Handwritten Digit Recognition

A Minor Project Report

Submitted in Partial Fulfillment of the

requirement for the award of the degree

Of

BACHELOR OF TECHNOLOGY (B.Tech)

In
Information Technology
By

Ananta Arora
(159103009)
Ankit Kumar
(159102017)

Under the Guidance of
Ms. Neha Sharma

School of Computing and Information Technology

MANIPAL UNIVERSITY JAIPUR
JAIPUR-303007
RAJASTHAN, INDIA

May/2018

CERTIFICATE

Date: 04-05-2018

This is to certify that the project entitled **“HANDWRITTEN DIGIT RECOGNITION”** is a record of the bonafide work done by **ANKIT KUMAR** (159102017) submitted in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology (B.Tech) in **(Information Technology)** of Manipal University Jaipur, during the academic year 2017-18.

Ms. Neha Sharma
Project Guide, Dept. of Information Technology
Manipal University Jaipur

Dr. Sumit Srivastava
HOD, Dept. of Information Technology
Manipal University Jaipur

CERTIFICATE

Date: 04-05-2018

This is to certify that the project entitled **"HANDWRITTEN DIGIT RECOGNITION"** is a record of the bonafide work done by **ANANTA ARORA** (159103009) submitted in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology (B.Tech) in **(Information Technology)** of Manipal University Jaipur, during the academic year 2017-18.

Ms. Neha Sharma
Project Guide, Dept. of Information Technology
Manipal University Jaipur

Dr. Sumit Srivastava
HOD, Dept. of Information Technology
Manipal University Jaipur

ABSTRACT

The aim of this project is to implement Support Vector Machine (SVM) algorithm and a multi-layer neural network, Convolutional Neural Network (CNN) to recognize and predict handwritten digits ranging from 0 to 9.

The recognition of handwritten digits is tough task as everyone has different writing style. We performed Principal Component Analysis (PCA) on the dataset which we acquired from MNIST, to reduce the dimensions. We got an accuracy of 88.7% when we changed the color map of the digits from gray to binary. But reduction in dimensions allowed us to increase the accuracy from 88.7% to 97.9% using SVM algorithm. Further the experiment done on model made by using Convolutional layers gave an accuracy of 99.7%.

Keywords—*Convolutional Neural Network, Support Vector Machine, Neural Network, Principal Component Analysis, MNIST, Machine Learning, Softmax.*

List of Figures

Table of Contents

1. INTRODUCTION

1.1 SCOPE OF THE WORK

Machine Learning is a practical approach for Artificial Intelligence. It uses concepts of Statistics, Probability, Data Science, Computer Algorithms and Programming. The goal of a machine learning model is to make predictions using the data. It is one of the most exciting fields in academic and industrial research.

Machine Learning finds various applications in

1. Driverless cars (Google, Tesla)
2. Digital Assistants (OK Google, Siri, Cortana)
3. Recommendation systems (Amazon, Netflix)
4. Spam/Fraud detection (Email, Payment Gateways)
5. Computer vision (OCR, Image Transcription)
6. Various other automation

Deep learning is one such field of Machine Learning, which has been introduced with the objective of moving Machine Learning to its original goals: Artificial Intelligence.

A popular demonstration of the capability of deep learning techniques is object recognition in image data. The “hello world” of object recognition for machine learning and deep learning is the MNIST dataset for handwritten digit recognition.

We used MNIST dataset for handwritten digit recognition task in Python using the Keras deep learning library and SciKit library for machine learning algorithms. Handwritten digit recognition has been used to recognise amounts written on cheques for banks, zip codes on envelops for postal services. It can also be helpful to recognize the house numbers which are written by hand.

1.2 OBJECTIVE

The objective of our project is to create a model capable enough to analyze and classify what handwritten digits from different pixels are generating once we scan and upload them through the layers of our model. We have used the MNIST dataset of training (42000 rows and 785 columns) and testing data (28000 rows and 784 columns) of handwritten digits.

2. REQUIREMENT ANALYSIS

2.1 SOFTWARE REQUIREMENTS

There are certain softwares which are necessarily required in the building of our machine learning and deep learning models.
Few of them to mention are:

1. Jupyter Notebook
2. Anaconda
3. Browser (Prefer - Google Chrome)
4. Terminal or Command Prompt
5. Python 3.x

We used Jupyter notebook as our development environment and trained our models in that application. Jupyter can be installed using two methods:

- If you have python installed you can use **pip** to install the Jupyter notebook
  - python3 –m pip install jupyter
- Another method to install jupyter is by using **Anaconda**.
- To open jupyter notebook we just need to write jupyter notebook in our terminal and it opens up in the browser as our home directory as the root directory in the tab.

2.2 LIBRARIES

We installed tons of libraries to train out our dataset in Python. Few of them are enlisted here:

- **Numpy** – It is a fundamental package for scientific computing in Python. It contains a powerful N-dimensional array object which has been helpful throughout of this project.
- **Pandas** – Pandas is python data analysis library which is used to make Series and Data Frames. It is a great tool for analyzing and preprocessing the data.
- **Matplotlib** – This library is useful for plotting the graphs.
- **Scikit** – We used this library to implement the machine learning algorithms.
- **Keras** – It was used to import the neural network.

3. METHODOLOGY

3.1 Support Vector Machine (SVM)

Support Vector Machine is used to find the strategy for linearly separable or non-linearly separable data. In other words, given labeled training data (supervised learning), the algorithm outputs an optimum hyperplane.
It uses non-linear mapping to transform the original training data into a higher dimension and within this dimension it searches for the optimal separating hyperplane also known as decision boundary.

From a high dimension data from two classes can always be separated by a hyperplane. SVM can be used for both classification and regression. However, it is mostly used in *classification* problems.

