Damon Ball

Nikhil Castelino

Peter Nolan

**Project 2 Report**

**Artificial Neural Networks**

**Base Case:**

Confusion Matrix:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Predicted Labels | | | | | | | | | | | |
| Real Labels |  | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| **0** | 163 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **1** | 131 | 51 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **2** | 159 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **3** | 168 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **4** | 164 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **5** | 142 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **6** | 166 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **7** | 172 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **8** | 150 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **9** | 161 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Total Accuracy:** 0.131530424093

**Experimentation**

After experimentation with numerous different aspects of the ANN, changing the amount of layers, neurons per layer, number of epochs, and the batch size, I found a combination of characteristics that successfully matched the majority of the provided images with their respective labels. In the end, I had developed an ANN with the following characteristics:

* **5 Total Layers**
* **Layer 1**: 500 Neurons
* **Layer 2:** 300 Neurons
* **Layer 3:** 100 Neurons
* **Layer 4:** 50 Neurons
* **Layer 5:** 10 Neurons
* **Number of Epochs:** 200
* **Batch Size:** 1024

**Process:**

In order to get these results, I started with the original Template that had been provided in the assignment, which only had 2 Layers and 10 epochs.

The first experiments involved increasing the amount of epochs that the training set would go through, so I had increased it from 10 to 100, then 1,000, then as high as 10,000. Despite these increases there was only a marginal increase in performance, while still suffering from wildly varying results, going as high as 80 percent accuracy and then dropping to as low as 40 percent. Since I was not seeing any definitive improvements, I decided that having such large epoch sizes was unnecessary, and reduced the size down to 200, providing me with an acceptable edge in performance without hindering runtime exponentially.

The next part of the experiment was to increase the number of layers, as well as the number of neurons that were in each layer. I started by adding another layer with 100 neurons to the algorithm, and noticed a slight increase in performance. I then added another layer of 300 neurons, and noticing a larger increase, put in another layer of 500, as well as a layer of fifty neurons in between the 10 and 100 layers. All together, this added a large amount of improvement to the ANN, bringing up accuracy to around 80 percent, and only fluctuating mildly for most cases. The results from my experimentation are shown below. Note: It is important to note that there are some cases that fail badly, going as low as 10 Percent. However, I felt that these cases were too few to consider true measures of the ANN’s success.

**Results:**

Confusion Matrix:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Predicted Labels | | | | | | | | | | | |
| Real Labels |  | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| **0** | 163 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **1** | 19 | 163 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **2** | 35 | 0 | 117 | 0 | 1 | 0 | 1 | 2 | 3 | 0 |
| **3** | 32 | 0 | 0 | 133 | 0 | 0 | 0 | 2 | 0 | 1 |
| **4** | 39 | 1 | 0 | 0 | 120 | 0 | 1 | 1 | 0 | 2 |
| **5** | 40 | 0 | 0 | 0 | 0 | 98 | 1 | 1 | 1 | 1 |
| **6** | 18 | 0 | 0 | 1 | 1 | 1 | 144 | 1 | 0 | 0 |
| **7** | 25 | 0 | 0 | 0 | 2 | 0 | 0 | 143 | 0 | 2 |
| **8** | 46 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 101 | 0 |
| **9** | 37 | 0 | 0 | 1 | 1 | 0 | 0 | 2 | 2 | 118 |

**Total Accuracy**: 0.79901659496

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Label** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| **Precision** | 0.36 | 0.99 | 1.00 | 0.98 | 0.96 | 0.98 | 0.97 | 0.94 | 0.94 | 0.95 |
| **Recall** | 1.00 | 0.90 | 0.74 | 0.79 | 0.73 | 0.69 | 0.87 | 0.83 | 0.67 | 0.73 |

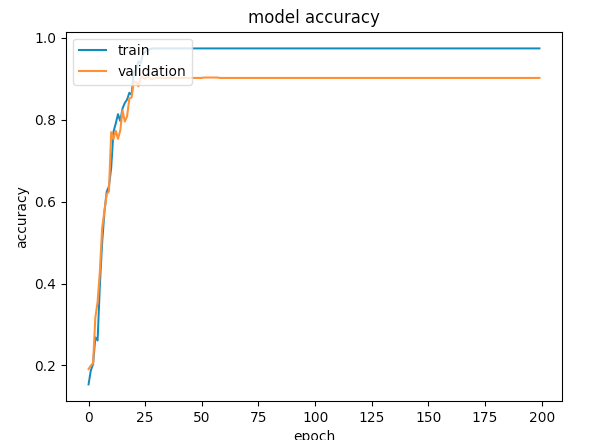
Overall precision = 91%

Overall Recall = 80%

From the above matrix, we can tell that the worst performing class in the list was 0, which usually had the most false positives amongst the classes. The cause for this could be anything from too few zeros in the training area, to too few of the other classes. However, since the proportions for the training set were preset, this could not be changed.

