**Project Report**

**Title:**

**Convolutional Neural Networks**

**Recognizing Hand-Written Digits with**

**99 Percent Accuracy.**

**Group Member:**

TALHA SAJJAD (9912)

USMAN KHAN (9911)

Instructor: Dr. Noman Islam

**Acknowledgement:**

I would like to express my special thanks of gratitude to my teacher (**DR NOMAN ISLAM**) who gave me*the golden*opportunity to do this wonderful project on the topic (Convolutional Neural Networks Recognizing Hand-Written Digits with 99 Percent Accuracy) which also helped me in doing a lot of Research and I came to know about so many new things I am thankful to sirSecondly, I would also like to thank my friends who helped me a lot in finalizing this project within*the limited*time frame.

**TABLE OF CONTANT:**

* Abstract
* Introduction
* Problem
* Literature review
* Proposed work
* Project architecture
* Details of work
* Experiments
* Result
* Discussion
* Conclusion
* Future work
* Reference
* Glossary of term

**Abstract:**

Convolutional Networks work by moving small filters across the input image. A popular demonstration of the capability of deep learning techniques is object recognition in image data. This means the filters are re-used for recognizing patterns throughout the entire input image. This makes the Convolutional Networks much more powerful than Fully-Connected networks with the same number of variables. This in turn makes the Convolutional Networks faster to train.

**Introduction:**

The [MNIST](http://yann.lecun.com/exdb/mnist/) problem is a dataset developed by Yann LeCun, Corinna Cortes and Christopher Burges for evaluating machine learning models on the handwritten digit classification problem. . We classify the images and hand written recognition with 99 percent accuracy. The dataset was constructed from a number of scanned document dataset available from the [National Institute of Standards and Technology](http://www.nist.gov/) (NIST). This is where the name for the dataset comes from, as the Modified NIST or MNIST dataset. Images of digits were taken from a variety of scanned documents, normalized in size and centered. This makes it an excellent dataset for evaluating models, allowing the developer to focus on the machine learning with very little data cleaning or preparation required. 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.so in short words We classify the images and hand written recognition with 99 percent accuracy.

**Problem:**

We solve the problem 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, which is nothing more than the inverted classification accuracy.

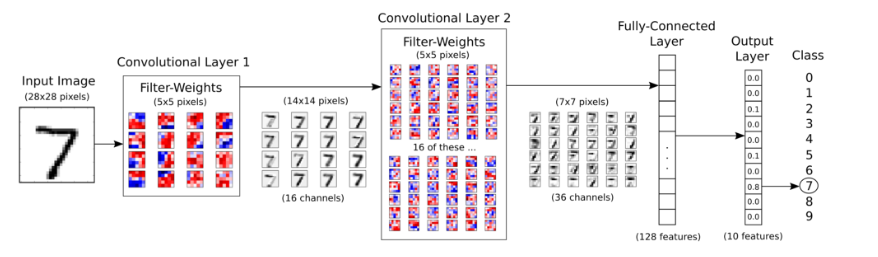
**Literature review:**

A lot of people solve this problem but our solution is 99 percent is accurate the people who solve this problem Jason brown lee and waheed Pandya, seraj and sentdex and many other person. Recognition is a Windows based Neural Network system to learn and accept mouse driven characters. Itcan be taught easily to recognize new characters [1].Sajjad S. Ahranjany and Farbod Razzazi proposed written. Recognition Results of different convolutional Neural Networks were fused with gradient descent.

**Proposed work:**

In hand written digit recognition, we give the input image and it pass in complete process of convolutional neural network and predict the output which digit you give me and in our project the accuracy of prediction is 99 percent correct and error chances is only 1 percent. Excellent results achieve a prediction error of less than 1%. State-of-the-art prediction error of approximately 0.2% can be achieved with large Convolutional Neural Networks. There is a listing of the state-of-the-art results.

**Project Architecture:**



The input image is processed in the first convolutional layer using the filter-weights. This results in 16 new images, one for each filter in the convolutional layer. The images are also down-sampled so the image resolution is decreased from 28x28 to 14x14.

These 16 smaller images are then processed in the second convolutional layer. We need filter-weights for each of these 16 channels, and we need filter-weights for each output channel of this layer. There are 36 output channels so there are a total of 16 x 36 = 576 filters in the second convolutional layer. The resulting images are down-sampled again to 7x7 pixels.

