Spoken Digits Recognition

Daniele Gilio

May 19, 2020

0.1 Introduction

This assignment is about spoken digits recognition. The dataset is comprised of 1760 training samples, 120 validation samples and 120 test samples. The dataset has already been processed so that each sample represents the spectrogram of a given digit. Spectrograms are visual representations of audio files: the power of the recorded sound wave is divided amongst 16 frequencies and 64 time periods creating images like the one in Figure 1. Our objective is to create a Neural Network (a Multi-Layer Perceptron to be precise) in order to recognize the spoken digits.

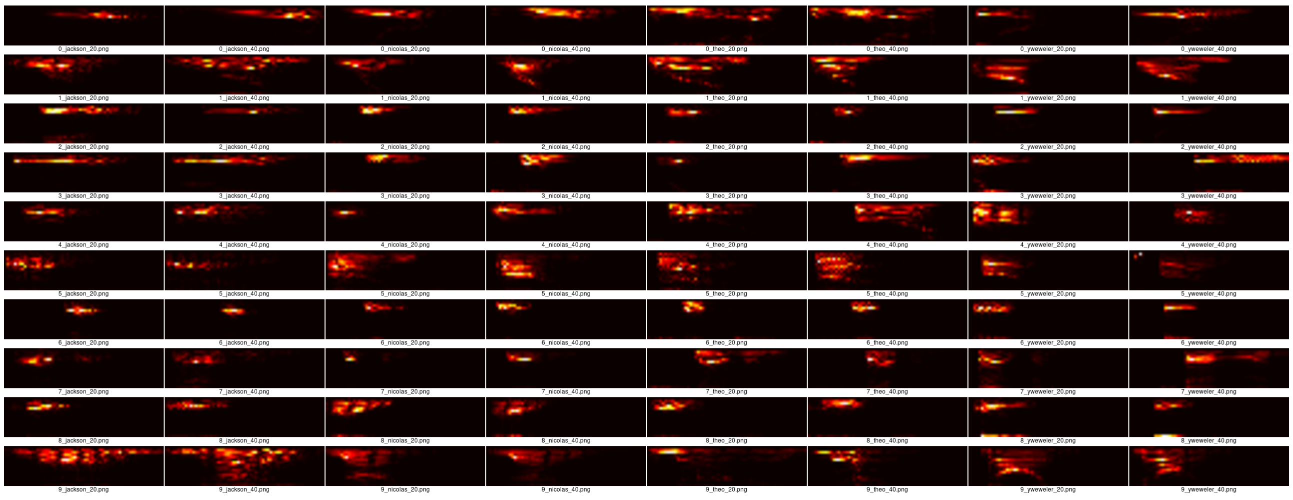

Figure 1: Example of the dataset spectrograms

0.2 Data Visualization and Analysis

In order to test our code we plotted spectrograms after we loaded the data and performed a normalization. The second plotting was also done in order to verify that the normalization did not interfere too much with the data. At this stage we opted for a Mean-Variance Normalization since it is the most common and we do not believe that the dataset geometry would be hugely altered. We can see the output of our code in Figure 2.

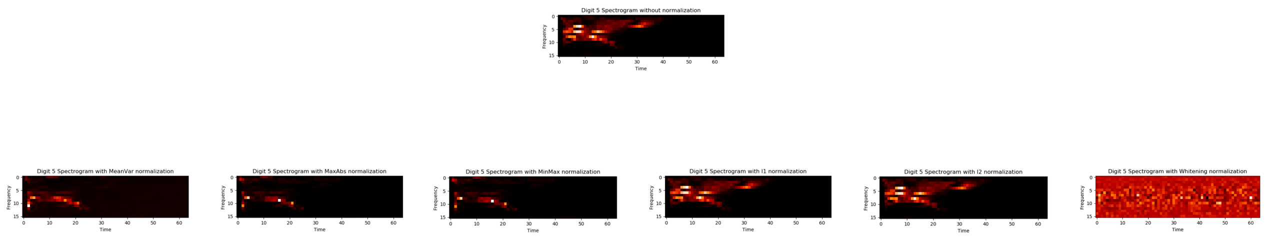

Figure 2: Original Spectrogram (Top) and Normalized Spectrograms (Bottom)

0.3 Neural Network

In order to experiment a bit with Neural Networks we built a Single-Layer Perceptron in which we have only an input layer and an output layer. As in the previous assignment we implemented a varying learning rate.

