Handwritten Gujarati Character Recognition

Submitted By

Kalaria Shreyashkumar Hitendrabhai

14MCEC11

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INSTITUTE OF TECHNOLOGY

NIRMA UNIVERSITY

AHMEDABAD-382481

May 2016

Handwritten Gujarati Character Recognition

Major Project

Submitted in partial fulfillment of the requirements

for the degree of

Master of Technology in Computer Science and Engineering

Submitted By

Kalaria Shreyashkumar Hitendrabhai

(14MCEC11)

Guided By

Dr. Priyank Thakkar

Co-Guided By

Prof. Ankit Sharma

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INSTITUTE OF TECHNOLOGY

NIRMA UNIVERSITY

AHMEDABAD-382481

May 2016

Certificate

This is to certify that the major project entitled **"Handwritten Gujarati Character Recognition"** submitted by **Kalaria Shreyashkumar Hitendrabhai (Roll No: 14MCEC11)**, towards the partial fulfillment of the requirements for the award of degree of Master of Technology in Computer Science and Engineering of Nirma University, Ahmedabad, is the record of work carried out by him under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this major project part-II, to the best of my knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Dr. Priyank Thakkar
Guide & Associate Professor,
CSE Department,
Institute of Technology,
Nirma University, Ahmedabad.

Dr. Priyanka Sharma
Professor,
Coordinator M.Tech - CSE
Institute of Technology,
Nirma University, Ahmedabad

Dr. Sanjay Garg
Professor and Head,
CSE Department,
Institute of Technology,
Nirma University, Ahmedabad.

Dr. P. N. Tekwani
I/c Director,
Institute of Technology,
Nirma University, Ahmedabad

Statement of Originality

I, **Kalaria Shreyashkumar Hitendrabhai**, Roll. No. **14MCEC11**, give undertaking that the Major Project entitled **"Handwritten Gujarati Character Recognition"** submitted by me, towards the partial fulfillment of the requirements for the degree of Master of Technology in **Computer Science & Engineering** of Institute of Technology, Nirma University, Ahmedabad, contains no material that has been awarded for any degree or diploma in any university or school in any territory to the best of my knowledge. It is the original work carried out by me and I give assurance that no attempt of plagiarism has been made. It contains no material that is previously published or written, except where reference has been made. I understand that in the event of any similarity found subsequently with any published work or any dissertation work elsewhere; it will result in severe disciplinary action.

_____

Signature of Student

Date:

Place:

Endorsed by

Dr. Priyank Thakkar

(Signature of Guide)

Acknowledgements

It gives me immense pleasure in expressing thanks and profound gratitude to **Dr. Priyank Thakkar**, Associate Professor, Computer Science Department, Institute of Technology, Nirma University, Ahmedabad for his valuable guidance and continual encouragement throughout this work. The appreciation and continual support he has imparted has been a great motivation to me in reaching a higher goal. His guidance has triggered and nourished my intellectual maturity that I will benefit from, for a long time to come. Thanks are also due for **Prof. Ankit Sharma**, Assistant Professor, Institute of Technology, Nirma University, Ahmedabad for providing Gujarati language database.

It gives me an immense pleasure to thank **Dr. Sanjay Garg**, Hon'ble Head of Computer Science and Engineering Department, Institute of Technology, Nirma University, Ahmedabad for his kind support and providing basic infrastructure and healthy research environment.

A special thank you is expressed wholeheartedly to **Dr. P. N. Tekwani**, Hon'ble I/c Director, Institute of Technology, Nirma University, Ahmedabad for the unmentionable motivation he has extended throughout course of this work.

I would also thank the Institution, all faculty members of Computer Engineering Department, Nirma University, Ahmedabad for their special attention and suggestions towards the project work.

- Kalaria Shreyash

14MCEC11

Abstract

Character recognition is a task of classifying character images into one of the many predefined classes. The focus of this thesis is on handwritten Gujarati character recognition. Specifically, thesis focuses on applying deep learning techniques for the task of handwritten Gujarati character recognition. Experiments are carried out on three datasets, out of which two are Gujarati numeral datasets while one is a Gujarati character dataset. LeNet - a well known deep neural network is used for the task. A significant contribution of the thesis is ILeNet which is inspired from LeNet and fine-tuned for the requirement of handwritten Gujarati character recognition. Experimental results demonstrate that classification accuracy of LeNet on all three datasets is significant. ILeNet improves the accuracy further and establishes the importance of ILeNet.

Conventions

Typesetting

This thesis is typeset using Latex software.
Font used in this thesis are of Times new roman family.

Referencing

Referencing and citation style adopted in this thesis is ieee transaction(ieeetr).
For electronic references, Last publication date is shown here.

Spelling

The Unites States English Spelling is adopted here.

Units

The Units used in This thesis are based in the International System of Units(SI Units), unless specified.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Problem Definition

During last several years, Handwritten character recognition is the topic of interest under the frame work of character recognition and pattern recognition.

Character recognition task is divided into online and offline character recognition. Online character recognition means recognition of character as soon as it is written from any input device like a special digitizer or from PDA and offline character recognition means recognition of character from static image of character or from any scanned document.

So, character recognition is a task of convert character from any character image, printed document or from handwritten document to machine encoded text. Here, character recognition is performed on Gujarati language.

1.2 Motivation

Character recognition helps in many ways like for saving handwritten page or form data to computer, reading bank cheque details, etc. Until now, there are many research done on English language and on other foreign language and many standard database are available for English language. For Indian language like Hindi, Gujarati, Bangla, etc. less research is done compare to English. So, here handwritten character recognition is done on Gujarati language which is official language of Gujarat.

1.3 Outline of the thesis

Chapter 2 contains various database details. It includes English language databases like MNIST, IAM Handwritten Dataset, NIST Dataset, etc. Gujarati language database which is preparing is also describe here.

Chapter 3 contains survey of different papers for character recognition.

Chapter 4 contains detail of caffe which is deep learning framework. In this chapter, installation steps and libraries of pycaffe are describe. Here, Caffe is use for implementation of various CNN models.

Chapter 5 contains implementation part. In this chapter, all CNN models are describe which are used for generating results. And also describe all accuracy result for all three Gujarati databases with different cases.

Chapter 6 contains conclusion part.

Chapter 2

Available Datasets

2.1 Other languages

There are many standard datasets are available for English language like MNIST, IAM handwritten dataset, NIST hand printed forms and characters dataset, UJI pencharacter dataset.

2.1.1 MNIST

- MNIST is dataset of handwritten English numerals. The digits have been size-normalized and centered in a fixed-size image[2].
- It contains 60,000 training examples and 10,000 testing examples.
- The original black and white (bilevel) images from NIST were size normalized to fit in a 20x20 pixel box while preserving their aspect ratio. The resulting images contain grey levels as a result of the anti-aliasing technique used by the normalization algorithm. the images were centered in a 28x28 image by computing the center of mass of the pixels, and translating the image so as to position this point at the center of the 28x28 field[2].

2.1.2 IAM Handwritten Dataset

- It contains handwritten English text.
- It was first published in ICDAR 1999 (International Conference on Document Analysis and Recognition).

- In version 3.0, 657 writers contributed samples of their handwriting.
- It contains 1,539 pages of scanned text, 5,685 isolated and labeled sentences, 13,353 isolated and labeled text lines and 1,15,320 isolated and labeled words.

2.1.3 NIST Hand printed Forms and Characters Database

- It contains English language hand printed characters.
- In includes hand printed sample forms from 3600 writers. From this forms 8,10,000 character images are collected.

2.1.4 UJI Pen Characters Data Set

- This database contains sample from 11 different writers.
- Each writer contribute with uppercase and lowercase letters and digits. And from each writer they collect 2 sample. So, database contains total 1364 samples.
- The handwriting samples were collected on a Toshiba Portg M400 Tablet PC using its cordless stylus.

2.2 Gujarati language

For Gujarati language databases are prepared. A numeral dataset and a character dataset are prepared in addition to one more numeral dataset which is availed from external sources. Numeral Database 1 contains 12000 images of Gujarati numerals and Database 2 contains 14000 images of Gujarati numerals. In database 1 for each numerals 1200 images are there and in database 2, there are 1400 images for each numerals. In both database image size is 16x16 pixels. Sample database images are shown in figure 2.1. Alphabet database contains 88751 images for different Gujarati alphabets. This database contains around 2000 images for each alphabets. Sample images from alphabet database is shown in figure 2.2. And table 2.1 shows images per alphabets in alphabet database.