Example:
Here we have two data. One in green and one in red. We need to separate them from each other. What are the possible ways using which we can separate the different datasets.

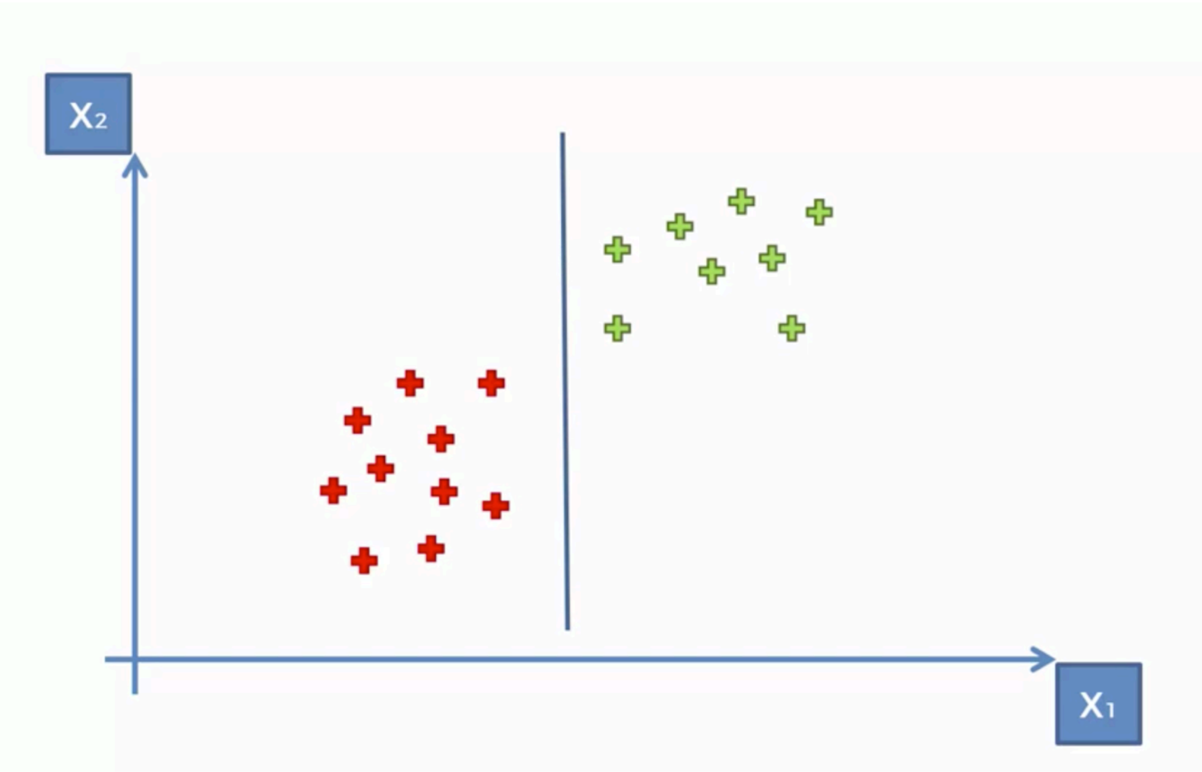

Figure 1-Separating data using hyperplane

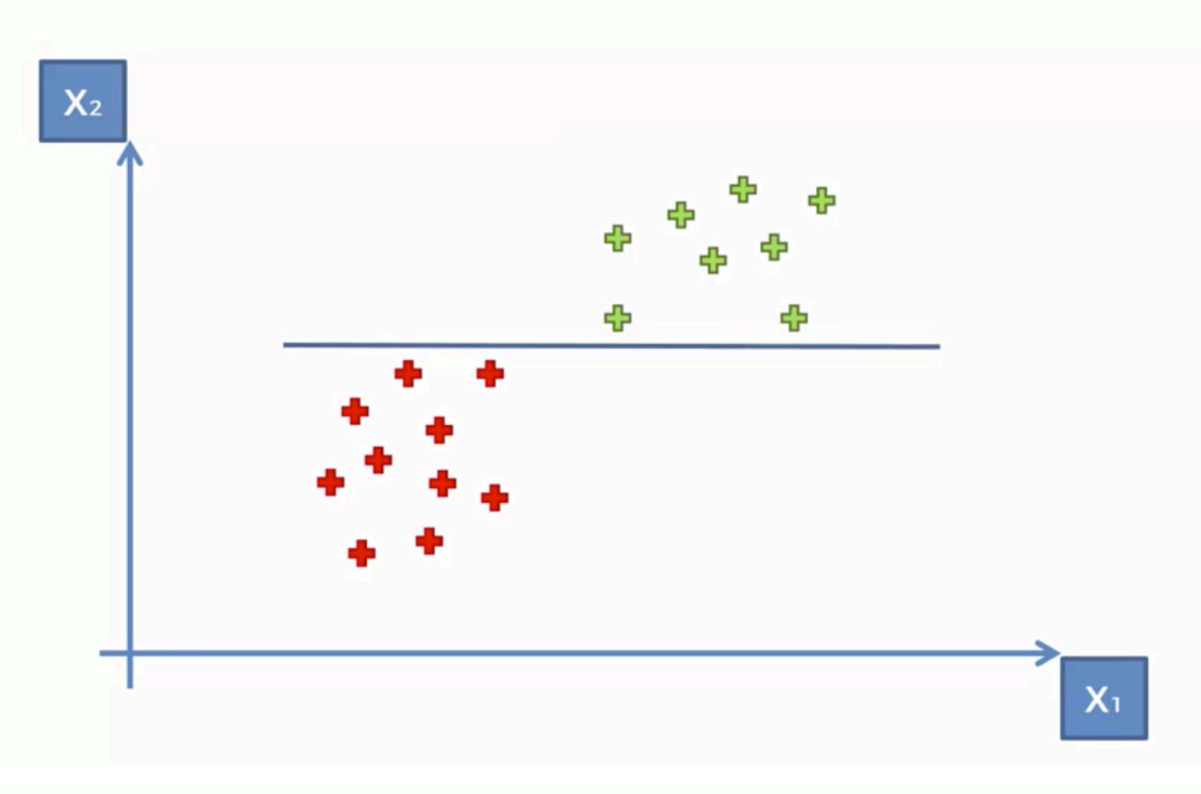

Figure 2-Horizontal Hyperplane

