A Robust Approach to Handwritten Digit Classification in Bangla and Hindi using Convolutional Neural Networks

Report submitted in partial fulfillment of the requirement for the award of the degree of

Bachelor of Computer Science and Engineering

In the Faculty of Engineering and Technology

Jadavpur University

By

JISHNU MUKHOTI

Roll No **: 001210501038**
Registration No **: 119621 of 2012 - 13**

Under the Guidance of

Dr. Ram Sarkar

Department of Computer Science and Engineering

Jadavpur University

Kolkata - 700 032

2016

ACKNOWLEDGEMENTS

I am extremely grateful to my project guide, **Dr. Ram Sarkar** for his constant guidance, prompt help, support and unwavering encouragement which helped me a lot in completing the project. Under his tutelage, I have been able to learn many new things on my project topic, including an in-depth understanding of deep learning and convolutional neural networks, for which I am forever thankful to him.

I am also grateful to **Dr. Debesh Das,** Head of the Department of Computer Science and Engineering, Jadavpur University for allowing me to carry out my research in the department.

I express my gratitude to my friend **Sukanya Dutta** for all the help and support which she has given me in the research that we have carried out together.

Finally, I am extremely thankful to my parents and all the members of my family who constantly encouraged and pushed me on in all my research endeavors.

Date :

Place : Kolkata

JISHNU MUKHOTI

B.C.S.E (C.S.E)

Roll No: 001210501038

Abstract

Optical Character Recognition (OCR) is a process by which digitized images of handwritten or printed text can be converted into machine-readable and editable text. There are several sub-domains in this field including handwritten word recognition, script or language recognition, digit recognition and others. Digit classification has many important applications in real life scenarios. For instance, it might often be required to automatically and intelligently note the car plate number of rashly driven vehicles. Hence, considering the relevance of the topic, in the present work, I have proposed a solution to the problem of handwritten digit classification.

The primary challenge in handwritten digit recognition is to correctly classify digits which are highly varied in their visual characteristics primarily due to the writing styles of different individuals. In the present work, I have proposed the use of Convolutional Neural Networks (CNN) for the purpose of classifying handwritten Bangla and Hindi numerals. The major advantage of using a CNN based classifier is that no prior hand-crafted feature needs to be extracted from the images for efficient and accurate classification. An added benefit of a CNN classifier is that it provides complete translational invariance and a certain extent of rotational invariance during recognition. Applications can be found in real-time OCR systems where input images are often not perfectly oriented along a vertical axis. In this work, I have used modified versions of the well-known LeNet CNN architecture. Extensive experiments have revealed a best-case classification accuracy of 98.2% for Bangla and 98.8% for Hindi numerals outperforming competitive models in the literature.

Contents

List of Figures:

List of Tables:

Chapter 1

Introduction

Optical Character Recognition (OCR) is the process by which images of handwritten or machine-printed text are electronically encoded into machine-readable form. Applications of OCR systems include the conversion of text available in different non-editable formats into a form which can be easily manipulated by general purpose text editors. Without OCR systems, the conversion of such non-editable formats would have to be carried out manually, a time consuming and laborious job. One of the challenging tasks of OCR is the recognition of handwritten characters [1]. Much of the most challenging material requiring recognition is obtained from handwritten sources like forms, bank cheques, handwritten mailing addresses and others. The main difficulty is dealing with a variety of writing styles of different individuals. Even a single individual can write in different styles depending on the environment or task. Hence, correct recognition of a character amidst thousands of such handwritten styles requires an OCR system with robust and agile algorithms, which in turn introduces the necessity for innovation, creativity and development in this field, thus making it interesting to the researchers.

In the present work, we address handwritten digit recognition, a special case of the general handwritten character classification problem. Classification necessitates the extraction of features which are optimized to recognize the discriminatory patterns that separate one image class from others. At the same time, these extracted features should be less sensitive to extraneous distortions that may produce visual differences between images belonging to the same pattern class. The primitive set of features is the set of pixel intensities describing an image, where each pixel in a size-normalized image is considered as a feature, hence, the number of features is equal to the number of pixels in the image. This approach, however, has a number of disadvantages. One is that an image can contain thousands of pixels giving rise to a very high dimensional feature space. Construction of optimal decision boundaries in such a feature space requires a large number of training samples for any pattern classifier. Furthermore, the discriminatory information required for differentiating between image classes

is often found in spatially localized regions of the image. The approach of training a classifier with pixel intensities ignores these localized pattern differences and is relatively inefficient. An approach to deal with this problem is to extract features from an image which can effectively convey the differences in spatially localized patterns while simultaneously reducing the dimensionality of the feature-space. Several innovative ideas [2]-[12] have been proposed in this direction.

Cao et al. [2] proposed a zone-based feature extraction method using directional histograms and multiple neural classifiers for the recognition of handwritten Roman numerals. In their work, the use of more than one expert system has been suggested for better classification accuracy. Park et al. [3] presented a classification mechanism using a hierarchical feature space and tested it on the NIST digit database. Given an input image of a character, the initial step involves extraction of coarse-resolution features from the image. The image is then divided into sub-images and the same procedure is recursively carried out at finer levels of resolution. The recursion ends when the given classification criterion is satisfied. The sub-images are chosen adaptively based on the level of discriminatory information that they contain.

Basu et al. [4] proposed a method to classify handwritten Bangla numerals by dividing the image into a fixed number of equal sized regions. A combination of shadow features, centroid features and longest-run features is then extracted from the image to train a Multi-Layer Perceptron (MLP) for distinguishing pattern classes. Another work is presented by Rajashekhararadhya et al. [5], where a given image of a handwritten character is divided into zones and the extracted features are based on image centroids as well as zone centroids. For each zone in the image, the mean distance between the image centroid and every pixel in the zone is computed. The same metric is also computed with respect to the zone centroid. In this way, the authors extract 2n features where n is the total number of zones. These features are then used to train nearest neighbor classifiers and feed-forward neural networks for the classification of handwritten numerals in Kannada, Telugu, Tamil and Malayalam scripts. Other notable works in this field include the application of genetic algorithms [6], [7], fuzzy-geno approaches [8], quad-tree based longest run feature extractors [9], MLP based classifiers [10], [11], Support Vector Machines (SVM) [12], [28], script based classifiers [29] and others.

It should be noted that all the works mentioned above focus primarily on features extracted from an image based on human intuition. The performance of such hand-crafted heuristics is, thus, heavily dependent on how well they have been designed to capture the shape characteristics. Moreover, the feature extractors are often optimized to solve a particular problem. This problem-specific approach to manually designing features leads to a loss of generality in the models. In the present work, we use a method which can integrate the automated extraction of features from a given image with its subsequent classification, convolutional neural networks (CNNs) [13]. Their architecture can be specially tailored for the task of 2D image recognition.