Alphabets	Number of images	Alphabets	Number of images
ka	2001	kha	2002
ga	2007	gha	2011
cha	2048	chha	2048
ja	2000	jha	2006
t	2004	th	2068
da	2002	dha	2003
ana	2024	ta	2005
tha	2033	da	2006
dha	2005	na	2002
pa	2020	pha	2012
ba	2002	bha	2104
ma	2002	ya	2014
ra	2005	la	2049
ala	2009	va	2005
sha	2007	sa	2005
ha	2027	ksha	2015
gna	2004	a	2016
e	2005	ee	2008
u	2003	uu	2102
roo	2012	am	2000

Table 2.1: Number of images per alphabet.

Figure 2.1: Sample images for Gujarati numerals.

Figure 2.2: Sample images for Gujarati Alphabets.

Chapter 3

Literature Survey

A lot of research work is done in the area of character recognition for various languages. For English language lots of work is done in compare of any other language. For Gujarati language less research work is done in compare of other Indian languages.

Character recognition can classify in two classes, character recognition for printed character and character recognition for handwritten character.

Generally character recognition flow is like first from input images are preprocessed and then from these preprocessed images features are extracted. This feature vector is used for either train the classification model or to classify the image. This classical technique flow of character recognition is shown in figure 3.2. Figure **??** shows character recognition using deep learning / convolutional neural network. Here, in this no need for hand crafted feature extraction. Convolutional neural network works for both feature extraction and classification.

Image perprocessing is required because all scanned images may not be in same size or format. In image preprocessing task, first images are scanned from any handwritten or printed document. On that scanned image, preprocessing task like size normalization, background noise removal and skeletonization are preformed.

For feature extraction, lots of techniques are available. Task of feature extraction is divided in two main categories, structural and statistical feature[3]. Structural Feature are like strokes and bays in various directions, end points, intersections of line segments, loops and stroke relations, etc. Statistical features are derived from statistical distribution of points like zoning, moments, n-tuples, etc. Form these feature a feature vector of size 124 is generated.

Figure 3.1: Character recognition using classical techniques.

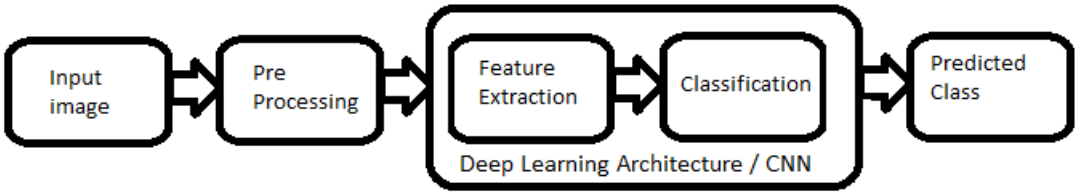

Figure 3.2: Character recognition using deep learning / Convolutional neural network.

Some can use direct image pixel values as feature means feature vector includes pixel value as feature. For example if our image size is 16x16 pixel than our feature vector is of size 256 and it contains pixel value.

For classification tasks, many techniques are available. Yann LeCun et al. [1] used convolutional neural network and build LeNet-5 for document recognition on English language. Also there is other classifier available like KNN, SVM and neural networks for classification.

For Devnagri language, Sethi and Chatterjee [4] developed system to recognize Devnagari hand printed character using binary decision tree classifier.

For Gujarati character recognition, Desai [5] applies neural network for classification,

for this numeral database of 610 images used for training and 2650 images used for testing. With this they get accuracy of 81.66%. Other paper is for offline handwritten character recognition of Gujarati Script using pattern matching[6]. In this paper they use preprocessing technique like Inversion, Gray scaling, Thinning. For classification they use Back propagation neural network and Template Matching Algorithm. With this, they get accuracy of 71.66%.

Chapter 4

Caffe - Deep Learning Framework

4.1 What is Caffe?

Caffe is a deep learning framework made with expression, speed, and modularity in mind. It is developed by the Berkeley Vision and Learning Center (BVLC)[7].

It is pure C++ and CUDA based architecture for deep learning.

Caffe can run on Ubuntu, OS X, RHEL, CentOS, Fedora.

We can use caffe by three different interfaces:

- Command line
- Python
- MATLAB

4.2 Why Caffe?

To use caffe, we can configure our models without hard coding, this is because of its expressive architecture. We can switch between its two available mode, CPU and GPU. To switch between each other, we need to set a single flag to train[7].

Now a days, speed is more important for computing. Caffe provides that to us means if we use caffe on NVIDIA K40 GPU, it processes 60M images per day, that proves caffe is fastest available deep learning framework[7].

4.3 Installation Step

4.3.1 Prerequisites

CUDA installation

CUDA is required for GPU mode. CUDA 7.0 and latest driver for GPU is required for caffe. If CUDA 7.0 is not supported with your GPU than it will also work fine with 6.0 too.

To install CUDA to your GPU system, first you need to download CUDA from its official site (prefer .deb file to install CUDA) then run following command in terminal.

```
$ sudo dpkg -i <path to .deb file>
$ sudo apt-get update
$ sudo apt-get install cuda
```

After this we need to set following environment variable to .bashrc file of home directory.

```
$ export CUDA_HOME = /usr/local/cuda-7.5
$ export LD_LIBRARY_PATH = ${CUDA_HOME}/lib64
$ PATH = ${CUDA_HOME}/bin:${PATH}
$ export PATH
```

Now we can check installation by running command "nvcc -V" or "nvcc –version" to check the version of CUDA.

Install libprotobuf, leveldb, blast, boost, lmdb and hdf5

```
$ sudo apt-get install libprotobuf-dev protobuf-compiler
  gfortran \ libboost-dev cmake libleveldb-dev libsnappy-dev
  \ libboost-thread-dev libboost-system-dev \ libatlas-base-dev
  libhdf5-serial-dev libgflags-dev \ libgoogle-glog-dev liblmdb-dev
  gcc-4.7 g++-4.7
```

Install OpenCV

- install dependencies

```
$ sudo apt-get -y install libopencv-dev build-essential
```

```
cmake git libgtk2.0-dev pkg-config python-dev python-numpy
libdc1394-22 libdc1394-22-dev libjpeg-dev libpng12-dev
libtiff4-dev libjasper-dev libavcodec-dev libavformat-dev
libswscale-dev libxine-dev libgstreamer0.10-dev
libgstreamer-plugins-base0.10-dev libv4l-dev libtbb-dev
libqt4-dev libfaac-dev libmp3lame-dev libopencore-amrnb-dev
libopencore-amrwb-dev libtheora-dev libvorbis-dev
libxvidcore-dev x264 v4l-utils unzip
```

- Download OpenCV

```
$ mkdir opencv
$ cd opencv
$ wget -O OpenCV-$version.zip http://sourceforge.net/projects
  /opencvlibrary/files/opencv-unix/$version/opencv-3.0.0.zip
  /download
$ unzip opencv-3.0.0.zip
```

- Install OpenCV

```
$ cd opencv-3.0.0-alpha
$ mkdir build
$ cd build
$ cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=
  /usr/local -D WITH_TBB=ON -D WITH_V4L=ON -D WITH_QT=ON -D
  WITH_OPENGL=ON ..
$ make    j 2
$ sudo make install
```

- Finishing installation

```
$ sudo sh -c 'echo "/usr/local/lib" > /etc/ld.so.conf.d/
  opencv.conf'
$ sudo ldconfig
```

Dependencies for pycaffe

Before "make pycaffe", we need to install following python dependencies.

- Install numpy

```
$ sudo apt-get install python-numpy
```

- Install scipy

```
$ sudo apt-get install python-scipy
```

- Install scikit-image

```
$ sudo apt-get install python-skimage
```

- Install matplotlib

```
$ sudo apt-get install python-matplotlib
```

- Install ipython

```
$ sudo apt-get install ipython
```

- Install h5py

```
$ sudo apt-get install python-h5py
```

- Install networkx

```
$ sudo apt-get install python-networkx
```

- Install nose

```
$ sudo apt-get install python-nose
```

- Install pandas

```
$ sudo apt-get install python-pandas
```

- Install python-dateutil

```
$ sudo apt-get install python-dateutil
```

- Install protobuf

```
$ sudo pip install protobuf
```

- Install python-gflags

```
$ sudo apt-get install python-gflags
```

- Install python-pyyaml

```
$ sudo apt-get install python-pyyaml
```

- Install python-pillow

```
$ sudo apt-get install python-pillow
```

- Install python-six

```
$ sudo apt-get install python-six
```

4.3.2 Download and Install Caffe

Installation with CUDA

```
$ sudo apt-get install -y git git clone https://github.com
  /BVLC/caffe.git    cd caffe && git checkout dev cp
  Makefile.config.example Makefile.config sed -i "s/# CU
      STOM_CXX := g++/CUSTOM_CXX := g++-4.7/" Makefile.config
$ cmake ..
$ make all
$ make pycaffe
$ make runtest
```

Installation without CUDA

For CPU only caffe uncomment the "CPU_ONLY := 1" flag in "makefile.config" to configure and build the caffe and don't need to install CUDA for this.

