# Recognizing Alphanumeric Characters with a Feed-Forward Neural Network

## COMP815 - Assignment 1, Part 2

Eirik Folkestad - Student no: 17966805

## I. Introduction

In this project we are going to classify binary images of the 36 alphanumeric characters '0'-'9' and 'A'-'Z' with the help of a Feed-Forward Neural Network. We will process the input data and transform it to a suitable format, determine the best way to split the data into datasets for training, validation and testing, choose the best architecture for prediction and test the trained system for robustness as well as see if a better architecture can be produced in terms of robustness.

## II. Data

### A. Original Dataset

The dataset consists of **39** binary images of each **36** alphanumeric character categories, '0'-'9' and 'A'-'Z', which means that the dataset contains **36x39 = 1404** samples in total and **36x39 = 1404** corresponding target labels. Each binary image is a 2 dimensional matrix of size **20x16**. The positions in the matrix, described by row and column, takes part in describing the shape of an alphanumeric character. A **1** means that the matrix position describes a part of the object and a **0** means that it does not. An example of a data sample representing the alphanumeric character, 'O', although reduced in size, can be viewed in Table I.

TABLE I. Data Sample of size 3x3 Representing the Alphanumeric Character 'O'

| | | |
|---|---|---|
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 0 |

### B. Data Processing

Before the data can be input into the FFNN, *patternnet* [1], that Matlab provides in its Neural Network Toolbox, we need to reshape it to a suitable format. The FFNN expects a matrix with each sample as a column for the data. First we convert every **20x16** sample into a **320x1** vector, then we put every sample side by side so that the **36x39** matrix with **20x16** elements becomes a **320x1404** matrix. This means that there are **1404** samples of size **320x1**. The FFNN also require that we need to convert the corresponding target labels of the samples into a one-hot encoded representation before we can use them. A one-hot encoded label is a vector of size equal the number of labels, and every element is **0** except for the one element that represents the label which is **1**. This means that from the **1x1404** vector of target labels, we now create a **36x1404** matrix of one-hot encoded labels. An example of a one-hot encoding can be seen in Table II.

TABLE II. A One-Hot Encoding Example of the Label 1 when the Set of All Labels is (1,2,3)

| | | |
|---|---|---|
| 1 | 0 | 0 |

## III. Results

An experiment was conducted to see the Cross-Entropy Error and the Percent Error with different dataset splits into training, validation and test sets. Each configuration's trained model have been evaluated by averaging the Cross-Entropy Error produced by training the prediction model 10 times. See Figures 1 and 2 for the results.

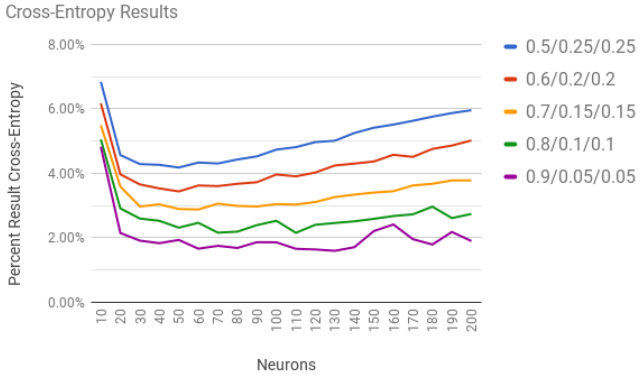


Fig. 1. Cross-Entropy Error Results Averaged over 10 Iterations of the Same Configuration