CNNs represent an image as an agglomeration of sub-patterns of lower dimensions. These sub-patterns are the learnable features for a CNN and the location of such features with respect to one another is used as the discriminatory information for distinguishing pattern classes. An additional benefit of convolutional networks is their tolerance for geometric transformations including translation and rotation that may affect the classification result. In other words, CNNs can effectively recognize an object in a 2D image even when the object is shifted from the center of the image or rotated by a relatively small angle about its center. These advantages have made CNNs, the classifier of choice for our study. In fact, CNN based classifiers have been applied to various problems including 2D face detection [14], handwritten character recognition [15], time-series modeling and speech recognition [16] and others [17]-[20]. In this paper, we apply convolutional networks for classification of handwritten numerals in Bangla and Hindi, two of the most popular languages in India.

Extensive experiments have been carried out to test three areas of performance for CNN based classifiers: a) the dependence of classification accuracy on the size of the training set, b) the changes in classification accuracy obtained by applying cross-validation schemes for training and c) the effect of inducing rotated images in the training set in order to achieve rotational invariance for large angles of rotation. In the experiments, we use customized versions of the well-known LeNet [15] architecture. The results of our experiments reveal a best-case classification accuracy of 98.2% for Bangla and 98.8% for Hindi numerals, outperforming competitive models by a significant margin. With the high classification performance, the experiments indicate the applicability of CNNs for the task of handwritten digit recognition in Bangla and Hindi.

The remainder of this report is organized as follows. Chapter 2 deals with an in-depth description of Convolutional Neural Networks along with specific details on the LeNet-5 architecture. Chapter 3 discusses Caffe, a deep learning framework used for both research as well as industrial purposes. In this chapter, I also present a few illustrative samples of code for the reader's better understanding. In Chapter 4, the experiments that have been carried out in the research have been detailed along with the results. Chapter 5 presents the concluding thoughts on the topic.

Chapter 2

Convolutional Neural Network

The working principles of a CNN [13] are primarily two-fold. Firstly, an image is viewed as a composite collection of patterns or sub-images. The network identifies such patterns as visual features in a hierarchical bottom-up fashion depending on the network's depth. The lower order features detected in the initial hidden layers of a CNN include elementary patterns like edges and corners. These features are further combined to form higher order features which are identified at deeper layers of the network. Finally, the location of each detected feature with respect to other features is used by the network to differentiate among pattern classes.

2.1 Overview

A CNN is a deep learning [21] architecture specialized for the task of image recognition and classification. The input instances for a CNN include the pixel intensities of an image arranged in the form of a 2D array. Each input unit corresponds to a single pixel intensity in the image. Multiple such input planes may be required for color images (such as RGB images). The output of the CNN is a vector $\vec{V}$ of length k as given below

$$\vec{V} = [c_1, c_2, ..., c_k] \tag{1}$$

where k is the number of pattern classes and the element c_i indicates the confidence of the given image belonging to the ith pattern class. There are three main types of layers in a typical CNN architecture which map the input to the output: i) the convolution layer, ii) the pooling or sub-sampling layer and iii) the fully connected layer.

The convolution layer is responsible for the detection and extraction of visual features from spatially localized neighborhoods of the image. Let a grayscale image of dimensions $m \times n$ be represented as an $m \times n$ matrix $\mathbf{I}$ of real numbers, where the element $\mathbf{I}_{i,j}$ corresponds to the pixel intensity of the (i,j)th pixel in the image at the ith row and jth column. For the sake of

simplicity, we assume the image to be grayscale so that a single matrix of pixel intensities suffices to describe the image. Each element in the matrix $\mathbf{I}$ is an input for the CNN. In order to detect localized features, the image is first partitioned into a number of small partially overlapping windows. These windows are also known as **local receptive fields** in the literature. Let the dimensions of each window be $w \times w$ where $w << m, n$ and let the window be indexed according to the location of its top left pixel. Thus, a window $\mathbf{W}_{p,q}$ corresponds to a square region of pixels with $\mathbf{I}_{p,q}$ at the top left and $\mathbf{I}_{(p+w-1),(q+w-1)}$ at the bottom right. It should also be noted that the row-wise adjacent windows of $\mathbf{W}_{p,q}$ are $\mathbf{W}_{p,q-1}$ (left) and $\mathbf{W}_{p,q+1}$ (right) whereas the column-wise adjacent windows are $\mathbf{W}_{p-1,q}$ (above) and $\mathbf{W}_{p+1,q}$ (below). This indicates that two consecutive windows represent local regions of the image which are overlapping except for two rows or two columns of pixels.

The job of the convolution layer is to recognize the presence of one or more visual patterns or features in each local receptive field of the image i.e., from $\mathbf{W}_{1,1}$ to $\mathbf{W}_{(m-w+1),(n-w+1)}$. Clearly, the detection of a single feature, for example feature A, in a given window $\mathbf{W}_{i,j}$ is in itself, a full-fledged computational task. Hence, it is sensible for the convolution layer to allocate a hidden neuron for identifying feature A in the window $\mathbf{W}_{i,j}$. However, the same feature may appear in other local regions or windows of the image as well. Each such window should be allotted its own feature A detector. This calls for a set of hidden neurons, one for each window, performing the task of detecting the presence of feature A in their respective windows. The replication of the same operation over a set of multiple neurons requires each of them to possess an identical set of $w \times w$ trainable weights and a single trainable bias. Such a set of neurons in the convolution layer is called a feature map. Let the output of the convolution neuron connected to the window $\mathbf{W}_{p,q}$ be $c_{p,q}$ as given below

$$c_{p,q} = \rho\left(\left(\sum_{i=1}^{w}\sum_{j=1}^{w} a_{i,j}\mathbf{I}_{(p+i-1),(q+j-1)}\right)+b\right) \tag{2}$$

where w represents the dimensionality of the window $\mathbf{W}_{p,q}$, $a_{i,j}$ is the weight corresponding to the (i,j)th pixel intensity $\mathbf{I}_{(p+i-1),(q+j-1)}$ in the window $\mathbf{W}_{p,q}$, b denotes the bias and ρ is the activation function of the neuron. Common candidates for the activation function include the sigmoid and the hyperbolic tangent functions. It is worth noting that in equation (2), the weights and the bias are independent of the location (p,q) of the window, indicating that they are replicated over all the neurons in a feature map. A generic convolution layer possesses multiple such feature maps, representing multiple learnable visual features where all hidden neurons in a feature map share the same set of weights and bias.