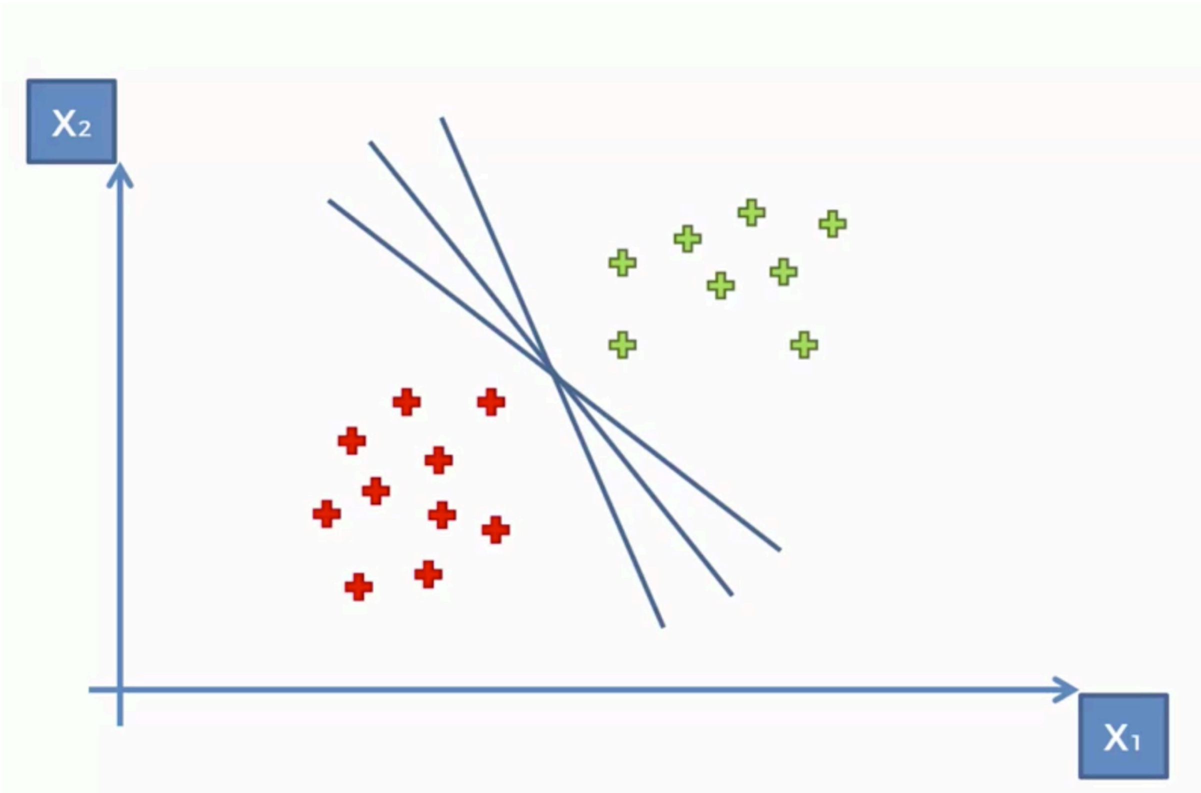

Figure 3-Possible Hyperplanes

3.1.1 Support Vector

It is an input vector that just touch the boundary of the margin that are circled in the diagram. We separate the data such that they have maximum margin between them. SVM selects the hyperplane which classifies the classes accurately prior to maximizing margin.