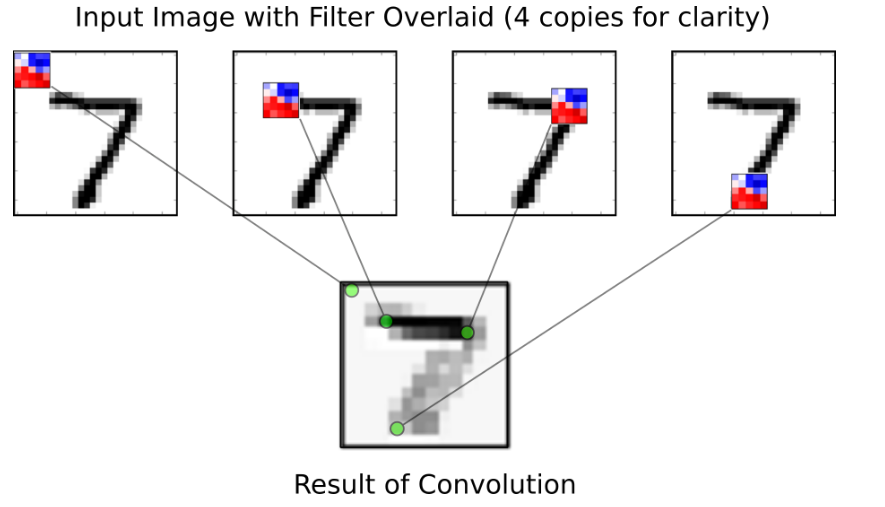
The output of the second convolutional layer is 36 images of 7x7 pixels each. These are then flattened to a single vector of length 7 x 7 x 36 = 1764, which is used as the input to a fully-connected layer with 128 neurons (or elements). This feeds into another fully-connected layer with 10 neurons, one for each of the classes, which is used to determine the class of the image, that is, which number is depicted in the image.

The convolutional filters are initially chosen at random, so the classification is done randomly. The error between the predicted and true class of the input image is measured as the so-called cross-entropy. The optimizer then automatically propagates this error back through the Convolutional Network using the chain-rule of differentiation and updates the filter-weights so as to improve the classification error. This is done iteratively thousands of times until the classification error is sufficiently low.

These particular filter-weights and efficient. This means the flowchart actually has one more data-dimension when implemented in TensorFlow intermediate images are the results of one optimization run and may look different if you re-run this Notebook.

Note that the computation in TensorFlow is actually done on a batch of images instead of a single image, which makes the computation more .

**Detail of work:**

****

The step-size for moving the filter across the input is called the stride. There is a stride for moving the filter horizontally (x-axis) and another stride for moving vertically (y-axis).

In the source-code below, the stride is set to 1 in both directions, which means the filter starts in the upper left corner of the input image and is being moved 1 pixel to the right in each step. When the filter reaches the end of the image to the right, then the filter is moved back to the left side and 1 pixel down the image. This continues until the filter has reached the lower right corner of the input image and the entire output image has been generated.

When the filter reaches the end of the right-side as well as the bottom of the input image, then it can be padded with zeroes (white pixels). This causes the output image to be of the exact same dimension as the input image.

Furthermore, the output of the convolution may be passed through a so-called Rectified Linear Unit (ReLU), which merely ensures that the output is positive because negative values are set to zero. The output may also be down-sampled by so-called max-pooling, which considers small windows of 2x2 pixels and only keeps the largest of those pixels. This halves the resolution of the input image e.g. from 28x28 to 14x14 pixels.

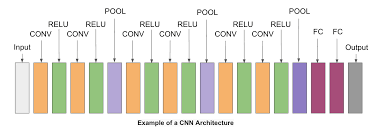
Note that the second convolutional layer is more complicated because it takes 16 input channels. We want a separate filter for each input channel, so we need 16 filters instead of just one. Furthermore, we want 36 output channels from the second convolutional layer, so in total we need 16 x 36 = 576 filters for the second convolutional layer. It can be a bit challenging to understand how this works.

