ENTS 669D-Introduction to Machine Learning

Vivek Bhave

115628496

Project 2: Hand-Written Digits Recognition

Aim: My main aim of the project to achieve hand-written digit recognition by using different conventional classifiers. I have used Support Vector Machine (SVM) algorithm and built a Convolutional Neural Network (CNN) using Deep Learning. I have tested on the MNIST dataset, a public dataset of handwritten digits to see how the classifiers perform. Also, I have experimented with several parameters to notice the effect of small changes in the model to the accuracy of the classifiers.

The Report has been organized as follows:

1) SVM Results
   A. SVM with PCA Results
   B. SVM with LDA Results
2) Deep Learning (CNN Results)
   I. Single-Layer CNN
   II. Multi-Layer CNN
3) Conclusion

Important Notes:

1. The MNIST dataset has 60000 training digit images and their labels and 10000 test digit images and labels. These images have been preprocessed and gray-scaled, so that the digits are fit and centered in the 28x28 image.
2. Each image has a dimensionality of 784 and there are 10 classes.
3. Effectiveness Measure: Accuracy is chosen as the effectiveness measure, where Accuracy= $\frac{\text{number of correctly classified test points}}{\text{total number of test points}}$

1. SVM Results

Brief summary of the classifier: A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples. In two dimensional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side.

I implemented SVM using MATLAB. I used LIBSVM, a machine-learning library for support-vector classification. It contains functions such as loadMNISTImages and loadMNISTLabels used to extract images and labels from the MNIST dataset. I used the function called svmtrain to train the classifier.

The results obtained were as follows:
Accuracy: 93.98%
Training time: 588.69 seconds
Prediction time: 218.10 seconds

Takeaways:
SVM is a fairly easy classifier to implement with a good accuracy rate. It's training and prediction time is high. It also needs a fair amount of processing power.

A. SVM with PCA Results

Brief summary about PCA: A component of along which the variance among the data is maximized is known as the first principle component. Using a number of these principle components the dimensions of the data is reduced. This significantly reduces computational complexity.

After reducing the dimensions along the principle components of the training and testing data, this new “dimension-reduced” data is passed through the SVM classifier.

Results:
Accuracy with 60000 training samples: 61%
Training time: 230.86 seconds
Prediction time: 14.89 seconds

Although the accuracy went down quite a bit the execution time is almost a fourth of that when PCA was not applied.

But implementing PCA with 60000 training samples took a large amount of time. If I reduced the training samples to 10000, the accuracy went down sharply to 9.8%.

B. SVM with LDA Results

Brief summary about LDA: This is primarily a classification technique (known as Fisher's discriminant analysis). But this can also be used to reduce dimensions of the image to c-1 dimensions. We not only find the axis that maximizes the scatter between classes (between class scatter matrix) but also find the axis that corresponds to minimum scatter within a class (within class scatter matrix). Unlike PCA, LDA is supervised as we need to provide class labels.

After reducing the dimensions using LDA of the training and testing data, this new "dimension-reduced" data is passed through the SVM classifier.

Results:
Accuracy with 60000 training samples: 49%
Training time: 212.76 seconds
Prediction time: 10.78 seconds

Although the accuracy went down quite a bit the execution time is almost a fourth of that when LDA was not applied.

But implementing PCA with 60000 training samples took a large amount of time.

If I reduced the training samples to 10000, the accuracy went down sharply to 9.1%.

Takeaways:

Although the execution time lowers but implementing PCA and LDA on 60000 training samples takes a very long processing time. Even after that the accuracy goes down quite a bit.

By reducing the number of training sample, the classifier's performance completely deteriorates.

2. Deep Learning (CNN) Results

Brief summary of classifier: Convolutional Neural Networks (**ConvNets** or **CNNs**) are a category of Neural Networks that have proven very effective in areas such as image recognition and classification. ConvNets have been successful in identifying faces, objects and traffic signs apart from powering vision in robots and self-driving cars.

I have built CNNs on Tensorflow (Python). TensorFlow is a powerful library for doing large-scale numerical computation. One of the tasks at which it excels is implementing and training deep neural networks. Tensorflow has the MNIST dataset built into one of its import statement.

I. Single Layer CNN

I built a single layer CNN using tensorflow using a softmax regression model. After importing the MNIST dataset, initializing placeholders, weights and biases, I implemented the regression model by multiplying the input images with the weights and adding biases. Then I specified the loss function using tf.nn.softmax_cross_entropy_with_logits which internally applies the softmax on the model's unnormalized model prediction and sums across all classes, and tf.reduce_mean which takes the average over these sums.

Training:
I trained the model using a Gradient Descent Optimizer with batches of the training sample.

Results:
Accuracy: 91.9%
Execution time: 0.89 seconds

Variation of training batch size:

Batch size	Accuracy (%)
100	91.9
200	91.94
500	92.15
1000	92.10

Overall the accuracy remains fairly constant, but there is slight increase in execution time with increase of batch size.

II. Multi-Layer CNN

I built a 4-layer CNN using tensorflow. Firstly, I initialized the weights and biases. I used ReLu neurons which need to be initialized to avoid “dead neurons”.

First Layer:
The first layer consists of convolution followed by max pooling. The convolution will compute 32 features for each 5x5 patch. Its weight tensor will have a shape of [5, 5, 1, 32]. The first two dimensions are the patch size, the next is the number of input channels, and the last is the number of output channels. There is also have a bias vector with a component for each output channel.

Second Layer:
The second layer has 64 features for each 5x5 patch.

Third Layer (Densely Connected Layer):
The image size has been reduced to 7x7. A fully-connected layer with 1024 neurons is added to allow processing on the entire image. The tensor is reshaped from the pooling layer into a batch of vectors, multiplyed by a weight matrix, added with a bias, and applied a ReLU.

Dropout:
To reduce overfitting, a dropout is applied before the readout layer. A placeholder is created for the probability that a neuron's output is kept during dropout. This allows us to turn dropout on during training and turn it off during testing. TensorFlow's tf.nn.dropout op automatically handles scaling neuron outputs in addition to masking them.

Readout Layer:
It is a layer like the one-layer softmax regression.

Training:
ADAM optimizer is used which is a more sophisticated optimizer.
A parameter keep_prob is added to feed_dict to control dropout rate.
It has 20000 training iterations.

Results:
Accuracy: 99.189%
Execution time: close to one hour

Takeaways:

A single layer CNN is very easy to build and train, having the a very low execution time and processing required.

A Multilayer CNN has a huge training time and processing required. It is much more complex compared to single-layer CNN. But it gives a very high accuracy rate.

3. Conclusion

In conclusion:

1. SVM is easy to implement and gives a good accuracy rate with decent execution time.
2. Using PCA and LDA with SVM, reduced the execution time but also lowered the accuracy. The processing time to implement PCA and LDA was very large compared to the execution time of SVM alone.
3. Single-layer CNN was easy to implement (not easier than SVM) and gave a decent accuracy rate and execution time.
4. Multi-layer CNN was the most complex and most difficult to implement. It required a large processing time and power. But its accuracy is the best among all the classifiers.