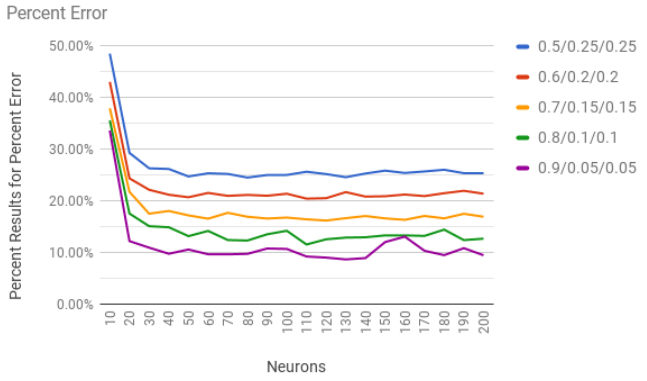


Fig. 2. Percent Error Results Averaged over 10 Iterations

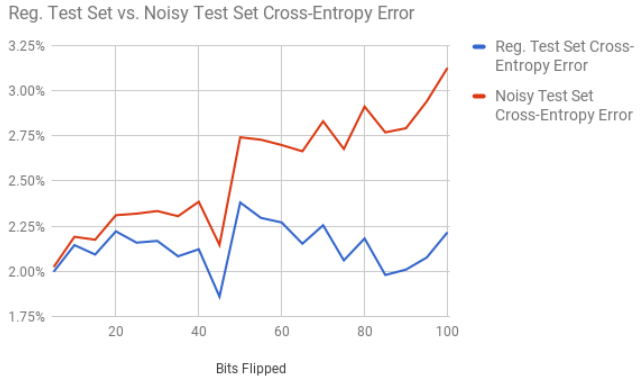


Fig. 3. Reg. Test Set vs. Noisy Test Set Cross-Entropy Error Averaged over 10 Iterations with a 0.8/0.1/0.1 Split and 70 Neurons in the Hidden Layer and Different Number of Bits Flipped

The critical area after analyzing the Figures 1 and 2 can be seen in Tab. III.

TABLE III. CRITICAL AREA OF CROSS-ENTROPY ERROR AND PERCENT ERROR FOR THE 0.8/0.1/0.1 SPLIT RESULTS AVERAGED OVER 10 ITERATIONS OF THE SAME CONFIGURATION ON THE REGULAR TEST SET

| Performance Measure | 40 | 50 | 60 | 70 | 80 | 90 |
|---|---|---|---|---|---|---|
| Cross-Entropy Error | 2.53% | 2.32% | 2.47% | 2.16% | 2.19% | 2.40% |
| Percent Error | 14.87% | 13.16% | 14.16% | 12.40% | 12.30% | 13.51% |

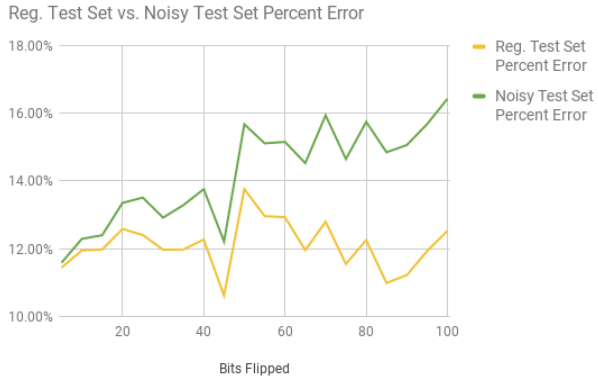


Fig. 4. Reg. Test Set vs. Noisy Test Set Percent Error Averaged over 10 Iterations with a 0.8/0.1/0.1 Split and 70 Neurons in the Hidden Layer and Different Number of Bits Flipped

An experiment was conducted to evaluate the Robustness of the trained prediction model. By altering unique random bits in the test set, we can create a noisy test set and compare the performance of the FFNN prediction model on this noisy test set and the test set without noise. This would determine the robustness of the FFNN prediction model. The results can be seen in Table IV. The results when a different number of random bits are flipped can be observed in Figures 3 and 4. All these experiments are done with a *0.8/0.1/0.1* split and **70** neurons in the hidden layer.

TABLE IV. COMPARISON OF PERFORMANCE MEASUREMENTS FROM THE REGULAR TEST SET TO THE SINGLE BIT-FLIPPED TEST SET AVERAGED OVER 10 ITERATIONS WITH A 0.8/0.1/0.1 SPLIT AND 70 NEURONS IN THE HIDDEN LAYER

| Performance Measurement | Regular Test Set | Bit-Flipped Test Set |
|---|---|---|
| Cross-Entropy Error | 2.25% | 1.300% |
| Percent Error | 2.26% | 1.315% |

An experiment was conducted where the cross-entropy error and the percent error where measured for both the regular test set and a the test set with flipped bits. The test set with flipped bits is called a noisy test set. The experiment was to see if we could find a better architecture with a different number of neurons for data with enough noise to significantly reduce the performance of the prediction model. The results can be seen in Figures 5 and 6.

