THE UNIVERSITY OF TEXAS AT DALLAS

MACHINE LEARNING PROJECT

CS 6375.001

**TOPIC: DIGIT RECOGNITION**

Team Members:

Nandita Koppisetty - nxk174230

Priyanka Gadde - pxg170630

Sushma Sunkollu Nagaraj - sxs173433

# **Introduction:**

Visual texts are widely used as one of the most important forms of human language. Extracting contents from images has always been a challenge and a lot of research is focused towards this field. Also, today more and more information is transformed into digital form and the visual texts are embedded in various forms of digital media. Converting printed or handwritten document into electronic data that can be used in various computer systems will be of great importance.

# **Implementation Details:**

In this project we have made a small attempt to demonstrate the capability of Machine learning techniques in recognition of handwritten text. The MNIST dataset obtained will be trained by applying deep learning techniques using TensorFlow. TensorFlow is a framework used to create deep learning models. The entire project is aimed at applying the best possible technique which aids in increasing the accuracy of results.

# **Experimental Evaluation:**

**Algorithm 1 : Simple and Larger Convolutional Neural Network (CNN)**

**Data Set Source : MNIST**

**Framework : TensorFlow**

**Library : Keras**

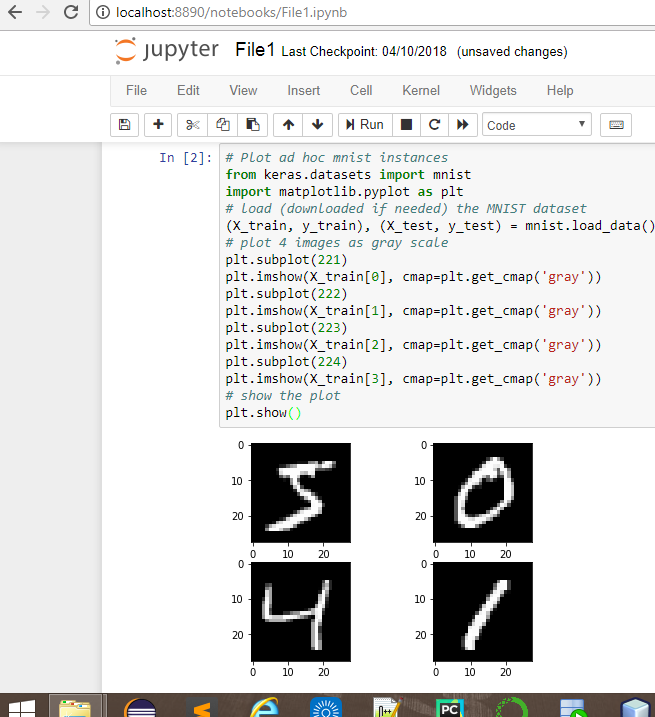
Keras is an open source neural network interface written in Python. It is capable of running on tensor flow.

The MNIST is a dataset generally used for evaluating machine learning models on handwritten digit classification problem. All the images in the dataset are taken from variety of scanned document normalized in size and centered normalized in size and centered. Each image is a 28 by 28 pixel square (784 pixels total). A standard spit of the dataset is used to evaluate and compare models, where 60,000 images are used to train a model and a separate set of 10,000 images are used to test it.

It is a digit recognition task. As such there are 10 digits (0 to 9) or 10 classes to predict. Results are reported using prediction error and scores.

# **Methodology:**

The images are taken from the MNIST datasets. The datasets are in the following format:



This data is further reshaped so that the data can be used to train the CNN.  In Keras, the layers used for two-dimensional convolutions expect pixel values with the dimensions [pixels] [width] [height].We normalize the data so that we get consistent results. All the features are normalized from 0 -255 to 0 to 1, to give equal importance to all features.

# **Working:**

Convolutional neural networks are more complex than multilayer perceptron. They generally give better results.

