Handwritten Digit Recognition

Rahul Shah

April 2023

1 Introduction

This project is on handwritten digit recognition involving neural networks and other classification machines. A predefined defined CNN architecture is given to optimize it to increase baseline accuracy and report tradeoffs from hyper-parameters. An independent SVM is created using LIBSVM and hyper-parameters are varied to report tradeoffs on run-time and accuracy.

2 CNN

Baseline CNN architecture:

- Convolutional #1 (C1) → Filters: 6. Filter size: 5x5. Activation: ReLU
- Pooling #1 (P1) → Pool Size: 2x2. Strides: 2x2
- Convolutional #2 (C1) → Filters: 16. Filter size: 3x3. Activation: ReLU
- Pooling #2 (P2) → Pool Size: 2x2. Strides: 2x2
- Fully Connected #1 (FC1) → Activation: ReLU. Outputs: 120
- Fully Connected #2 (FC2) → Activation: ReLU. Outputs: 84
- Fully Connected #3 (FC3) → Activation: Softmax. Outputs: 10

Initial training parameters:

- Epochs: 2
- Batch Size: 256
- Loss Function: Categorical Cross Entropy
- Optimizer: Adam

Training accuracy: 97.52%. Test accuracy: 97.75%

2.1 Architecture Changes

First, the CNN's hyper-parameters are changed to show how each parameter affects the accuracy. As expected, simply increasing the number of filters in each convolutions layer improves the accuracy. With 16 filters in C1 and 32 in C2, the accuracy increased to 0.9924 for training and 0.9852 for test.

Increasing the kernel size means reducing the number of details seen in the images. Increasing both kernel sizes to 8x8 slightly reduces the accuracy, with 0.9805 training and 0.9794 test.

This table shows all of the different changes in architecture and how the accuracy is affected by each. Note that not every available parameter was changed as this creates too much variance in the data making it hard to draw conclusions.

Model#	C1 (filters, kernel size)	C2 (filters, kernel size)	Activation (C1,C2)	FC1 (units, activation	FC2 (units, activation	FC3 (units, activation	Accuracy (train, test)
0 - Initial	6,(5x5)	16,(3x3)	relu,relu	120, relu	84,relu	10,softmax	97.52,97.75
1	16,(3x3)	32,(2x2)	relu,relu	120,relu	84,relu	10,softmax	97.62,97.46
2	16,(8x8)	24,(5x5)	relu,softmax	120,relu	84,relu	10,softmax	96.91,97.29
3	6,(8x8)	10,(5x5)	relu,tanh	120,relu	84,softmax	10,softmax	77.53,77.90
4	16,(4x4)	10,(5x5)	softmax,sig	120,relu	84,sigmoid	10,softmax	93.57,93.72
5	10,(4x4)	10,(4x4)	softmax,relu	84,relu	84,relu	10,sigmoid	94.01,94.66
6	32,(3x3)	64,(3x3)	relu,relu	200,relu	100,relu	10,softmax	98.32,98.31
7	16,(4x4)	32,(3x3)	relu,relu	140,relu	84,relu	10,softmax	98.45,98.51

Model #0 is the architecture of the initial model and its accuracy. From the seven runs of the model with different architectures, only Models #6 and 7 had accuracies above the baseline. While an increased number of filters improves accuracy, increasing kernel size at the same time reduces it as less data is able to be gathered by the networks. Changing the activation methods did not seem to improve the accuracy as for most functions besides ReLU, all of the neurons are activated at the same time, creating more bias between neighboring neurons.

The last two models both had more filters and a smaller kernel size for the Convolutional layers. The Fully Connected layers also had greater or equal units. All of the activation methods were the same as discusses earlier changing them seemed to decrease accuracy. Model #6 had much more parameters and total connections than Model #7 and still had a slightly lower accuracy. It is important to optimize connections and not create too many paths as it increases total execution run-time as well as surprisingly lowers accuracy. The complexity of that CNN is much higher and led to bias between neurons and more incorrect predictions than Model #7. This is why Model #7 has the highest accuracy and is most optimal from all the models in the table.

2.2 Archtiecture and Parameter Changes

From the previous section, this CNN performs the best with accuracy and run-time

- Convolutional #1 (C1) → Filters: 16. Filter size: 4x4. Activation: ReLU
- Pooling #1 (P1) → Pool Size: 2x2. Strides: 2x2
- Convolutional #2 (C1) → Filters: 32. Filter size: 3x3. Activation: ReLU
- Pooling #2 (P2) → Pool Size: 2x2. Strides: 2x2
- Fully Connected #1 (FC1) → Activation: ReLU. Outputs: 140
- Fully Connected #2 (FC2) → Activation: ReLU. Outputs: 84
- Fully Connected #3 (FC3) → Activation: Softmax. Outputs: 10

The two hyper-parameters are epochs and batch size. Epochs refer to the number of times the dataset is passed through the CNN. Batch size refers to the number of samples to work with before updating the weights. With this CNN, the hyper-parameters, loss functions, and optimizers are changed and accuracies shown in the table.

