

Handwritten Digit Recognition Using Convolutional Neural Network

A project report submitted in partial fulfilment
of the requirements for the degree of

Bachelor of Engineering

In

Computer Engineering

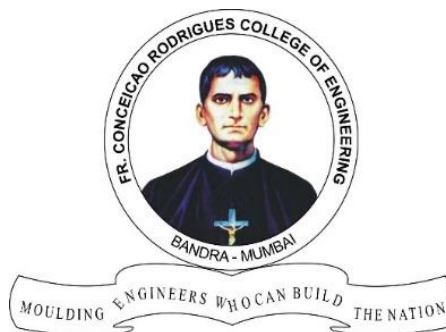
Submitted by

Nivea Dabre (Roll No. 7364)

Valencia Dias (Roll No. 7368)

Under the guidance of

Mrs. Kalpana Prasanna Deorukhkar
(Assistant Professor, Computer Engineering Department)



Department of Computer Engineering
Fr. Conceicao Rodrigues College of Engineering
Bandra, Mumbai-400 050
Year: 2017-2018

Internal Approval Sheet

CERTIFICATE

This is to certify that the project entitled "**Handwritten Digit Recognition using Convolutional Neural Network** " is a bonafide work of **Nivea Dabre(7364)**, **Valencia Dias(7368)** submitted to the University of Mumbai in partial fulfillment of the requirement for the award of the degree of "**Bachelor of Engineering**" in "**Computer Engineering**".

Prof .Kalpana Prasanna Deorukhkar

Supervisor/Guide

Dr. Sunil Surve

Head of Department

Dr. Srija Unnikrishnan

Principal

Project Report Approval

This project report entitled by *Handwritten Digit Recognition using Convolutional Neural Network* by *Nivea Dabre, Valencia Dias* is approved for the degree of **Bachelor of Engineering** in **Computer Engineering**.

Examiners

1. _____

2. _____

Date:

Place:

Declaration

We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Nivea Dabre (Roll No. 7364) _____

Valencia Dias (Roll No. 7368) _____

Date: April 15, 2018

Abstract

Convolutional Neural Networks(CNNs) have been confirmed as a powerful technique for classification of visual inputs like handwritten digits and faces recognition. Traditional convolutional layer's input feature maps are convolved with learnable kernel then combined for achieving better performance. The biggest drawback is that the combination of feature maps can lose features and do not apply well to large-scale neural networks. Handwritten digit recognition has recently been of very interest among the researchers because of the evolution of various Machine Learning, Deep Learning and Computer Vision algorithms. Nowadays, convolutional neural networks have become a hot topic in machine learning community, showing significant gains over start-of-the-art machine learning methods used in various applications, like speech, image processing and sentence classification. Each application

requires higher accuracy performing closely to human by neural networks. More importantly, we need higher accuracy rate, even to one hundred percent, and faster training. To achieve these goals, some methods and tricks are proposed for CNNs. Deep big simple neural net without convolutional layers gets 0.35% error rate. In order to avoid overfitting, elastic distortion was introduced to expand dataset and their simple network results in 0.4% error rate. Multi-column deep neural networks average many networks' prediction and get 0.23% error rate. Big networks with billions of parameters may easily overfit even with the largest datasets. compared the results of some of the most widely used Machine Learning Algorithms like SVM, KNN & RFC bandwidth. Deep Learning algorithm like multilayer CNN using Keras with Tensorflow. Using these, I was able to get the accuracy of 98.70% using CNN (Keras+Tensorflow) as compared to 97.91% using SVM, 96.67% using KNN, 96.89% using RFC.

Acknowledgments

We have great pleasure in presenting the report on "**Handwritten Digit Recognition using Convolutional Neural Network**". We take this opportunity to express our sincere thanks towards the guide **Prof . Kalpana Prasanna Deorukhkar**, C.R.C.E, Bandra (W), Mumbai, for providing the technical guidelines, and the suggestions regarding the line of this work. We enjoyed discussing the work progress with her during our visits to department.

We thank Dr. Sunil Surve, Head of Computer Dept., Principal and the management of C.R.C.E., Mumbai for encouragement and providing necessary infrastructure for pursuing the project.

We also thank all non-teaching staff for their valuable support, to complete our project.

Nivea Dabre (Roll No. 7364)

Valencia Dias (Roll No. 7368)

Date: April 15, 2018

Table of Contents

Abstract

List of Figures

List of Tables

1. Introduction	1
1.1 Aim of the project	1
1.2 Scope of the project	2
2. Literature Survey	3
2.1 Handwritten digit Recognition using deep learning	3
2.2 Handwritten digit Recognition using no combination of feature maps	3
2.3 Accurate Handwritten Digit Recognition using convolutional neural network	4
3. Proposed System	6
3.1 Analysis of the proposed system	6
3.1.1 MNIST Dataset	6
3.1.2 MNIST Dataset format Analysis	7
3.1.3 Layers of the convolutional neural network	8
3.2 Design of the proposed system	8
3.2.1 Phases of the convolutional neural network	9
3.3 Module Description	10
3.3.1 Flowchart	10
3.3.2 Use-case Diagram	11
3.3.3 Activity Diagram	13
3.3.4 Sequence Diagram	13
3.3.5 System Diagram	14
4. System Design and Requirements	15
4.1 Tensorflow	15
4.2 NumPy	15

4.3 SciPy	15
4.4 Scikit-learn	16
4.5 Pillow	16
4.6 H5py	16
4.7 Keras	16
4.8 Python 3.6	17
5. Hardware and Software Requirements	18
6. Implementation Details	19
6.1 Raw optimization	19
6.1.1 Loading the MNIST Dataset in Keras	19
6.1.2 Baseline model with Multilayer Perceptron	20
6.1.3 Simple Convolutional Neural Network for MNIST	21
6.1.4 Large Convolutional Neural Network for MNIST	22
6.2 Final optimization	23
6.2.1 Filter size and number of filters for convolutional layers	23
6.2.2 Defining weights, biases and layers of convolutional neural network	23
6.2.3 Defining cost and optimizer	23
6.2.4 Plotting images of the test Set which are incorrectly classified	24
6.2.5 Plotting Confusion Matrix	25
6.2.6 Printing test accuracy depending on correctly classified images	26
7. Conclusion and Future Enhancement	29
7.1 Result Analysis	29
7.1.1 Performance Analysis	29
7.2 Future Enhancements	39
7.3 Conclusion	39
8. References	41

List of Figures

Fig 1.1.1 Convolutional Neural Network Basic Layout

Fig 1.2.1 Arrangements of Neurons in CNN

Fig 3.1.2.1 Conversion of the image

Fig 3.1.3.1 Feature Extraction For CNN

Fig 3.2.1.1 Phases of Convolutional Neural Network

Fig 3.2.1.1 Phases of Convolutional Neural Network

Fig 3.3.1.1 Flowchart

Fig 3.3.2.1 Use Case Diagram

Fig 3.3.2.2 User Module

Fig 3.3.2.3 Pre-processing Module

Fig 3.3.3.1 Activity Diagram

Fig 3.3.4.1 Sequence Diagram

Fig 3.3.5.1 System Architecture

Fig 6.1.1.1 Loading the MNIST dataset in Keras

Fig 6.1.2.1 Baseline Model with Multi-Layer Perceptrons

Fig 6.1.3.1 Simple Convolutional Neural Network for MNIST

Fig 6.1.4.1 Larger Convolutional Neural Network for MNIST

Fig .7.1.1.1 Extracting the data from the MNIST dataset

Fig 7.1.1.1 Performance of the algorithm before any optimization on the dataset

Fig 7.1.1.3 Performance after 1000 optimization iterations

Fig 7.1.1.4 Performance after 10000 optimization iterations

Fig 7.1.1.5 Plotting the Confusion Matrix

Fig 7.1.1.6 Printing the precision and recall values

Fig7.1.1.7 Inputting the 1st and 13th image from the dataset

Fig.7.1.1.8 Printing the output of 1st Convolution layer

Fig.7.1.1.9 All Conversion of the image input to the 2nd Convolutional layer

Fig .7.1.1.10 Convolutional Layer 2

Fig 7.1.1.11 Output of 2nd Convolutional Layer

List of Tables

Fig. 2.1. Revolution in Neural Network

Fig 2.2 Percent Accuracy of Each Classification Technique

Fig 2.3. Classifier Error Rate Comparison

Fig 5.1 Hardware Requirement

Fig 5.2 Software Requirement

Chapter 1

Introduction

1.1 Aim of the project

Handwritten digit recognition is the ability of a computer system to recognize the handwritten inputs like digits, characters etc. from a wide variety of sources like emails, papers, images, letters etc. This has been a topic of research for decades. Some of the research areas include signature verification, bank check processing, postal address interpretation from envelopes etc. A Convolutional Neural Network (CNN) is a type of feed- forward Artificial Neural Network in which the connectivity pattern between its neurons is inspired by the organization of the animal visual cortex. Convolutional Neural Networks consist of neurons that have learnable weights and biases. Each neuron receives some input, performs a dot product and optionally follows it with a non-linearity. The whole Convolutional Neural Network expresses a differentiable scorefunction that is further followed by a Softmax function. The data input into the Convolutional Neural Network is arranged in the form of its width, height and depth. The Artificial Neural Networks can almost mimic the human brain and are a key ingredient in image processing field. For example, Convolutional Neural Networks with Back Propagation for Image Processing, Deep Mind by Google for creating Art by learning from existing artist styles etc.

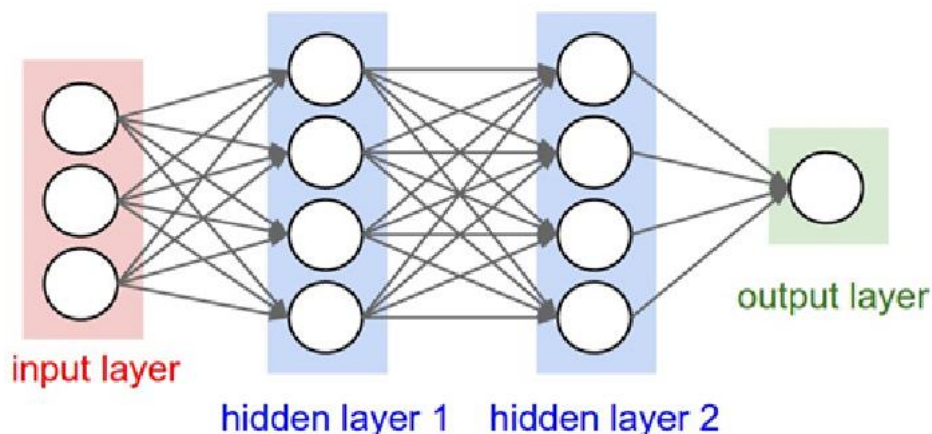


Fig 1.1.1 Convolutional Neural Network Basic Layout

1.2 Scope of the project

Nowdays, Convolutional Neural Networks (CNNs) have become an emergent topic in machine learning. It has lead its importance in various different applications such as music and speech recognition, image processing, and sentence classification to name a few. We need to achieve high accuracy rate close to human brain's neural network, to gain high accuracy as close to 100% as possible and minimization of the error rate. Various techniques have been proposed over the years to achieve these goals.

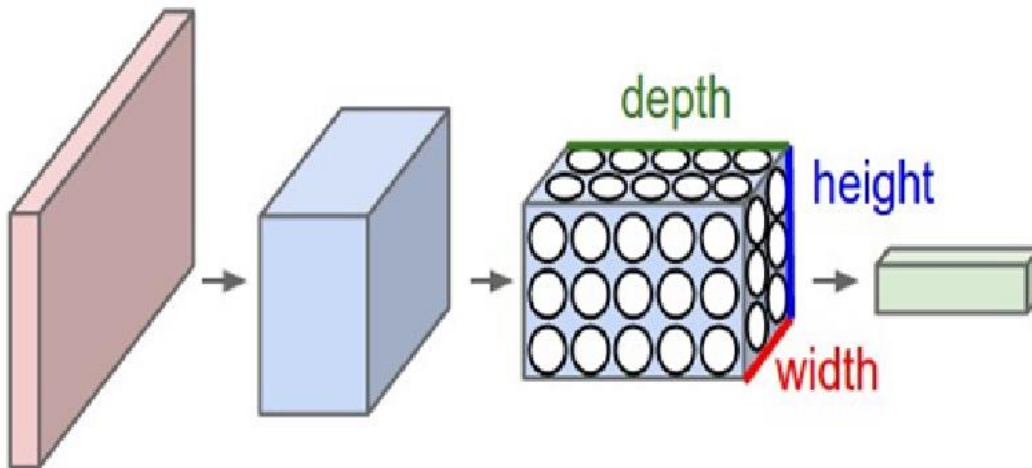


Fig 1.2.1 Arrangements of Neurons in CNN

Chapter 2

Literature Survey

2.1 Handwritten Digit Recognition Using Deep Learning

(Anuj Dutt, Aashi Dutt)

A lot of classification techniques using Machine Learning have been developed and used for this like K-Nearest Neighbours, SVM Classifier, Random Forest Classifier etc. but these methods although having the accuracy of 97% are not enough for the real world applications. One example of this is, if you send a letter with addressee name as “Anuj” and the system detects and recognizes it as “Tanuj” then it will not be delivered to “Anuj” but “Tanuj”. Although eventually it may come to the right address but if the mail is important, this delay can cost a lot. In short, the accuracy in these applications is very critical but these techniques do not provide the required accuracy due to very little knowledge about the topology of a task. Here comes the use of Deep Learning. In the past decade, deep learning has become the hot tool for Image Processing, object detection, handwritten digit and character recognition etc.