Installation with NVIDIA cuDNN

For fastest operation Caffe is accelerated by drop-in integration of NVIDIA cuDNN. To speedup your Caffe models, install cuDNN then uncomment the "USE_CUDNN := 1" flag in "Makefile.config" when installing Caffe.

4.4 Liberaries of pycaffe

For use of caffe with python, pycaffe interface is available. And all files and scripts are available in caffe/python directory.

To use caffe in python "import caffe" will provide access to all its liberaries. By this we can handle models, do forward and backward, handle input and optputs and network visulations.

4.4.1 caffe.Net

caffe.Net is the central interface for loading, configuring, and running models. caffe.Classsifier and caffe.Detector provide convenience interfaces for common tasks[8].

caffe.Classifier

caffe.Classifier extends caffe.Net for image class prediction. For this prediction is done by scaling, oversampling or center cropping.

- Parameters:
  image_dims : dimensions to scale input for cropping/sampling. Default is to scale to net input size for whole-image crop. mean, input_scale, raw_scale, channel_swap: params for preprocessing options.

def predict(self, inputs, oversample=True):

Predict classification probabilities of inputs.

- inputs :
  iterable of (H x W x K) input ndarrays.
- oversample :
  boolean
  average predictions across center, corners, and mirrors when True (default). Center-only prediction when False.
- Returns:
  predictions: (N x C) ndarray of class probabilities for N images and C

caffe.Dectector :

Detector extends Net for windowed detection by a list of crops or selective search proposals.

- Parameters:
  mean, input_scale, raw_scale, channel_swap : params for preprocessing options.
- context_pad : amount of surrounding context to take s.t. a 'context_pad' sized border of pixels in the network input image is context, as in R-CNN feature extraction.

def detect_windows(self, images_windows):

Do windowed detection over given images and windows. Windows are extracted then warped to the input dimensions of the net.

- Parameters:
  images_windows: (image filename, window list) iterable.
  context_crop: size of context border to crop in pixels.
- Returns:
  detections: list of filename: image filename, window: crop coordinates,
  predictions: prediction vector dicts.

def detect_selective_search(self, image_fnames):

Do windowed detection over Selective Search proposals by extracting the crop and warping to the input dimensions of the net.

- Parameters:
  image_fnames: list
- Returns:
  detections: list of filename: image filename, window: crop coordinates,
  predictions: prediction vector dicts.

def crop(self, im, window):

Crop a window from the image for detection. Include surrounding context according to the 'context_pad' configuration.

- Parameters:

  im: H x W x K image ndarray to crop.

  window: bounding box coordinates as ymin, xmin, ymax, xmax.

- Returns: crop: cropped window.

def configure_crop(self, context_pad):

Configure crop dimensions and amount of context for cropping. If context is included, make the special input mean for context padding.

- Parameters:

  context_pad : amount of context for cropping.

4.4.2 caffe.SGDSolver

caffe.SGDSolver exposes the solving interface[8].

4.4.3 caffe.io

caffe.io handles input / output with preprocessing and protocol buffers[8].

def blobproto_to_array(blob, return_diff=False):

Convert a blob proto to an array. In default, we will just return the data, unless return_diff is True, in which case we will return the diff.

def array_to_blobproto(arr, diff=None):

Converts a 4-dimensional array to blob proto. If diff is given, also convert the diff. You need to make sure that arr and diff have the same shape, and this function does not do sanity check.

def arraylist_to_blobprotovecor_str(arraylist):

Converts a list of arrays to a serialized blobprotovec, which could be then passed to a network for processing.

def blobprotovector_str_to_arraylist(str):

Converts a serialized blobprotovec to a list of arrays.

def array_to_datum(arr, label=0):

Converts a 3-dimensional array to datum. If the array has dtype uint8, 0the output data will be encoded as a string. Otherwise, the output data will be stored in float format.

def datum_to_array(datum):

Converts a datum to an array. Note that the label is not returned, as one can easily get it by calling datum.label.

4.4.4 Class Transformer

Transform input for feeding into a Net. This is mostly for illustrative purposes and it is likely better to define your own input preprocessing routine for your needs.

- Parameters:
  net : a Net for which the input should be prepared

def preprocess(self, in_, data):

- Format input for Caffe:
  - convert to single
  - resize to input dimensions (preserving number of channels)
  - transpose dimensions to K x H x W
  - reorder channels (for instance color to BGR)
  - scale raw input (e.g. from [0, 1] to [0, 255] for ImageNet models)
  - subtract mean
  - scale feature
- Parameter:
  in_ : name of input blob to preprocess for
  data : (H' x W' x K) ndarray
- Returns:
  caffe_in : (K x H x W) ndarray for input to a Ne

def deprocess(self, in_, data):

Invert Caffe formatting

def set_transpose(self, in_, order):

Set the input channel order for e.g. RGB to BGR conversion as needed for the reference ImageNet model.

- Parameters:
  in_ : which input to assign this channel order
  order : the order to transpose the dimensions

def set_channel_swap(self, in_, order):

Set the input channel order for e.g. RGB to BGR conversion as needed for the reference ImageNet model. N.B. this assumes the channels are the first dimension AFTER transpose.

- Parameters:
  in_ : which input to assign this channel order order : the order to take the channels. (2,1,0) maps RGB to BGR for example.

def set_raw_scale(self, in_, scale):

Set the scale of raw features s.t. the input blob = input * scale. While Python represents images in [0, 1], certain Caffe models like CaffeNet and AlexNet represent images in [0, 255] so the raw_scale of these models must be 255.

- Parameters: in_ : which input to assign this scale factor scale : scale coefficient

def set_mean(self, in_, mean):

Set the mean to subtract for centering the data.

- Parameters:
  in_ : which input to assign this mean. mean : mean ndarray (input dimensional or broadcastable)

def set_input_scale(self, in_, scale):

Set the scale of preprocessed inputs s.t. the blob = blob * scale. N.B. input_scale is done AFTER mean subtraction and other preprocessing while raw_scale is done BEFORE.

- Parameters:
  in_ : which input to assign this scale factor scale : scale coefficient

def load_image(filename, color=True):

Load an image converting from grayscale or alpha as needed.

- Parameters:
  filename : string
  color : boolean flag for color format. True (default) loads as RGB while False loads as intensity (if image is already grayscale).
- Returns: image : an image with type np.float32 in range [0, 1] of size (H x W x 3) in RGB or of size (H x W x 1) in grayscale.

def resize_image(im, new_dims, interp_order=1):

Resize an image array with interpolation.

- Parameters:
  im : (H x W x K) ndarray
  new_dims : (height, width) tuple of new dimensions.
  interp_order : interpolation order, default is linear.
- Returns:
  im : resized ndarray with shape (new_dims[0], new_dims[1], K)

def oversample(images, crop_dims):

Crop images into the four corners, center, and their mirrored versions.

- Parameters:
  image : iterable of (H x W x K) ndarrays
  crop_dims : (height, width) tuple for the crops.
- Returns: crops : (10*N x H x W x K) ndarray of crops for number of inputs N.

4.4.5 caffe.draw

caffe.draw visualizes network architecture[8].

def get_pooling_types_dict():

Get dictionary mapping pooling type number to type name

def get_edge_label(layer):

Define edge label based on layer type.

def get_layer_label(layer, rankdir):

Define node label based on layer type.

- Parameters:

  layer : ?

  rankdir : 'LR', 'TB', 'BT' : Direction of graph layout.

- Returns:

  string : A label for the current layer

def choose_color_by_layertype(layertype):

Define colors for nodes based on the layer type.

def get_pydot_graph(caffe_net, rankdir, label_edges=True):

Create a data structure which represents the 'caffe_net'.