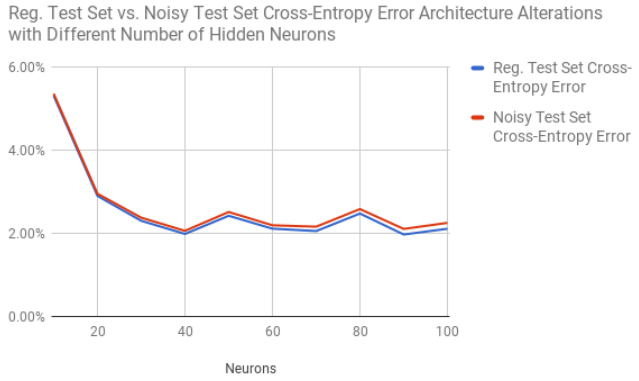


Fig. 5. Reg. Test Set vs. Noisy Test Set Cross-Entropy Error Averaged over 10 Iterations with a 0.8/0.1/0.1 Split and 20 Randomly Flipped Bits and a Different Number of Neurons In the Hidden Layer

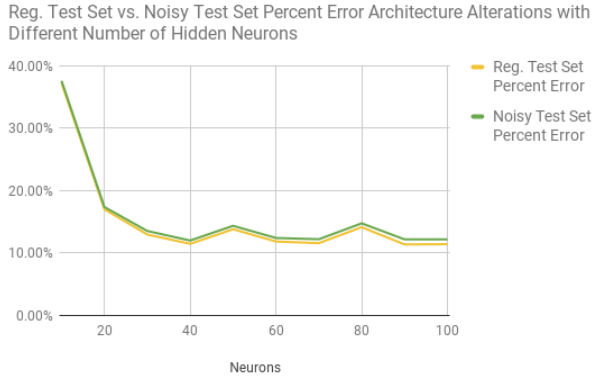


Fig. 6. Reg. Test Set vs. Noisy Test Set Percent Error Averaged over 10 Iterations with a 0.8/0.1/0.1 Split and 20 Randomly Flipped Bits and a Different Number of Neurons In the Hidden Layer

TABLE V. CRITICAL AREA OF CROSS-ENTROPY ERROR AND PERCENT ERROR FOR THE 0.8/0.1/0.1 SPLIT RESULTS AVERAGED OVER 10 ITERATIONS OF THE SAME CONFIGURATION ON THE NOISY TEST SET

| Performance Measure | 30 | 40 | 50 | 60 | 70 | 80 |
|---|---|---|---|---|---|---|
| Cross-Entropy Error | 2.38% | 2.06% | 2.51% | 2.19% | 2.16% | 2.58% |
| Percent Error | 13.55% | 12.02% | 14.38% | 12.43% | 12.23% | 14.78% |

## IV. DISCUSSION

### A. Division of Dataset

Dividing the dataset into three subsets is necessary for training the FFNN prediction model, validating that the training done is useful for predicting labels for other data and also to test the trained prediction model after training is complete and from this determine its quality. Hence, we split our data in the three subsets as follows:

- **Training**
- **Validation**
- **Test**

However, how we split the dataset into the three subsets is important both for performance and correct performance measurements. It is important to let the training set be as big as possible because more data to train on means a better prediction model and it is important that both the validation and test sets should contain a few samples of each category so that the measured performance is representable for the true performance of the prediction model. However, the two are opposites and increasing one, decreases the other which means that there is a trade-off problem occurring. We should from the knowledge of this try to find acceptable sizes for the subsets which do not decrease performance too much and so that we still can measure a representable performance. We can see this effect in Figure 2 and Figure 1. The performance clearly increases with the increased training set size and a decrease in both errors can be observed. The number of support each class approximately has the validation and test sets have in each *Training/Validation/Test* split is as follows:

- **0.5/0.25/0.25** - $36 * 0.25 = 9$
- **0.6/0.20/0.20** - $36 * 0.20 = 7.2$
- **0.7/0.15/0.15** - $36 * 0.15 = 5.4$
- **0.8/0.10/0.10** - $36 * 0.10 = 3.6$
- **0.9/0.05/0.05** - $36 * 0.05 = 1.8$

Statistically, we would get one from each category just by having validation and test sets of $\frac{1}{36}$ of the dataset ($36 * \frac{1}{36} = 1$). However, this is usually not going to happen. We should instead choose subset sizes that will produce a validation and test set that includes at least one sample from each category with a higher probability. With this in mind, the **0.7/0.15/0.15** and **0.8/0.10/0.10** splits seem like reasonable choices for a split and maybe the **0.8/0.10/0.10** would be the best choice of the two.