A **pooling or sub-sampling layer** is used to create compressed versions of the feature maps produced by the convolution layer. The objective of pooling is to lower the resolution of the feature maps by discarding the knowledge about the exact positions of features detected in an image. Lowering the dimension of the feature maps reduces the computational load for later layers along the network. This can be compensated for by increasing the number of feature maps in deeper convolution layers, providing a rich representation of visual features. Two common pooling strategies employed in our work are max pooling and the average pooling. Let a feature map from the convolution layer be represented as a matrix $\mathbf{C}$, where the element $c_{p,q}$ corresponding to the pth row and qth column is given by equation (2). The matrix $\mathbf{C}$ is divided into disjoint 2×2 regions and for each such region, a hidden neuron is allocated in the pooling layer, thereby downscaling the feature map $\mathbf{C}$ by a factor of 4 (both the width and the height are reduced by a factor of 2). In the max pooling scheme, the neuron outputs the maximum activation in the region and in the average pooling scheme, it outputs the mean activation in the region. Let a region $\mathbf{R}_{p,q}$ be represented by the values $c_{p,q}$, $c_{p,q+1}$, $c_{p+1,q}$ and $c_{p+1,q+1}$. The output of the max pooling and average pooling neuron corresponding to the region $\mathbf{R}_{p,q}$ a $Mx_{p,q}$ and $Av_{p,q}$ respectively as follows:

$$Mx_{p,q} = \max(c_{p,q}, c_{p,q+1}, c_{p+1,q}, c_{p+1,q+1}) \tag{3}$$

$$Av_{p,q} = \frac{(c_{p,q} + c_{p,q+1} + c_{p+1,q} + c_{p+1,q+1})}{4}. \tag{4}$$

With our pooling strategies, given that there are h feature maps in the convolution layer each with dimensions $l \times k$, the output of the pooling layer is a set of h compressed feature maps with dimensions $\frac{l}{2} \times \frac{k}{2}$. In Figure 1, the max and average pooling operations have been illustrated.

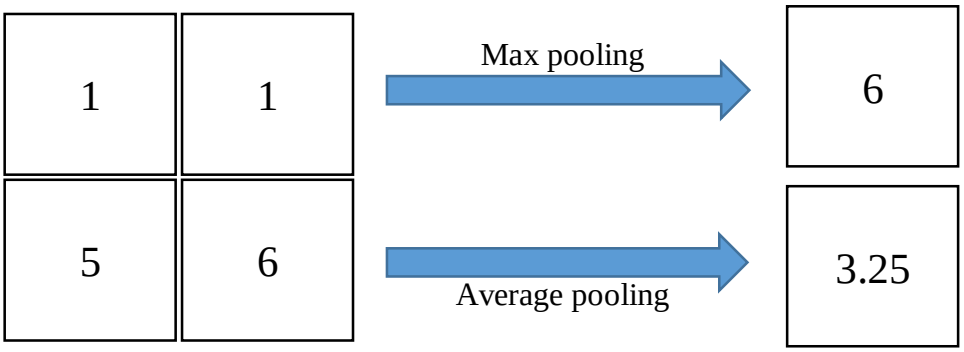

Fig. 1. An illustration of max and average pooling schemes

The final hidden layer in a typical CNN architecture is a **fully connected layer**. Each neuron of the previous layer (generally a pooling layer) is connected to all the neurons of a fully connected layer. The main purpose of this layer is to learn the relative positions of different sub-patterns or features with respect to one another that can serve as the discriminatory information to identify a pattern class. A schematic diagram of a simplified CNN architecture is illustrated in Figure 2. The input is a grayscale image of dimensions 10×10. The image is partitioned into 5×5 local receptive fields and connected to a convolution layer having two feature maps, A and B respectively. The feature maps are then condensed using a pooling layer which finally leads to a fully connected output layer having four neurons representative of four pattern classes.

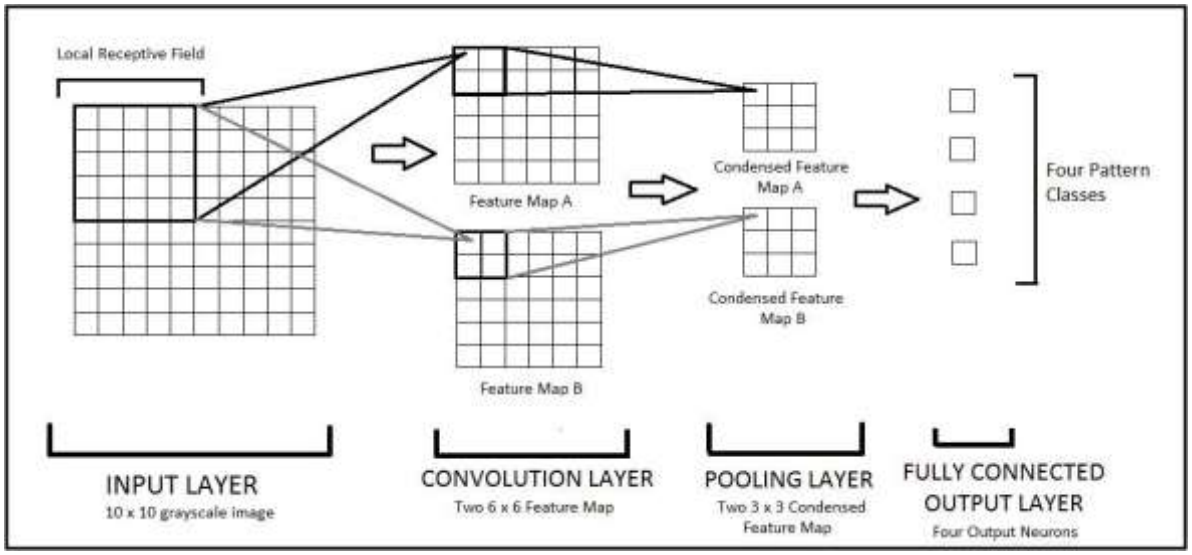

Fig. 2. A simple CNN architecture for solving a 4-class classification problem

2.2 Lenet-5 Architecture

In our work, we make use of the well-known LeNet [15] CNN architecture for all our experiments. The input is a 32 x 32 grayscale image with each input neuron representing a pixel intensity. There are 5 main hidden layers and a single output layer in the concerned CNN architecture. The structure of the LeNet CNN is illustrated in Figure 2. The first hidden layer C1, is a convolution layer mapping each 5×5 receptive field from the input layer to a single neuron in each feature map. There are a total of 20 feature maps in this layer, each having dimensions $28\times28.$ The convolution layer C1 is followed by the pooling layer P2 which is also the second hidden layer in the network. The layer P2 produces 20 condensed feature maps from the 20 feature maps in C1. This is done by connecting each disjoint 2x2 region in a convolution feature map to a single neuron in the corresponding condensed feature map, thereby downscaling the dimensions of each convolution feature map by a factor of 2. Naturally, the condensed feature maps have a dimension of 14x14. The pooling neuron produces either the maximum or the average of its four inputs depending on the pooling scheme used.