- Parameters:

  caffe_net : object

  rankdir : 'LR', 'TB', 'BT' : Direction of graph layout.

  label_edges : boolean, optional : Label the edges (default is True).

- Returns:

  pydot graph object

def draw_net(caffe_net, rankdir, ext='png'):

Draws a caffe net and returns the image string encoded using the given extension.

- Parameters:

  caffe_net : a caffe.proto.caffe_pb2.NetParameter protocol buffer.

  ext : string, optional

  The image extension (the default is 'png').

- Returns:

  string : Postscript representation of the graph.

def draw_net_to_file(caffe_net, filename, rankdir='LR'):

Draws a caffe net, and saves it to file using the format given as the file extension. Use '.raw' to output raw text that you can manually feed to graphviz to draw graphs.

- Parameters:
  caffe_net : a caffe.proto.caffe_pb2.NetParameter protocol buffer.
  filename : string : The path to a file where the networks visualization will be stored.
  rankdir : 'LR', 'TB', 'BT' : Direction of graph layout.

4.4.6 caffe blob

Caffe blobs are exposed as numpy ndarrays for ease-of-use and efficiency[8]. Compile pycaffe by make pycaffe. The module dir caffe/python/caffe should be installed in your PYTHONPATH for import caffe. **Pycaffe properties:**

def _Net_blobs(self):

An OrderedDict (bottom to top, i.e., input to output) of network blobs indexed by name.

def _Net_blob_loss_weights(self):

An OrderedDict (bottom to top, i.e., input to output) of network blob loss weights indexed by name

def _Net_params(self):

An OrderedDict (bottom to top, i.e., input to output) of network parameters indexed by name; each is a list of multiple blobs (e.g.,weights and biases)

def _Net_forward(self, blobs=None, start=None, end=None, **kwargs):

Forward pass: prepare inputs and run the net forward.

- Parameters:
  blobs : list of blobs to return in addition to output blobs.
  kwargs : Keys are input blob names and values are blob ndarrays. For formatting inputs for Caffe, see Net.preprocess(). If None, input is taken from data layers.
  start : optional name of layer at which to begin the forward pass
  end : optional name of layer at which to finish the forward pass (inclusive)

- Returns: outs : blob name: blob ndarray dict.

def _Net_backward(self, diffs=None, start=None, end=None, **kwargs):

Backward pass: prepare diffs and run the net backward.

- Parameters:
  diffs : list of diffs to return in addition to bottom diffs.
  kwargs : Keys are output blob names and values are diff ndarrays. If None, top diffs are taken from forward loss.
  start : optional name of layer at which to begin the backward pass
  end : optional name of layer at which to finish the backward pass (inclusive)
- Returns: outs: blob name: diff ndarray dict.

def _Net_forward_all(self, blobs=None, **kwargs):

Run net forward in batches.

- Parameters:
  blobs : list of blobs to extract as in forward()
  kwargs : Keys are input blob names and values are blob ndarrays. Refer to forward().
- Returns: all_outs : blob name: list of blobs dict.

def _Net_forward_backward_all(self, blobs=None, diffs=None, **kwargs):

Run net forward + backward in batches.

- Parameters:
  blobs: list of blobs to extract as in forward()
  diffs: list of diffs to extract as in backward()
  kwargs: Keys are input (for forward) and output (for backward) blob names and values are ndarrays. Refer to forward() and backward(). Prefilled variants are called for lack of input or output blobs.
- Returns:
  all_blobs: blob name: blob ndarray dict.
  all_diffs: blob name: diff ndarray dict.

def _Net_set_input_arrays(self, data, labels):

Set input arrays of the in-memory MemoryDataLayer. (Note: this is only for networks declared with the memory data layer.)

def _Net_batch(self, blobs):

Batch blob lists according to net's batch size.

- Parameters:

  blobs: Keys blob names and values are lists of blobs (of any length). Naturally, all the lists should have the same length.

  Yields:

  batch: blob name: list of blobs dict for a single batch.

Chapter 5

Gujarati Handwritten Numeral and Character Recognition using Deep Learning

Here in this work, convolutional neural network is used for character recognition. This convolutional neural network is implemented using Caffe deep learning framework. This networks are trained using two numerals and one alphabet database of Gujarati language with different cases and also trained and tested using MNIST for comparison purpose.

5.1 Convolutional Neural Network

Here, experiment is done using 5 different architecture of CNN. From these five network one is LeNet-5[1] which is used for English character recognition. And other four architecture are presented and used for purpose of obtaining higher accuracy.

This proposed CNNs consists of 6 layers except input layer with different value of feature maps and trainable parameters. The input image size is 16x16 pixel. Here in all this network architecture "xavier" weight filler is used. Xavier weight filler make sure that all weight in the network are just right, not too small or too large. In caffe weight initialization is done from a distribution form a zero mean and specific variance. Here

shows the xavier equation used for weight filler.

$$Var(W) = \frac{1}{n_{in}} \tag{5.1}$$

5.1.1 LeNet-5[1]

LeNet-5[1] is 7 layer CNN architecture except input layer. Figure 5.1 shows architecture of LeNet-5[1]. In this CNN model, first layer is convolutional layer with 6 feature map and 5x5 of weight matrix. So the size of each feature map is 12x12. Second layer is subsampling layer with 6 feature maps of size 6x6 with stride of 2 and kernel size of 2. Third layer is convolutional layer with 16 feature map and 5x5 of weight matrix, so feature map size is 2x2. Fourth layer is subsampling layer with 16 feature map of size 1x1 with stride of 2 and kernel size of 2. Fifth layer is fully connected layer contains 120 units and is connected with fourth subsampling layer. sixth layer is full connection with fifth layer with 84 connections. Last and seventh layer is output layer of size 10 for numerals and of size 44 for alphabets.

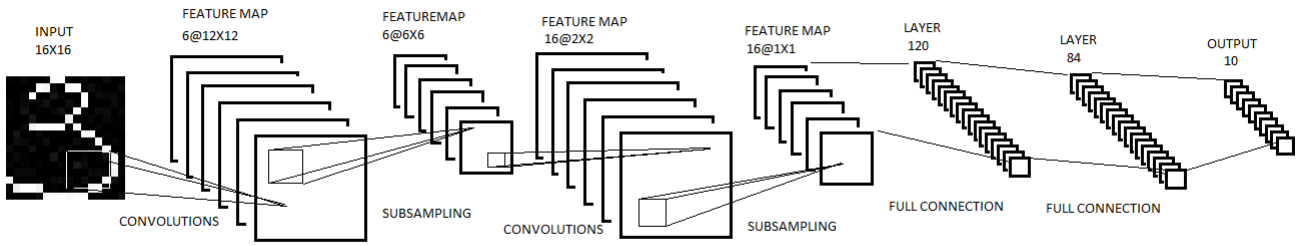

Figure 5.1: Architecture of LeNet-5[1]

5.1.2 Model-1

In this CNN model, first layer is convolutional layer with 20 feature map and 5x5 of weight matrix. So the size of each feature map is 12x12. Second layer is subsampling layer with 20 feature maps of size 6x6 with stride of 2 and kernel size of 2. Third layer is convolutional layer with 50 feature map and 5x5 of weight matrix, so feature map size is 2x2. Fourth layer is subsampling layer with 50 feature map of size 1x1 with stride of 2 and kernel size of 2. Fifth layer is fully connected layer contains 500 units and is

connected with fourth subsampling layer. Last and sixth layer is output layer of size 10 for numerals and of size 44 for alphabets. Figure 5.2 shows architecture of Model-1 for numerals means in this figure output layer shows 10 units.