**Graph for Model Accuracy**

Below, we see that Accuracy increases exponentially as more epochs are initiated. We see that the validation and training set both proceed to a high level within the first 25 epochs, and then remain at a high level of accuracy for the remainder of the epochs.



**Missed Images for ANN**

While the most common mistakes occurred do to the other classes being compared to the zero class, we can see that there are still some other errors within the matrix as well. These three provide more examples:

****

**Real 2, Predict 8**

****

**Real 6, Predict 5**

****

**Real 9, Predict 7**

**Decision Trees**

**Baseline results:**

Confusion matrix:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Predicted Labels | | | | | | | | | | | |
| Real Labels |  | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| **0** | 144 | 0 | 4 | 2 | 2 | 3 | 2 | 0 | 4 | 1 |
| **1** | 0 | 155 | 6 | 4 | 3 | 2 | 3 | 3 | 3 | 2 |
| **2** | 1 | 5 | 110 | 7 | 2 | 5 | 6 | 7 | 6 | 9 |
| **3** | 2 | 2 | 9 | 119 | 2 | 11 | 4 | 1 | 13 | 3 |
| **4** | 0 | 2 | 6 | 2 | 123 | 3 | 3 | 2 | 5 | 16 |
| **5** | 3 | 1 | 4 | 9 | 3 | 99 | 6 | 2 | 7 | 7 |
| **6** | 2 | 0 | 7 | 3 | 6 | 2 | 137 | 2 | 6 | 0 |
| **7** | 0 | 0 | 4 | 1 | 4 | 0 | 2 | 152 | 2 | 6 |
| **8** | 2 | 4 | 4 | 13 | 4 | 5 | 3 | 5 | 96 | 14 |
| **9** | 2 | 1 | 1 | 3 | 12 | 6 | 3 | 5 | 0 | 127 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Label** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| **Precision** | 0.923077 | 0.911765 | 0.709677 | 0.730061 | 0.763975 | 0.727941 | 0.810651 | 0.849162 | 0.676056 | 0.686486 |
| **Recall** | 0.888889 | 0.856354 | 0.696203 | 0.716867 | 0.759259 | 0.702128 | 0.830303 | 0.888889 | 0.64 | 0.79375 |

Overall accuracy = 76.2%

**Experiments:**

**Variation on the baseline:**

I tested changing the values for the minimum number of samples required for a split and the minimum number of samples required to form a leaf. First I tried changing each value by itself, and both the recall and precision went down drastically for both. I also tried a decision tree with both values changed, and still all performance measures went down.

Then I tested changing the max depth of the tree. Here, changing the max depth actually resulted in interesting differences. Using a max depth of 20 gave better results for some labels and worse results for others. However, overall it was slightly worse than no max depth. I then messed around with other similar values for max depth and found that a max depth of 18 was generally the best. However, the changes were small overall when comparing the overall accuracy.

**Max Depth = 18**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Predicted Labels | | | | | | | | | | | |
| Real Labels |  | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| **0** | 140 | 0 | 7 | 2 | 2 | 3 | 3 | 0 | 4 | 1 |
| **1** | 0 | 158 | 6 | 4 | 3 | 2 | 3 | 3 | 0 | 2 |
| **2** | 1 | 4 | 117 | 7 | 1 | 3 | 13 | 6 | 3 | 3 |
| **3** | 2 | 2 | 11 | 121 | 0 | 12 | 2 | 0 | 11 | 5 |
| **4** | 1 | 1 | 4 | 0 | 122 | 2 | 4 | 5 | 5 | 18 |
| **5** | 8 | 1 | 4 | 7 | 1 | 101 | 6 | 1 | 5 | 7 |
| **6** | 2 | 0 | 8 | 1 | 6 | 2 | 136 | 1 | 8 | 1 |
| **7** | 0 | 1 | 5 | 2 | 2 | 0 | 3 | 149 | 2 | 7 |
| **8** | 2 | 4 | 7 | 10 | 3 | 4 | 8 | 5 | 94 | 13 |
| **9** | 1 | 2 | 3 | 3 | 6 | 4 | 4 | 7 | 2 | 128 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Label** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| **Precision** | 0.89172 | 0.913295 | 0.680233 | 0.770701 | 0.835616 | 0.759398 | 0.747253 | 0.841808 | 0.701493 | 0.691892 |
| **Recall** | 0.864198 | 0.872928 | 0.740506 | 0.728916 | 0.753086 | 0.716312 | 0.824242 | 0.871345 | 0.626667 | 0.8 |

Since I was unable to find any variation that consistently improved the baseline model, I decided to continue using the unaltered decision tree.

