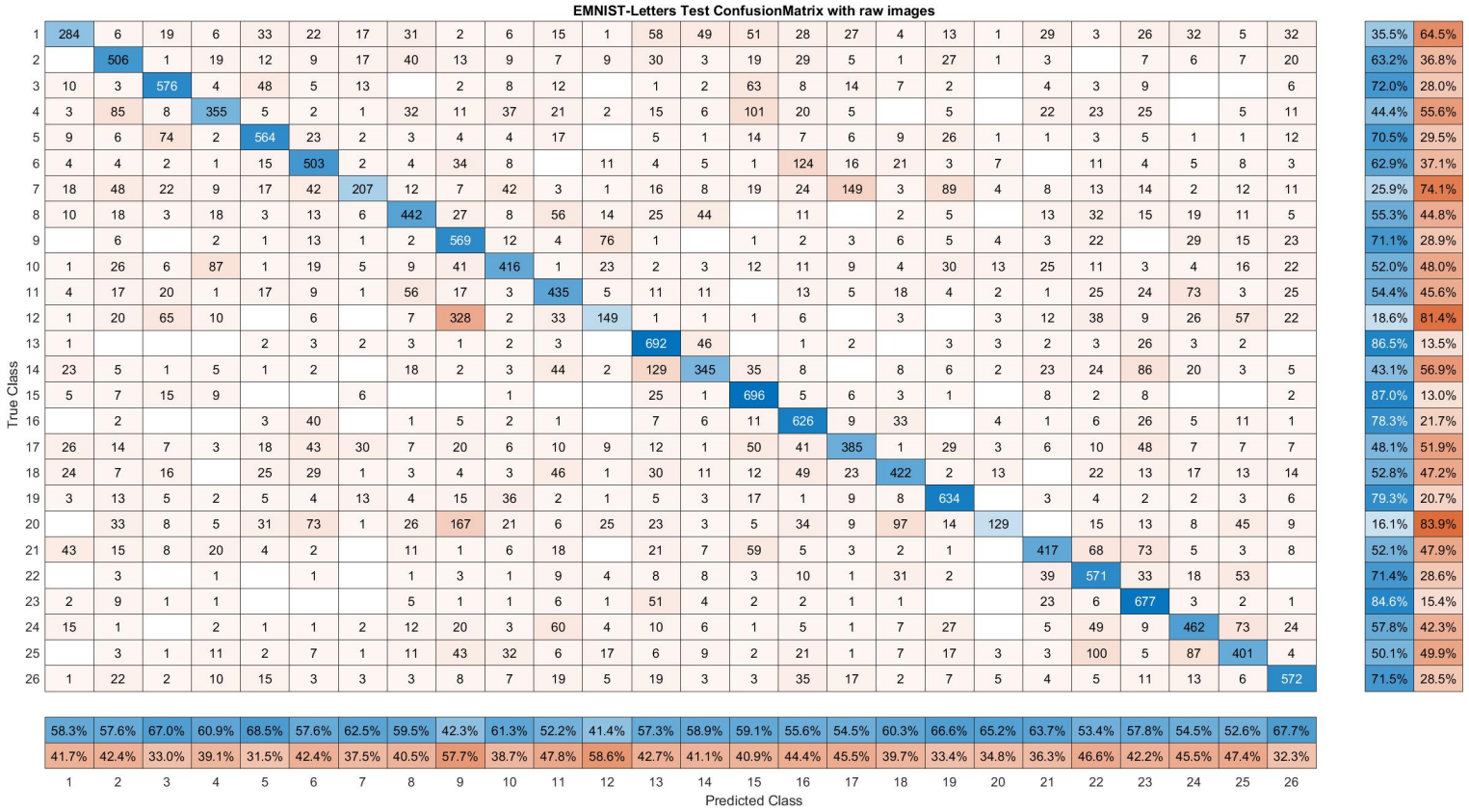


Figure 3: Confusion matrix at layer "raw images", using test set

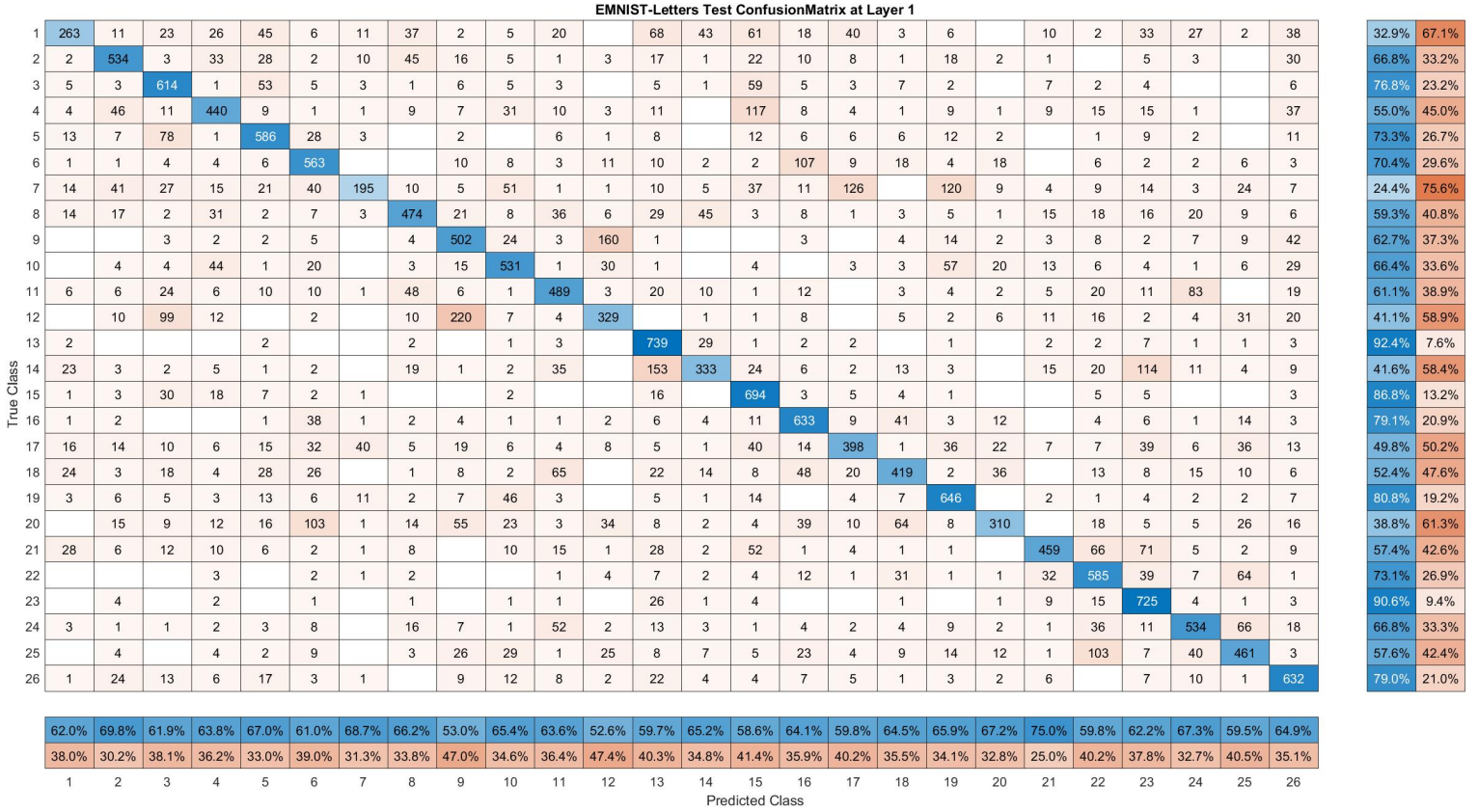


Figure 4: Confusion matrix at layer 1, using test set

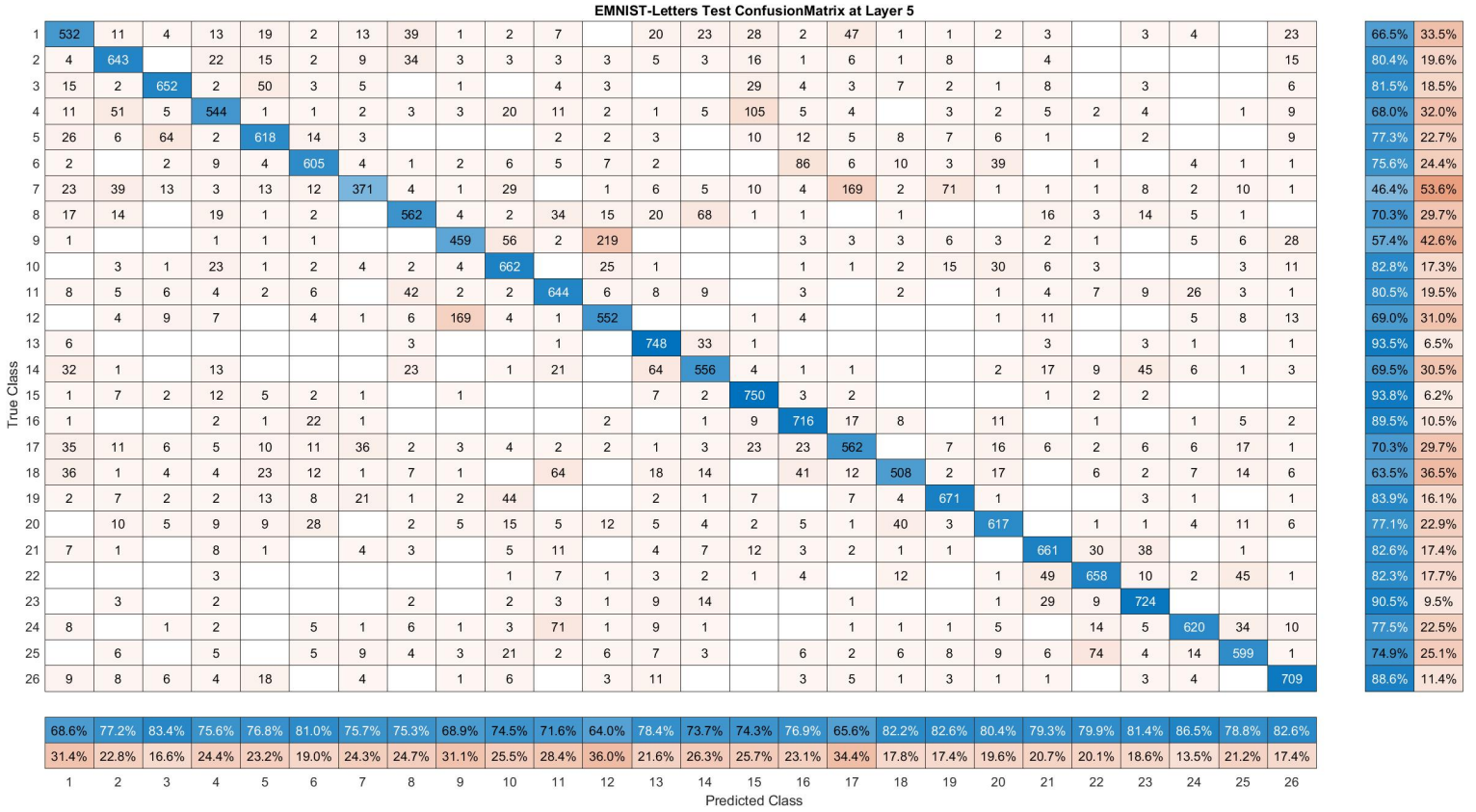


Figure 5: Confusion matrix at layer 5, using test set

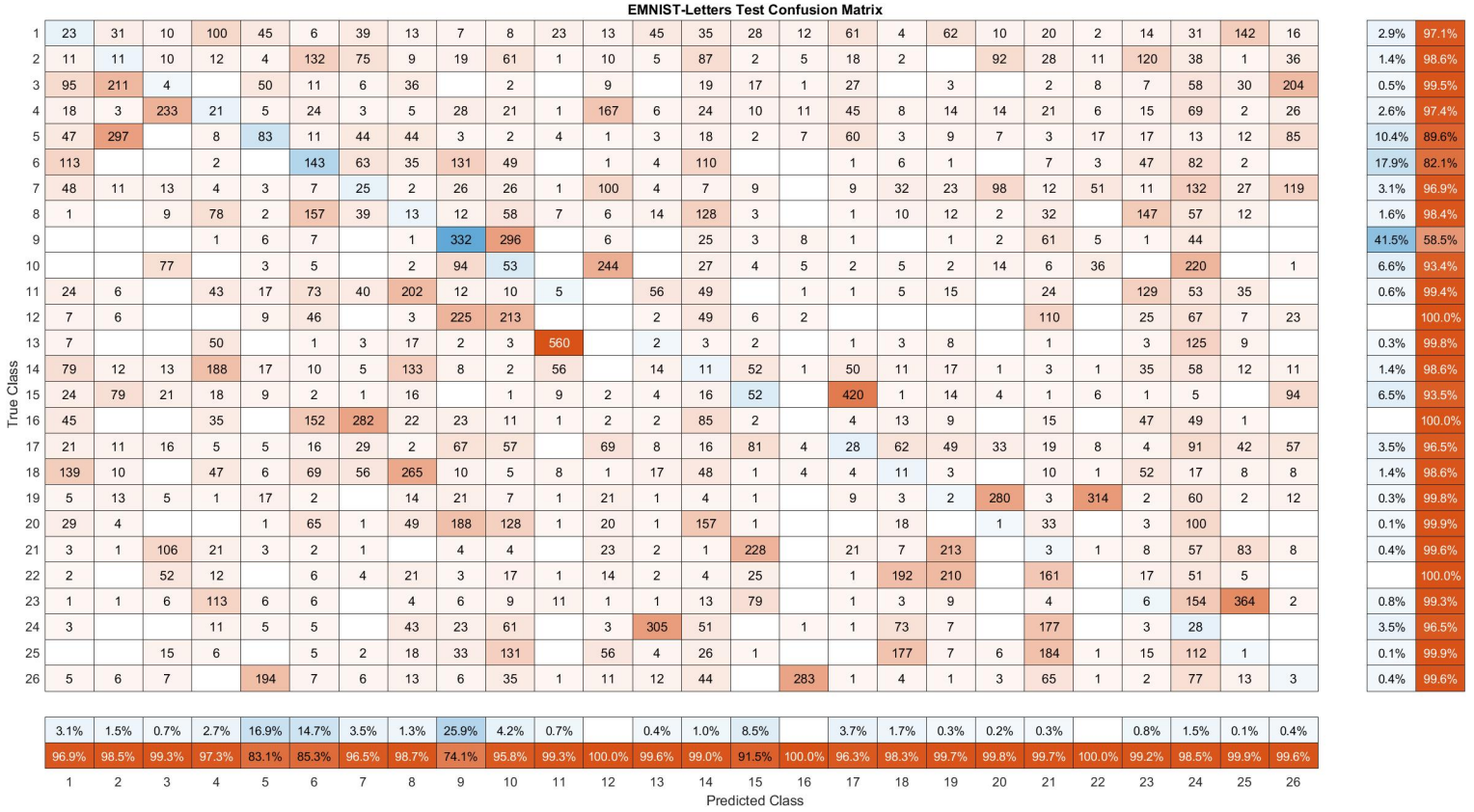
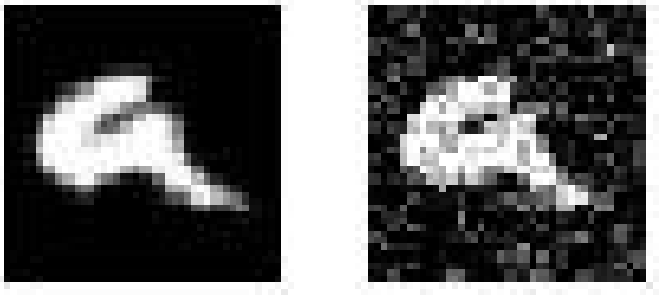


Figure 6: Confusion matrix about test set using k-means

I also tried to use k-means (with k = 26) to find another way to group test set. Moreover I wanted to compare two different approaches. K-means returns a matrix really bad as we can see in fig. 6, almost all samples are missclassified. This could happen because a lot of samples contains the same structure and k-means cannot generalize very well. In order to see the best k for k-means we should use Silhouette which attributes a score to each k (and pick the best one). Almost surely that Silhouette won't return k = 26 because the matrix is really bad. If k is different from 26 it means that samples are not clustered as "letter" but with others characteristics. Therefore the model above is always better than k-means.



(a) (b)

Figure 7: fig. (a) shows the image without noise, while in fig. (b) is represented the same character with Gaussian noise

2.5 Effect of adding noise to the test images

In my experience I decided to add Gaussian noise with mean 0 and variance 0.05. In Fig. 7 is represented how one letter changes after added the noise. To see the performance after added the noise we have to look at fig.8. We easily see how the performance decreases. Since there is noise if we try to classify with pixel of course we cannot generalize very well, model thinks "all samples are the same". Another possible explanation is that model trained how to classify a patter pretty well but, introducing Gaussian noise, I create "weird thing" that the model interprets as primitive and this is why it cannot classify the samples correctly.