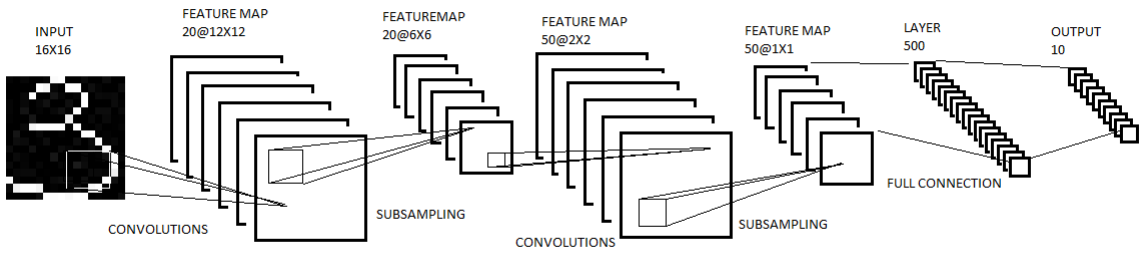

Figure 5.2: Architecture of Model-1

5.1.3 Model-2

In this CNN model, first layer is convolutional layer with 20 feature map and 3x3 of weight matrix. So the size of each feature map is 14x14. Second layer is subsampling layer with 20 feature maps of size 7x7 with stride of 2 and kernel size of 2. Third layer is convolutional layer with 50 feature map and 5x5 of weight matrix, so feature map size is 5x5. Fourth layer is subsampling layer with 50 feature map of size 3x3 with stride of 2 and kernel size of 2. Fifth layer is fully connected layer contains 500 units and is connected with fourth subsampling layer. Last and sixth layer is output layer of size 10 for numerals and of size 44 for alphabets. Figure 5.3 shows architecture of Model-2 for numerals means in this figure output layer shows 10 units.

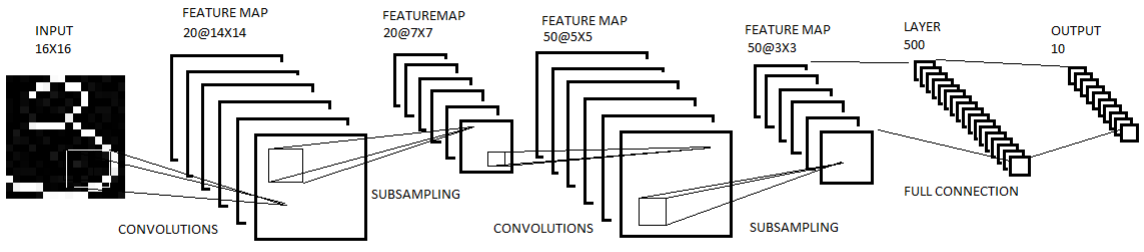

Figure 5.3: Architecture of Model-2

5.1.4 Model-3

In this CNN model, first layer is convolutional layer with 30 feature map and 5x5 of weight matrix. So the size of each feature map is 12x12. Second layer is subsampling layer with 30 feature maps of size 6x6 with stride of 2 and kernel size of 2. Third layer is convolutional layer with 75 feature map and 5x5 of weight matrix, so feature map size is 2x2. Fourth layer is subsampling layer with 75 feature map of size 1x1 with stride of 2 and kernel size of 2. Fifth layer is fully connected layer contains 750 units and is connected with fourth subsampling layer. Last and sixth layer is output layer of size 10 for numerals and of size 44 for alphabets. Figure 5.4 shows architecture of Model-3 for numerals means in this figure output layer shows 10 units.

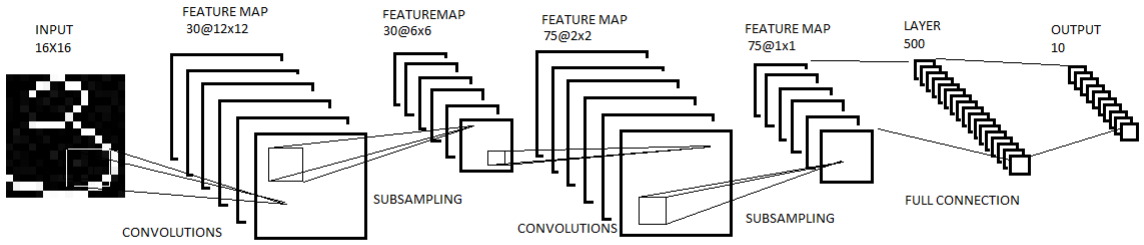

Figure 5.4: Architecture of Model-3

5.1.5 Model-4

In this CNN model, first layer is convolutional layer with 30 feature map and 3x3 of weight matrix. So the size of each feature map is 14x14. Second layer is subsampling layer with 30 feature maps of size 7x7 with stride of 2 and kernel size of 2. Third layer is convolutional layer with 75 feature map and 3x3 of weight matrix, so feature map size is 5x5. Fourth layer is subsampling layer with 75 feature map of size 3x3 with stride of 2 and kernel size of 2. Fifth layer is fully connected layer contains 750 units and is connected with fourth subsampling layer. Last and sixth layer is output layer of size 10 for numerals and of size 44 for alphabets. Figure 5.5 shows architecture of Model-1 for numerals means in this figure output layer shows 10 units.

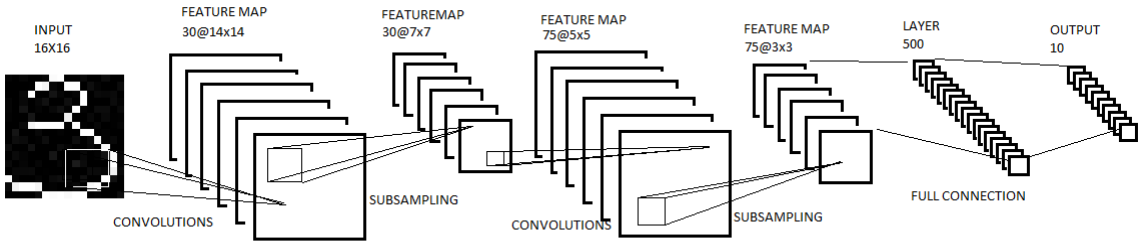

Figure 5.5: Architecture of Model-4

Chapter 6

Experimental Evaluation

All CNNs described in chapter 5 are trained and tested using two numerals and one characters dataset. This training and testing is done in five different ways as describe below.

- 5 fold cross validation - Numeral dataset 1
- 5 fold cross validation - Numeral dataset 2
- Numeral dataset 1 for training and dataset 2 for testing
- Numeral dataset 2 for training and dataset 1 for testing
- 5 fold cross validation - Character dataset

In case of single dataset for training and testing, 80% of images are used for train the network and 20% images are used for testing the trained network and accuracy is calculated using this tested images.

All this accuracy results are generated using 5 fold cross validation. And average accuracy of these 5 fold is considered as average accuracy of that particular case. All testing results for all proposed CNN and how those results are generated is described below.

6.1 5 fold cross validation - Numeral dataset 1

As described earlier numeral dataset 1 contain 12000 images means for each numeral there is 1200 images. Now this dataset is divided into five fold (Fold 1, Fold 2, Fold 3,

	LeNet-5	Model-1	Model-2	Model-3	Model-4
Fold 1	95.80%	97.10%	96.95%	96.89%	96.95%
Fold 2	95.75%	96.80%	97.50%	97.15%	97.15%
Fold 3	95.05%	96.80%	97.25%	97.10%	97.89%
Fold 4	95.85%	97.10%	97.55%	97.05%	97.60%
Fold 5	96.45%	96.30%	96.95%	97.00%	97.05%
Average Accuracy	95.78%	96.82%	97.24%	97.03%	97.32%

Table 6.1: Numeral dataset 1 : Accuracy result of five fold cross validation for fold 1 training images for all CNN models.

	LeNet-5	Model-1	Model-2	Model-3	Model-4
Fold 1	95.95%	96.60%	97.30%	97.25%	97.70%
Fold 2	95.35%	96.20%	97.30%	96.70%	97.45%
Fold 3	95.45%	97.13%	97.39%	96.89%	97.20%
Fold 4	94.55%	96.39%	96.39%	96.30%	96.20%
Fold 5	94.89%	96.39%	97.20%	96.85%	97.25%
Average Accuracy	95.24%	96.53%	97.11%	96.79%	97.16%

Table 6.2: Numeral dataset 1 : Accuracy result of five fold cross validation for fold 2 training images for all CNN models.

Fold 4, Fold 5), from these folds five cases are generated, in first case Fold 1 is keep aside for testing and with Fold 2, Fold 3, Fold 4 and Fold 5 network is trained, in second case Fold 2 is keep aside for testing and with other four network is trained, in third case Fold 3 is keep aside for testing and other four are used for train the network and so on. In each fold 9600 images for training and 2400 images for testing. Now to know which CNN model performs better to train the folds, training set of all folds are further divided into five fold, for example for Fold 1 if training images are divided into folds, in each case 7680 images for training and 1920 images for testing and the procedure is same as with whole dataset for a particular fold case. And this folds are trained and tested for all five CNN models, to get best model from all five and train with this model and get accuracy for numeral dataset 1. These accuracy results are shown in table 6.1 to 6.5.