0.3.1 Architecture

After playing a bit with network architectures we settled on the following sequence $[1024, 256, 128, 32, 10]$, which values represent the width of each layer. Bigger models tended to overfit and took a long time to train, on the other hand tinier models were faster to train but less precise. We believe that optimal values are not very far from the ones we found.

0.3.2 Hyperparameters and Training

In order to train our network we tested Gradient Descent, SGD and mini-batch SGD. The first one proved to be the smoothest but the slowest and hardware demanding, SGD was the fastest but the most erratic. Mini-batch SGD was a good compromise between the two. We obtained the best results with a batch size of 8 for the Single-Layer Network and a size of 256 for the Multi-Layer one. The number of steps was adjusted accordingly to the batch size, so that $batch\_size \cdot steps \simeq training\_samples$. Both networks were trained for 1000 epochs to ensure they reached convergence. The starting learning rate was set at 0.01, the regularization coefficient and the momentum coefficient were left untouched, respectively at 10^{-5} and 0.99. Note that with mini-batch SGD we updated the learning rate at each epoch, not at each step.

0.3.3 Normalization Techniques

We decided to further investigate normalization techniques, so to do that we performed a training for each one. We noticed that the batch size is very impactful on this test, small batch sizes tend to favour L2 Regularization whereas bigger batch sizes favour MeanVar Normalization. We can see a plot of all the tests performed on the Multi-Layer Network (with the hyperparameters listed before) in Figure 3. From the data we collected we can

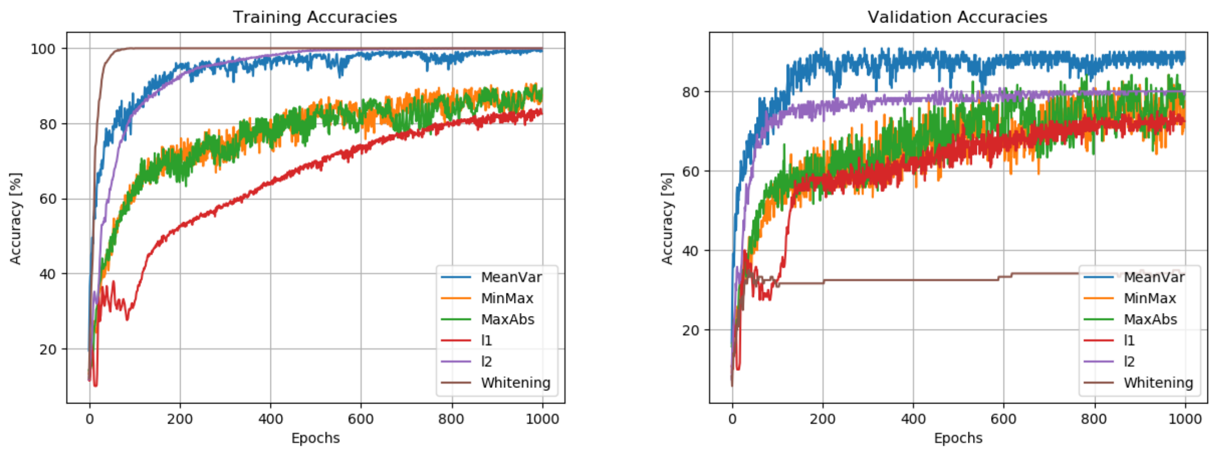

Figure 3: Training Accuracies (Left) and Validation Accuracies (Right)

say that MeanVar is the best overall performer, whereas Whitening is the worst. There is no clear separation between instance normalizations and statistical based ones since L2 normalization is the second most performing normalization.