The following summarizes the working of simple convolutional networks:

1. The first hidden layer is a convolutional layer called a Convolution2D. The layer has 32 feature map, each of size 5×5 and a rectifier activation function. The input layer has a structure outline above [pixels] [width] [height].
2. Next we define a pooling layer that takes the max called MaxPooling2D. It is configured with a pool size of 2×2.
3. The next layer is a regularization layer using dropout called Dropout. It is configured to randomly exclude 20% of neurons in the layer in order to reduce overfitting.
4. Next is a layer that converts the 2D matrix data to a vector called Flatten (). It allows the output to be processed by standard fully connected layers.
5. Next a fully connected layer with 128 neurons and rectifier activation function.
6. Finally, the output layer has 10 neurons for the 10 classes and a softmax activation function to output probability-like predictions for each class.

The Convolutional Neural Networks converge by Gradient Descent.  The CNN is fit over 10 epochs (iterations) with a batch size of 200. An epoch is one forward pass and one backward pass through the neural network. Batchsize defines number of samples that going to be propagated through the network.

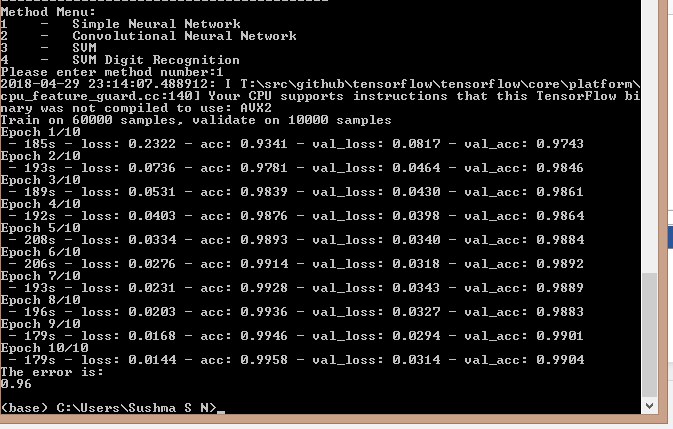
Running the example, the accuracy on the training and validation test is printed each epoch and at the end of the classification error rate is printed.

For larger convolutional neural networks we have additional convolutional, max pooling layers and fully connected layers. The network topology can be summarized as follows.

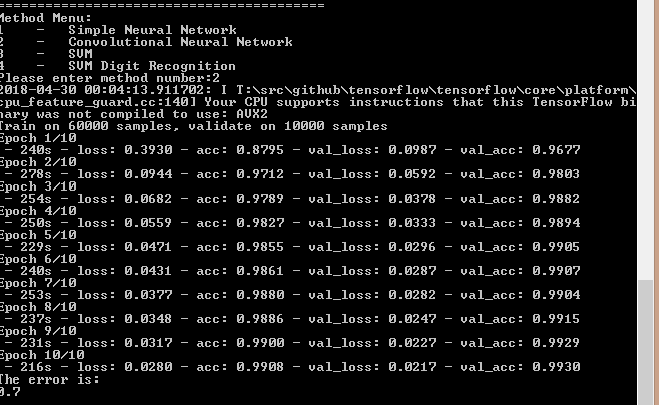
1. Convolutional layer with 30 feature maps of size 5×5.
2. Pooling layer taking the max over 2\*2 patches.
3. Convolutional layer with 15 feature maps of size 3×3.
4. Pooling layer taking the max over 2\*2 patches.
5. Dropout layer with a probability of 20%.
6. Flatten layer.
7. Fully connected layer with 128 neurons and rectifier activation.
8. Fully connected layer with 50 neurons and rectifier activation.
9. Output layer.

# **OUTPUT:**

## **Simple Convolutional Neural Networks:**



**Larger Convolutional Neural Networks**:



# **Conclusion:**

From the above screenshots we can observe that larger convolutional networks provide better accuracy than simple convolutional networks. This is because we stack more layers and pooling for the larger CNN which helps to train more and more features of higher abstraction which results in the training samples being classified more accurately.