Here, from all this accuracy results, Model-4 works better than other models. So, now this Model-4 is used for training five fold cases of numeral dataset 1 and average accuracy is calculated. Also this test performs for LeNet-5 for comparison purpose and generated results are shown in table 6.6. And confusion matrix for all cases are also shown in table 6.7 to 6.11.

	LeNet-5	Model-1	Model-2	Model-3	Model-4
Fold 1	96.15%	96.55%	97.15%	96.80%	97.35%
Fold 2	94.15%	96.50%	97.15%	96.75%	97.70%
Fold 3	95.70%	95.55%	97.30%	96.89%	97.50%
Fold 4	95.45%	96.89%	96.80%	96.45%	97.30%
Fold 5	95.30%	96.39%	96.89%	96.95%	97.10%
Average Accuracy	95.35%	96.57%	97.05%	96.76%	97.39%

Table 6.3: Numeral dataset 1 : Accuracy result of five fold cross validation for fold 3 training images for all CNN models.

	LeNet-5	Model-1	Model-2	Model-3	Model-4
Fold 1	95.90%	97.05%	97.00%	97.50%	97.65%
Fold 2	96.25%	96.15%	97.15%	96.45%	97.45%
Fold 3	94.60%	96.70%	97.35%	96.85%	97.30%
Fold 4	96.00%	96.75%	97.20%	96.95%	97.45%
Fold 5	95.85%	96.20%	97.25%	96.55%	96.95%
Average Accuracy	95.72%	96.57%	97.19%	96.86%	97.56%

Table 6.4: Numeral dataset 1 : Accuracy result of five fold cross validation for fold 4 training images for all CNN models.

	LeNet-5	Model-1	Model-2	Model-3	Model-4
Fold 1	95.45%	97.45%	98.05%	97.39%	97.80%
Fold 2	95.25%	96.15%	96.60%	95.89%	96.45%
Fold 3	95.89%	96.39%	97.15%	96.55%	97.50%
Fold 4	94.65%	96.30%	97.25%	96.55%	96.95%
Fold 5	96.05%	97.15%	97.39%	97.45%	97.45%
Average Accuracy	95.45%	96.68%	97.28%	96.76%	97.23%

Table 6.5: Numeral dataset 1 : Accuracy result of five fold cross validation for fold 5 training images for all CNN models.

	LeNet-5	Model-4
Fold 1	95.62%	97.54%
Fold 2	97.12%	97.87%
Fold 3	96.25%	97.79%
Fold 4	94.95%	97.25%
Fold 5	95.04%	96.91%
Average Accuracy	95.79%	97.47%

Table 6.6: Numeral dataset 1 : Accuracy result of five fold cross validation for LeNet-5 and Model-4.

	0	1	2	3	4	5	6	7	8	9
0	236	0	0	0	0	0	0	1	3	0
1	0	232	1	0	1	5	1	0	0	0
2	0	4	229	1	0	0	1	1	2	2
3	1	0	0	236	0	1	1	1	0	0
4	0	0	0	3	234	0	0	0	1	2
5	1	0	0	0	2	234	1	2	0	0
6	0	0	0	0	0	0	239	0	0	1
7	4	3	3	0	0	1	4	224	0	1
8	1	0	0	0	1	0	0	0	238	0
9	0	0	1	0	0	0	0	0	0	239

Table 6.7: Numeral dataset 1 : Confusion matrix of fold 1 for Model-4.

	0	1	2	3	4	5	6	7	8	9
0	237	0	0	0	0	0	0	3	0	0
1	0	231	2	0	1	3	0	2	1	0
2	0	3	235	0	1	0	1	0	0	0
3	0	0	0	238	0	0	0	2	0	0
4	0	0	0	0	240	0	0	0	0	0
5	0	1	2	0	3	234	0	0	0	0
6	0	1	0	4	1	1	231	2	0	0
7	4	0	0	4	0	1	2	229	0	0
8	0	0	0	1	0	0	0	0	238	1
9	1	0	0	1	1	1	0	0	1	235

Table 6.8: Numeral dataset 1 : Confusion matrix of fold 2 for Model-4.

	0	1	2	3	4	5	6	7	8	9
0	237	1	0	0	0	0	0	2	0	0
1	0	236	0	0	0	3	0	0	1	0
2	0	4	232	1	0	0	0	2	1	0
3	0	0	2	232	0	0	3	3	0	0
4	0	1	0	0	238	1	0	0	0	0
5	0	0	0	1	3	235	0	1	0	0
6	0	1	1	1	0	2	229	5	1	0
7	1	2	0	1	0	0	0	234	1	1
8	1	0	0	0	0	0	0	0	238	1
9	0	0	0	0	1	1	0	0	2	236

Table 6.9: Numeral dataset 1 : Confusion matrix of fold 3 for Model-4.

	0	1	2	3	4	5	6	7	8	9
0	233	0	0	0	1	1	0	3	2	0
1	0	233	2	0	0	1	1	3	0	0
2	0	3	232	0	0	1	1	2	0	1
3	0	0	1	232	0	0	1	6	0	0
4	0	1	3	1	233	2	0	0	0	0
5	0	1	1	0	4	230	3	1	0	0
6	0	1	0	2	0	0	234	3	0	0
7	1	2	1	0	0	0	2	234	0	0
8	0	1	0	0	3	0	0	0	236	0
9	0	0	2	0	1	0	0	0	1	236

Table 6.10: Numeral dataset 1 : Confusion matrix of fold 4 for Model-4.

	0	1	2	3	4	5	6	7	8	9
0	237	0	0	1	0	0	0	0	2	0
1	1	227	6	0	0	2	0	0	3	1
2	0	2	232	0	1	4	0	0	1	0
3	0	0	1	233	0	0	2	4	0	0
4	1	0	0	0	233	1	2	1	2	0
5	1	5	0	0	0	233	1	0	0	0
6	1	1	0	2	0	3	229	4	0	0
7	0	1	1	0	1	2	1	233	1	0
8	0	0	2	0	0	0	0	0	237	1
9	2	1	0	0	0	0	1	0	6	230

Table 6.11: Numeral dataset 1 : Confusion matrix of fold 5 for Model-4.

	LeNet-5	Model-1	Model-2	Model-3	Model-4
Fold 1	95.17%	96.73%	97.08%	96.47%	97.04%
Fold 2	95.69%	97.13%	97.52%	97.00%	97.65%
Fold 3	95.73%	97.13%	97.69%	97.34%	97.56%
Fold 4	96.78%	97.26%	97.60%	97.39%	97.56%
Fold 4	96.60%	97.30%	97.56%	97.52%	98.26%
Average Accuracy	95.99%	97.11%	97.49%	97.14%	97.61%

Table 6.12: Numeral dataset 2 : Accuracy result of five fold cross validation for fold 1 training images for all CNN models.

	LeNet-5	Model-1	Model-2	Model-3	Model-4
Fold 1	96.26%	96.65%	97.21%	97.08%	97.43%
Fold 2	95.30%	96.34%	97.30%	96.60%	96.73%
Fold 3	95.86%	97.00%	97.69%	97.13%	97.47%
Fold 4	96.34%	96.68%	96.60%	97.17%	97.86%
Fold 5	96.43%	97.69%	97.69%	97.52%	97.69%
Average Accuracy	96.03%	96.90%	97.49%	97.10%	97.43%

Table 6.13: Numeral dataset 2 : Accuracy result of five fold cross validation for fold 2 training images for all CNN models.

6.2 5 fold cross validation - Numeral dataset 2

As described earlier numeral dataset 2 contain 14000 images means for each numeral there is 1400 images. Folds and cases are generated as describe for numeral dataset 1. In each fold 11200 images for training and 2800 images for testing. Now to know which CNN model performs better to train the folds, training part of all folds are further divided into five fold, for example for Fold 1 if training images are divided into folds, in each case 8960 images for training and 2240 images for testing and the procedure is same as with whole dataset for a particular fold case. And this folds are trained and tested for all five CNN models, to get best model from all five and train with this model and get accuracy for numeral dataset 2. These accuracy results are shown in table 6.12 to 6.16.

Here, from all this accuracy results, Model-4 works better than other models. So, now this Model-4 is used for training five fold cases of numeral dataset 1 and average accuracy is calculated. Also this test performs for LeNet-5 for comparison purpose and generated results are shown in table 6.17. And confusion matrix for all folds are also shown in table 6.18 to 6.22.