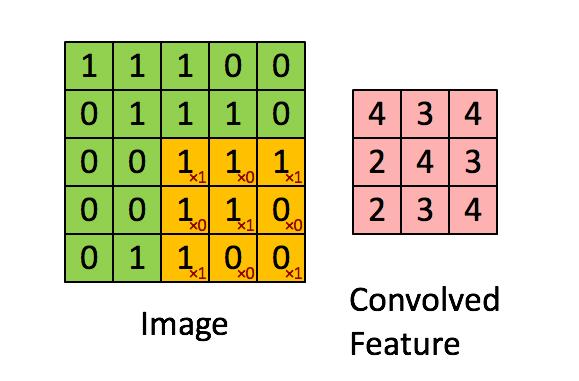
## **Load Data**

In our project The MNIST data-set is about 12 MB and will be downloaded automatically if it is not located in the given path.

We apply relu + pooling again and again to reduce the feature and pict the most important feature whose value is high and important and pass the fully connected layer and give tha output to predict the answer.

Here is the process on convolutional neural network,





Convolutional neural networks are more complex than standard multi-layer perceptrons, so we will start by using a simple structure to begin with that uses all of the elements for state of the art results. Below summarizes the network architecture.

1. The first hidden layer is a convolutional layer called a Convolution2D. The layer has 32 feature maps, which with the size of 5×5 and a rectifier activation function. This is the input layer, expecting images with the 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.

**Experiments:**

After research and learning a lot of things we make many experiments to complete our project so that project code is below

**Result:**

## **Performance before any optimization**

Accuracy on Test-Set: 10.8% (1080 / 10000)

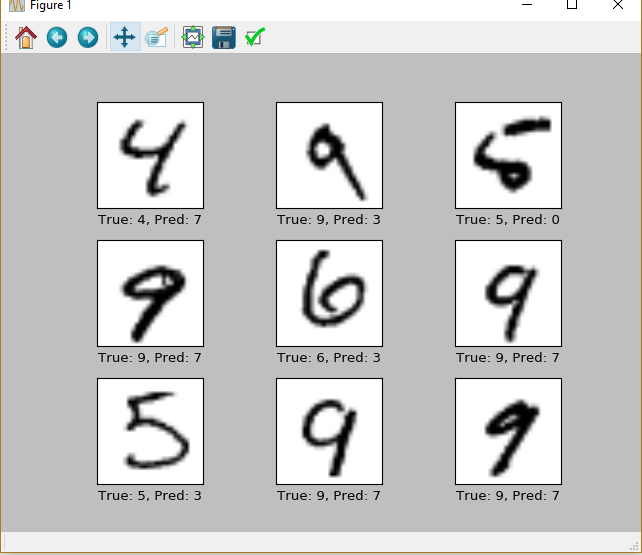
## **Performance after 1 optimization iteration**

Accuracy on Test-Set: 11.2% (1120 / 10000)

## **Performance after 100 optimization iterations**

Accuracy on Test-Set: 55.7% (5568 / 10000)

Example errors:



## Performance after 1000 optimization iterations

Optimization Iteration: 101, Training Accuracy: 71.9%

Optimization Iteration: 201, Training Accuracy: 76.6%

Optimization Iteration: 301, Training Accuracy: 71.9%

Optimization Iteration: 401, Training Accuracy: 85.9%

Optimization Iteration: 501, Training Accuracy: 89.1%

Optimization Iteration: 601, Training Accuracy: 95.3%

Optimization Iteration: 701, Training Accuracy: 90.6%

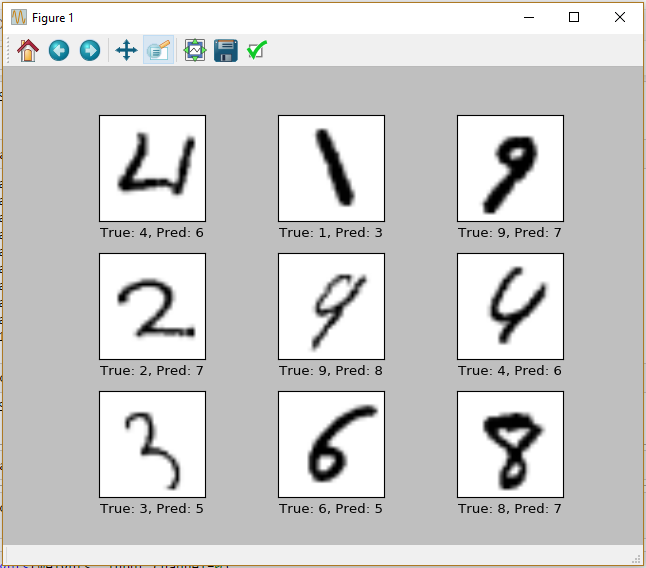
Optimization Iteration: 801, Training Accuracy: 92.2%