# **Algorithm 2: Linear SVM**

**Data set Source : MNIST**

**Library : Scikit, Sci-image, Sklearn, cv2, Collections**

# **Methodology:**

Linear Support Vector Machines try to find a hyperplane that separates the training data into two classes with a maximum margin. In our case the class of a data point is the digit it represents. We want to maximize the margin between the hyperplane and the two classes to minimize the error of incorrectly recognizing a digit. The hyperplane then divides the data, so that everything above the hyperplane belongs to one class and everything below the hyperplane belongs to the other class. Each pixel value of the 28 by 28 image is represented in its own dimension, meaning that an image is a point in a space with 28 \* 28 = 784 dimensions. And the hyperplane divides the data points into two classes in this 784 dimensional space.

In this analysis Support Vector Machines (SVM) are used to train a model to classify digits. The feature representation method Histogram of Oriented Gradients (HOG) are used as the feature representation.

The histogram of oriented gradients (HOG) is a [feature descriptor](https://en.wikipedia.org/wiki/Feature_descriptor) used in [computer vision](https://en.wikipedia.org/wiki/Computer_vision) and [image processing](https://en.wikipedia.org/wiki/Image_processing) for the purpose of [object detection](https://en.wikipedia.org/wiki/Object_detection). The technique counts occurrences of gradient orientation in localized portions of an image. This method is similar to that of [edge orientation histograms](https://en.wikipedia.org/w/index.php?title=Edge_orientation_histogram&action=edit&redlink=1), [scale-invariant feature transform](https://en.wikipedia.org/wiki/Scale-invariant_feature_transform) descriptors, and [shape contexts](https://en.wikipedia.org/wiki/Shape_context), but differs in that it is computed on a dense grid of uniformly spaced cells and uses overlapping local contrast normalization for improved accuracy.

# **Working:**

1. First we load data from the MNIST dataset.
2. We rescale the features using histogram of oriented gradients (HOG) features.
3. The samples are then normalized to give equal importance to all features.
4. Next we implement Linear Support Vector Classification. Similar to SVC with parameter kernel=’linear’, but implemented in terms of liblinear rather than libsvm, so it has more flexibility in the choice of penalties and loss functions and should scale better to large numbers of samples. This class supports both dense and sparse input and the multiclass support is handled according to a one-vs-the-rest scheme.

Note: We use the loaded MNIST data to train the classifier.

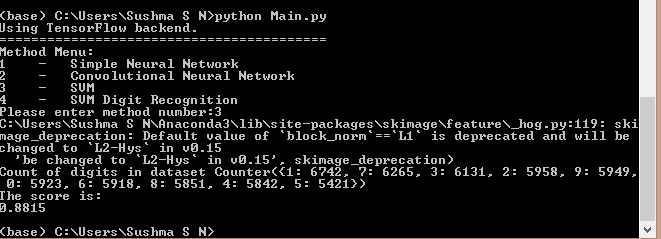
1. When we execute the DigitClassifier\_SVM.py file a classifier file called DigitClassifier.pkl is created. It’s a script file which aids in digit recognition.
2. The result of the DigitClassifier\_SVM.py is a score value which depicts the level of accuracy of the digits of the test data using hog features. The higher the score better is the accuracy.
3. We further input image with handwritten digits to DigitRecognition\_SVM.py. The DigitRecognition.py uses the DigitClassifier.pkl file as a template to recognize the digits on the image.
4. The input image is converted to grayscale as it is required for thresholding. Applying the Gaussian filter which helps to extract the edges of target object in an image which helps to match with the template as in digit classifier file.
5. Performing image thresholding, the first argument being the input image and Second argument is the threshold value which is used to classify the pixel values. Third argument is the maxVal which represents the value to be given if pixel value is more than (sometimes less than) the threshold value and the fourth parameter has the output after thresholding.
6. We then find contours in the image as it helps in shape analysis and object recognition.