	LeNet-5	Model-1	Model-2	Model-3	Model-4
Fold 1	95.78%	97.00%	97.86%	97.13%	97.73%
Fold 2	95.95%	96.91%	97.21%	97.30%	97.56%
Fold 3	96.08%	97.04%	97.47%	97.26%	97.65%
Fold 4	95.56%	97.30%	97.69%	97.47%	97.82%
Fold 5	96.13%	96.30%	97.82%	97.43%	97.26%
Average Accuracy	95.90%	96.91%	97.61%	96.71%	97.60%

Table 6.14: Numeral dataset 2 : Accuracy result of five fold cross validation for fold 3 training images for all CNN models.

	LeNet-5	Model-1	Model-2	Model-3	Model-4
Fold 1	96.26%	97.13%	97.56%	97.21%	97.60%
Fold 2	96.34%	97.17%	97.60%	96.69%	97.69%
Fold 3	96.78%	97.04%	97.43%	97.08%	97.73%
Fold 4	95.65%	96.82%	97.47%	97.26%	97.69%
Fold 5	95.73%	97.34%	98.43%	97.69%	98.43%
Average Accuracy	96.15%	97.10%	97.69%	97.18%	97.82%

Table 6.15: Numeral dataset 2 : Accuracy result of five fold cross validation for fold 4 training images for all CNN models.

	LeNet-5	Model-1	Model-2	Model-3	Model-4
Fold 1	95.95%	97.08%	97.56%	97.17%	97.73%
Fold 2	95.65%	97.34%	96.60%	97.26%	97.73%
Fold 3	96.34%	97.26%	97.39%	97.17%	97.56%
Fold 4	96.65%	96.91%	97.26%	97.39%	98.00%
Fold 5	95.78%	97.56%	97.73%	97.52%	97.78%
Average Accuracy	96.07%	97.23%	97.50%	96.30%	97.76%

Table 6.16: Numeral dataset 2 : Accuracy result of five fold cross validation for fold 5 training images for all CNN models.

	LeNet-5	Model-4
Fold 1	96.07%	97.75%
Fold 2	96.60%	97.89%
Fold 3	95.89%	97.75%
Fold 4	96.14%	97.60%
Fold+ 5	95.60%	97.71%
Average Accuracy	96.18%	97.74%

Table 6.17: Numeral dataset 2 : Accuracy result of five fold cross validation for LeNet-5 and Model-4.

	0	1	2	3	4	5	6	7	8	9
0	273	0	0	0	1	0	0	4	2	0
1	0	271	4	0	0	2	2	0	0	1
2	0	3	270	0	0	0	1	1	4	1
3	0	0	1	274	0	0	2	3	0	0
4	1	0	1	0	275	1	2	0	0	0
5	0	0	0	0	0	278	1	1	0	0
6	0	0	0	0	1	3	268	8	0	0
7	0	0	0	2	0	0	0	278	0	0
8	0	0	0	0	2	0	0	0	277	1
9	1	0	0	0	1	0	1	0	7	270

Table 6.18: Numeral dataset 2 : Confusion matrix of fold 5 for Model-4.

	0	1	2	3	4	5	6	7	8	9
0	278	0	0	0	0	0	1	0	1	0
1	0	277	2	0	0	1	0	0	0	0
2	0	6	271	1	2	0	0	0	0	0
3	0	0	1	269	0	0	3	7	0	0
4	0	2	0	1	274	1	1	0	1	0
5	0	2	0	0	4	272	2	0	0	0
6	0	0	0	1	0	0	278	1	0	0
7	1	0	0	3	0	0	6	270	0	0
8	1	1	0	0	0	0	0	0	277	1
9	1	0	0	0	1	0	0	1	1	276

Table 6.19: Numeral dataset 2 : Confusion matrix of fold 5 for Model-4.

	0	1	2	3	4	5	6	7	8	9
0	274	0	0	0	0	0	1	2	3	0
1	0	272	6	0	1	0	0	0	0	1
2	0	5	271	1	0	1	1	0	1	0
3	0	0	0	272	0	0	3	5	0	0
4	0	0	0	0	274	1	2	3	0	0
5	0	1	2	1	1	275	0	0	0	0
6	0	0	1	1	2	0	269	7	0	0
7	1	0	0	5	0	1	1	272	0	0
8	0	1	0	0	0	0	0	0	278	1
9	0	0	0	0	0	0	0	0	1	279

Table 6.20: Numeral dataset 2 : Confusion matrix of fold 5 for Model-4.

	0	1	2	3	4	5	6	7	8	9
0	268	0	1	0	0	0	0	4	5	2
1	0	276	1	0	1	0	0	0	0	2
2	0	7	270	0	0	1	0	1	1	0
3	0	0	1	277	0	0	1	1	0	0
4	0	1	0	1	278	0	0	0	0	0
5	0	0	1	0	0	277	2	0	0	0
6	1	2	1	1	3	0	265	6	0	1
7	5	0	0	2	0	0	7	265	0	1
8	0	0	1	0	0	0	0	0	278	1
9	0	0	0	0	0	0	0	0	3	277

Table 6.21: Numeral dataset 2 : Confusion matrix of fold 5 for Model-4.

	0	1	2	3	4	5	6	7	8	9
0	276	0	1	1	0	0	0	0	1	1
1	0	276	2	0	0	0	0	1	1	0
2	0	5	272	1	0	1	0	1	0	0
3	0	0	1	271	0	0	5	3	0	0
4	0	0	1	0	279	0	0	0	0	0
5	0	2	1	0	0	277	0	0	0	0
6	0	1	0	2	2	2	269	4	0	0
7	5	0	2	3	1	3	0	264	0	2
8	5	0	0	0	0	0	0	0	274	1
9	0	0	0	0	0	0	0	0	2	278

Table 6.22: Numeral dataset 2 : Confusion matrix of fold 5 for Model-4.

	LeNet-5	Model-1	Model-2	Model-3	Model-4
Fold 1	95.62%	96.58%	97.20%	96.70%	97.54%
Fold 2	97.12%	97.41%	97.05%	97.87%	97.87%
Fold 3	96.25%	96.70%	97.54%	97.16%	97.79%
Fold 4	94.95%	96.20%	97.12%	96.66%	97.25%
Fold 5	95.04%	96.66%	97.20%	97.20%	96.91%
Average Accuracy	95.79%	96.71%	97.22%	97.11%	97.47%

Table 6.23: Numeral dataset 1 : Accuracy result of five fold cross validation for all CNN models.

	0	1	2	3	4	5	6	7	8	9
0	1378	0	3	1	0	0	0	5	7	6
1	0	1366	10	1	5	4	2	0	6	6
2	0	55	1296	1	9	11	1	2	21	4
3	0	0	7	1329	5	3	24	32	0	0
4	3	7	0	2	1377	3	2	0	6	0
5	0	5	7	0	5	1378	4	0	0	1
6	1	12	0	9	13	12	1234	116	0	3
7	17	4	2	22	2	6	20	1325	0	2
8	5	3	0	0	2	0	0	0	1383	7
9	0	3	0	0	0	0	1	2	28	1366

Table 6.24: Confusion matrix of numeral dataset 2 tested on trained Model-4 with numeral dataset 1.

6.3 Numeral dataset 1 for training and dataset 2 for testing

In this case whole numeral dataset 1 of 12000 is used for training and dataset 2 of 14000 images is used for testing. Here, in this case model is trained with whole dataset 1, thus first it is divided into five folds and with these folds all five models are trained and tested, generated results are shown in table 6.23. Model-4 gives best accuracy compared to other model, so it is used for train with whole dataset 1. And this trained model is tested using dataset 2. Accuracy obtained with this experiment is 95.94% with Model-2 and 93.36% with LeNet-5. Confusion matrix for this result is shown in table 6.24.

	LeNet-5	Model-1	Model-2	Model-3	Model-4
Fold 1	96.07%	96.71%	97.64%	97.07%	97.75%
Fold 2	96.60%	97.35%	97.92%	97.89%	97.89%
Fold 3	95.89%	97.50%	97.92%	97.39%	97.75%
Fold 4	96.14%	97.07%	97.67%	97.57%	97.60%
Fold 5	95.60%	97.25%	97.67%	97.03%	96.71%
Average Accuracy	96.18%	97.17%	97.76%	97.39%	97.74%

Table 6.25: Numeral dataset 2 : Accuracy result of five fold cross validation for all CNN models.