2.2 Handwritten Digit Recognition System Using No Combination of Feature Maps

(Drashti Dobariya, Dhvani Prajapati)

This paper introduced a replacement No combination of feature maps(NCFM) technique for abstraction of features. Every input feature map will generate an output feature map without combining, which lead to minimal data loss and achieve high precision rate. On analysis of results we can conclude that high performance can be achieved and that we can accomplish the acceptable precision rate on the dataset. The organization of the paper is as follows: Section II describes the outline of all the related previous research papers then follows introduction of NCFM technique to enhance the performance of the CNN. Section III then describes NCFM technique thoroughly and discusses the benefits. Section IV describes the architectures of the CNN with NCFM and CFM, that are used throughout section V. Section VI designs 3 experiments to verify the validity and analysis of the Ncfm technique.

2.3 Accurate Handwritten Digits Recognition using Convolutional Neural Networks

(Yan Yin, JunMin Wu, HuanXin Zheng University of Science and Technology of China)

In this paper, we introduce a novel Ncfm (No combination of feature maps) technique to abstract a variety of features. Each input feature map can get a output feature map without combining, which can avoid information loss and get higher accuracy. Evaluation results show that performance can be improved and we achieve the state-of-the-art accuracy rate with 99.81 % on the MNIST datasets. Specifically we make the following contributions:

1. We propose CNNs with Ncfm technique to improve accuracy and speed up convergence. We reach the state-of-the-art performance with 99.81% accuracy rate on the MNIST dataset.
2. We compare the Cfm(Combination of feature maps) technique with Ncfm technique and analyze the advantages. Firstly, Ncfm converges faster than Cfm. Secondly, Ncfm performs better than Cfm with fewer convolutional filters.

Method	Accuracy	Description
Hand printed symbol recognition.	97% overall.	Extract geometrical, topological and local measurements required to identify the character.
OCR for cursive handwriting.	88.8% for exicon size 40,000.	To implement segmentation and recognition algorithms for cursive handwriting.
Recognition handwritten numerals based upon fuzzy model.	95% for Hindi and 98.4% for English numerals overall.	The aim is to utilize the fuzzy technique to recognize handwritten numerals for Hindi and English numerals.
Combining decision multiple connectionist classifiers for Devanagari numeral recognition.	89.6% overall.	To use a reliable and an efficient technique for classifying numerals.
Hill climbing algorithm for handwritten character recognition.	93% for uppercase letters.	To implement hill climbing algorithm for selecting feature subset.
Optimization of feature selection for recognition of Arabic characters.	88% for numbers and 70% for letters.	To apply a method of selecting the features in an optimized way.
Handwritten numeral recognition for six popular Indian scripts.	99.56% for Devanagari, 98.99% for Bangla, 99.37% for Telugu, 98.40% for Oriya, 98.71% for Kannada and 98.51% for Tamil overall.	To find out the recognition rate for the six popular Indian scripts.

Fig. 2.1. Revolution in Neural Network

	RFC	KNN	SVM	CNN
Trained Classifier Accuracy	99.71%	97.88%	99.91%	99.98%
Accuracy on Test Images	96.89%	96.67%	97.91%	98.72%

Fig 2.2 Percent Accuracy of Each Classification Technique

Model	Test Error Rate
Random Forest Classifier	3.11%
K Nearest Neighbours	3.33%
Supervised Vector Machine	2.09%
Convolutional Neural Network	1.28%

Fig 2.3. Classifier Error Rate Comparison

Chapter 3

Proposed System

3.1 Analysis of the proposed system

3.1.1 MNIST Dataset

The MNIST dataset, a subset of a larger set NIST, is a database of 70,000 handwritten digits, divided into 60,000 training examples and 10,000 testing samples. The images in the MNIST dataset are present in form of an array consisting of 28x28 values representing an image along with their labels. This is also the same in case of the testing images.

The data is stored in four files:

1. train-images-idx3-ubyte: training set images
2. train-labels-idx1-ubyte: training set labels
3. t10k-images-idx3-ubyte: test set images
4. t10k-labels-idx1-ubyte: test set labels

The Training Set Label file has the data in the following representation:

[offset]	[type]	[value]	[description]
0000	32 bit integer	0x00000801(2049)	magic number (MSB first)
0004	32 bit integer	60000	number of items
0008	unsigned byte	??	label
0009	unsigned byte	??	label
.....			
xxxx	unsigned byte	??	label

The labels values are 0 to 9.

The Training Set Image file has the data in following representation:

[offset]	[type]	[value]	[description]
0000	32 bit integer	0x00000803(2051)	magic number
0004	32 bit integer	60000	number of images
0008	32 bit integer	28	number of rows
0012	32 bit integer	28	number of columns
0016	unsigned byte	??	pixel
0017	unsigned byte	??	pixel
.....			
xxxx	unsigned byte	??	pixel

3.1.2 MNIST Dataset Format Analysis

As you can see from above, the MNIST data is provided in a specific format. So, to be able to read the dataset it is first important to know that in what format the data is available to us. Both the Training and Testing images and labels have the first two columns consisting of the “Magic Number” and the number of items in the file.

The magic number has its first two bytes equal to zero. This magic number is read as MSB first and its format is as shown below:

2 Bytes	1 Byte	1 Byte
00	Data Type	Dimensions

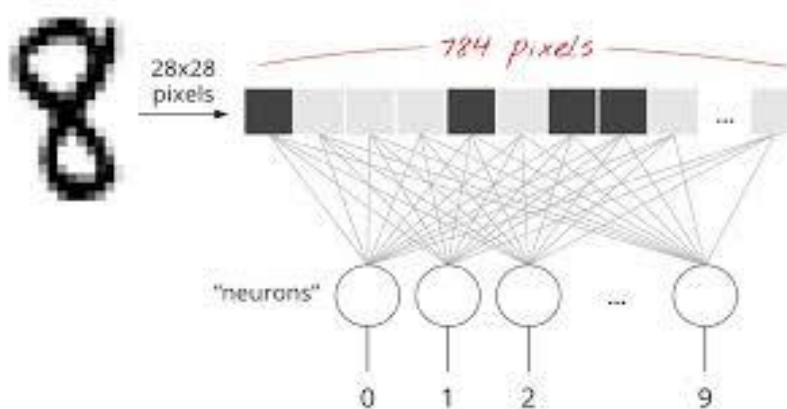


Fig 3.1.2.1 Conversion of the image

3.1.3 Layers of Convolutional Neural Network

A CNN consists of a lot of layers. These layers when used repeatedly, lead to a formation of a Deep Neural Network. Three main types of layers used to build a CNN are:

1. **Input:** This layer holds the raw pixel values of image.
2. **Convolutional Layer:** This layer gets the results of the neuron layer that is connected to the input regions. We define the number of filters to be used in this layer. Each

filter may be a 5x5 window that slider over the input data and gets the pixel with the maximum intensity as the output.

3. **Rectified Linear Unit [ReLU] Layer:** This layer applies an element wise activation function on the image data. We know that a CNN uses back propagation. So in order to retain the same values of the pixels and not being changed by the back propagation, we apply the ReLU function.
4. **Pooling Layer:** This layer perform a down-sampling operation along the spatial dimensions (width, height), resulting in volume.
5. **Fully Connected Layer:** This layers is used to compute the score classes i.e which class has the maximum score corresponding to the input digits.

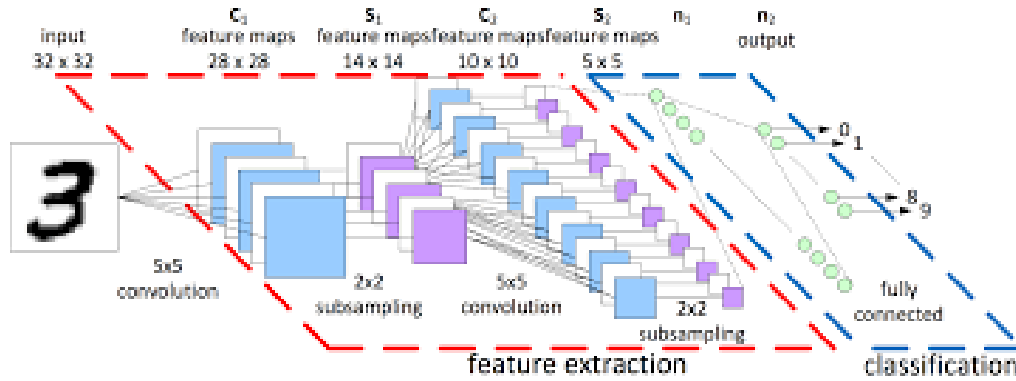


Fig 3.1.3.1 Feature Extraction For CNN

3.2 Design of the proposed System

3.2.1 Phases of the Convolutional Neural Network

The CNN for Handwritten Digit Recognition works in three main phases.

Phase1- Input MNIST Data1: The first phase is to input the MNIST data. The MNIST data is provided as 784-d array of pixels. So firstly we convert it to grayscale images using 28x28 matrix of pixels.

Phase2- Building Network Architecture: In the second phase, we define the models to be used to build a convolutional neural network. Here, we use the *Sequential* class from *Keras* to build the network. In this network, we have three layer sets of layers “*CONV* => *ReLU* => *POOL*”.

a)First Convolution Layer: In the first layer, we take 20 convolutional filters that go as a sliding window of size 5x5 over all the images of 28x28 matrix size and try to get the pixels with most intensity value.

b)ReLU Function: We know that convolution is a method that uses *Back Propagation*. So using the ReLU function as the activation function just after the convolutional layer reduces the likelihood of the vanishing gradient and avoids sparsity. This way we don't lose the important data and even get rid of redundant data like a lot of 0's in the pixels.

c) Pooling Layer: The pooling layer gets the data from the ReLU function and down-samples the steps in the 3D tensor. In short it pools all the pixels obtained from previous layers and again forms a new image matrix of a smaller size. These images are again input into the second set of layers i.e. "CONV => ReLU => POOL" and this process goes on till we get to a smallest set of pixels from which we can classify the digit.

Phase 3- Fully Connected Layer: The fully connected layer is used to connect each of the previous layers to the next layers. This layer consists of 500 neurons. Finally, we apply a Softmax Classifier that returns a list of probabilities for each of the 10 class labels. The class label with the largest probability is chosen as the final classification from the network and shown in the output. This output received is used to make the confusion matrix for the model. In this we can add more number of layers but adding more layers might affect the accuracy of the system. Since, it uses multiple layers, so it's called a Deep Learning system.