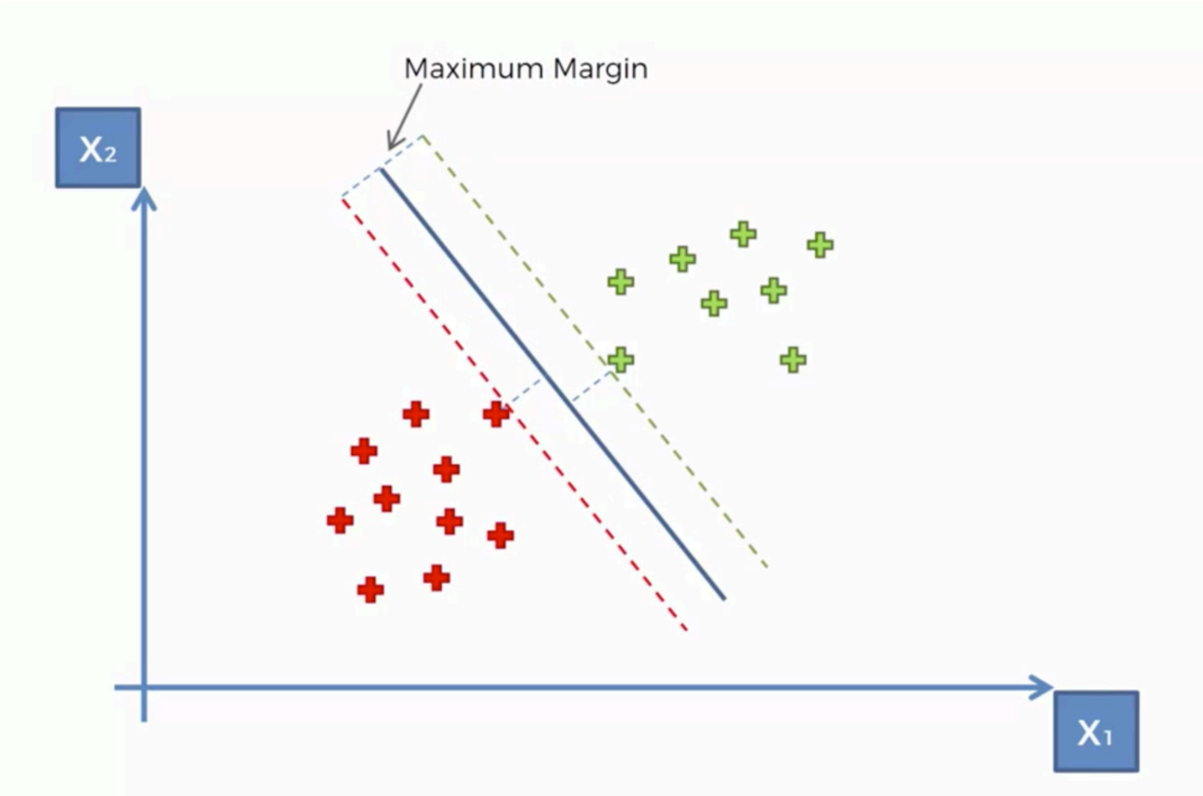

Figure 4- Maximum Margin

There are two distances available, d^+ and d^-
D^+: Shortest distance to the closest positive point from the support vector hyperplane.
D^-: Shortest distance to the closest negative point from the support vector hyperplane.