The hidden layer C3 is a convolution layer which maps 5x5 receptive fields in each of the condensed feature maps from layer P2 to a single neuron in a feature map. There are a total of 50 feature maps in this layer. The reason behind an increase in the number of feature maps is that the reduction of the dimensionality of the image facilitates a larger number of feature maps,

thereby leading to a comparatively richer representation of visual patterns or features in the image. It should be noted that a neuron in a feature map of layer C3 is connected to 20 receptive fields in the previous layer. Clearly, with 14x14 condensed feature maps in P2 being sampled using 5x5 receptive fields, the dimension of each feature map in C3 is 10x10. Each of the 50 feature maps in C3 is further compressed using a second pooling layer P4 to half of its dimensions (i.e., 5x5). In our experiments, we employ the same pooling scheme (either max pooling or average pooling) in both the pooling layers. Hence, the pooling layer P4 also uses either the max or average pooling method. The condensed feature maps of layer P4 are fully connected to 500 neurons in the fully connected layer F5. This layer leads up to the output layer containing 10 neurons corresponding to the 10 digits (pattern classes). The softmax loss function [22] is used for computing error during training and the well-known backpropagation algorithm [15] is used to train the weights and biases of the network.

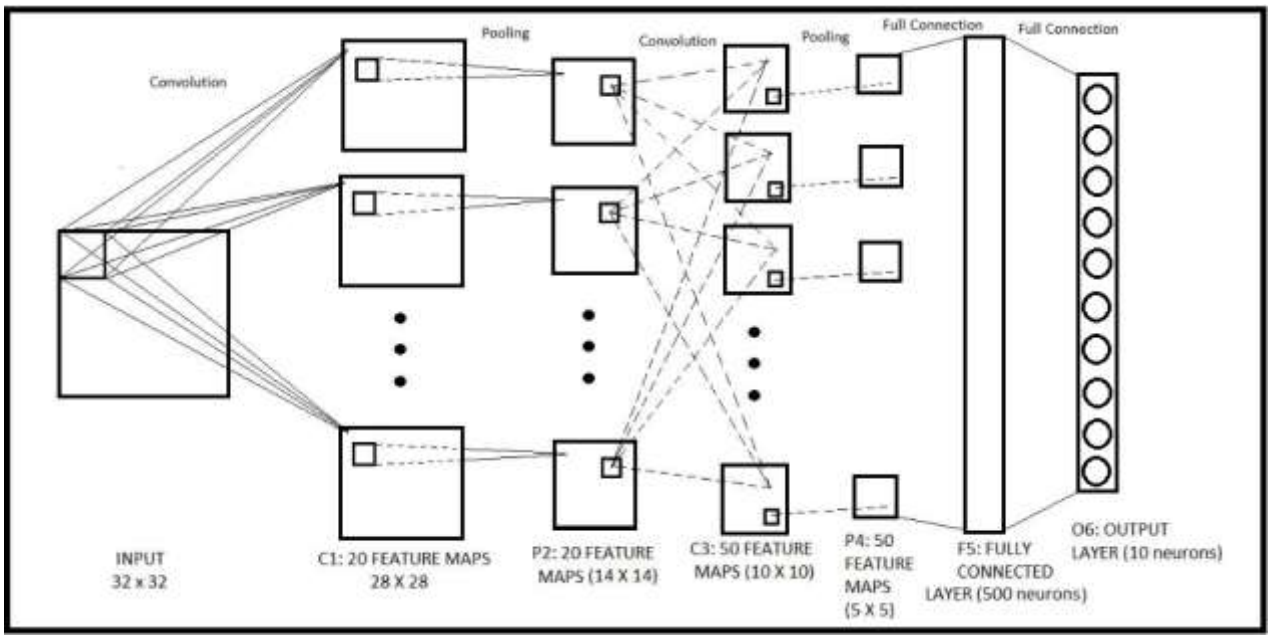

Fig. 3. Architecture of the LeNet CNN used in the experiments

Chapter 3

Caffe: A Deep Learning Framework

3.1 Overview

Caffe [27] is a deep learning framework made with expression, speed, and modularity in mind. It is developed by the Berkeley Vision and Learning Center (BVLC) and by community contributors. Caffe provides multimedia scientists and practitioners with a clean and modifiable framework for state-of-the-art deep learning algorithms and a collection of reference models. The framework is a BSD-licensed C++ library with Python and MATLAB bindings for training and deploying general-purpose convolutional neural networks and other deep models efficiently on commodity architectures. Caffe fits industry and internet-scale media needs by CUDA GPU computation, processing over 40 million images a day on a single K40 or Titan GPU (2.5 ms per image). By separating model representation from actual implementation, Caffe allows experimentation and seamless switching among platforms for ease of development and deployment from prototyping machines to cloud environments.

Caffe provides a complete toolkit for training, testing, tuning, and deploying models, with well-documented examples for all of these tasks. As such, it's an ideal starting point for researchers and other developers looking to jump into state-of-the-art machine learning. At the same time, it's likely the fastest available implementation of these algorithms, making it immediately useful for industrial deployment.

Modularity: The software is designed from the beginning to be as modular as possible, allowing easy extension to new data formats, network layers, and loss functions. Lots of layers and loss functions are already implemented, and plentiful examples show how these are composed into trainable recognition systems for various tasks.

Separation of representation and implementation: Caffe model definitions are written as config files using the Protocol Buffer language. Caffe supports network architectures in the form of arbitrary directed acyclic graphs. Upon instantiation, Caffe reserves exactly as much

memory as needed for the network, and abstracts from its underlying location in host or GPU. Switching between a CPU and GPU implementation is exactly one function call.

Test coverage: Every single module in Caffe has a test, and no new code is accepted into the project without corresponding tests. This allows rapid improvements and refactoring of the codebase, and imparts a welcome feeling of peacefulness to the researchers using the code. Python and MATLAB bindings. For rapid prototyping and interfacing with existing research code, Caffe provides Python and MATLAB bindings. Both languages may be used to construct networks and classify inputs. The Python bindings also expose the solver module for easy prototyping of new training procedures.

Pre-trained reference models: Caffe provides (for academic and non-commercial use|not BSD license) reference models for visual tasks, including the landmark \AlexNet ImageNet model with variations and the R-CNN detection model. More are scheduled for release. We are strong proponents of reproducible research: we hope that a common software substrate will foster quick progress in the search over network architectures and applications.