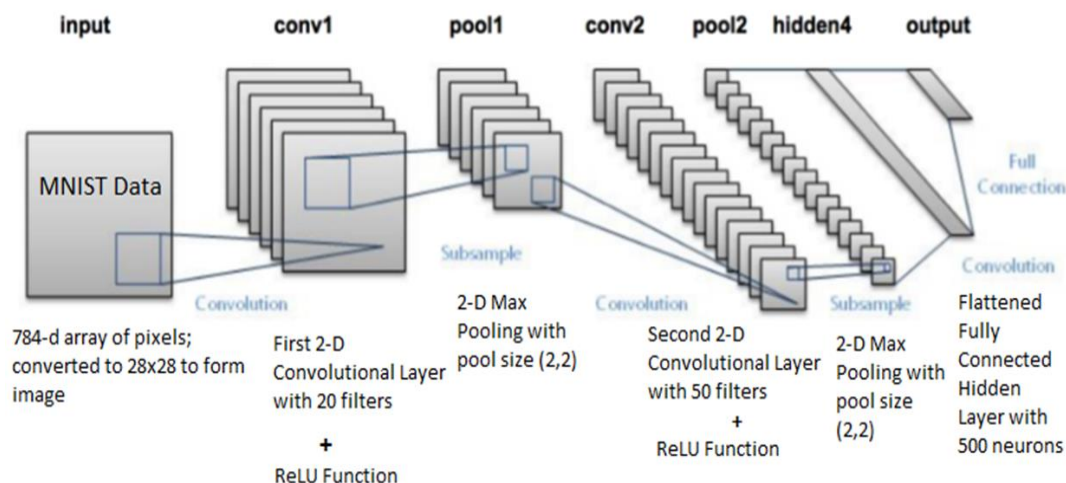


Fig 3.2.1.1 Phases of Convolutional Neural Network

3.3 Module Description

3.3.1 Flow Chart

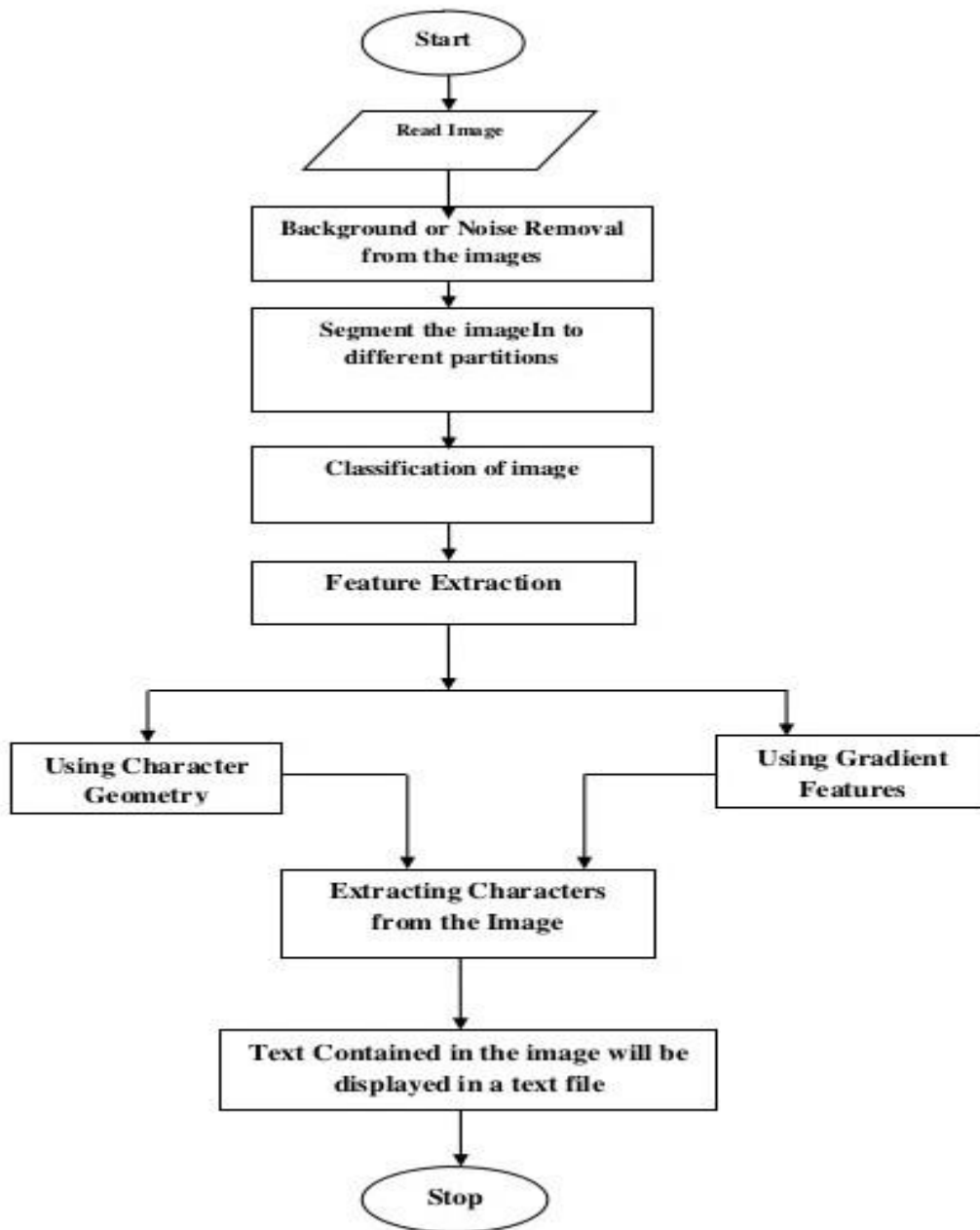


Fig 3.3.1.1 Flowchart

3.3.2 Use Case Diagram

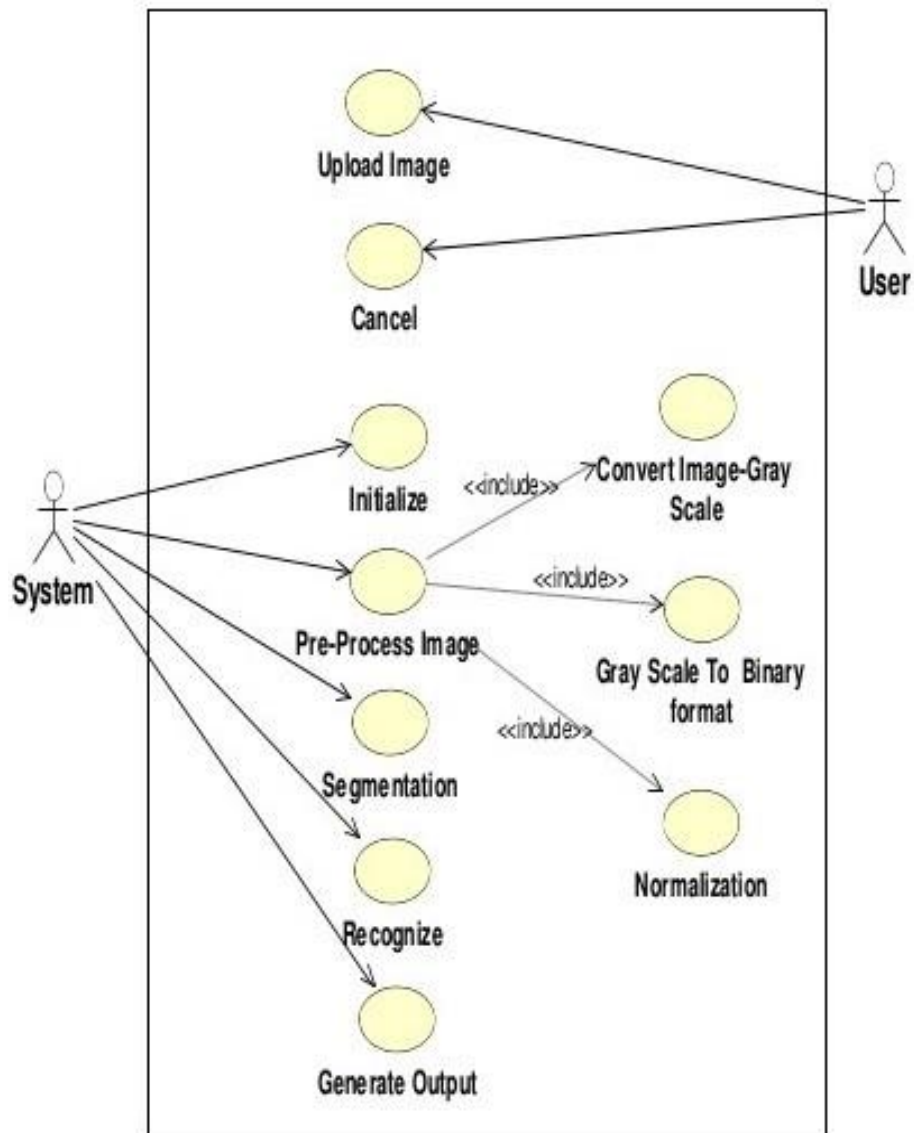


Fig 3.3.2.1 Use Case Diagram

User Module



User Case	Description
Actor	User
Precondition	Input image should be available.
Main Scenario	User uploads image.
Extension Scenario	If the image is not compatible. Not possible to upload file.
Post Condition	Image successfully uploaded.

Fig 3.3.2.2 User Module

Pre-processing Module



User Case	Description
Actor	System
Precondition	Uploaded input image
Main Scenario	Pre-processing is carried out by converting the image from RGB format to binary format.
Post Condition	Extract characters Before segmentation

Fig 3.3.2.3 Pre processing Module

3.3.3 Activity Diagram

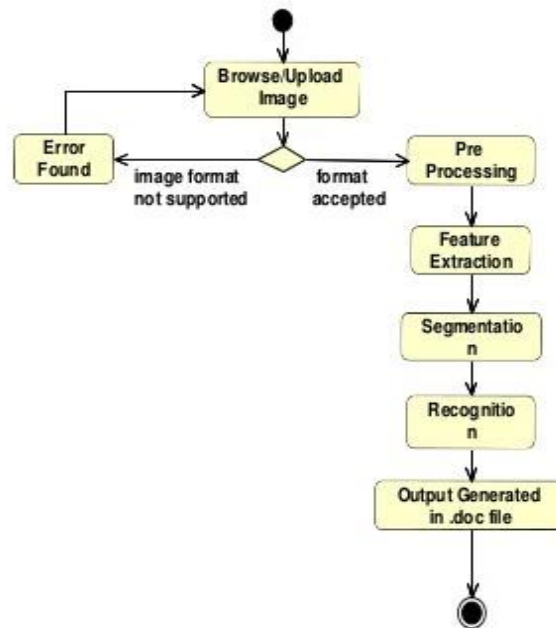


Fig 3.3.3.1 Activity Diagram

3.3.4 Sequence Diagram

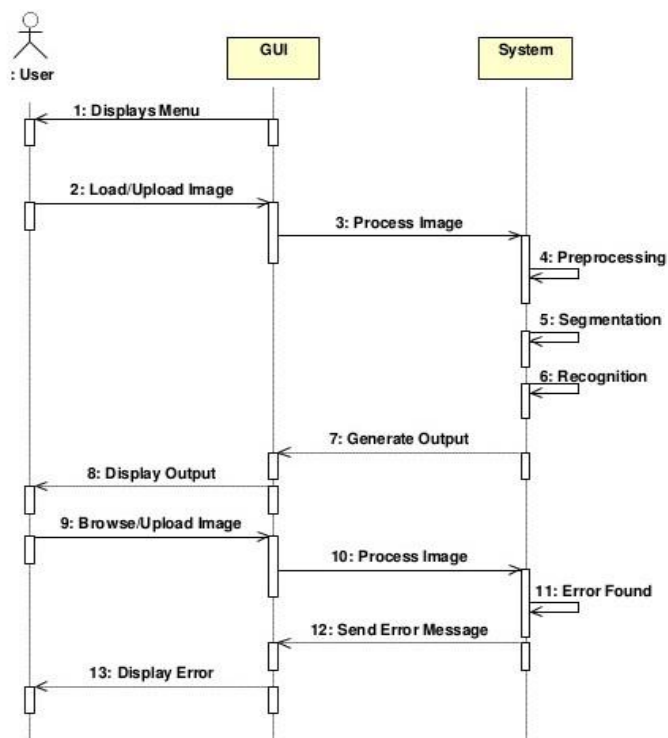


Fig 3.3.4.1 Sequence Diagram

3.3.5 System Architecture

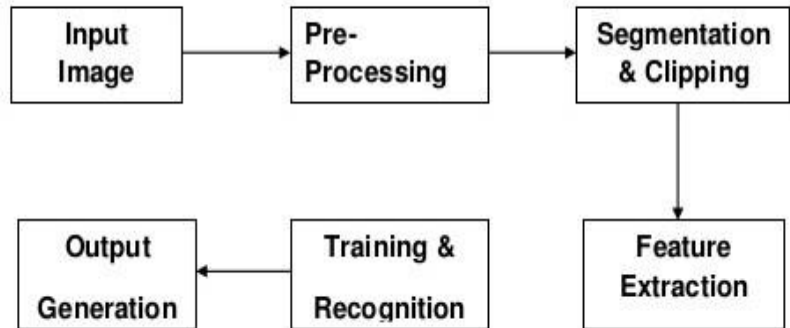


Fig 3.3.5.1 System Architecture

Chapter 4

System Design and Requirements

4.1 Tensorflow

TensorFlow is a Python library for fast numerical computing created and released by Google. It is a foundation library that can be used to create Deep Learning models directly or by using wrapper libraries that simplify the process built on top of TensorFlow. Installation of TensorFlow is straightforward if you already have a Python SciPy environment. TensorFlow works with Python 2.7 and Python 3.3+. Installation is probably simplest via PyPI and specific instructions of the pip command.

4.2 NumPy

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- A powerful N-dimensional array object
- Sophisticated (broadcasting) functions
- Tools for integrating C/C++ and Fortran code
- Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

4.3 SciPy

SciPy is an open-source Python library used for scientific computing and technical computing.

SciPy contains modules for optimization, linear algebra, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers and other tasks common in science and engineering.

4.4 Scikit-learn

Scikit-learn is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, *k*-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

4.5 Pillow

Python Imaging Library is a free library for the Python programming language that adds support for opening, manipulating, and saving many different image file formats. It is available for Windows, Mac OS X and Linux.

4.6 H5py

It lets you store huge amounts of numerical data, and easily manipulate that data from NumPy. For example, you can slice into multi-terabyte datasets stored on disk, as if they were real NumPy arrays. Thousands of datasets can be stored in a single file, categorized and tagged however you want.

4.7 Keras

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research. Keras is compatible with: Python 2.7-3.6.

Use Keras if you need a deep learning library that:

1. Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).
2. Supports both convolutional networks and recurrent networks, as well as combinations of the two.
3. Runs seamlessly on CPU and GPU.

4.8 Python 3.6

1. New library modules:

- typing: PEP 484 – Type Hints.
- zipfile: PEP 441 Improving Python ZIP Application Support.

2. New built-in features:

- bytes % args, bytearray % args: PEP 461 – Adding % formatting to bytes and bytearray.