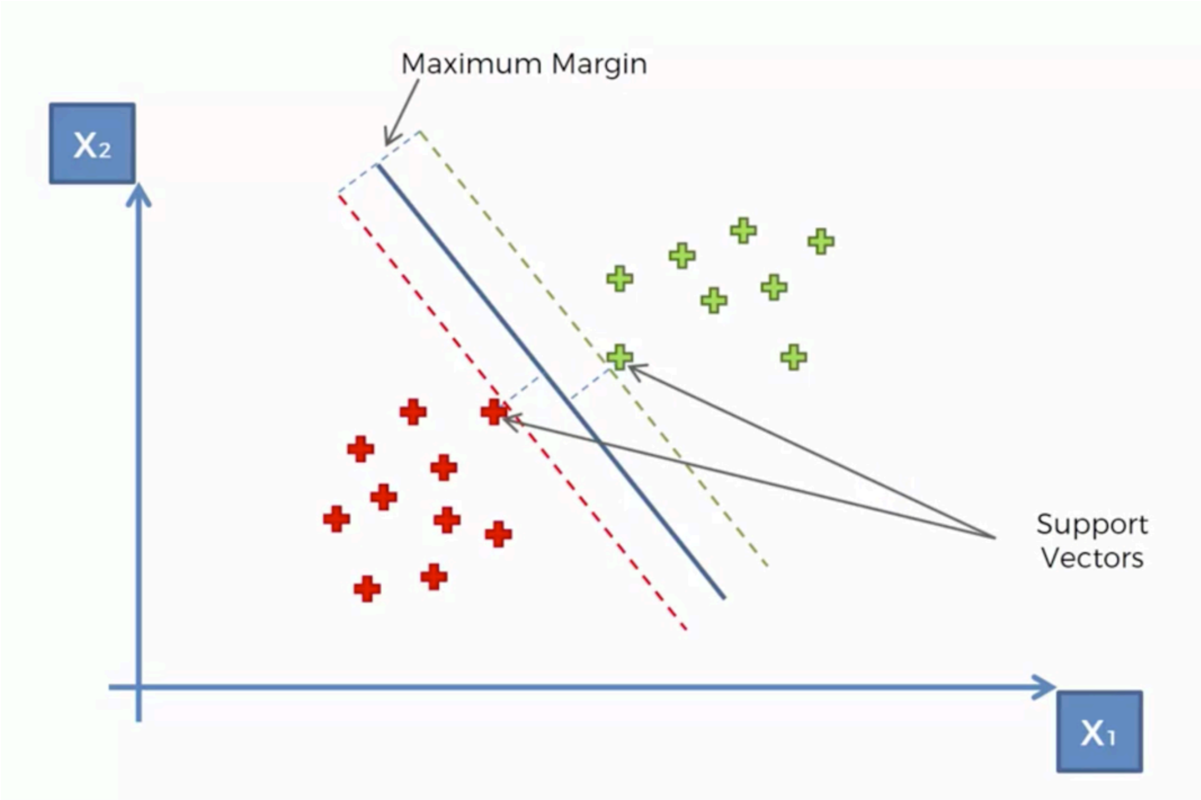

Figure 5-Support Vectors

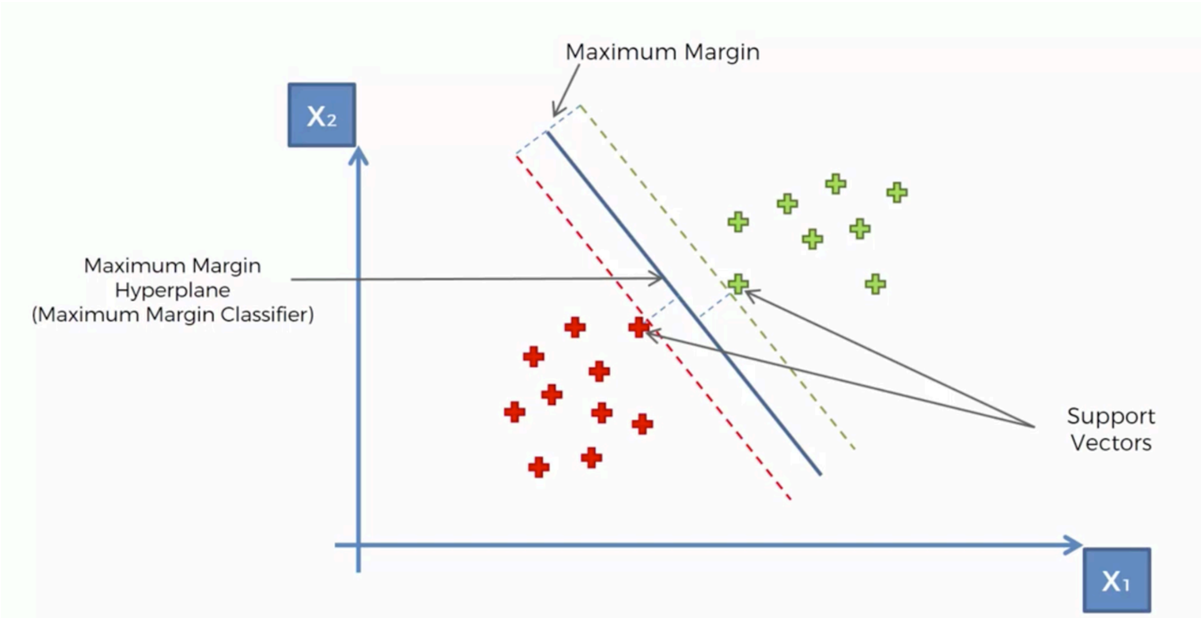

Figure 6-Hyperplanes

3.2 Principal Component Analysis (PCA)

Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called **principal components**. It is used to make data easy to explore and visualize.

2D Example

First, consider a dataset in only two dimensions, like (height, weight). This dataset can be plotted as points in a plane. But if we want to tease out variation, PCA finds a new coordinate system in which every point has a new (x, y) value. The axes don't actually mean anything physical; they're combinations of height and weight called "principal components" that are chosen to give one axes lots of variation.

Drag the points around in the following visualization to see PC coordinate system adjusts.

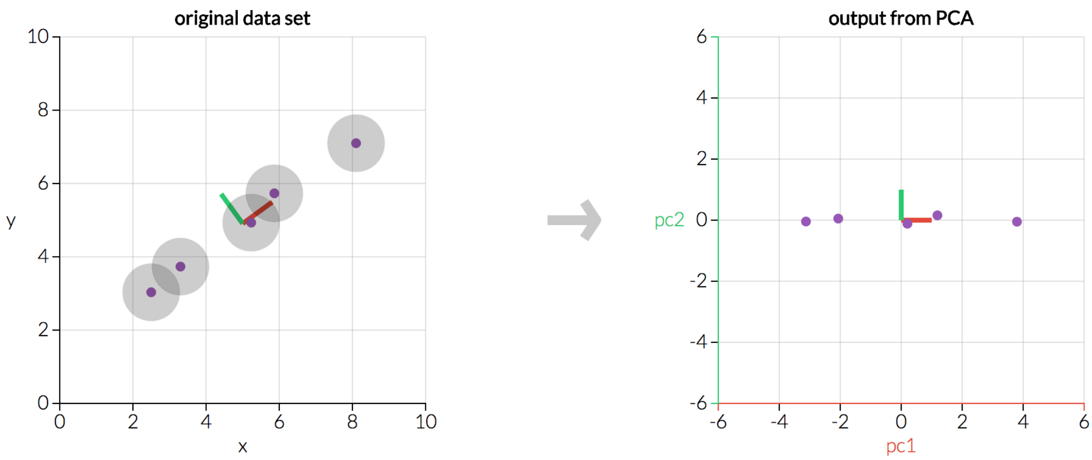

Figure 7-PCA on 2D data

PCA is mainly used for dimensionality reduction in a data set.

3.3 Convolutional Neural Network (CNN)

Convolutional Neural Networks (CNN), is a type of advanced artificial neural network. It differs from regular neural networks in terms of the flow of signals between neurons. Typical neural networks pass signals along the input-output channel in a single direction, without allowing signals to loop back into the network. This is called a forward feed.

Convolution has the nice property of being **translational invariant**. Intuitively, this means that each convolution filter represents a feature of interest (e.g. whiskers, fur), and the CNN algorithm learns which features comprise the resulting reference (i.e. cat). The output signal strength is not dependent on where the features are located, but simply whether the features are present. Hence, a cat could be sitting in different positions, and the CNN algorithm would still be able to recognize it.

CNN can be summarized in 4 major steps:

1. Convolution
2. Max Pooling
3. Flattening
4. Full Connection

3.3.1 Convolution

The convolutional layer is applied and a convolution operation is done for the input, passing the results to the next layer.
It combines and integrates two functions to generate third function. A feature map is generated based on the feature detector which acts as the filters in CNN.