Model #	Epochs	Batch Size	Loss fn	Optimzer	Accuracy (train, test)
0 - Initial	2	256	Categorical Cross Entropy	Adam	98.43, 98.56
1	10	128	Categorical Cross Entropy	Adam	99.82, 99.11
2	10	64	Categorical Cross Entropy	Adagrad	99.96, 99.16
3	5	64	Categorical Cross Entropy	SGD	99.99, 99.14
4	5	128	Mean Squared Error	Adam	99.79, 98.96
5	10	128	Mean Squared Error	Adagrad	99.91, 99.12
6	5	256	Mean Squared Error	SGD	99.92, 99.20
7	5	128	Hinge	Adam	99.58, 98.98
8	4	32	Hinge	Adagrad	99.90, 99.23
9	5	64	Hinge	SGD	99.92, 99.25

As expected, accuracy is proportional to epochs and inversely proportional to batch size. All of the new models have accuracies higher than Model #0 (baseline) due to the increase in epochs and decrease in batch size. The loss function and optimizers were changed for each unique combination, and some combinations outperformed others.
Model #4 with Mean Squared Error loss and Adam optimization performed the least for testing accuracy, at 98.96%. Model #7 with Hinge and Adagrad followed after with 98.98%.
From the three optimizers analyzed, SGD seemed to perform the best. The low learning rate allows it to converge to more optimal solutions. Adam converges faster but is generally less optimal for image classification problems like digit recognition due to its momentum changes.
In no way are these the most optimal classifiers, but their accuracies are improved from the baseline's, which was 97.75% test accuracy.

2.3 CNN Conclusions

From these results, it is difficult to create the 'perfect' or most optimal classifier, as there are still so many untested parameters that can be varied.
However, the conclusions that can be drawn are that a greater number of filters and smaller kernel size can improve accuracy. Activation methods cause a great change in the accuracy as different neurons are fired at different or simultaneous times. While increasing the number of connections in a CNN may improve accuracy, it can also increase run-time, which is not optimal.
More epochs means that the data is passed through the CNN more times and can perform better. However, an excessive number of epochs can cause over-fitting to training data, and may lead to the model 'memorizing' the data instead of learning it. A smaller batch size means that the weights are updated more frequently, but a too small batch size can greatly increase run-time.
Different loss functions and optimizers are used in different scenarios and may help or hurt the test accuracy. Adagrad, for example, is a very simple optimizer compared to Adam or SGD, but decays very quickly and can end up in less than optimal solutions more frequently.
Changing these hyper-parameters gives more freedom in how to construct the CNN and test it to incrase accuracy while keeping in mind run-time.

3 SVM

A C-SVM is constructed using LIBSVM as a different classification method for the digit recognition problem. LDA is used to reduce the dimensions of data. Because we prioritize class separation, LDA is the better choice over PCA. There are 10 classes for the digits, so LDA was applied with 9 parameters. The hyper-parameters for the C-SVM are cost, gamma, and kernel type. The table shows the effect to run-time and accuracy when changing these parameters. SVM run-time is heavily influenced by these hyper-parameters, so it is important to note the relation.

Model #	Cost	Gamma	Kernel Type	Run-time	Accuracy (%)
1	5	0.05	radial basis	55s	92.53
2	10	0.1	linear	3m 5s	89.33
3	5	0.1	polynomial	1m 32s	91.56
4	1	0.01	sigmoid	1m 7s	89.03
5	100	0.2	radias basis	1m 24s	90.67
6	1000	0.01	polynomial	1m	91.64
7	60000	0.1	radial basis	2m 16s	89.49
8	10000	0.05	sigmoid	39s	78.41
9	5000	0.1	radial basis	1m 32s	89.75
10	500	0.01	polynomial	46s	91.52
11	100	0.1	sigmoid	1m 30s	64.27
12	50	0.05	polynomial	1m 18s	91.49
13	50	0.05	radial basis	50s	92.32
14	25	0.05	radial basis	41s	92.48
15	10	0.05	radial basis	38s	92.55

These 15 tests have variations in the hyper-parameters and show their relation to run-time and accuracy. With a minimum accuracy of 64.27% and a maximum of 92.55%, there are key factors that affect it. The three tested models with sigmoid kernel type have the three lowest accuracies. While this could be attested to the change in other hyper-parameters, the sigmoid function does not seem to be the most optimal kernel function for this dataset.

The model with linear kernel function had the highest run-time out of all other models, and was almost a minute more than the next highest. Due to this extremely high run-time, the linear model was disregarded for the rest of the models as it doesn't make sense to sacrifice run-time for an accuracy that can achieved with other models.

The polynomial kernel function has pretty low run-times compared to the other two, but the accuracy was slightly lower than radial basis with similar cost and gamma hyper-parameters. This is why most of the models were ran with RBF. The cost and gamma hyper-parameters were varied to see how they affect accuracy.

In theory, a high cost would increase accuracy as it a regularization parameter used to change the decision function margin. A higher gamma would reduce the influence that each training sample has in the entire model. It turns out that an extremely high cost decreases the accuracy regardless of what gamma is, as seen in Models #7-9.

The models with RBF tend to outperform the other models, with cost and gamma updated to optimize it. Surprisingly, Model #15 had both highest accuracy and lowest run-time, making it much more optimal than the other models.

More testing and fitting can be done to make a more optimal classifier, but a relatively high accuracy was still able to be achieved.

4 Closing Thoughts

From first glance, the CNN has a much higher accuracy than SVM classification. In general CNNs are more suitable for image recognition tasks than SVMs. This is attributed to CNNs Hierarchical feature extraction, convolutional layers, pooling layers, and regularization techniques. CNNs are able to extract the most important featuers of an image and then reduce the dimensionality with pooling layers while still keeping important information. CNNs are well-suited for image recognition tasks such as handwritten digit recognition. SVMs on the other hadn are more commonly used for classification tasks involving tabular or text data.