- New `bytes.hex()`, `bytearray.hex()` and `memoryview.hex()` methods. (Contributed by Arnon Yaari in bpo-9951.)
- `memoryview` now supports tuple indexing (including multi-dimensional). (Contributed by Antoine Pitrou in bpo-23632.)
- Generators have a new `gi_yieldfrom` attribute, which returns the object being iterated by `yield from` expressions. (Contributed by Benno Leslie and Yury Selivanov in bpo-24450.)
- A new `RecursionError` exception is now raised when maximum recursion depth is reached. (Contributed by Georg Brandl in bpo-19235.)

3. Significant improvements in the standard library:

- `collections.OrderedDict` is now implemented in C, which makes it 4 to 100 times faster.
- The `ssl` module gained support for Memory BIO, which decouples SSL protocol handling from network IO.
- The new `os.scandir()` function provides a better and significantly faster way of directory traversal.
- `functools.lru_cache()` has been mostly reimplemented in C, yielding much better performance.
- The new `subprocess.run()` function provides a streamlined way to run subprocesses.
- The `traceback` module has been significantly enhanced for improved performance and developer convenience.

4. Security improvements:

- SSLv3 is now disabled throughout the standard library. It can still be enabled by instantiating a `ssl.SSLContext` manually. (See bpo-22638 for more details; this change was backported to CPython 3.4 and 2.7.)
- HTTP cookie parsing is now stricter, in order to protect against potential injection attacks. (Contributed by Antoine Pitrou in bpo-22796.)

Chapter 5

Hardware and Software Requirement

5.1 Hardware Requirement

Device	Computer , Laptop (min 3)
Screen	15 inch
Storage	2 GB RAM, 40 GB HDD
Processors	Dual core processor with clock speed of 2 GHZ

5.2 Software Requirement

Operating System	Microsoft Windows XP/7/8/10 (*.arff files only)
Language used	Python 3.6
IDE used	Anaconda-Spyder 3.2.6
Libraries	numpy scipy, scikit-learn, pillow, h5py, keras,tensorflow

Chapter 6

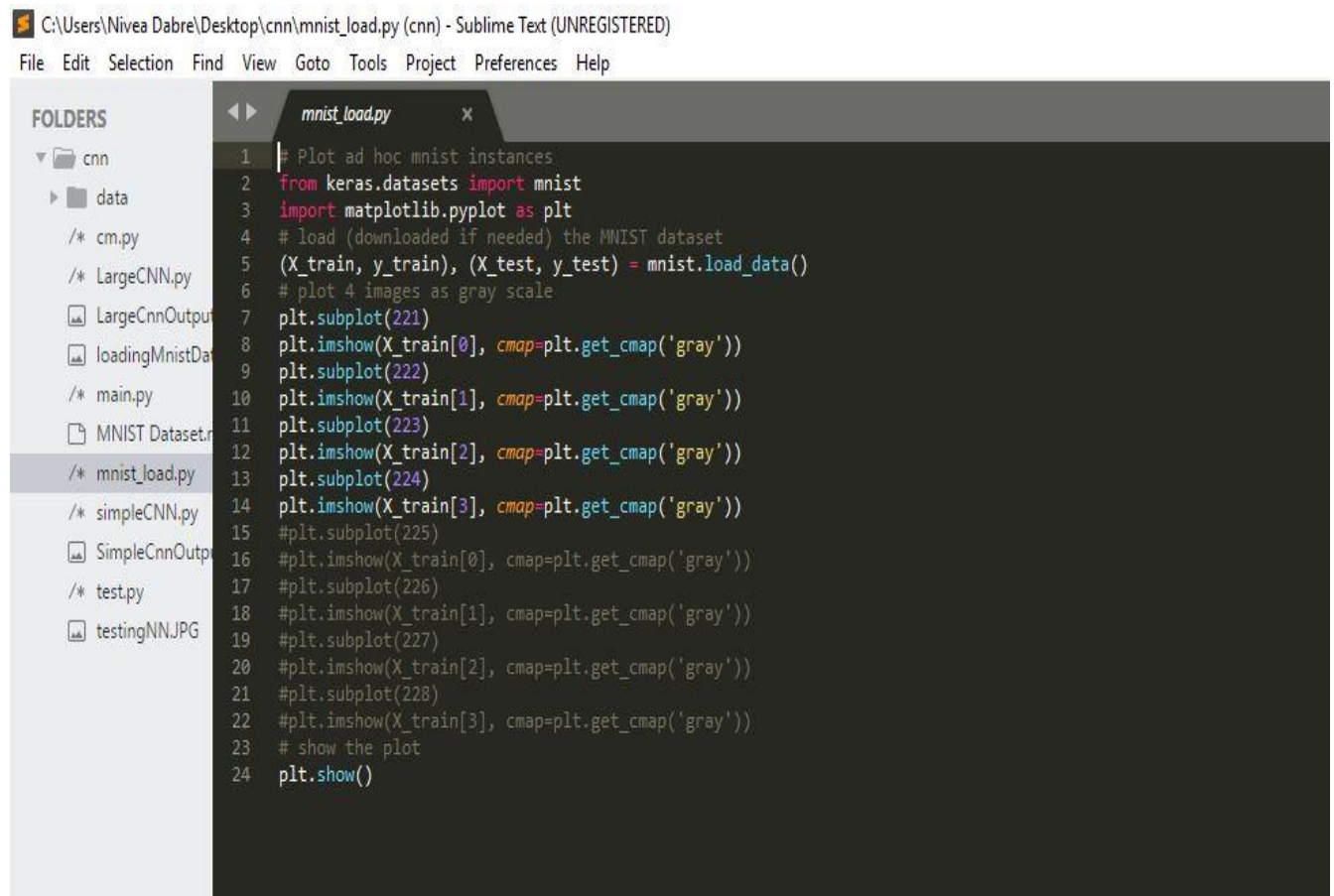
Implementation Details

6.1 Raw Optimization

Comparison of various models based on the Accuracy Rate

6.1.1 Loading the MNIST dataset in Keras

The Keras deep learning library provides a convenience method for loading the MNIST dataset. The dataset is downloaded automatically the first time this function is called and is stored in your home directory in `~/.keras/datasets/mnist.pkl.gz` as a 15MB file. To demonstrate how easy it is to load the MNIST dataset, we will first write a little script to download and visualize the first 4 images in the training dataset.



```
C:\Users\Nivea Dabre\Desktop\cnn\mnist_load.py (cnn) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

FOLDERS
└─ cnn
  └─ data
    /* cm.py
    /* LargeCNN.py
    LargeCnnOutput
    loadingMnistData
    /* main.py
    MNIST Dataset.r
    /* mnist_load.py
    /* simpleCNN.py
    SimpleCnnOutput
    /* test.py
    testingNN.JPG

1  # Plot ad hoc mnist instances
2  from keras.datasets import mnist
3  import matplotlib.pyplot as plt
4  # load (downloaded if needed) the MNIST dataset
5  (X_train, y_train), (X_test, y_test) = mnist.load_data()
6  # plot 4 images as gray scale
7  plt.subplot(221)
8  plt.imshow(X_train[0], cmap=plt.get_cmap('gray'))
9  plt.subplot(222)
10 plt.imshow(X_train[1], cmap=plt.get_cmap('gray'))
11 plt.subplot(223)
12 plt.imshow(X_train[2], cmap=plt.get_cmap('gray'))
13 plt.subplot(224)
14 plt.imshow(X_train[3], cmap=plt.get_cmap('gray'))
15 #plt.subplot(225)
16 #plt.imshow(X_train[0], cmap=plt.get_cmap('gray'))
17 #plt.subplot(226)
18 #plt.imshow(X_train[1], cmap=plt.get_cmap('gray'))
19 #plt.subplot(227)
20 #plt.imshow(X_train[2], cmap=plt.get_cmap('gray'))
21 #plt.subplot(228)
22 #plt.imshow(X_train[3], cmap=plt.get_cmap('gray'))
23 # show the plot
24 plt.show()
```

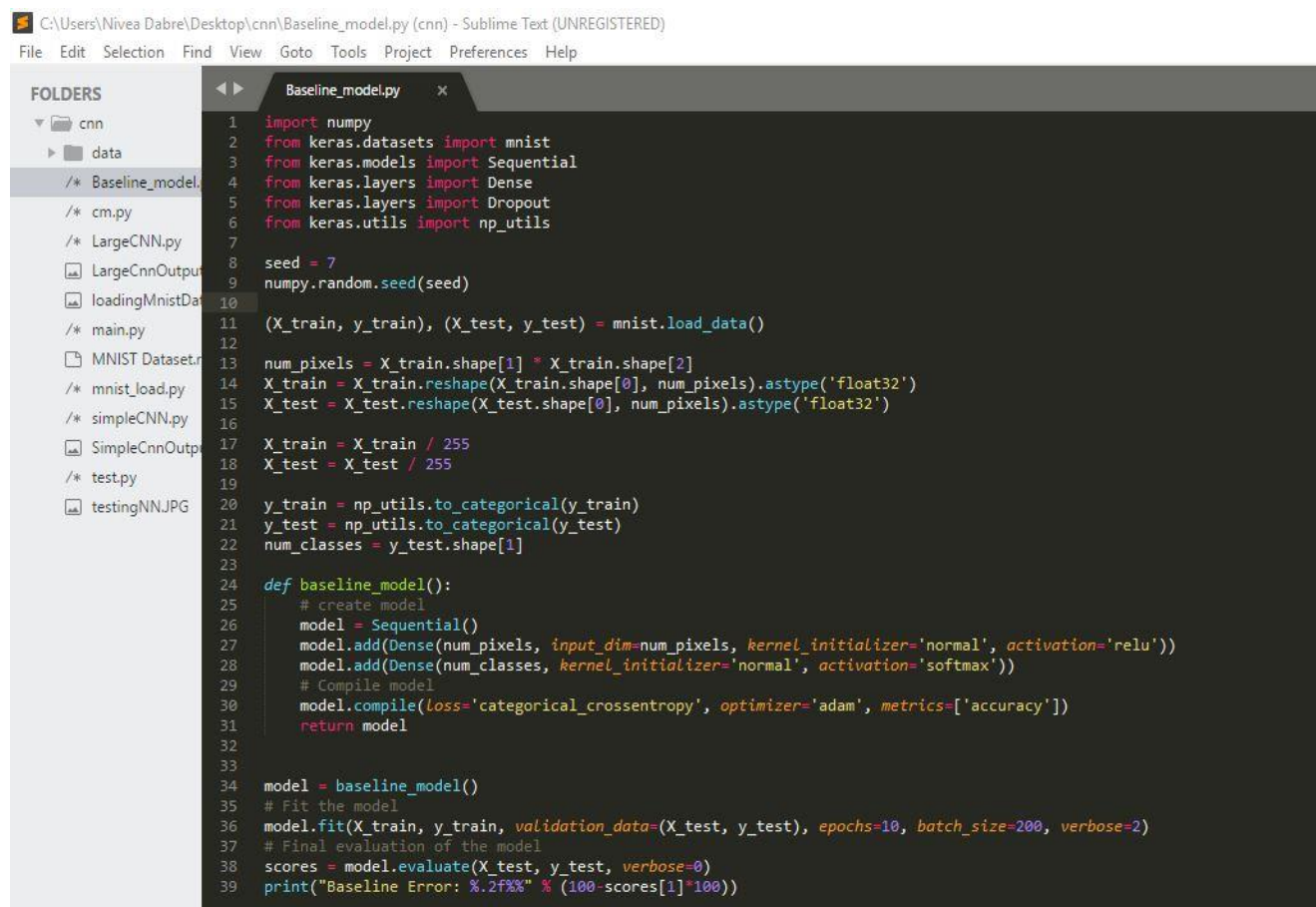
Fig 6.1.1.1 Loading the MNIST dataset in Keras

6.1.2 Baseline Model with Multi-Layer Perceptrons

In this section we will create a simple multi-layer perceptron model that achieves an error rate of 1.89%. We will use this as a baseline for comparing more complex convolutional neural network models.

A softmax activation function is used on the output layer to turn the outputs into probability-like values and allow one class of the 10 to be selected as the model's output prediction.