Optimization Iteration: 901, Training Accuracy: 95.3%

Time usage: 0:00:03

Accuracy on Test-Set: 93.0% (9302 / 10000)

Example errors:



**Performance after 10,000 optimization iterations** After 10,000 optimization iterations, the model has a classification accuracy on the test-set of about 99%.

Accuracy on Test-Set: 98.7% (9867 / 10000)

Example errors:

Confusion Matrix:

[[ 974 0 1 0 0 1 0 1 3 0]

[ 0 1122 4 0 1 0 1 1 6 0]

[ 2 2 1018 2 1 0 0 3 4 0]

[ 0 0 0 1006 0 1 0 0 3 0]

[ 0 0 2 0 974 0 1 0 1 4]

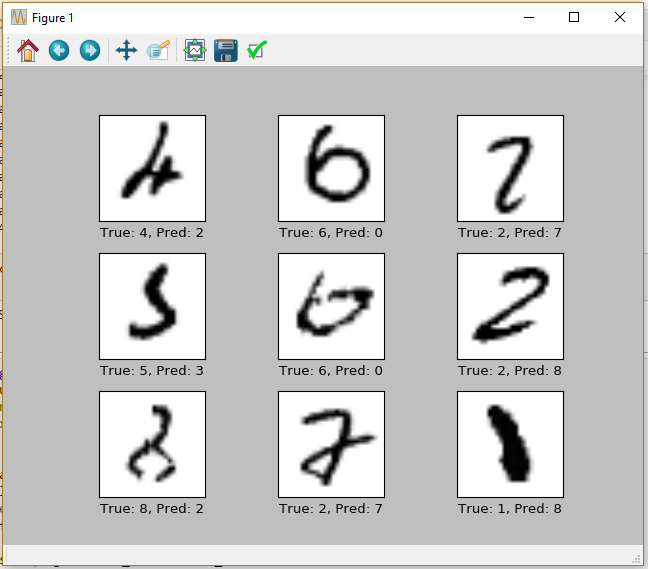
[ 2 0 0 5 0 881 1 0 2 1]

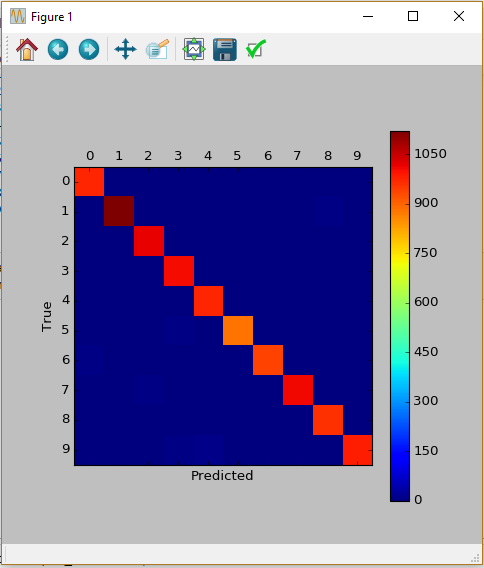
[ 5 2 1 1 4 3 939 0 3 0]

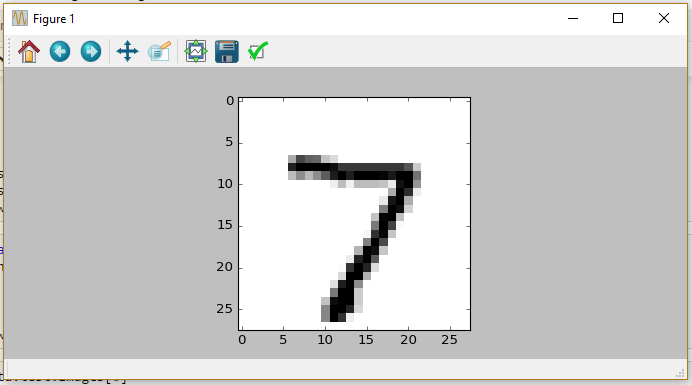
[ 1 2 7 3 0 0 0 1009 2 4]