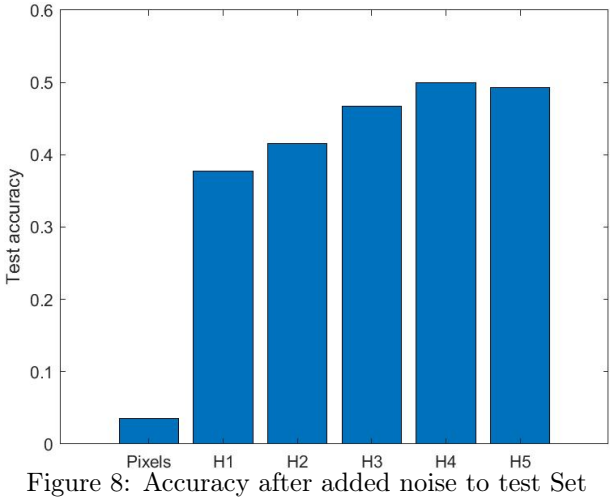


Figure 8: Accuracy after added noise to test Set

Since in H5 the performance on test set decrease, while performance in training set improves (seen by code), the model is maybe overfitting. I decided to see how my network works with 6 layers and I saw that the test set further improves (Use code to check it) and does not decrease. Hence my model is not overfitting (up to now).

Another possible exploration is to see the accuracy on the test set but, in this case, with a neural network trained with a noisy Training set. I added the same noise error as before (0 mean, 0.05 variance) and the final performance are observable in fig. 9. It is easily to see that the results are better. In particular the model understood how to classify the raw images. This could happen because the model learnt how to "classify the noise" since training set contains Gaussian noise too. Moreover all the performance increases considerably (respect the last case).

I also see the new confusion matrix and they are really bad. In particular model does not make the same error as human (for example confusing "N" with "M"). For example, for the model is really hard to recognize letters "I" and "L" because with a little bit of noise these 2 characters are "transformed" in others (I noticed it plotting the Confusion matrix in the relative code).

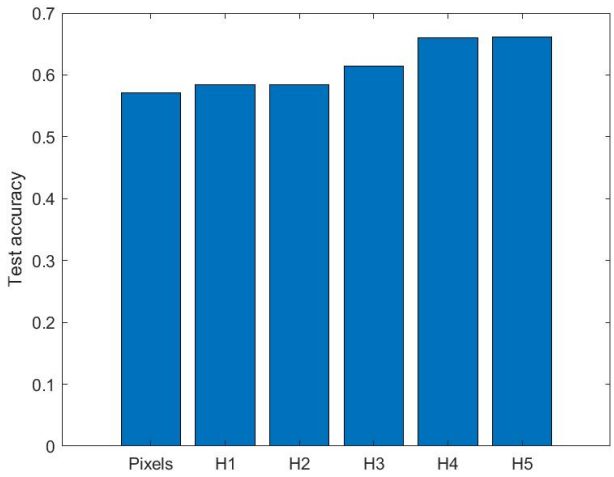


Figure 9: Accuracy after added noise to test Set. In this case my network is trained with noisy Train set