Logarithmic loss is used as the loss function (called `categorical_crossentropy` in Keras) and the efficient ADAM gradient descent algorithm is used to learn the weights.



```
C:\Users\Nivea Dabre\Desktop\cnn\Baseline_model.py (cnn) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

FOLDERS
└─ cnn
  └─ data
    └─ Baseline_model.py
    └─ cm.py
    └─ LargeCNN.py
    └─ LargeCnnOutput.py
    └─ loadingMnistData.py
    └─ main.py
    └─ MNIST Dataset.py
    └─ mnist_load.py
    └─ simpleCNN.py
    └─ SimpleCnnOutput.py
    └─ test.py
    └─ testingNN.JPG

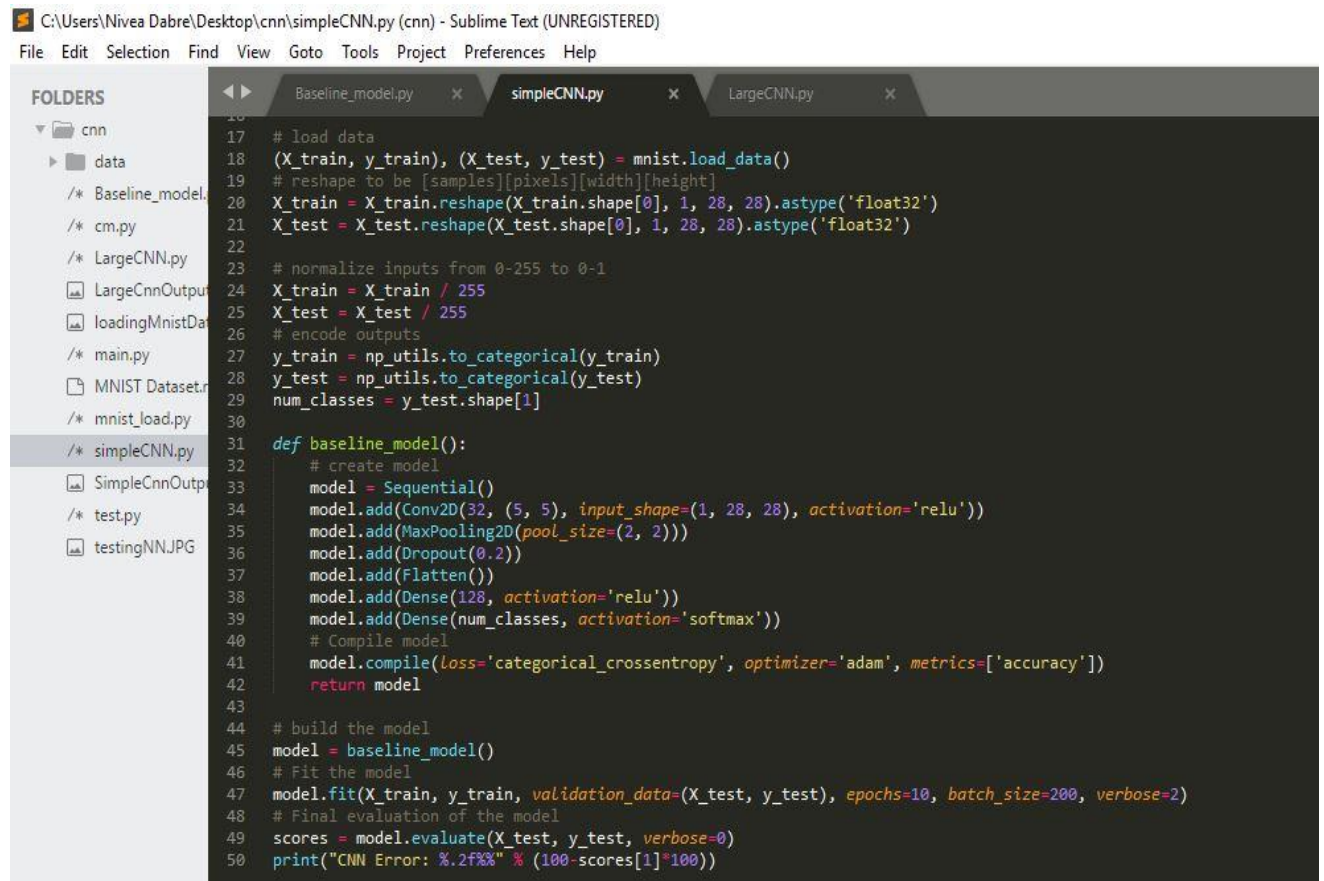
1 import numpy
2 from keras.datasets import mnist
3 from keras.models import Sequential
4 from keras.layers import Dense
5 from keras.layers import Dropout
6 from keras.utils import np_utils
7
8 seed = 7
9 numpy.random.seed(seed)
10
11 (X_train, y_train), (X_test, y_test) = mnist.load_data()
12
13 num_pixels = X_train.shape[1] * X_train.shape[2]
14 X_train = X_train.reshape(X_train.shape[0], num_pixels).astype('float32')
15 X_test = X_test.reshape(X_test.shape[0], num_pixels).astype('float32')
16
17 X_train = X_train / 255
18 X_test = X_test / 255
19
20 y_train = np_utils.to_categorical(y_train)
21 y_test = np_utils.to_categorical(y_test)
22 num_classes = y_test.shape[1]
23
24 def baseline_model():
25     # create model
26     model = Sequential()
27     model.add(Dense(num_pixels, input_dim=num_pixels, kernel_initializer='normal', activation='relu'))
28     model.add(Dense(num_classes, kernel_initializer='normal', activation='softmax'))
29     # Compile model
30     model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
31     return model
32
33
34 model = baseline_model()
35 # Fit the model
36 model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, batch_size=200, verbose=2)
37 # Final evaluation of the model
38 scores = model.evaluate(X_test, y_test, verbose=0)
39 print("Baseline Error: %.2f%%" % (100-scores[1]*100))
```

Fig 6.1.2.1 Baseline Model with Multi-Layer Perceptrons

This very simple network achieves a respectable **error rate of 1.89%**.

6.1.3 Simple Convolutional Neural Network for MNIST

In this we create a simple CNN for MNIST that demonstrates how to use all of the aspects of a modern CNN implementation, including Convolutional layers, Pooling layers and Dropout layers.



```
C:\Users\Nivea Dabre\Desktop\cnn\simpleCNN.py (cnn) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

FOLDERS
  ▼ cnn
    ► data
      /* Baseline_model.py
      /* cm.py
      /* LargeCNN.py
      LargeCnnOutput
      loadingMnistData
      /* main.py
      MNIST Dataset
      /* mnist_load.py
      /* simpleCNN.py
      SimpleCnnOutput
      /* test.py
      testingNN.JPG

17 # load data
18 (X_train, y_train), (X_test, y_test) = mnist.load_data()
19 # reshape to be [samples][pixels][width][height]
20 X_train = X_train.reshape(X_train.shape[0], 1, 28, 28).astype('float32')
21 X_test = X_test.reshape(X_test.shape[0], 1, 28, 28).astype('float32')
22
23 # normalize inputs from 0-255 to 0-1
24 X_train = X_train / 255
25 X_test = X_test / 255
26 # encode outputs
27 y_train = np_utils.to_categorical(y_train)
28 y_test = np_utils.to_categorical(y_test)
29 num_classes = y_test.shape[1]
30
31 def baseline_model():
32     # create model
33     model = Sequential()
34     model.add(Conv2D(32, (5, 5), input_shape=(1, 28, 28), activation='relu'))
35     model.add(MaxPooling2D(pool_size=(2, 2)))
36     model.add(Dropout(0.2))
37     model.add(Flatten())
38     model.add(Dense(128, activation='relu'))
39     model.add(Dense(num_classes, activation='softmax'))
40     # Compile model
41     model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
42     return model
43
44 # build the model
45 model = baseline_model()
46 # Fit the model
47 model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, batch_size=200, verbose=2)
48 # Final evaluation of the model
49 scores = model.evaluate(X_test, y_test, verbose=0)
50 print("CNN Error: %.2f%%" % (100-scores[1]*100))
```

Fig 6.1.3.1 Simple Convolutional Neural Network for MNIST

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.

The model is trained using logarithmic loss and the ADAM gradient descent algorithm.

This achieves a respectable **error rate of 1.04%**.

6.1.4 Larger Convolutional Neural Network for MNIST

This time we define a large CNN architecture with 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.

```

C:\Users\Nivea Dabre\Desktop\cnn\LargeCNN.py (cnn) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

FOLDERS
  cnn
    data
  /* Baseline_model.py
  /* cm.py
  /* LargeCNN.py
  LargeCnnOutput
  loadingMnistData
  /* main.py
  MNIST Dataset
  /* mnist_load.py
  /* simpleCNN.py
  SimpleCnnOutput
  /* test.py
  testingNN.JPG

12 K.set_image_dim_ordering('th')
13 # fix random seed for reproducibility
14 seed = 7
15 numpy.random.seed(seed)
16 # load data
17 (X_train, y_train), (X_test, y_test) = mnist.load_data()
18 # reshape to be [samples][pixels][width][height]
19 X_train = X_train.reshape(X_train.shape[0], 1, 28, 28).astype('float32')
20 X_test = X_test.reshape(X_test.shape[0], 1, 28, 28).astype('float32')
21 # normalize inputs from 0-255 to 0-1
22 X_train = X_train / 255
23 X_test = X_test / 255
24 # encode outputs
25 y_train = np_utils.to_categorical(y_train)
26 y_test = np_utils.to_categorical(y_test)
27 num_classes = y_test.shape[1]
28
29 # define the larger model
30 def larger_model():
31     # create model
32     model = Sequential()
33     model.add(Conv2D(30, (5, 5), input_shape=(1, 28, 28), activation='relu'))
34     model.add(MaxPooling2D(pool_size=(2, 2)))
35     model.add(Conv2D(15, (3, 3), activation='relu'))
36     model.add(MaxPooling2D(pool_size=(2, 2)))
37     model.add(Dropout(0.2))
38     model.add(Flatten())
39     model.add(Dense(128, activation='relu'))
40     model.add(Dense(50, activation='relu'))
41     model.add(Dense(num_classes, activation='softmax'))
42     # Compile model
43     model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
44     return model
45
46 # build the model
47 model = larger_model()
48 # Fit the model
49 model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, batch_size=200)
50 # Final evaluation of the model
51 scores = model.evaluate(X_test, y_test, verbose=0)
52 print("Large CNN Error: %.2f%%" % (100-scores[1]*100))

```

Fig 6.1.4.1 Larger Convolutional Neural Network for MNIST

This achieves a respectable **error rate of 0.75%**.

6.2 Final optimization

6.2.1 Filter size and number of filters for convolutional layers

```
#Convolutional Layer 1.
filter_size1 = 5      # Convolution filters are 5 x 5 pixels.
num_filters1 = 16     # There are 16 of these filters.

#Convolutional Layer 2.
filter_size2 = 5      # Convolution filters are 5 x 5 pixels.
num_filters2 = 36     # There are 36 of these filters.

# Fully-connected layer.
fc_size = 128         # Number of neurons in fully-connected layer.
```

6.2.2 Defining weights, biases and layers of convolutional Neural Network

```
def new_weights(shape):
    return tf.Variable(tf.truncated_normal(shape, stddev=0.05))

def new_biases(length):
    return tf.Variable(tf.constant(0.05, shape=[length]))

def new_conv_layer(input,          # The previous layer.
                   num_input_channels, # Num. channels in prev. layer.
                   filter_size,     # Width and height of each filter.
                   num_filters,     # Number of filters.
                   use_pooling=True): # Use 2x2 max-pooling.
    def flatten_layer(layer):
        # Flatten the layer.
        shape = layer.get_shape().as_list()
        new_shape = [-1, shape[1], shape[2], shape[3]]
        return tf.reshape(layer, new_shape)

    # Convolve the input with the filters.
    conv = tf.nn.conv2d(input, new_weights([num_input_channels, filter_size, filter_size, num_filters]),
                        [1, 1, 1, 1], use_pooling)
    # Add the biases.
    new_biases = new_biases(num_filters)
    return conv + new_biases

def new_fc_layer(input,          # The previous layer.
                 num_inputs,    # Num. inputs from prev. layer.
                 num_outputs,   # Num. outputs.
                 use_relu=True): # Use Rectified Linear Unit (ReLU)
    # Fully connected layer.
    shape = input.get_shape().as_list()
    new_shape = [-1, shape[1]]
    layer = tf.reshape(input, new_shape)
    layer = tf.nn.matmul(layer, new_weights([num_inputs, num_outputs]))
    if use_relu:
        layer = tf.nn.relu(layer)
    return layer
```

6.2.3 Defining cost and optimizer

```
y_pred = tf.nn.softmax(layer_fc2);
y_pred_cls = tf.argmax(y_pred, axis=1);
cross_entropy = tf.nn.softmax_cross_entropy_with_logits(logits=layer_fc2, labels=y_true);
cost = tf.reduce_mean(cross_entropy);
optimizer = tf.train.AdamOptimizer(learning_rate=1e-4).minimize(cost);
```

Softmax cross entropy

Computes softmax cross entropy between logits and labels. Measures the probability error in discrete classification tasks in which the classes are mutually exclusive (each entry is in exactly one class).

Apply softmax on the output from layer_fc2(the fully connected second layer)

The softmax function, or normalized exponential function is a generalization of the logistic function that "squashes" a K -dimensional vector of arbitrary real values to a K -dimensional vector of real values in the range (0, 1) that add up to 1.

The softmax function is used in the final layer of a convolutional neural network-based classifier. This network is trained under a log loss (or cross-entropy) regime, giving a non-linear variant of multinomial logistic regression.

Adam Optimizer

The method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients.

The authors describe Adam as combining the advantages of two other extensions of stochastic gradient descent. Specifically: Adaptive Gradient Algorithm (AdaGrad) that maintains a per-parameter learning rate that improves performance on problems with sparse gradients (e.g. natural language and computer vision problems).

Root Mean Square Propagation (RMSProp) that also maintains per-parameter learning rates that are adapted based on the average of recent magnitudes of the gradients for the weight (e.g. how quickly it is changing). This means the algorithm does well on online and non-stationary problems (e.g. noisy).