### B. Architecture

A FFNN with a single hidden layer is sufficient to represent any function [2] and therefore we will not make changes to the default number of hidden layers in the *patternnet* of Matlab which uses exactly one hidden layer. The one thing that we will change is the number of neurons in the hidden layer. The number of hidden layer neurons helps determine the expressiveness of the network. With too few neurons, the network is going to have a hard time approximating a function for recognizing a pattern and with too many, the approximated function will most likely overfit to the training data. For determining the number of hidden neurons that make up the best architecture, we conducted an experiment to see how different the results were. As we observe in Figure 1, a small number of neurons (below around **20-30**) makes it hard to approximate a pattern recognizing function since the network is not able to fully express one and we get a high percentage of cross-entropy error. A high number on the other hand, make the approximated pattern recognition function too fitted to the training data and we observe a negative effect in performance when predicting other data like the one in the test set. In the splits with the smaller training sets the overfitting is more visible and this happens with even fewer hidden neurons than the splits with the larger training sets. This makes sense

since with less data to overfit on, the less generalizing the function is going to be and with more data to overfit on, the more generalizing the function is going to be just based on the diversity in the data. This is why we observe the overfitting as making less impact on the splits with the larger training sets. The interval that seem the most interesting to investigate closer is with **40-90** hidden neurons. As we see in Table III we can see that **70** hidden neurons seem to produce the best architecture which is shown in Figure 7.

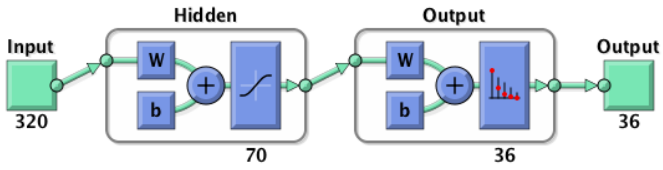


Fig. 7. The Feed-Forward Neural Network Architecture

### C. Robustness of the Prediction Model

The robustness of the model is something that was not taken into consideration before training and thus we did nothing to prevent noisy data from being falsely predicted to another class than the true class. However, through experiments done by introducing noise to the test set we can see indications for how robust the trained FFNN prediction model is to noisy data. From the results in Table IV we can see that the difference in flipping only one bit in each sample in the test data does not make any significant difference in terms of performance. Yet, if we increase the number of randomly flipped bits like in Figure 3 and Figure 4, we can see an increasingly larger difference in the performance measurements on the regular test set compared to the noisy test set. The performance decreases significantly when around **20** random bits are flipped in each sample. After this the performance decreases even more at a steady rate. This means that in around when **20** bits are randomly flipped, the representation of the alphanumeric characters are beginning to be too different from what the approximated function for pattern recognition is not trained to handle. The architecture is also a little too biased to the dataset and could benefit from generalizing more by reducing the number of hidden layer neurons. As we see in Figure 5 and Figure 6, the difference seem to decrease with the number of hidden neurons. The model generalizes better with fewer neurons neurons and is more biased with many neurons. Since this is a trade-off situation, we should focus on balancing what is important; robustness or performance by making the approximated function generalized or biased. Since we are looking to increase robustness, we would decrease the number of hidden layer neurons to get the required effect. The new optimal architecture if we randomly flip as much as **20** bits should have around **40** hidden layer neurons. This also causes the deviance in performance between the regular test set and the noisy test set to be small which is a desirable effect in the sense that this means that the prediction model predicts noisy data up to **20** randomly flipped bits just as well as data without noise. This indicates that the new model is generalizing better than before and it seems to be able to handle a larger amount of noise than the model with **70** neurons in the hidden layer.

## V. Conclusion

In this project we have trained a FFNN prediction model which is capable of classifying binary images to 36 alphanumeric character classes. To make the FFNN suitable for the task, we have adjusted the configurations of the model so that we have an optimal *training/validation/test* split of the dataset, we have optimized architecture for data without noise of **70** neurons in the hidden layer and we have later altered the architecture to take noisy data into account by reducing the number of hidden layer neurons to **40** so that the model was generalizing better. As we have seen, there are many factors that influence the performance of a prediction model like the FFNN prediction model. It is a difficult task to accommodate all trade-off problems and at the same time produce a prediction model capable of predicting with good performance. However, the answer to what a good prediction model is, relies on the task at hand and the scope of which we wish to solve it.

## References

[1] The MathWorks, Inc., "Pattern recognition network," 2017, accessed on 22.09.2017. [Online]. Available: https://au.mathworks.com/help/nnet/ref/patternnet.html

[2] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.