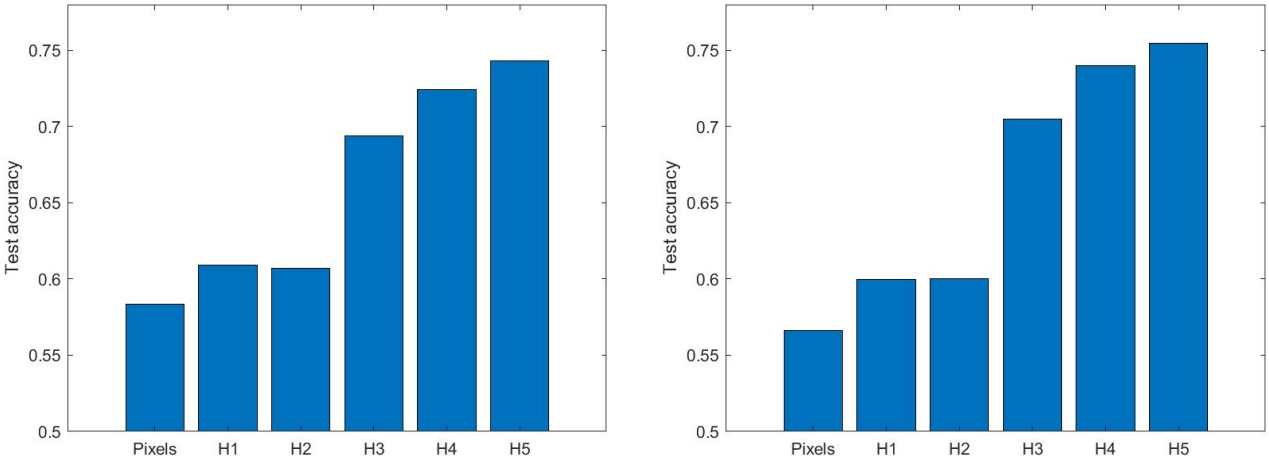


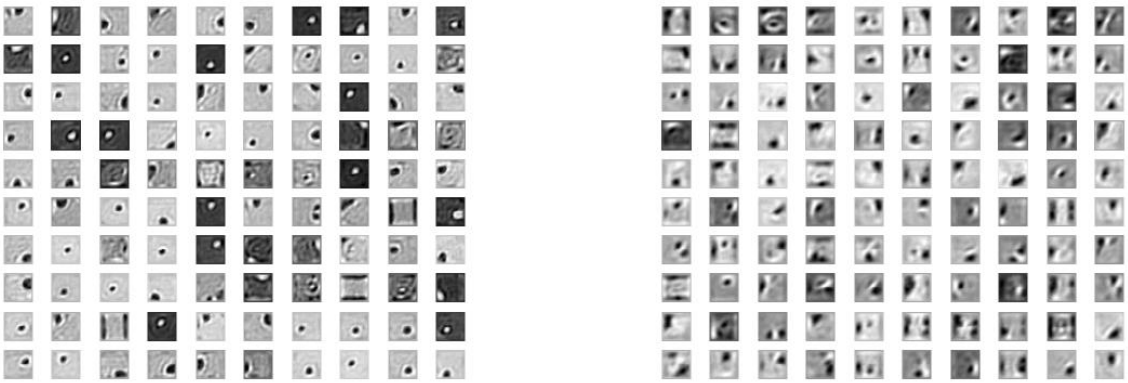
Figure 10: fig. (a) shows the accuracy using test set without noise, while in fig. (b) is represented the accuracy using test set with noise

2.6 Data Augmentation

I decided to learn my model with both Training set and Noisy Training set (249600 samples) in order to investigate how the performances change. In fig. 10(a) we can observe the performance given by test set with no noise. In this case we can affirm that the performance are really good and they are really similar to accuracy in fig. 1 but a little bit less. This could happen because training the network with both training set and noise training set some characters can be confused (due to noise, maybe letter "I" looks a "T").
On the other side, the network can classify really well the samples even if test set is composed with only noise characters. In fact, as we can see in fig. 10(b) the performance improve exponentially. Model learned how to classify both noise and no noise character (indeed the accuracy are more or less the same).

2.7 Explore receptive fields

In fig. 11 and 12 we see the receptive field for layer 1, 3 and 5. At layer 1 the receptive fields are tuned to capture "poor" visual features (they look like primitives). Every neuron is able to capture just a circle, bar, ecc... At layer 3, instead, the receptive fields are becoming more abstract and look similar to our characters. At last layer 5 the receptive fields improve and, for example, neuron in position [2 2] seems it can identify the letter "B". In any case we know, from tab. 1, that the two accuracy (layer 3 and 5) are more or less the same and, that's why the receptive fields could look similar to us. Nevertheless receptive fields at layer 5 are more abstract.
I also looked at receptive field for the network trained with Noisy training set (to check load **DataAugementation_DBN_300.mat**) and, in this case, the receptive fields seem as if they contained noise.



(a) (b)

Figure 11: fig. (a) shows the receptive field at layer1, while in fig. (b) is represented receptive field at layer 3

Figure 12: Receptive field at layer 5

Moreover I explored the connection weights and they weren't too big (in absolute value) but I did not know what to plot and discuss. [2] [3] [**6**] [4] [5]

REFERENCES

[1] Gregory Cohen et al. "EMNIST: an extension of MNIST to handwritten letters". In: *arXiv preprint arXiv:1702.05373* (2017).

[2] MultiMedia LLC. *MS Windows NT Kernel Description*. 1999. URL: http://web.archive.org/web/20080207010024/http://www.808multimedia.com/winnt/kernel.htm (visited on 09/30/2010).

[3] URL: https://gist.github.com/Neo23x0/e4c8b03ff8cdf1fa63b7d15db6e3860b.

[4] URL: https://nvd.nist.gov/vuln/detail/CVE-2021-45046.

[5] URL: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-45105.