6.2.4 Plotting images of the test Set which are incorrectly classified

```
def plot_example_errors(cls_pred, correct):
    # This function is called from print_test_accuracy() below.

    # cls_pred is an array of the predicted class-number for
    # all images in the test-set.

    # correct is a boolean array whether the predicted class
    # is equal to the true class for each image in the test-set.

    # Negate the boolean array.
    incorrect = (correct == False);

    # Get the images from the test-set that have been
    # incorrectly classified.
    images = data.test.images[incorrect];

    # Get the predicted classes for those images.
    cls_pred = cls_pred[incorrect];

    # Get the true classes for those images.
```

```

cls_true = data.test.cls[incorrect];

# Plot the first 9 images.
plot_images(images=images[0:9],
            cls_true=cls_true[0:9],
            cls_pred=cls_pred[0:9]);

```

6.2.5 Plotting Confusion Matrix

Confusion matrix C is such that $C_{i,j}$ is equal to the number of observations known to be in group i but predicted to be in group j .

Thus in binary classification, the count of true negatives is $C_{0,0}$, false negatives is $C_{1,0}$, true positives is $C_{1,1}$ and false positives is $C_{0,1}$.

```

def plot_confusion_matrix(cls_pred):
    cm = confusion_matrix(y_true=cls_true, y_pred=cls_pred);
    TP = np.diag(cm)
    print("Printing True Positive Elements")
    print(TP)
    # array([ 963, 1119,  972,  975,  953,  818,  938,  975,  906,  949])

    print("Printing False Negative Values")
    FP = np.sum(cm, axis=0) - TP
    print(FP)
    # array([50, 28, 39, 56, 37, 11, 66, 42, 54, 49])

    print("Printing False Negative Values")
    FN = np.sum(cm, axis=1) - TP
    print(FN)
    # array([17, 16, 60, 35, 29, 74, 20, 53, 68, 60])
    # calculating precision and Recall
    print("printing precision and recall")
    precision = TP/(TP+FP)
    recall = TP/(TP+FN)
    print("Precision:-")
    print(precision)
    print("Recall:-")
    print(recall)

```

6.2.6 Printing test accuracy depending on the number of correctly classified images

```
def print_test_accuracy(show_example_errors=False,
                        show_confusion_matrix=False):

    # Number of images in the test-set.
    num_test = len(data.test.images);

    # Allocate an array for the predicted classes which
    # will be calculated in batches and filled into this array.
    cls_pred = np.zeros(shape=num_test, dtype=np.int);

    # Now calculate the predicted classes for the batches.
    # We will just iterate through all the batches.
    # There might be a more clever and Pythonic way of doing this.

    # The starting index for the next batch is denoted i.
    i = 0

    while i < num_test:
        # The ending index for the next batch is denoted j.
        j = min(i + test_batch_size, num_test);

        # Get the images from the test-set between index i and j.
        images = data.test.images[i:j, :]

        # Get the associated labels.
        labels = data.test.labels[i:j, :]

        # Create a feed-dict with these images and labels.
        feed_dict = {x: images,
                     y_true: labels}

        # Calculate the predicted class using TensorFlow.
        cls_pred[i:j] = session.run(y_pred_cls, feed_dict=feed_dict)

        # Set the start-index for the next batch to the
        # end-index of the current batch.
        i = j;

    # Convenience variable for the true class-numbers of the test-set.
    cls_true = data.test.cls;

    # Create a boolean array whether each image is correctly classified.
    correct = (cls_true == cls_pred);
```

```

# Calculate the number of correctly classified images.
# When summing a boolean array, False means 0 and True means 1.
correct_sum = correct.sum();

# Classification accuracy is the number of correctly classified
# images divided by the total number of images in the test-set.
acc = float(correct_sum) / num_test

# Print the accuracy.
msg = "Accuracy on Test-Set: {0:.1%} ({1} / {2})"
print(msg.format(acc, correct_sum, num_test))

# Plot some examples of mis-classifications, if desired.
if show_example_errors:
    print("Example errors:")
    plot_example_errors(cls_pred=cls_pred, correct=correct)

# Plot the confusion matrix, if desired.
if show_confusion_matrix:
    print("Confusion Matrix:")
    plot_confusion_matrix(cls_pred=cls_pred)

# Performance before any optimization
print("Performance before any optimization");
print_test_accuracy();

# Performance after 1 optimization iteration
print("Performance after 1 optimization iteration");

optimize(num_iterations=1);

print_test_accuracy();

# Performance after 100 optimization iterations¶
print("Performance after 100 optimization iterations");
optimize(num_iterations=99); # We already performed 1 iteration above.

print_test_accuracy(show_example_errors=True);

# Performance after 1000 optimization iterations
print("Performance after 1000 optimization iterations");

optimize(num_iterations=900); # We performed 100 iterations above.

print_test_accuracy(show_example_errors=True);

# Performance after 10,000 optimization iterations

```

```
print("Performance after 10,000 optimization iterations");  
  
optimize(num_iterations=2000); # We performed 1000 iterations above.  
  
print_test_accuracy(show_example_errors=True,  
                    show_confusion_matrix=True);
```

Chapter 7










Conclusion and Future Enhancements

7.1 Result Analysis

7.1.1 Performance Analysis

Here we will sequentially present the outputs obtained in our project:

```
Extracting data/MNIST/t10k-images-idx3-ubyte.gz
Extracting data/MNIST/t10k-labels-idx1-ubyte.gz
WARNING:tensorflow:From C:\Users\Valencia Dias\Anaconda3\lib\site-packages\tensorflow\contrib\learn\python\learn\datasets\mnist.py:290:
DataSet.__init__ (from tensorflow.contrib.learn.python.learn.datasets.mnist) is deprecated and will be removed in a future version.
Instructions for updating:
Please use alternatives such as official/mnist/dataset.py from tensorflow/models.
Size of:
- Training-set:      55000
- Test-set:          10000
- Validation-set:    5000
```

		
True: 7	True: 2	True: 1
		
True: 0	True: 4	True: 1
		
True: 4	True: 9	True: 5

```
Tensor("Relu_0", shape=(?, 14, 14, 16), dtype=float32)
Tensor("Relu_1:0", shape=(?, 7, 7, 36), dtype=float32)
Tensor("Reshape_2:0", shape=(?, 1764), dtype=float32)
1764
Tensor("Relu_2:0", shape=(?, 128), dtype=float32)
Tensor("add_3:0", shape=(?, 10), dtype=float32)
```

Fig .7.1.1.1 : Extracting the data from the MNIST dataset

The above figure shows that how the mnist dataset is loaded that is extracting the data from the dataset. Also the size of the training set, validation set and test set is decided

Training set: a set of examples used for learning: to fit the parameters of the classifier In the MLP case, we would use the training set to find the “optimal” weights with the back-prop rule

Validation set: a set of examples used to tune the parameters of a classifier In the MLP case, we would use the validation set to find the “optimal” number of hidden units or determine a stopping point for the back-propagation algorithm

Test set: a set of examples used only to assess the performance of a fully-trained classifier In the MLP case, we would use the test to estimate the error rate after we have chosen the final model (MLP size and actual weights) After assessing the final model on the test set, YOU MUST NOT tune the model any further!

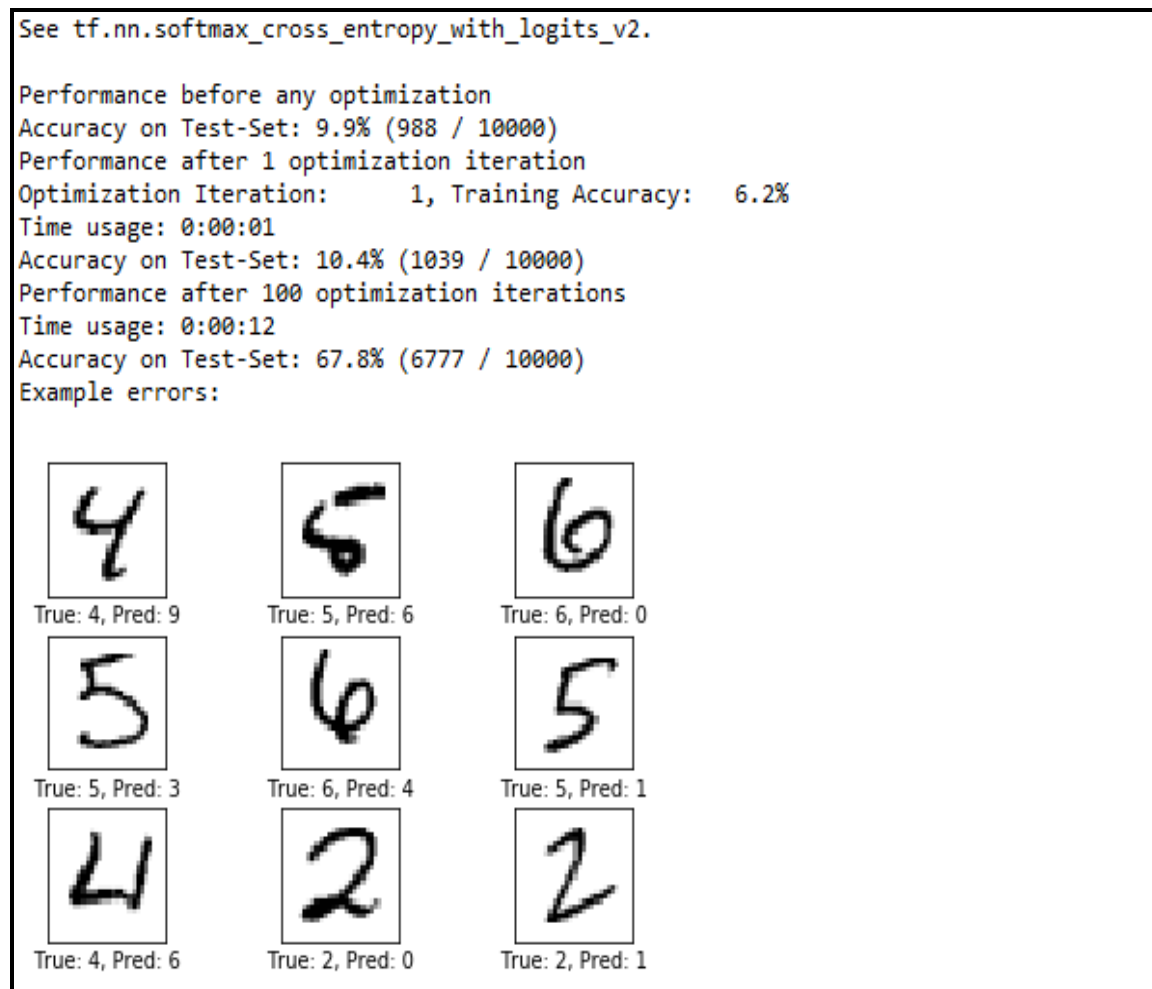


Fig 7.1.1.1: Performance of the algorithm before any optimization on the dataset

Computes softmax cross entropy between logits and labels. Measures the probability error in discrete classification tasks in which the classes are mutually exclusive (each entry is in exactly one class)

The softmax function, or normalized exponential function is a generalization of the logistic function that "squashes" a K-dimensional vector of arbitrary real values to a K-dimensional vector of real values in the range (0, 1) that add up to 1.

The softmax function is used in the final layer of a convolutional neural network-based classifier. This network is trained under a log loss (or cross-entropy) regime, giving a non-linear variant of multinomial logistic regression.

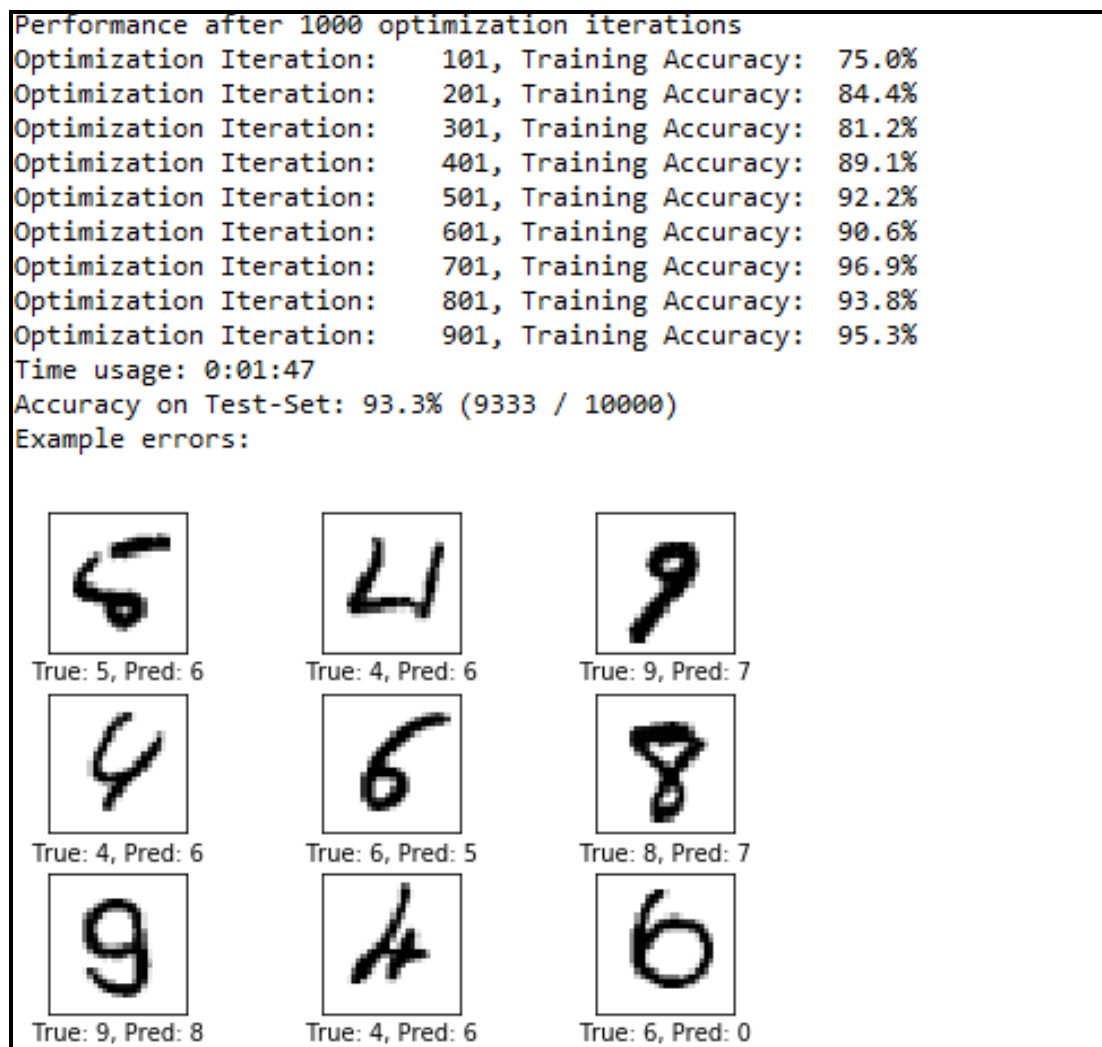
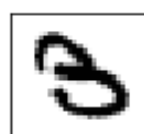