When executed the DigitRecognition.py is executed, a window is opened where the digits are bounded within rectangular boxes with their respective classification displayed above.

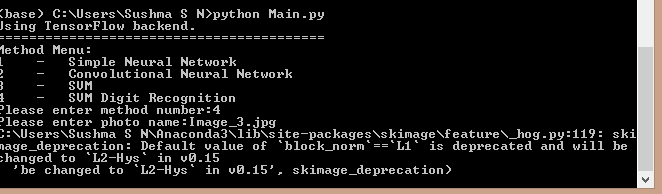
The bounding rectangles help in creating an approximate rectangular region around the digit which helps to compare with template in classifier file.

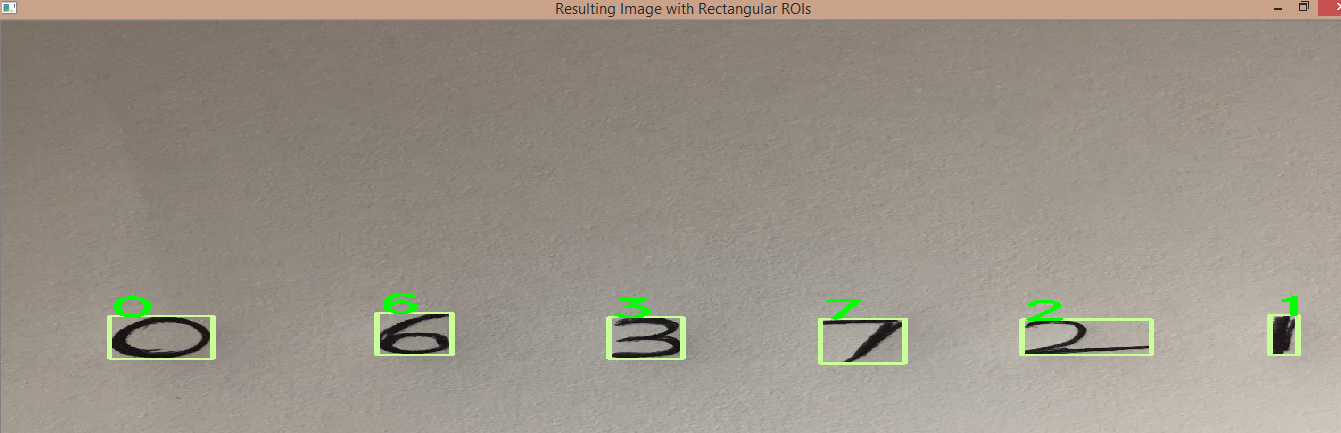
# **OUTPUT:**

a)Output obtained after running the DigitClassification\_SVM.py file.



b) Output obtained after running the DigitRecognition\_SVM.py file





# **Future Work:**

The accuracy for CNN can be further increased by performing the preprocessing to the MNSIT dataset:

The several techniques as shown below:

**1) Feature standardization to the MNSIT dataset:**

The result that standardizing images brings is slightly darkening and lightening different images which helps to obtain better results.

**2) ZCA Whitening:**

Using this technique we reduce the redundancy of certain pixels in order to highlight certain features of images it is very much similar to principal component analysis

**3)Random rotations:**

It is a known fact that different people write in different angles, in order to maintain the uniformity we can randomly rotate images up to 90 degrees.

**4)Random shifts:**

When people write sometimes numbers won't be exactly centered. Thus we can randomly shift numbers to be slightly off-centered to further increase the accuracy.

# **Conclusion:**

The comparison between accuracies for Simple and Convolutional networks:

Simple Convolutional Networks: **Error**: 0.96

Large Convolutional Neural Networks: **Error**: 0.7

We can observe that larger convolutional networks provide better accuracy than simple convolutional networks. This is because we stack more layers and pooling for the larger CNN which helps to train more and more features of higher abstraction which results in the training samples being classified more accurately.