3.2 Installation

The steps to install Caffe on Ubuntu 14.04 without GPU (on Virtual Box) are given below:

Step 1: Download the latest version of Ubuntu. The following is a link for Kubuntu: http://cdimage.ubuntu.com/kubuntu/releases/trusty/release/kubuntu-14.04.1-desktop-amd64.iso.

Step 2: Install the Virtual machine in Virtual Box and install all system updates.

Step 3: Install build essentials: `sudo apt-get install build-essential`

Step 4: Install latest version of Kernel headers: `sudo apt-get install linux-headers-`uname -r``

Step 5: Install LATEST VBox Additions.

Step 6: Activate bidirectional keyboard in Virtual Box.

Step 7: Install curl: `sudo apt-get install curl`

Step 8: Download CUDA: `cd ~/Downloads/`

```
curl -
O  "http://developer.download.nvidia.com/compute/cuda/6_5/rel/installers/cuda_6.5.14_linux_64.run"
```

Step 9: Make the downloaded installer file runnable: `chmod +x cuda_6.5.14_linux_64.run`

Step 10: Run the CUDA installer: `sudo ./cuda_6.5.14_linux_64.run --kernel-source-path=/usr/src/linux-headers-`uname -r`/`

- Accept the EULA
- Do NOT install the graphics card drivers (since we are in a virtual machine)
- Install the toolkit (leave path at default)
- Install symbolic link
- Install samples (leave path at default)

Step 11: Update the library path:

```
1. echo 'export PATH=/usr/local/cuda/bin:$PATH' >> ~/.bashrc
2. echo 'export
   LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda/lib64:/usr/local/lib' >>
   ~/.bashrc
3. source ~/.bashrc
```

Step 12: Install dependencies:

```
1. sudo apt-get install -y libprotobuf-dev libleveldb-dev libsnappy-dev
   libopencv-dev libboost-all-dev libhdf5-serial-dev protobuf-compiler
   gfortran libjpeg62 libfreeimage-dev libatlas-base-dev git python-dev
   python-pip libgoogle-glog-dev libbz2-dev libxml2-dev libxslt-dev libffi-dev
   libssl-dev libgflags-dev liblmdb-dev python-yaml
2. sudo easy_install pillow
```

Step 13: Download Caffe:

```
1. cd ~
2. git clone https://github.com/BVLC/caffe.git
```

Step 14: Install python dependencies for Caffe:

```
1. cd caffe
2. cat python/requirements.txt | xargs -L 1 sudo pip install
```

Step 15: Add a couple of symbolic links:

```
1. sudo ln -s /usr/include/python2.7/ /usr/local/include/python2.7
2. sudo ln -s /usr/local/lib/python2.7/dist-packages/numpy/core/include/numpy/
   /usr/local/include/python2.7/numpy
```

Step 16: Create a Makefile.config from the example

```
1. cp Makefile.config.example Makefile.config
```

2. `nano Makefile.config`
3. Uncomment the line # CPU_ONLY := 1 (In a virtual machine we do not have access to the the GPU)
4. Under PYTHON_INCLUDE, replace /usr/lib/python2.7/dist-packages/numpy/core/include with /usr/local/lib/python2.7/dist-packages/numpy/core/include (i.e. add /local)

Step 17: Compile Caffe:

1. `make pycaffe`
2. `make all`
3. `make test`

Step 18: Download the ImageNet Caffe model and labels:

1. `./scripts/download_model_binary.py models/bvlc_reference_caffenet`
2. `./data/ilsvrc12/get_ilsvrc_aux.sh`

Step 19: Modify python/classify.py to add the --print_results option

- Compare https://github.com/jetpacapp/caffe/blob/master/python/classify.py (version from 2014-07-18) to the current version of classify.py in the official Caffe distributionhttps://github.com/BVLC/caffe/blob/master/python/classify.py

Step 20: Test your installation by running the ImageNet model on an image of a kitten:

1. `cd ~/caffe`
2. `python python/classify.py --print_results examples/images/cat.jpg foo`
3. Expected result: [('tabby', '0.27933'), ('tiger cat', '0.21915'), ('Egyptian cat', '0.16064'), ('lynx', '0.12844'), ('kit fox', '0.05155')]
4. Check https://github.com/BVLC/caffe/wiki/Ubuntu-14.04-VirtualBox-VM in case of any error in this step.

3.3 Illustrative Code

Blobs, Net and Layers: A Blob is a wrapper over the actual data being processed and passed along by Caffe, and also under the hood provides synchronization capability between the CPU and the GPU. Mathematically, a blob is an N-dimensional array stored in a C-contiguous fashion.

Caffe stores and communicates data using blobs. Blobs provide a unified memory interface holding data; e.g., batches of images, model parameters, and derivatives for optimization.

Blobs conceal the computational and mental overhead of mixed CPU/GPU operation by synchronizing from the CPU host to the GPU device as needed. Memory on the host and device is allocated on demand (lazily) for efficient memory usage.

The layer is the essence of a model and the fundamental unit of computation. Layers convolve filters, pool, take inner products, apply nonlinearities like rectified-linear and sigmoid and other element wise transformations, normalize, load data, and compute losses like softmax and hinge.

The net jointly defines a function and its gradient by composition and auto-differentiation. The net is a set of layers connected in a computation graph – a directed acyclic graph (DAG) to be exact. Caffe does all the bookkeeping for any DAG of layers to ensure correctness of the forward and backward passes. A typical net begins with a data layer that loads from disk and ends with a loss layer that computes the objective for a task such as classification or reconstruction.

The net is defined as a set of layers and their connections in a plaintext modeling language. A simple logistic regression classifier.