**Hand-engineered features:**

The first feature I tested was simply taking the average of all pictures in an image. Since taking the average divides each total by the same number of pixels, the total of the pixels values was used instead as it was equivalent to using the average but used less operations. However, the pixel sum feature did not make any major improvement on the model, and instead had slightly worse results. Another similar feature I attempted was using the number of 0 pixels as a feature. This feature would correspond to the amount of empty space in an image, which I thought would help differentiate between numbers like 8 which takes up a lot of pixel space, and 1 which takes up much less. However, this too did not significantly improve results.

The next feature I tested was the pixel total for each row, which is the same as the pixel average for each row. This feature worked very well, with an approximate increase in overall accuracy of about 2%. While the precision and recall scores for some labels did fall, it was only by a small amount compared to the increase in the scores of other labels. I then also tested a fourth feature of the pixel total for each column, which also increased the performance scores but in to a much less degree than the row total. My theory on why this happened is because there is more variation between numbers vertically than horizontally for most numbers.

Next I tried dividing each image into 4 by 4 blocks and taking the pixel total of each block as a feature. This did result in a slight increase in the performance metrics, but not by much. I then tried the same feature but with 2 by 2 blocks, 7 by 7 blocks (dividing the image evenly into 16 blocks) and 14 by 14 blocks(cutting the image in fourths). 14 by 14 blocks performed much worse, and 2 by 2 performed about as well as 4 by 4. However, 7 by 7 blocks actually did improve results by a slight margin, around 0.5%.

**Results**:

The final decision tree using all the best methods I tested was one that used no pruning, no maximum depth, and had hand-engineered features of average row pixels, average column pixels, and average pixels of each 7 by 7 block. This model obtained an overall accuracy of about 78.5% when testing on the testing data set. Below are the results obtained with this tree.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Predicted Labels | | | | | | | | | | | | | | | | | | | | | |
| Real Labels |  | | **0** | | **1** | | **2** | | **3** | | **4** | | **5** | | **6** | | **7** | | **8** | **9** | |
| **0** | | 137 | | 1 | | 9 | | 2 | | 2 | | 2 | | 3 | | 0 | | 5 | 1 | |
| **1** | | 0 | | 160 | | 5 | | 1 | | 4 | | 3 | | 2 | | 4 | | 1 | 1 | |
| **2** | | 2 | | 6 | | 111 | | 5 | | 2 | | 5 | | 7 | | 6 | | 8 | 6 | |
| **3** | | 1 | | 2 | | 9 | | 119 | | 1 | | 13 | | 3 | | 1 | | 11 | 6 | |
| **4** | | 1 | | 1 | | 6 | | 3 | | 129 | | 1 | | 2 | | 3 | | 5 | 11 | |
| **5** | | 4 | | 1 | | 4 | | 9 | | 2 | | 101 | | 6 | | 4 | | 4 | 6 | |
| **6** | | 2 | | 1 | | 9 | | 2 | | 8 | | 2 | | 136 | | 0 | | 3 | 2 | |
| **7** | | 0 | | 0 | | 5 | | 2 | | 3 | | 0 | | 1 | | 151 | | 4 | 5 | |
| **8** | | 1 | | 3 | | 7 | | 11 | | 2 | | 5 | | 6 | | 5 | | 98 | 12 | |
| **9** | | 0 | | 1 | | 3 | | 4 | | 10 | | 6 | | 3 | | 5 | | 1 | 127 | |
| **Label** | | **0** | | **1** | | **2** | | **3** | | **4** | | **5** | | **6** | | **7** | | **8** | | | **9** | | |
| **Precision** | | 0.925676 | | 0.909091 | | 0.660714 | | 0.753165 | | 0.791411 | | 0.731884 | | 0.804734 | | 0.843575 | | 0.7 | | | 0.717514 | | |
| **Recall** | | 0.845679 | | 0.883978 | | 0.702532 | | 0.716867 | | 0.796296 | | 0.716312 | | 0.824242 | | 0.883041 | | 0.653333 | | | 0.79375 | | |