0.4 Results

The final predictions on the test set reached an accuracy between 87.5% and 90% for the Multi-Layer Network and between 69% and 72% for the Single-Layer one. We can take a look at the *Confusion Matrix* to better understand how the networks perform. Examining the confusion matrices in Figure 4, we can see that the Multi-Layer Network performs better as expected. We can also see that the most incorrect predictions were given for digits 0 and 3 in the case of the Single-Layer Network and 1 and 3 for the Multi-Layer.

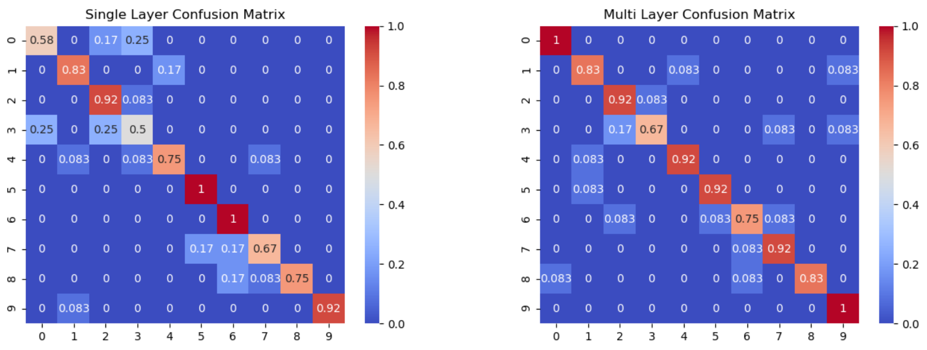

Figure 4: Single Layer (Left) and Multi Layer (Right) Confusion Matrices

With regards to the Single-Layer Network we can also take a look at the weights, which should represent the spectrograms used by the network to make predictions. In Figure 5 we see that 0 and 3 have very similar spectrograms as we expected from the confusion matrix. The Single Layer Network mostly uses the first part of the spectrogram, commonly called the "*attack*". It also uses the "*tail*" for the 0 which has a very distinct one. In the Multi-Layer case a direct weight interpretation is very difficult but we can listen to the samples which are misclassified with the most confidence. Some of them are perfectly discernible for a human while others can be a bit tricky since we feel that the signal-to-noise ratio is probably low.

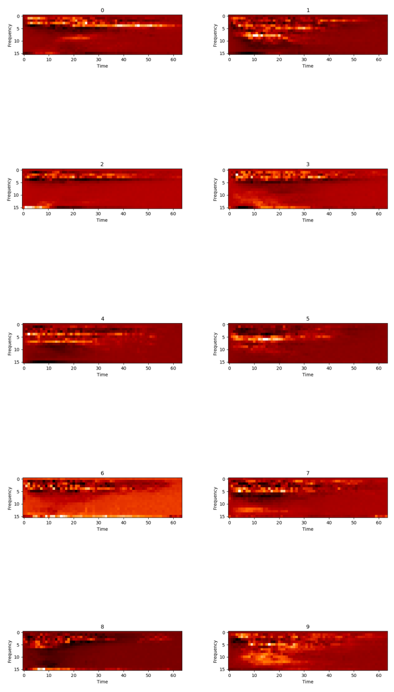

Figure 5: Single Layer Network Weights

0.5 Conclusions

In conclusion we can say that the Multi Layer Perceptron is a better choice for this particular classification problem. The Single Layer Perceptron is a good first try, but its linearity significantly limits its capabilities. Introducing some kind of memory in the neurons might help improving the Multi-Layer Network performance, since each instance in the feature vectors is strongly correlated to the ones around it, being a representation of an audio file. One could also think about improving the time resolution of the spectrograms in order to give the networks more information to learn from. As always expanding the dataset would also be a good idea, 1760 training samples are kind of on the tiny dataset side. That said our Multi Layer Network works quite well given the context.

I affirm that this report is the result of my own work and that I did not share any part of it with anyone else except the teacher.