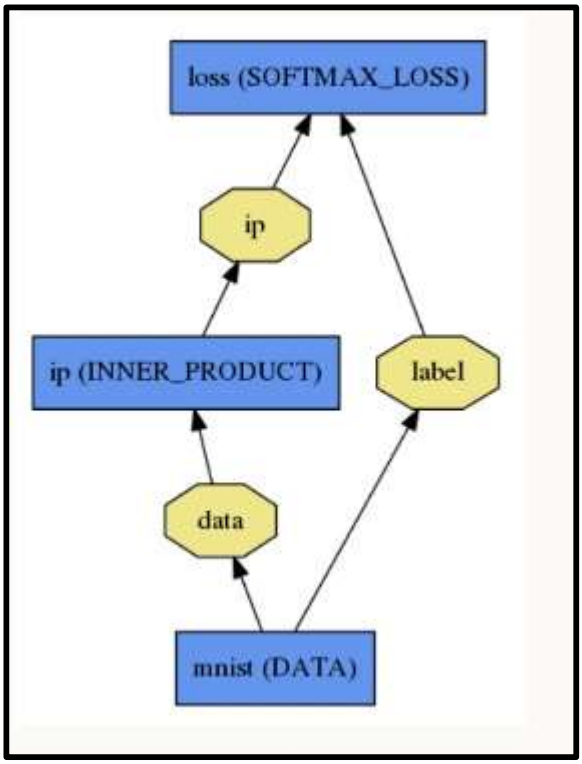

Fig. 4. A Caffe CNN

The above net is defined as follows:

```
name: "LogReg"
layer {
  name: "mnist"
  type: "Data"
  top: "data"
  top: "label"
  data_param {
    source: "input_leveldb"
    batch_size: 64
  }
}
layer {
  name: "ip"
  type: "InnerProduct"
  bottom: "data"
  top: "ip"
  inner_product_param {
    num_output: 2
  }
}
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "ip"
  bottom: "label"
  top: "loss"
}
```

Loss function definitions: The following piece of prototxt code uses the Softmax loss function as the chosen loss function in the net. There are quite a few other loss functions one may choose from.

```
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "pred"
  bottom: "label"
  top: "loss"
}

layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "pred"
  bottom: "label"
  top: "loss"
  loss_weight: 1
}
```

Training on the MNIST data:

Define the MNIST network.

```
name: "LeNet"
```

Defining the Data Layer:

```
layer {
  name: "mnist"
  type: "Data"
  transform_param {
    scale: 0.00390625
  }
  data_param {
    source: "mnist_train_lmdb"
    backend: LMDB
    batch_size: 64
  }
  top: "data"
  top: "label"
}
```

Defining the Convolutional Layer:

```
layer {
  name: "conv1"
  type: "Convolution"
  param { lr_mult: 1 }
  param { lr_mult: 2 }
  convolution_param {
    num_output: 20
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
  bottom: "data"
  top: "conv1"
}
```

Defining the Pooling Layer:

```
layer {
  name: "pool1"
  type: "Pooling"
  pooling_param {
    kernel_size: 2
    stride: 2
    pool: MAX
  }
  bottom: "conv1"
  top: "pool1"
```

```
}
```

Defining the Fully Connected Layer:

```
layer {
  name: "ip1"
  type: "InnerProduct"
  param { lr_mult: 1 }
  param { lr_mult: 2 }
  inner_product_param {
    num_output: 500
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
  bottom: "pool2"
  top: "ip1"
}
```

Defining the Inner Product Layer:

```
layer {
  name: "relu1"
  type: "ReLU"
  bottom: "ip1"
  top: "ip1"
}

layer {
  name: "ip2"
  type: "InnerProduct"
  param { lr_mult: 1 }
  param { lr_mult: 2 }
  inner_product_param {
    num_output: 10
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
  bottom: "ip1"
  top: "ip2"
}
```

Defining the Loss Layer:

```
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "ip2"
  bottom: "label"
}
```

Defining the LeNet Solver:

```
# The train/test net protocol buffer definition
net: "examples/mnist/lenet_train_test.prototxt"
# test_iter specifies how many forward passes the test should carry out.
# In the case of MNIST, we have test batch size 100 and 100 test iterations,
# covering the full 10,000 testing images.
test_iter: 100
# Carry out testing every 500 training iterations.
test_interval: 500
# The base learning rate, momentum and the weight decay of the network.
base_lr: 0.01
momentum: 0.9
weight_decay: 0.0005
# The learning rate policy
lr_policy: "inv"
gamma: 0.0001
power: 0.75
# Display every 100 iterations
display: 100
# The maximum number of iterations
max_iter: 10000
# snapshot intermediate results
snapshot: 5000
snapshot_prefix: "examples/mnist/lenet"
# solver mode: CPU or GPU
solver_mode: GPU
```

Training and testing the model:

```
cd $CAFFE_ROOT
./examples/mnist/train_lenet.sh
```

Chapter 4

Experiments and Results

Dataset 1: The dataset consists of 6000 images of Bangla handwritten digits. In order to construct this dataset, a larger database of 10,000 sample images was used, from which 600 images per digit were randomly selected. This larger database was created by the combined efforts of CVPR [23], ISI, Kolkata and CMATER [24], Jadavpur University. The dataset [25] has been made freely downloadable to the research community. Each sample in the dataset is a 32×32 grayscale image. A few sample images of this dataset are shown in Figure 5(a).

Dataset 2: The dataset [25] comprises 3000 bitmap images of Hindi handwritten numerals where each sample image has dimensions $32 \times 32.$ The process of construction of the dataset is similar to Dataset 1 and it is also freely available online. A few sample images of this dataset are shown in Figure 5(b).

Fig. 5. Sample handwritten numerals from datasets: (a) Dataset 1 (Bangla numerals), (b) Dataset 2 (Hindi numerals)

Experiment 1: The primary motivation behind the experiment detailed out in this section is to compare the relative performance of two well-known pooling schemes, Max pooling and Average pooling. The experiment also tests the variation of classification accuracy with the size of the training set. The following steps are involved for the experiment:

Step 1. A training set and a test set are constructed from the given dataset, where the size of the training set is varied from 500 to 5500 images for Dataset 1 and from 500 to 2500 images for Dataset 2. The size of the test set is kept constant at 500 images. A fixed number of images (i.e., 500) is kept aside for testing and a random sampling from the remaining images is used for training set construction.

Step 2. With each of the dataset configurations, the LeNet CNN is trained with two variations in its structure. In the first run, the network is configured with max pooling in both its pooling layers and in the second run, mean pooling or average pooling is employed in both the pooling layers. The results are summarized in Table 1 for Dataset 1 and in Table 2 for Dataset 2.

Table 1
Percentage accuracy for Max and Average pooling schemes for Dataset 1 (Best case accuracy values are given in bold)

Size of Training Set	Size of Test Set	Percentage Accuracy	
		Max Pooling	Average Pooling
500		84.4	82.4
1000		88.0	88.4
1500		91.4	91.0
2000		92.2	91.8
2500		92.4	91.4
3000	500	93.2	92.2
3500		95.4	93.8
4000		96.0	94.2
4500		95.6	94.4
5000		96.0	95.0
5500		96.6	**95.8**

Table 2
Percentage accuracy for Max and Average pooling schemes for Dataset 2 (Best case accuracy values are given in bold)

Size of Training Set	Size of Test Set	Percentage Accuracy	
		Max Pooling	Average Pooling
500		89.4	88.2
1000		93.6	94.2
1500	500	95.0	94.0
2000		95.2	94.8
2500		96.4	**95.4**

Experiment 2: In this experiment, a 10-fold cross-validation is performed on the datasets with the objective of improving classification accuracy. The max pooling scheme is employed for both the Bangla and Hindi scripts due to its relatively high classification accuracy as seen from the previous experiment. For this experiment, we carry out the following steps:

Step 1. The dataset is divided into a training set and a test set in a way so as to implement a 10-fold cross-validation. The size of the test set is set to a tenth the size of the entire dataset. The test set is shifted as a fixed size sliding-window along the dataset where two consecutive test sets chosen from the dataset are contiguous and disjoint. For Dataset 1, the training set consists of 5400 images and the test set contains 600 images. For Dataset 2, the training set comprises 2700 sample images and the test set has 300 images.