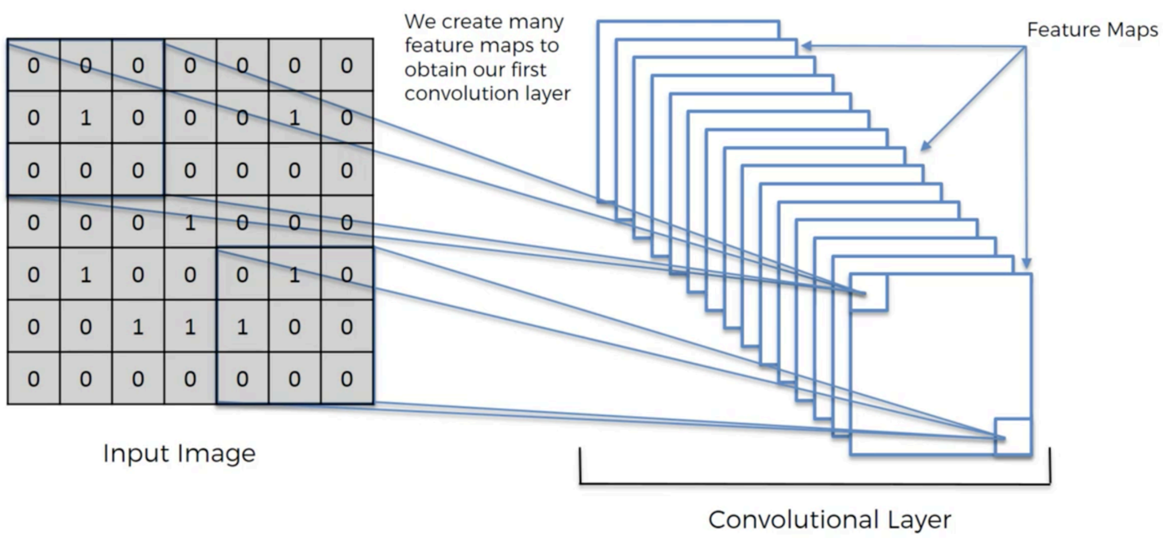

Figure 8-Convolutional Layer

ReLU Layer

- Rectified Linear Unit
- Additional step on top of our Convolutional step
- We apply rectifier function on the feature map because we want to increase non-linearity in our data.

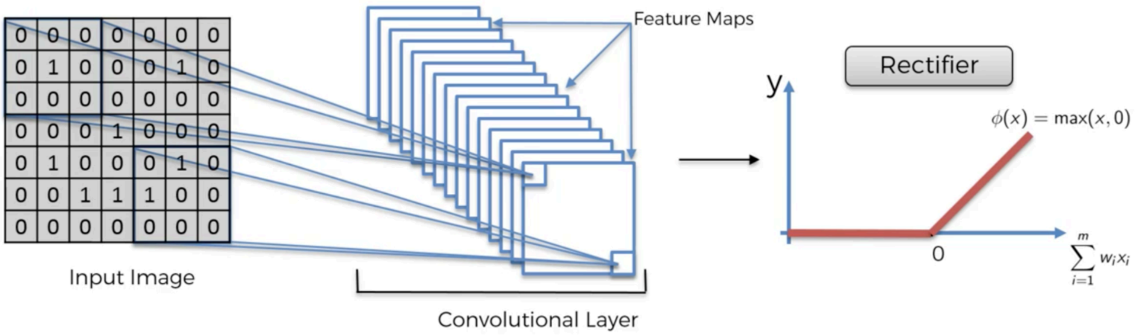

Figure 9-ReLU layer

3.3.2 Pooling

Convolutional neural networks may include one or more local or global pooling layers. For example, *max pooling* uses the maximum value from each of the cluster of neurons formed. So, we don't lose any important information.

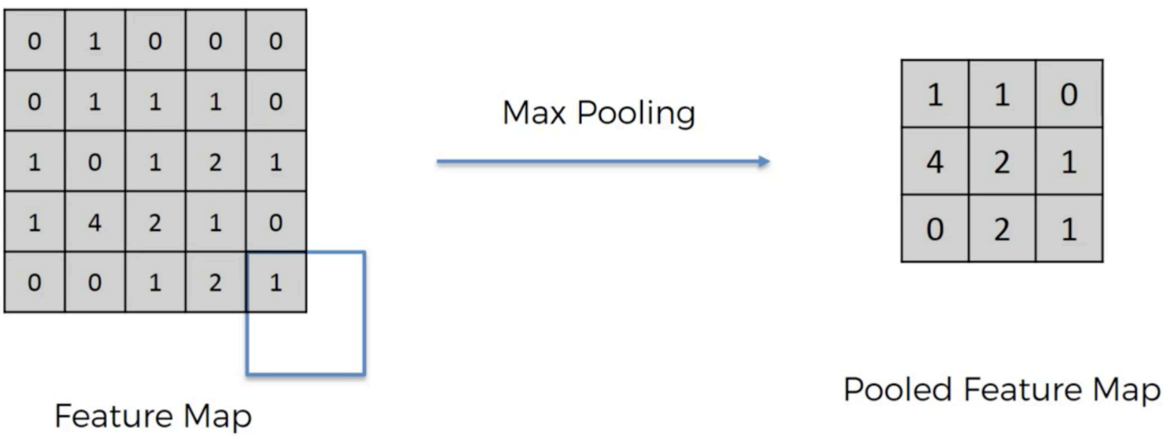

Figure 10-Pooling

3.3.3 Flattening

In flattening we convert the matrix of features into single column and pass it through a neural network. We practically do this as we take them as input for out CNN.

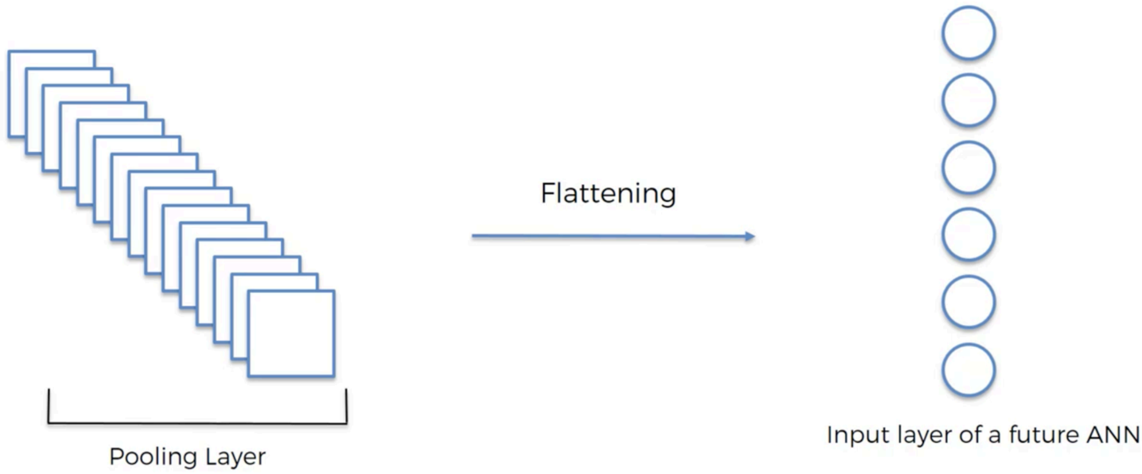

Figure 11-Flattening

3.3.4 Fully Connected

Fully connected layers connect every neuron in one layer to every neuron in another layer. It is in principle the same as the traditional multi-layer perceptron neural network.