Fig 7.1.1.3: Performance after 1000 optimization iterations

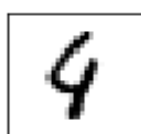
```

Optimization Iteration: 9301, Training Accuracy: 98.4%
Optimization Iteration: 9401, Training Accuracy: 100.0%
Optimization Iteration: 9501, Training Accuracy: 100.0%
Optimization Iteration: 9601, Training Accuracy: 98.4%
Optimization Iteration: 9701, Training Accuracy: 98.4%
Optimization Iteration: 9801, Training Accuracy: 98.4%
Optimization Iteration: 9901, Training Accuracy: 100.0%
Optimization Iteration: 10001, Training Accuracy: 98.4%
Optimization Iteration: 10101, Training Accuracy: 100.0%
Optimization Iteration: 10201, Training Accuracy: 100.0%
Optimization Iteration: 10301, Training Accuracy: 98.4%
Optimization Iteration: 10401, Training Accuracy: 98.4%
Optimization Iteration: 10501, Training Accuracy: 96.9%
Optimization Iteration: 10601, Training Accuracy: 100.0%
Optimization Iteration: 10701, Training Accuracy: 100.0%
Optimization Iteration: 10801, Training Accuracy: 100.0%
Optimization Iteration: 10901, Training Accuracy: 100.0%
Time usage: 0:19:29
Accuracy on Test-Set: 98.7% (9870 / 10000)
Example errors:

```



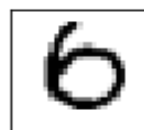
True: 3, Pred: 8



True: 4, Pred: 9



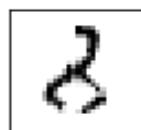
True: 4, Pred: 2



True: 6, Pred: 0



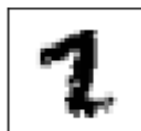
True: 9, Pred: 1



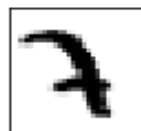
True: 8, Pred: 2



True: 1, Pred: 8



True: 2, Pred: 1



True: 7, Pred: 3

Fig 7.1.1.4: Performance after 10000 optimization iterations

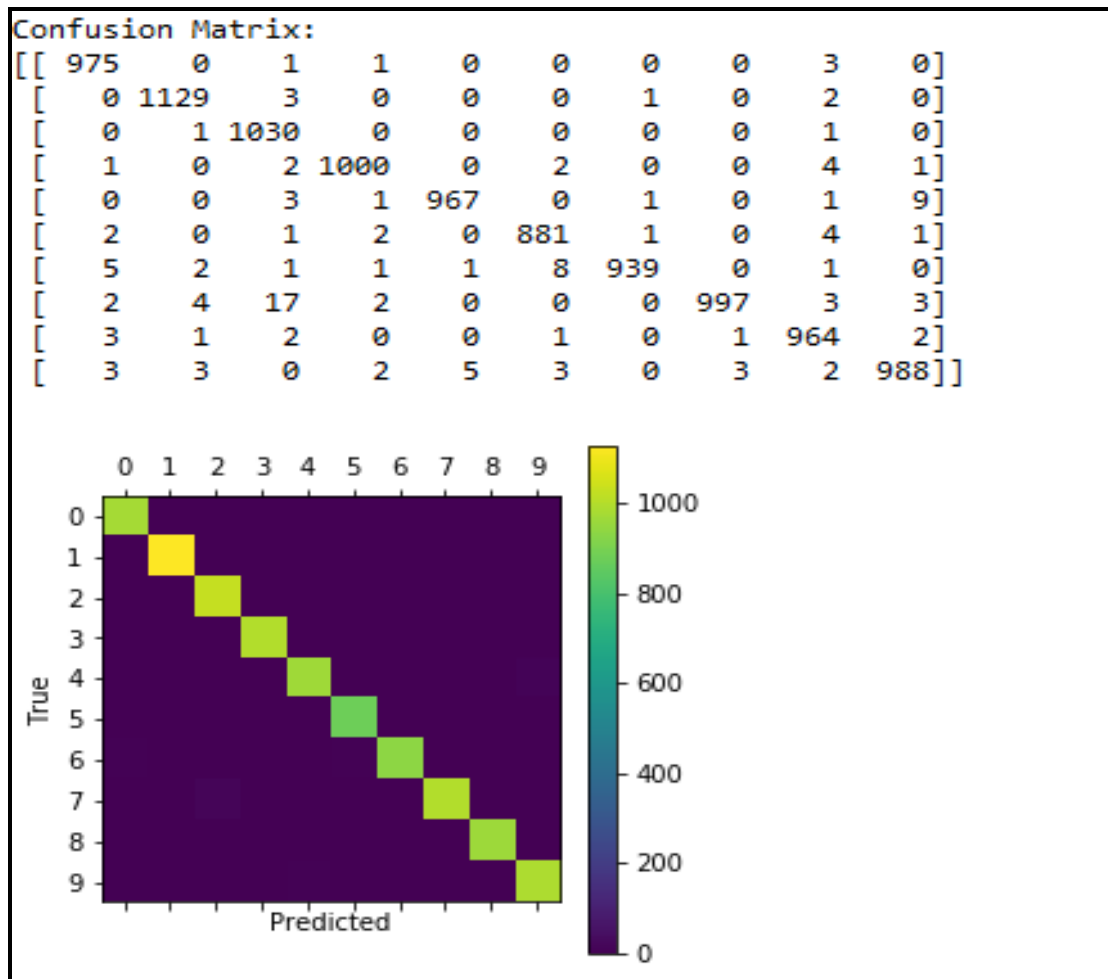


Fig 7.1.1.5:Plotting the Confusion Matrix

The above figure shows the plotting of the confusion matrix that describes the performance of classification model (or "classifier") on a set of test data for which the true values are known. The confusion matrix itself is relatively simple to understand, but the related terminology can be confusing.

```

Printing True Positive Elements
[ 975 1129 1030 1000 967 881 939 997 964 988]
Printing False Positive values
[16 11 30 9 6 14 3 4 21 16]
Printing False Negative Values
[ 5 6 2 10 15 11 19 31 10 21]
printing precision and recall
Precision:-
[0.98385469 0.99035088 0.97169811 0.99108028 0.9938335 0.98435754
0.99681529 0.996004 0.9786802 0.98406375]
Recall:-
[0.99489796 0.99471366 0.99806202 0.99009901 0.98472505 0.98766816
0.98016701 0.96984436 0.98973306 0.97918731]

```

Fig 7.1.1.6: Printing the precision and recall values

The above figure shows the values of recall and precision calculated from the true positive(TP), false positive(FP), false negative(FN) values.

Recall is calculated as the ratio of the total number of correctly classified positive examples divide to the total number of positive examples. High Recall indicates the class is correctly recognized (small number of FN). Recall is given by the relation:

$$\text{Recall} = \frac{TP}{TP + FN}$$

To get the value of precision we divide the total number of correctly classified positive examples by the total number of predicted positive examples. High Precision indicates an example labeled as positive is indeed positive (small number of FP). Precision is given by the relation:

$$\text{Precision} = \frac{TP}{TP + FP}$$

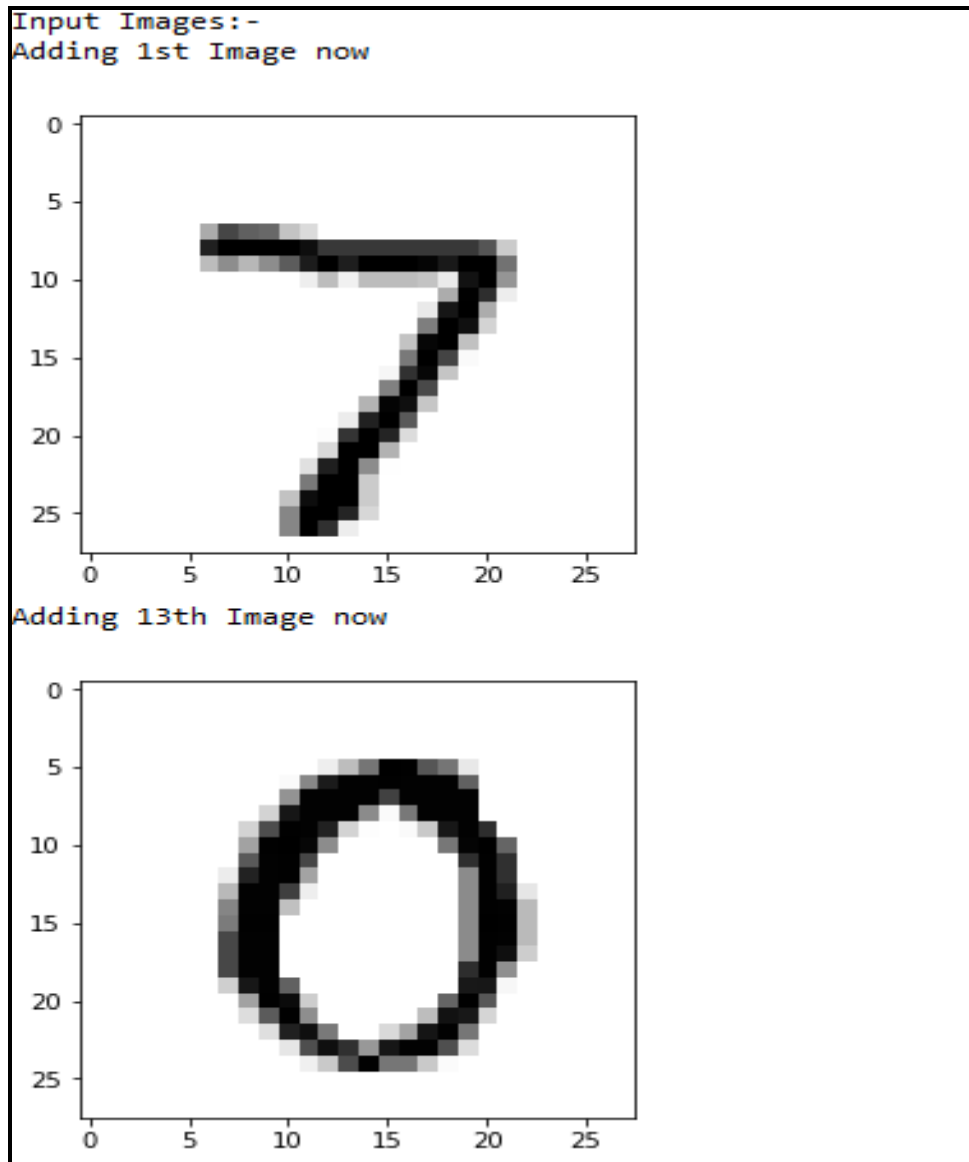


Fig7.1.1.7: Inputting the 1st and 13th image from the dataset

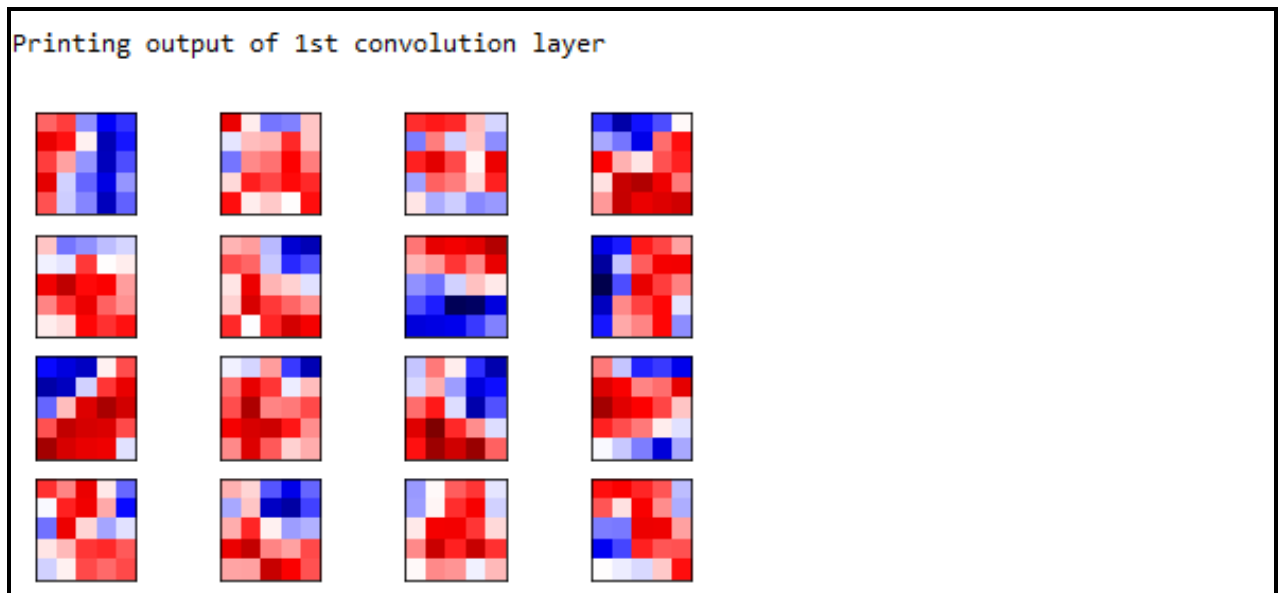


Fig.7.1.1.8 :Printing the output of 1st Convolution layer

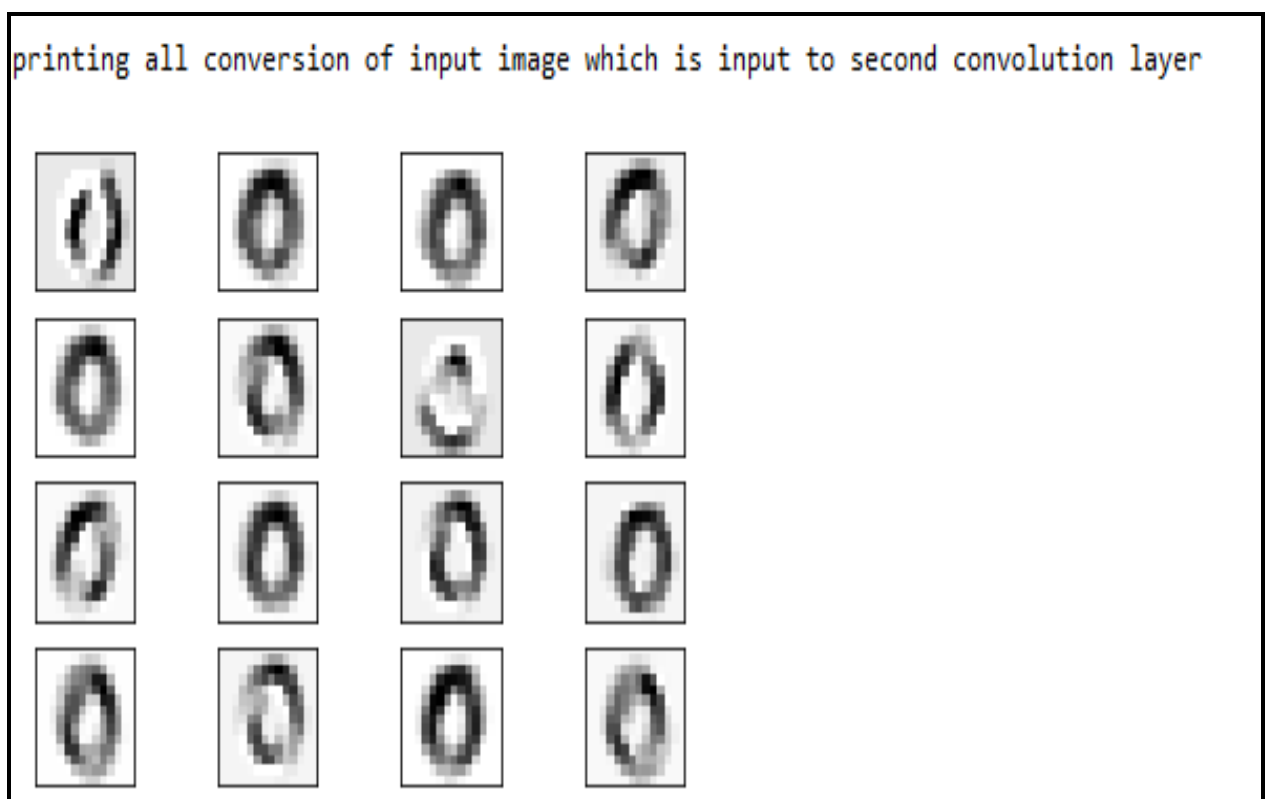


Fig.7.1.1.9 :All Conversion of the image input to the 2nd Convolutional layer

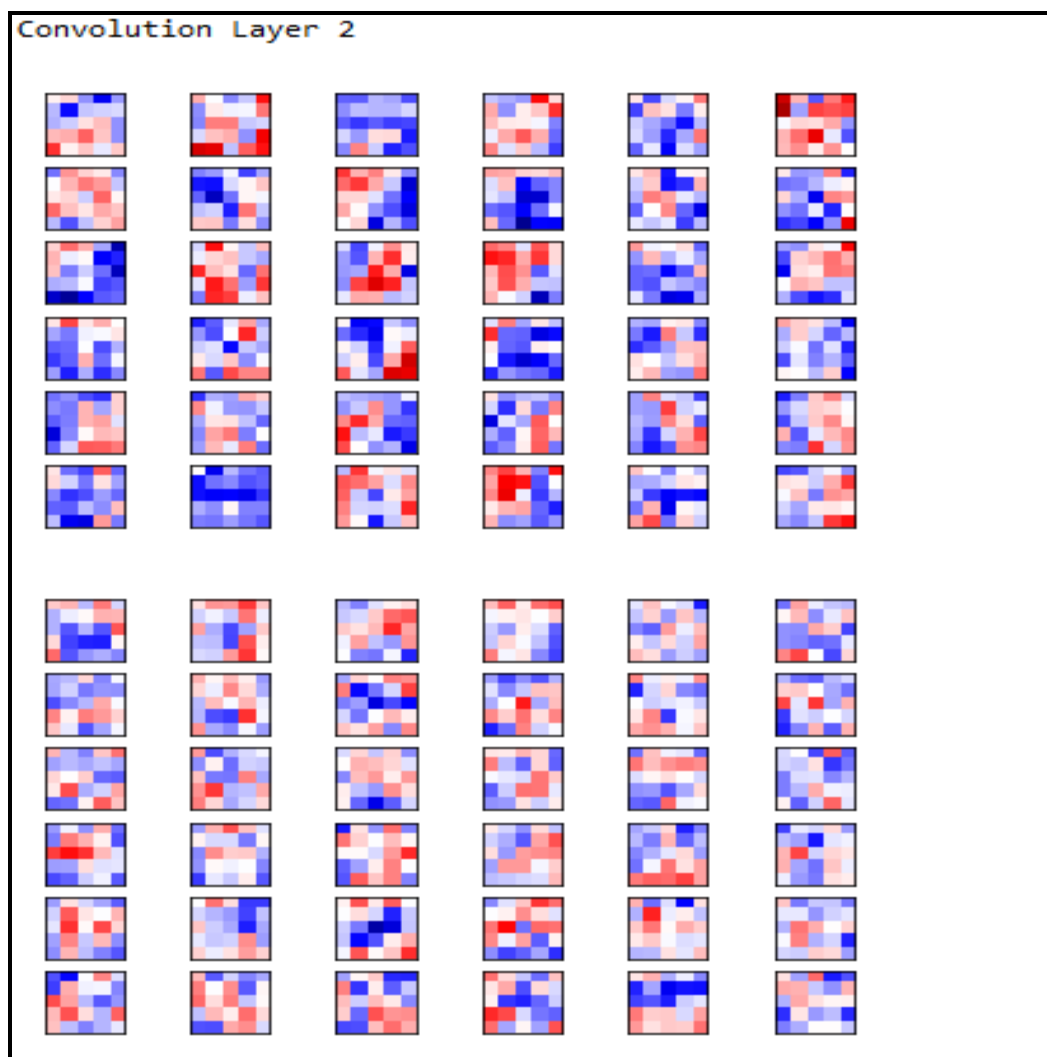


Fig .7.1.1.10:Convolutional Layer 2



Fig 7.1.1.11:Output of 2nd Convolutional Layer

7.2 Future Enhancement

We have restricted our study to isolated digit recognition because of lack of standard benchmarking Datasets. Based on our obtained results the following are the future directions for continuing the research

1. The work specifically belongs to isolated Digits with assumption that the images are of good quality and therefore complex normalization, skew removal and slant removal operations are not performed.
2. We would try to improve the learning and classification tasks, by the use of techniques like network pruning and variations in connection.
3. The developed model can be further extended for the holistic approach of conjunct/ word recognition.
4. The use of other training sets like alphanumeric character sets is an important challenge.

7.3 Conclusion

In our project we have applied convolutional neural network algorithm to the handwritten digits. Our goal was to recognize the handwritten digits from the MNIST dataset. Using the Convolutional Neural Network with Keras and TensorFlow as backend, we are able to get an accuracy of 98.7%. Every tool has its own complexity and accuracy. Although, we see that the complexity of the code and the process is bit more as compared to normal Machine Learning algorithms but looking at the accuracy achieved, it can be said that it is worth it. The recognition system is also able to recognize the images as quickly as possible for this large dataset. As the accuracy achieved is very high which means the error rate is very small in comparison to the neural network algorithm and other machine learning algorithms. Also, the current implementation is done only using the CPU.

References

1. Kai Ding, Zhibiniu, ianwen Jin, Xinghua Zhu, A Comparative study of GABOR feature and gradient feature for handwritten chinese character recognition, *International Conference on Wavelet Analysis and Pattern Recognition*, pp. 1182-1186, Beijing, China, 2007.
2. Pranob K Charles, V.Harish, M.Swathi, CH. Deepthi, "A Review on the Various Techniques used for Optical Character Recognition", *International Journal of Engineering Research and Applications*,2(1), pp. 659-662, 2012.
3. Bhatia Neetu, Optical Character Recognition Techniques, *International Journal of Advanced Research in Computer Science and Software Engineering*, 4(5), pp. 1219-1223, 2014.
4. Iana M. origo and VenuGovindaraju, Offline Arabic Handwriting Recognition: A Survey, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(5), pp. 712724, 2006.
5. K. Gaurav and Bhatia P. K., Analytical Review of Preprocessing Techniques for Offline Handwritten Character Recognition, *2 nd International Conference on Emerging Trends in Engineering & Management, ICETEM*, pp. 14-22, 2013.
6. Kai Ding, Zhibiniu, ianwen Jin, Xinghua Zhu, A Comparative study of GABOR feature and gradient feature for handwritten chinese character recognition, *International Conference on Wavelet Analysis and Pattern Recognition*, pp. 1182-1186, Beijing, China, 2007.
7. Pranob K Charles, V.Harish, M.Swathi, CH. Deepthi, "A Review on the Various Techniques used for Optical Character Recognition", *International Journal of Engineering Research and Applications*,2(1), pp. 659-662, 2012.
8. Bhatia Neetu, Optical Character Recognition Techniques, *International Journal of Advanced Research in Computer Science and Software Engineering*, 4(5), pp. 1219-1223, 2014.
9. Iana M. origo and VenuGovindaraju, Offline Arabic Handwriting Recognition: A Survey, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(5), pp. 712724, 2006
10. K. Gaurav and Bhatia P. K., Analytical Review of Preprocessing Techniques for Offline Handwritten Character Recognition, *2 nd International Conference on Emerging Trends in Engineering & Management, ICETEM*, pp. 14-22, 2013.
11. Y.LeCun,B.Baser,J.S.Denker,D.Henderson ,R.E.Howard,R.Hubbard,and L.D.Jackel , Handwritten digit recognition with a back- propogation network in D.Tourezky, *Advances in Neural Information Processing Systems 2*, Morgan Kaufman(1990)
12. Haider A.Alwzwazy, Hayder M. Albehadili, Younes S.Alwan ,Naz E Islam paper on "Handwritten Digit Recognition Using Convolutional Neural Networks" Vol 4 ,Issue 2,February 2016

13. Xuan Yang , Jing Pu paper on “Mdig:Multi-digit Recognition using Convolutional Neural Network on Mobile
14. Kumar, R., Goyal, M.K., Ahmed, P. and Kumar, A., 2012, December.
Unconstrained handwritten numeral recognition using majority voting classifier. In *Parallel Distributed and Grid Computing (PDGC), 2012 2nd IEEE International Conference on* (pp. 284-289). IEEE