Step 2. For each of the above mentioned training configurations, the LeNet CNN is trained and the test phase classification accuracy is noted. The accuracies are summarized in Table 3 for Dataset 1 and Table 4 for Dataset 2.

Table 3
Percentage accuracy for 10-fold cross-validation on Dataset 1
(Best case accuracy values are given in bold)

Test #	Training Set	Test Set	Percentage accuracy
1	601-6000	1-600	96.2
2	1-600,1201-6000	601-1200	96.6
3	1-1200, 1801-6000	1201-1800	97.0
4	1-1800, 2401-6000	1801-2400	97.8
5	1-2400, 3001-6000	2401-3000	97.8
6	1-3000, 3601-6000	3001-3600	**98.2**
7	1-3600, 4201-6000	3601-4200	97.2
8	1-4200, 4801-6000	4201-4800	96.6
9	1-4800, 5401-6000	4801-5400	96.2
10	1-5400	5401-6000	96.8

Table 4
Percentage accuracy for 10-fold cross-validation on Dataset 2
(Best case accuracy values are given in bold)

Test #	Training Set	Test Set	Percentage accuracy
1	301-3000	1-300	94.6
2	1-300,601-3000	301-600	96.2
3	1-600, 901-3000	601-900	91.8
4	1-900, 1201-3000	901-1200	86.8
5	1-1200, 1501-3000	1201-1500	83.6
6	1-1500, 1801-3000	1501-1800	93.0
7	1-1800, 2101-3000	1801-2100	97.2
8	1-2100, 2401-3000	2101-2400	98.2
9	1-2400, 2701-3000	2401-2700	**98.8**
10	1-2700	2701-3000	95.2

Experiment 3: One of the most significant advantages of a CNN in the field of object recognition is its ability for partial rotational invariance. In the field of OCR of which digit recognition is a part, it is often required to develop models which provide rotational invariance. In this experiment, we carry out extensive tests on the effectiveness of rotational invariance of a CNN when applied to images of Bangla and Hindi numerals. The experiment has three phases as follows:

Phase 1: In the first phase, we perform two tests. The first test is done by rotating a number of images in both the training and test sets by 90 degrees in the clockwise direction. The second test is carried out by rotating a number of images in both the training and test sets by 180 degrees. The number of images rotated in the training set is a small portion of the total number of training images, thereby making it difficult for the CNN to learn the parameters. The details of the first phase of this experiment are as follows:

Dataset 1 (Bangla Numerals): We divide the dataset of 6000 images into a training set of 4000 and a test set of 2000 images with the purpose of implementing a 3-fold cross-validation scheme. There are four major steps involved for Dataset 1. In the first step, we rotate 500 images in the training set and 100 images in the test set by 90 degrees in the clockwise direction. In the second step, we rotate 1000 images in the training set and 200 images in the test set by 90 degrees clockwise. The third and fourth steps are similar to the first two, the only difference being that the angle of rotation is 180 degrees instead of 90 degrees. The classification accuracies obtained for the above mentioned tests are summarized in Table 5.

Dataset 2 (Hindi Numerals): The dataset of 3000 images is divided into a training set of 2000 and a test set of 1000 images, for 3-fold cross-validation. We perform two steps similar to the first two steps carried out for Dataset 1. In the first step, we rotate 500 images in the training set and 100 images in the test set by an angle of 90 degrees in the clockwise direction. The second step is identical to the first except for the angle of rotation being 180 degrees. Table 6 summarizes the obtained classification accuracies corresponding to each test for Dataset 2.

It should be noted that the training and test sets are not augmented by inserting rotated versions of images in them. Instead, a portion of the existing images are rotated with the overall number of training and test images remaining same.

Table 5
Results of rotational invariance tests (Phase 1) for Dataset 1
(Best case accuracy value is shown in bold)

Number of images rotated	Angle of rotation in degrees	Training Set	Test Set	Percentage accuracy
		2001-6000	1-2000	95.4
	90	1-2000, 4001-6000	2001-4000	**97.0**
Training set: 500		1-4000	4001-6000	94.6
Test set: 100		2001-6000	1-2000	95.4
	180	1-2000, 4001-6000	2001-4000	96.8
		1-4000	4001-6000	93.6
		2001-6000	1-2000	93.2
	90	1-2000, 4001-6000	2001-4000	95.6
Training set: 1000		1-4000	4001-6000	92.4
Test set: 200		2001-6000	1-2000	95.6
	180	1-2000, 4001-6000	2001-4000	95.2
		1-4000	4001-6000	92.8

Table 6
Results of rotational invariance tests (Phase 1) for Dataset 2
(Best case accuracy value is shown in bold)

Number of images rotated	Angle of rotation in degrees	Training Set	Test Set	Percentage accuracy
		1001-3000	1-1000	92.8
	90	1-1000, 2001-3000	1001-2000	89.6
Training set: 500		1-2000	2001-3000	**94.2**
Test set: 100		1001-3000	1-1000	88.8
	180	1-1000, 2001-3000	1001-2000	92.0
		1-2000	2001-3000	92.6

Phase 2: The second phase of the experiment is similar to the first phase except for the fact that among the images rotated in both the training and test sets, half of the images are rotated by 90 degrees clockwise and the other half by 180 degrees. Dataset 1 is divided into a training set of 4000 images and a test set of 2000 images. A total of 1000 images in the training set are rotated out of which 500 images are rotated by 90 degrees clockwise and the other 500, by 180 degrees. In the test set, we randomly choose 200 images and rotate half of them by 90 degrees clockwise and the other half by 180 degrees.

We divide Dataset 2 into a training set of 2000 images and a test set of 1000 images. A total of 500 images in the training set are selected out of which 250 images are rotated by 90 degrees clockwise and the other 250 are rotated by 180 degrees. In the test set, we select 100 images and rotate 50 of them by 90 degrees in the clockwise direction and the other 50 by 180 degrees. For both the datasets, we implement a 3-fold cross-validation. The results for Dataset 1 and Dataset 2 are presented in Tables 7 and 8 respectively and the corresponding confusion matrices are illustrated in Figures 12 and 13 respectively.