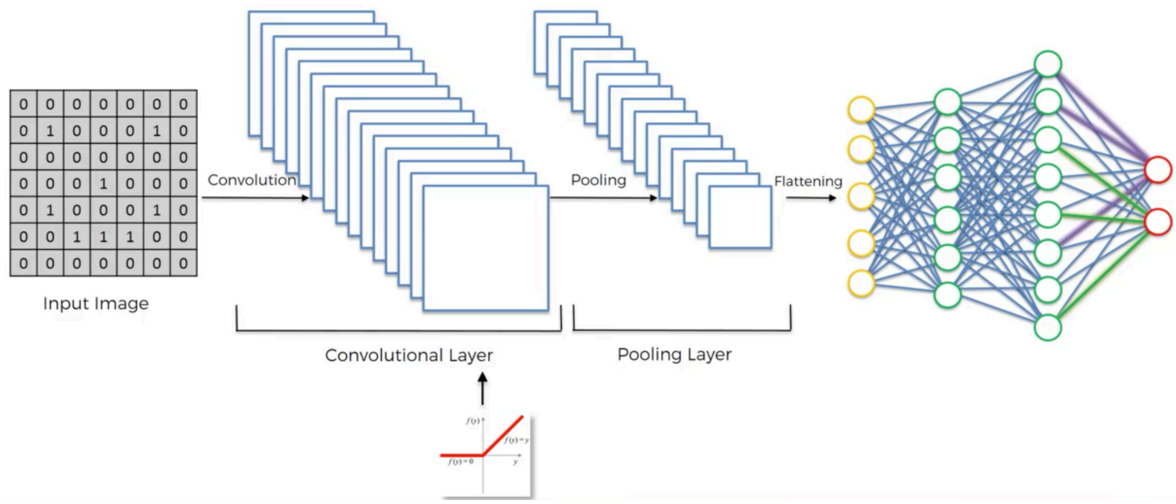

Figure 12-full connection

4. IMPLEMENTATION AND RESULTS

4.1 Using SVM

- Training our model for grayscale image for a training dataset of 5000 pixels.

```
In [4]: i = 1
        img = train_images.iloc[i].as_matrix()
        img = img.reshape((28, 28))
        plt.imshow(img, cmap = "gray")
        plt.title(train_labels.iloc[i,0])
        executed in 240ms, finished 11:17:41 2018-04-03
Out[4]: Text(0.5,1,'6')
```

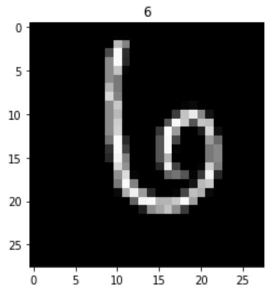

Figure 13-SVM_Gray

- Training out model and predicting the accuracy

```
In [6]: clf = svm.SVC()
        clf.fit(train_images, train_labels.values.ravel())
        clf.score(test_images, test_labels)
        executed in 28.9s, finished 11:18:23 2018-04-03
Out[6]: 0.10000000000000001
```

Figure 14-SVM_output

- We found that our model predicted extremely low score of 10%.
- To improve the result, we simply removed the gray color and replaced it with the binary i.e. black and white color mapping.

```
Out[7]: Text(0.5,1,'label    6\nName: 3275, dtype: int64')
```

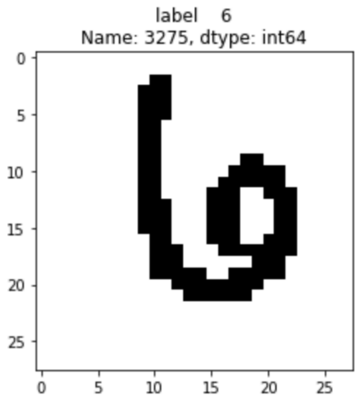

Figure 15-binary color mapping

```
In [10]: clf = svm.SVC()
         clf.fit(train_images, train_labels.values.ravel())
         clf.score(test_images, test_labels)
         executed in 8.68s, finished 02:13:40 2018-03-09

Out[10]: 0.8870000000000001
```

Figure 16-Retraining

4.2 PCA + SVM

We used PCA to reduce the dimensionality of our dataset and then performed the SVM algorithm to classify the digits. After applying PCA our accuracy significantly increased from 88.7% to 97.8%

```
PCA analysis begins with components:  0.721428571429
Starting SVM
accuracy:  0.979404761905 time elapsed:  21
PCA analysis begins with components:  0.732142857143
Starting SVM
accuracy:  0.979404761905 time elapsed:  22
PCA analysis begins with components:  0.742857142857
Starting SVM
accuracy:  0.980119047619 time elapsed:  22
PCA analysis begins with components:  0.753571428571
Starting SVM
accuracy:  0.979761904762 time elapsed:  23
PCA analysis begins with components:  0.764285714286
Starting SVM
accuracy:  0.979880952381 time elapsed:  24
PCA analysis begins with components:  0.775
Starting SVM
accuracy:  0.979166666667 time elapsed:  24
PCA analysis begins with components:  0.785714285714
Starting SVM
accuracy:  0.97880952381 time elapsed:  25
PCA analysis begins with components:  0.796428571429
Starting SVM
accuracy:  0.979047619048 time elapsed:  25
PCA analysis begins with components:  0.807142857143
Starting SVM
accuracy:  0.979166666667 time elapsed:  27
PCA analysis begins with components:  0.817857142857
Starting SVM
accuracy:  0.978928571429 time elapsed:  26
PCA analysis begins with components:  0.828571428571
Starting SVM
accuracy:  0.97880952381 time elapsed:  26
PCA analysis begins with components:  0.839285714286
Starting SVM
accuracy:  0.97880952381 time elapsed:  28
PCA analysis begins with components:  0.85
Starting SVM
accuracy:  0.978095238095 time elapsed:  29
```

Figure 17-pca+svm

4.3 Using CNN

- We changed the size of images to 28x28 matrix.
- Input shape = (28, 28, 1) where 1 is the color channel
- We used 32 filters of size 3x3
- We used ReLU at the input layer and Softmax at the output layer.