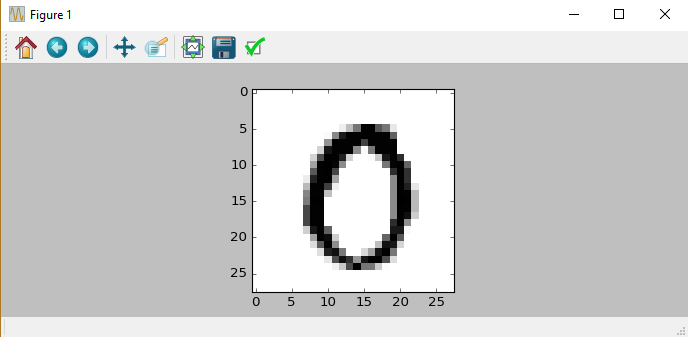
[ 2 0 1 3 0 2 0 2 962 2]

[ 2 4 0 6 9 1 0 3 2 982]]



****

****

****

## **Visualization of Weights and Layers**

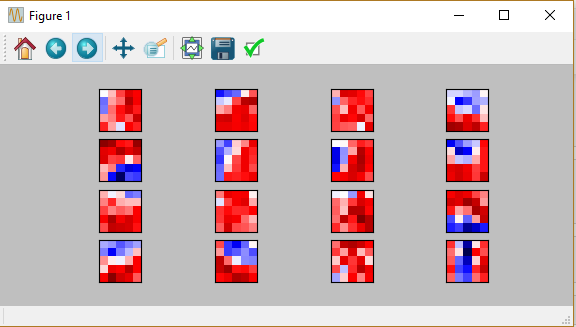
In trying to understand why the convolutional neural network can recognize handwritten digits, we will now visualize the weights of the convolutional filters and the resulting output images.

Plot an image from the test-set which will be used as an example below.

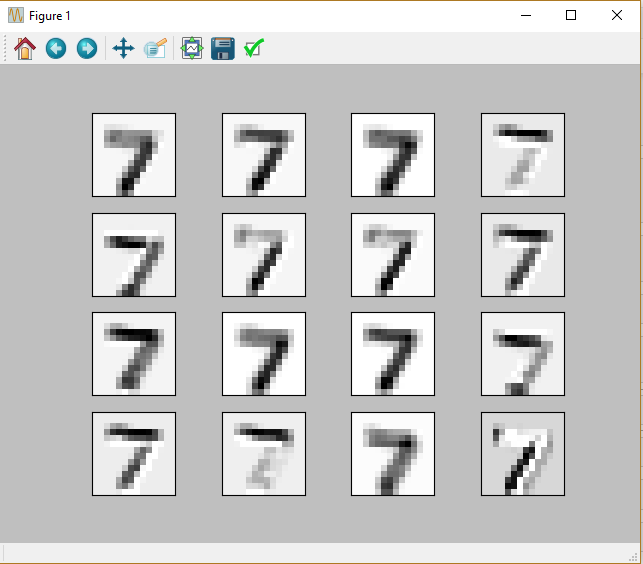
### **Convolution Layer 1**

Now plot the filter-weights for the first convolutional layer.

Note that positive weights are red and negative weights are blue.

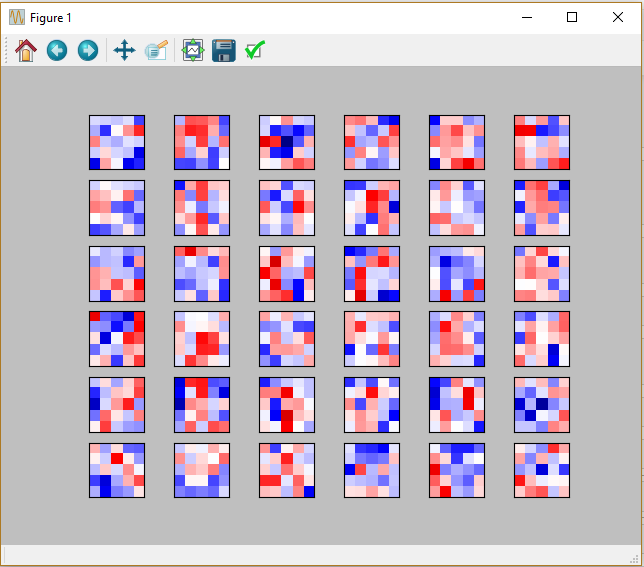
****

Applying each of these convolutional filters to the first input image gives the following output images, which are then used as input to the second convolutional layer. Note that these images are down-sampled to 14 x 14 pixels which is half the resolution of the original input image