	0	1	2	3	4	5	6	7	8	9
0	1174	0	0	2	1	0	0	13	7	3
1	0	1151	17	1	2	12	8	4	5	0
2	0	25	1157	2	1	5	1	6	1	2
3	0	1	6	1162	0	1	16	14	0	0
4	2	5	12	4	1129	27	13	1	5	2
5	2	29	9	1	8	1143	4	2	0	2
6	1	6	3	33	1	5	1128	14	3	6
7	12	10	8	4	0	5	24	1131	5	1
8	3	2	3	2	4	0	0	0	1182	4
9	5	0	5	1	6	3	3	4	20	1153

Table 6.26: Confusion matrix of numeral dataset 1 tested on trained Model-2 with numeral dataset 2.

6.4 Numeral dataset 2 for training and dataset 1 for testing

In this case whole numeral dataset 2 of 14000 is used for training and dataset 1 of 12000 images is used for testing. Here, in this case model is trained with whole dataset 2, thus first it is divided into five folds and with these folds all five models are trained and tested, generated results are shown in table 6.25. Model-2 gives best accuracy compared to other model, so it is used for train with whole dataset 2. And this trained model is tested using dataset 1. Accuracy obtained with this experiment is 95.91% with Model-4 and 93.72% with LeNet-5. Confusion matrix for this result is shown in table 6.26.

6.5 5 fold cross validation - Character dataset

As described earlier character dataset contain 88751 images. Folds and cases are generated as describe for numeral dataset 1. Now to know which CNN model performs better

	LeNet-5	Model-1	Model-2	Model-3	Model-4
Fold 1	92.31%	95.66%	96.47%	96.21%	96.86%
Fold 2	92.23%	95.43%	96.54%	95.95%	96.74%
Fold 3	92.12%	95.64%	96.52%	95.73%	96.65%
Fold 4	92.09%	95.42%	96.54%	95.85%	96.75%
Fold 5	92.31%	95.46%	96.43%	95.89%	96.71%
Average Accuracy	92.21%	95.52%	96.50%	95.92%	96.74%

Table 6.27: character dataset : Accuracy result of five fold cross validation for fold 1 training images for all CNN models.

	LeNet-5	Model-1	Model-2	Model-3	Model-4
Fold 1	92.19%	95.33%	96.69%	95.80%	96.78%
Fold 2	92.53%	95.45%	96.53%	95.76%	95.71%
Fold 3	92.47%	95.44%	95.60%	96.05%	96.54%
Fold 4	92.48%	95.57%	96.56%	95.97%	96.93%
Fold 5	92.35%	95.60%	96.45%	95.97%	96.60%
Average Accuracy	92.40%	95.47%	96.56%	95.91%	96.71%

Table 6.28: character dataset : Accuracy result of five fold cross validation for fold 2 training images for all CNN models.

to train the folds, training part of all folds are further divided into five fold, for example for Fold 1 if training images are divided into folds, and the procedure is same as with whole dataset for a particular fold case. And this folds are trained and tested for all five CNN models, to get best model from all five and train with this model and get accuracy for character dataset. Model-4 gives more accuracy than other models. This model is used for five fold cross validation and accuracy obtained from this is 97.09%. And it is also train and tested using LeNet-5 for comparision and its accuracy is 92.73%.

Here, from all this accuracy results, Model-4 works better than other models. So, now

	LeNet-5	Model-1	Model-2	Model-3	Model-4
Fold 1	92.82%	95.56%	96.64%	95.83%	96.65%
Fold 2	92.44%	95.51%	96.75%	95.97%	96.78%
Fold 3	92.54%	95.54%	96.73%	95.95%	96.71%
Fold 4	92.15%	95.54%	96.59%	96.03%	96.66%
Fold 5	92.16%	95.43%	96.49%	95.52%	96.48%
Average Accuracy	92.42%	95.51%	96.64%	95.86%	96.65%

Table 6.29: character dataset : Accuracy result of five fold cross validation for fold 3 training images for all CNN models.

	LeNet-5	Model-1	Model-2	Model-3	Model-4
Fold 1	92.30%	95.76%	96.76%	96.14%	96.86%
Fold 2	92.61%	95.78%	96.55%	96.09%	96.68%
Fold 3	91.89%	95.38%	96.39%	95.75%	96.62%
Fold 4	92.64%	95.55%	96.47%	95.65%	96.89%
Fold 5	92.64%	95.86%	96.73%	95.59%	96.81%
Average Accuracy	92.41%	95.67%	96.58%	95.84%	97.82%

Table 6.30: character dataset : Accuracy result of five fold cross validation for fold 4 training images for all CNN models.

	LeNet-5	Model-1	Model-2	Model-3	Model-4
Fold 1	91.74%	95.35%	96.18%	95.73%	96.64%
Fold 2	92.44%	95.43%	96.53%	96.04%	96.88%
Fold 3	91.78%	95.34%	96.52%	95.88%	96.78%
Fold 4	92.50%	95.72%	96.58%	95.80%	96.58%
Fold 5	92.59%	95.74%	96.70%	96.18%	96.85%
Average Accuracy	92.21%	95.51%	96.50%	95.92%	96.74%

Table 6.31: character dataset : Accuracy result of five fold cross validation for fold 5 training images for all CNN models.

this Model-4 is used for training five fold cases of character dataset and average accuracy is calculated. Also this test performs for LeNet-5 for comparison purpose and generated results are shown in table 6.32.

6.6 MNIST dataset for training and testing

MNIST contains 60000 images for training and 10000 images for testing. Now for comparison purpose, all five models are trained and tested using MNIST. First training images of dataset is divided into five fold and with this folds, five fold cross validation is performed

	LeNet-5	Model-4
Fold 1	92.87%	97.09%
Fold 2	93.11%	97.07%
Fold 3	92.43%	97.10%
Fold 4	92.50%	96.97%
Fold 5	92.78%	97.24%
Average Accuracy	92.73%	97.09%

Table 6.32: character dataset : Accuracy result of five fold cross validation for LeNet-5 and Model-4.

	LeNet-5	Model-1	Model-2	Model-3	Model-4
Fold 1	98.91%	98.90%	98.96%	98.98%	98.99%
Fold 2	98.77%	98.89%	99.09%	99.10%	99.11%
Fold 3	98.92%	99.07%	99.01%	99.14%	98.99%
Fold 4	98.79%	98.91%	98.90%	98.98%	98.87%
Fold 5	98.94%	99.18%	99.09%	99.15%	99.13%
Average Accuracy	98.86%	98.99%	99.01%	99.07%	99.01%

Table 6.33: MNIST dataset : Accuracy result of five fold cross validation for all CNN models.

for all models. For this dataset model-3 gives accuracy in five fold cross validation of training set. So, with Model-3 and LeNet-5 are used for training and accuracy results are generated with test set of MNIST. Accuracy obtained with Model-3 is 99.25% and with LeNet-5 is 99.14%.

Chapter 7

Conclusion

The thesis addresses the problem of handwritten Gujarati character recognition through deep learning techniques. Experiments are carried out on two large numeral datasets and one character dataset. LeNet5 and 4 other deep neural networks are employed for the task. All the models perform well but the best performance is achieved through ILeNet.

Bibliography

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[2] "Mnist handwritten dataset." http://yann.lecun.com/exdb/mnist/.

[3] L. Heutte, T. Paquet, J.-V. Moreau, Y. Lecourtier, and C. Olivier, "A structural/statistical feature based vector for handwritten character recognition," *Pattern recognition letters*, vol. 19, no. 7, pp. 629–641, 1998.

[4] I. K. Sethi and B. Chatterjee, "Machine recognition of constrained hand printed devanagari," *Pattern recognition*, vol. 9, no. 2, pp. 69–75, 1977.

[5] A. A. Desai, "Gujarati handwritten numeral optical character reorganization through neural network," *Pattern recognition*, vol. 43, no. 7, pp. 2582–2589, 2010.

[6] J. R. Prasad, U. Kulkarni, and R. S. Prasad, "Offline handwritten character recognition of gujrati script using pattern matching," in *Anti-counterfeiting, Security, and Identification in Communication, 2009. ASID 2009. 3rd International Conference on*, pp. 611–615, IEEE, 2009.

[7] "Caffe." http://caffe.berkeleyvision.org/.

[8] "Caffe python interface." http://caffe.berkeleyvision.org/tutorial/interfaces.html.