4.3.1 Model

In [11]:
```
model = Sequential()
model.add(Conv2D(32, (3,3),
                 activation = "relu",
                 input_shape = (28,28,1),
                 strides = (2,2)
                ))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Flatten(name  = "flatten"))
model.add(Dense(10, activation="softmax"))
model.summary()
```
executed in 52ms, finished 14:50:38 2018-05-03

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 13, 13, 32)        320
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 32)          0
_________________________________________________________________
flatten (Flatten)            (None, 1152)              0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                11530
=================================================================
Total params: 11,850
Trainable params: 11,850
Non-trainable params: 0
_________________________________________________________________
```

Figure 18-CNN_Model

```
batch_size = 64
nb_epochs = 25

model_fitting = model.fit(X_train_image, y_train,
                          batch_size= batch_size,
                          epochs= nb_epochs,
                          verbose=2,
                          validation_data=(X_test_image, y_test),
                          initial_epoch= 0
                         )
```
Figure 19-predicting the result

4.3.2 Output

```
Epoch 7/25
 - 3s - loss: 0.0170 - acc: 0.9959 - val_loss: 0.0859 - val_acc: 0.9767
Epoch 8/25
 - 3s - loss: 0.0165 - acc: 0.9960 - val_loss: 0.0853 - val_acc: 0.9779
Epoch 9/25
 - 3s - loss: 0.0161 - acc: 0.9961 - val_loss: 0.0896 - val_acc: 0.9771
Epoch 10/25
 - 3s - loss: 0.0161 - acc: 0.9960 - val_loss: 0.0842 - val_acc: 0.9771
Epoch 11/25
 - 4s - loss: 0.0157 - acc: 0.9963 - val_loss: 0.0916 - val_acc: 0.9760
Epoch 12/25
 - 4s - loss: 0.0156 - acc: 0.9964 - val_loss: 0.0859 - val_acc: 0.9783
Epoch 13/25
 - 4s - loss: 0.0151 - acc: 0.9967 - val_loss: 0.0924 - val_acc: 0.9757
Epoch 14/25
 - 4s - loss: 0.0149 - acc: 0.9963 - val_loss: 0.0989 - val_acc: 0.9752
Epoch 15/25
 - 4s - loss: 0.0146 - acc: 0.9965 - val_loss: 0.0898 - val_acc: 0.9755
Epoch 16/25
 - 3s - loss: 0.0149 - acc: 0.9965 - val_loss: 0.0860 - val_acc: 0.9771
Epoch 17/25
 - 3s - loss: 0.0146 - acc: 0.9965 - val_loss: 0.0854 - val_acc: 0.9781
Epoch 18/25
 - 4s - loss: 0.0142 - acc: 0.9966 - val_loss: 0.0895 - val_acc: 0.9769
Epoch 19/25
 - 3s - loss: 0.0142 - acc: 0.9966 - val_loss: 0.0888 - val_acc: 0.9769
Epoch 20/25
 - 3s - loss: 0.0143 - acc: 0.9965 - val_loss: 0.0924 - val_acc: 0.9757
Epoch 21/25
 - 4s - loss: 0.0139 - acc: 0.9970 - val_loss: 0.0879 - val_acc: 0.9771
Epoch 22/25
 - 4s - loss: 0.0136 - acc: 0.9969 - val_loss: 0.0911 - val_acc: 0.9752
Epoch 23/25
 - 3s - loss: 0.0131 - acc: 0.9967 - val_loss: 0.1045 - val_acc: 0.9740
Epoch 24/25
 - 4s - loss: 0.0135 - acc: 0.9967 - val_loss: 0.0914 - val_acc: 0.9764
Epoch 25/25
 - 3s - loss: 0.0133 - acc: 0.9970 - val_loss: 0.0975 - val_acc: 0.9757
```

Figure 20-Output

- We got an accuracy of 99.7% using CNN on training on 37800 samples.
- We got a validation score of 97.57% on validating on 4200 samples.

5. CONCLUSION

Our models improved significantly from 10% to 88.7% of accuracy using SVM. Further using better techniques to remove dimensions we achieved a score of 97.8%. Overall CNN model was the best one and predicted results with an accuracy of 99.7%. To improve the accuracy of the CNN model we can attach more Convolutional layers.

6. REFERENCES

- MNIST dataset, www.kaggle.com
- Support Vector Machine, www.analyticsvidhya.com
- Machine Learning A-Z, www.udemy.com
- Deep Neural Network, www.coursera.com
- Xiao-Xiao Niu and Ching Y. Suen, A novel hybrid CNN–SVM classifier for recognizing handwritten digits, ELSEVIER, The Journal of the Pattern Recognition Society, Vol. 45, 2012, 1318– 1325.

- https://en.wikipedia.org/wiki/Support_vector_machine
- https://en.wikipedia.org/wiki/Convolutional_neural_network
- http://neuralnetworksanddeeplearning.com/
- Machine Learning for Dummies, John Paul Mueller