Overall accuracy = 78.5%

**Miss predicted images for decision trees:**

Real6Predict5

Image 1: Actual = 6, Guessed = 5

Real2Predict7

Image 2: Actual = 2, Guessed = 7

Real0Predict5

Image 3: Actual = 0, Guessed = 5

**K-Nearest-Neighbors**

**Baseline results:**

Confusion matrix:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Predicted Labels | | | | | | | | | | | |
| Real Labels |  | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| **0** | 91 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| **1** | 0 | 98 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **2** | 3 | 1 | 88 | 1 | 0 | 1 | 0 | 2 | 1 | 0 |
| **3** | 1 | 0 | 0 | 87 | 0 | 2 | 0 | 0 | 0 | 1 |
| **4** | 1 | 2 | 0 | 0 | 73 | 0 | 1 | 0 | 0 | 3 |
| **5** | 2 | 0 | 0 | 1 | 0 | 67 | 2 | 0 | 1 | 0 |
| **6** | 2 | 0 | 0 | 1 | 0 | 1 | 97 | 0 | 0 | 0 |
| **7** | 1 | 6 | 0 | 0 | 0 | 0 | 0 | 81 | 0 | 1 |
| **8** | 4 | 6 | 3 | 1 | 0 | 1 | 2 | 0 | 69 | 0 |
| **9** | 1 | 0 | 0 | 3 | 0 | 0 | 0 | 3 | 0 | 85 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Label** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| **Precision** | 0.99 | 0.87 | 0.97 | 0.93 | 1.00 | 0.93 | 0.94 | 0.94 | 0.96 | 0.94 |
| **Recall** | 0.98 | 1.00 | 0.91 | 0.96 | 0.91 | 0.92 | 0.96 | 0.91 | 0.80 | 0.92 |

Average Precision = 95%

Average Recall = 93%

**Experiments:**

**Variation on the value:**

By changing the value for the n\_neighbors, the precision and recall values change each time. Multiple values were tested in order to find the correct n\_neighbors value that would provide the highest average recall value with a good average precision.

**N\_neighbors = 4**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Predicted Labels | | | | | | | | | | | |
| Real Labels |  | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| **0** | 83 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **1** | 0 | 103 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **2** | 10 | 2 | 78 | 0 | 0 | 0 | 10 | 0 | 0 | 0 |
| **3** | 8 | 1 | 0 | 78 | 0 | 0 | 0 | 0 | 0 | 0 |
| **4** | 6 | 2 | 0 | 0 | 83 | 0 | 1 | 0 | 0 | 1 |
| **5** | 8 | 0 | 0 | 1 | 0 | 72 | 1 | 0 | 0 | 0 |
| **6** | 2 | 0 | 0 | 0 | 0 | 0 | 86 | 0 | 0 | 0 |
| **7** | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 87 | 0 | 1 |
| **8** | 13 | 2 | 1 | 0 | 0 | 2 | 2 | 0 | 79 | 1 |
| **9** | 3 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 75 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Label** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| **Precision** | 1.00 | 0.93 | 0.99 | 0.97 | 1.00 | 0.97 | 0.96 | 1.00 | 1.00 | 0.96 |
| **Recall** | 0.99 | 1.00 | 0.87 | 0.90 | 0.89 | 0.88 | 0.98 | 0.93 | 0.79 | 0.94 |

Average Precision = 98%

Average Recall = 91%

Since I was unable to find any variation that consistently improved the baseline model, I decided to continue using the unaltered decision tree.

**Results**:

While there were certain values that gave out a higher value for precision, the recall value reduces. Additionally, it was noticed that an increase in the value for n\_neighbors, the recall value keeps decreasing. As a result, the n\_neighbors value was kept as 3

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Predicted Labels | | | | | | | | | | | |
| Real Labels |  | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| **0** | 91 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| **1** | 0 | 98 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **2** | 3 | 1 | 88 | 1 | 0 | 1 | 0 | 2 | 1 | 0 |
| **3** | 1 | 0 | 0 | 87 | 0 | 2 | 0 | 0 | 0 | 1 |
| **4** | 1 | 2 | 0 | 0 | 73 | 0 | 1 | 0 | 0 | 3 |
| **5** | 2 | 0 | 0 | 1 | 0 | 67 | 2 | 0 | 1 | 0 |
| **6** | 2 | 0 | 0 | 1 | 0 | 1 | 97 | 0 | 0 | 0 |
| **7** | 1 | 6 | 0 | 0 | 0 | 0 | 0 | 81 | 0 | 1 |
| **8** | 4 | 6 | 3 | 1 | 0 | 1 | 2 | 0 | 69 | 0 |
| **9** | 1 | 0 | 0 | 3 | 0 | 0 | 0 | 3 | 0 | 85 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Label** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| **Precision** | 0.99 | 0.87 | 0.97 | 0.93 | 1.00 | 0.93 | 0.94 | 0.94 | 0.96 | 0.94 |
| **Recall** | 0.98 | 1.00 | 0.91 | 0.96 | 0.91 | 0.92 | 0.96 | 0.91 | 0.80 | 0.92 |

Average Precision = 95%

Average Recall = 93%

**Images:**