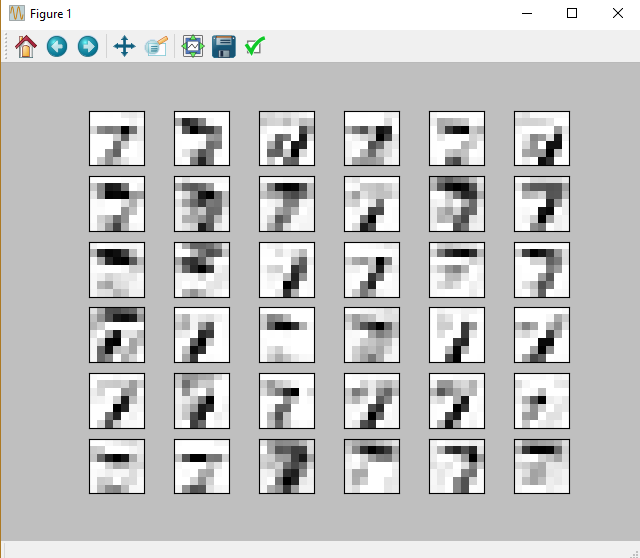


### **Convolution Layer 2**

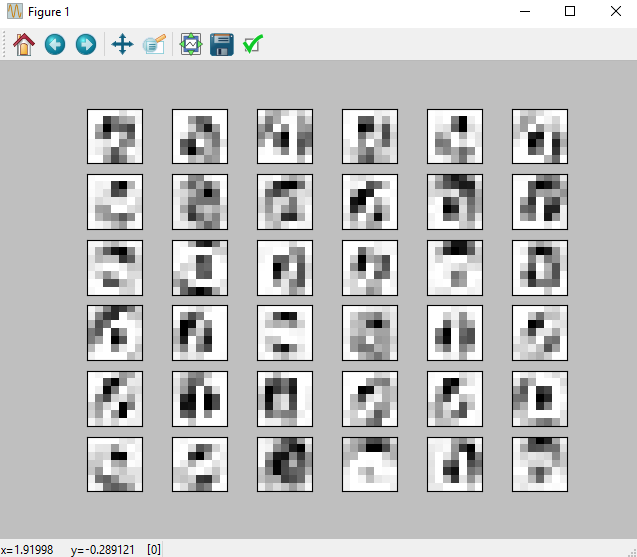
Now plot the filter-weights for the second convolutional layer.There are 16 output channels from the first conv-layer, which means there are 16 input channels to the second conv-layer. The second convlayer has a set of filter-weights for each of its input channels. We start by plotting the filter-weigths for the first channel.Note again that positive weights are red and negative weights are blue.



It can be difficult to understand and keep track of how these filters are applied because of the high dimensionality.Applying these convolutional filters to the images that were ouput from the first conv-layer gives the following images.Note that these are down-sampled yet again to 7 x 7 pixels which is half the resolution of the images from the first conv-layer.



And these are the results of applying the filter-weights to the second image.



From these images, it looks like the second convolutional layer might detect lines and patterns in the input images, which are less sensitive to local variations in the original input images.

These images are then flattened and input to the fully-connected layer, but that is not shown here.

**Future uses:**

In order to cater our needs in industrial application and many other places and now easy to implements such intelligence to identify objects into machine and computers.

**CONCLUSION:**

We have seen that a Convolutional Neural Network works much better at recognizing hand-written digits than the simple linear model . The Convolutional Network gets a classification accuracy of about 99%, or even more if you make some adjustments, compared to only 91% for the simple linear model.

However, the Convolutional Network is also much more complicated to implement, and it is not obvious from looking at the filter-weights why it works and why it sometimes fails.

So we would like an easier way to program Convolutional Neural Networks and we would also like a better way of visualizing their inner workings.