Table 7
Results of rotational invariance tests (Phase 2) for Dataset 1
(Best case accuracy value is shown in bold)

Training Set	Test Set	Percentage accuracy
2001-6000	1-2000	92.2
1-2000, 4001-6000	2001-4000	**95.2**
1-4000	4001-6000	92.4

Table 8
Results of rotational invariance tests (Phase 2) for Dataset 2
(Best case accuracy value is shown in bold)

Training Set	Test Set	Percentage accuracy
1001-3000	1-1000	92.4
1-1000, 2001-3000	1001-2000	90.6
1-2000	2001-3000	**95.6**

Phase 3: In the third phase of the experiment, we do not rotate any image in the training set. However, we rotate a certain number of images in the test set by 30 degrees clockwise in the first test and by 45 degrees clockwise in the second test. Given that there are no rotated images in the training set, these tests approximate the amount of rotation that can be induced in a given image provided that there remains a high chance for it to be correctly classified.

In this phase, there are two main steps performed for both Datasets 1 and 2. First, the datasets are divided into a training set and a test set in the ratio of 2:1 for the purpose of 3-fold cross-validation. Next, we rotate 100 images in the test set by 30 degrees clockwise in the first step and by 45 degrees clockwise in the second step. The results of this experiment are shown in Tables 9 and 10 for Datasets 1 and 2 respectively and the related confusion matrices are presented in Figures 14 and 15 respectively.

Table 9
Results of rotational invariance tests (Phase 3) for Dataset 1
(Best case accuracy value is shown in bold)

Angle of rotation in degrees	Training Set	Test Set	Percentage accuracy
	2001-6000	1-2000	**94.6**
30	1-2000, 4001-6000	2001-4000	94.2
	1-4000	4001-6000	93.0
	2001-6000	1-2000	93.0
45	1-2000, 4001-6000	2001-4000	94.2
	1-4000	4001-6000	91.2

Table 10
Results of rotational invariance tests (Phase 3) for Dataset 2
(Best case accuracy value is shown in bold)

Angle of rotation in degrees	Training Set	Test Set	Percentage accuracy
	1001-3000	1-1000	90.2
30	1-1000, 2001-3000	1001-2000	**92.2**
	1-2000	2001-3000	90.6
	1001-3000	1-1000	88.6
45	1-1000, 2001-3000	1001-2000	89.6
	1-2000	2001-3000	89.4

The confusion matrices convey a few interesting observations. If we factor in the complete absence of transformed images in the training set, the classification performance on rotated test images is quite satisfactory with a best case accuracy of 87 % for Bangla numerals and 88 % for Hindi numerals, where both of the best case scenarios occur when the angle of rotation is 30 degrees clockwise for test images. However, the experimental runs with test images rotated by 45 degrees clockwise indicate a lower performance with best case classification accuracy being 78% for Bangla numerals and 79% for Hindi numerals. The reasons behind these results are primarily two-fold. Firstly, rotation of an image by relatively low angles does not change its constituent visual features beyond the recognition capacity for a CNN. Since the features are successfully detected and the location of visual features with respect to one another remains almost the same even after rotation, a CNN can effectively recognize the image. However, in case of large angles of rotation, the individual features themselves become distorted enough to not be correctly recognized by the feature maps in a CNN, thereby leading to a concomitant decrease in classification accuracy. This is clearly reflected in the results obtained from Phase 3 of our experiment. We can conclude that the rotation of images by a maximum of approximately 30 degrees does not reduce the classification accuracy of a CNN below satisfactory levels. In Table 11, we present a comparative overview of the best-case performance of various zone or region based handcrafted feature extraction and classification mechanisms with that of our proposed approach of CNN classifiers for the task of classifying Bangla numerals. Due to the unavailability of significant works on Hindi handwritten numeral classification, we could not provide the comparative study on the same.

Table 11
Comparative overview of various region based handcrafted feature extraction and classification methods with proposed approach on Dataset 1

Work reference	Sample distribution for training and test sets		Classifier employed	Percentage classification accuracy on test images
	Training set	Test set		
Basu et al. [4]	4000	2000	MLP	96.65%
Basu et al. [28]	3600	400	SVM	97.15%
Basu et al. [29]	4000	2000	MLP	96.45%
Das et al. [6]	2000	1000	MLP	93.4%
Das et al. [7]	4000	2000	SVM	97.70%
Proposed approach	**5400**	**600**	**CNN**	**98.2%**

Chapter 5

Conclusions

In this work, the use of CNNs for the classification of handwritten digits in Bangla and Hindi scripts has been demonstrated. CNNs provide a compact architecture which, by the virtue of its design, automates the extraction of spatially localized visual features. The relative locations of each detected feature provides the discriminatory information used to differentiate among image classes or pattern classes.

Extensive experiments carried out on standard datasets indicate that the CNN can outperform most of the traditional classifiers. The experimental results yield a best case classification accuracy of 98.2% for Bangla and 98.8% for Hindi which are significantly better than most of the existing models in the literature. It is also observed that a satisfactory amount of invariance to rotation for large angles can be obtained by inserting a relatively small number of rotated images in the training set. Furthermore, the CNN can properly classify most images rotated by angles of approximately 30 degrees without needing any rotated images in its training set. It was also noted that quite a few of the misclassifications were due to the inherent visual similarity between the actual pattern class and the classified pattern class.

The experiments provide conclusive evidence of the usefulness and efficacy of CNNs for the recognition of handwritten digits in Bangla and Hindi. In the future, we intend to remodel the architecture of the CNN for handling images rotated by large angles without any prior training on such rotated images.

Bibliography

1. M. Cheriet, H. Bunke, J. Hu, F. Kimura, C.Y. Suen, New frontiers in handwriting recognition, Pattern Recognition 42 (2009) 3129–3130.

2. J. Cao, M. Ahmadi, M. Shridhar, Recognition of handwritten numerals with multiple feature and multistage classifier, Pattern Recognition 28 (1995), 153–160.

3. J. Park, V. Govindaraju, S.N. Srihari, OCR in a hierarchical feature space, IEEE Transactions on Pattern Analysis and Machine Intelligence 22 (2000), 400–407.

4. S. Basu, N. Das, R. Sarkar, M. Kundu, M. Nasipuri, D. Basu, An MLP based approach for recognition of handwritten 'Bangla' numerals, in: B. Prasad (Ed.), 2nd Indian International Conference on Artificial Intelligence, Pune, India, 2005, pp. 407–417.

5. S.V. Rajashekararadhya, P.V. Ranjan, Efficient zone based feature extraction algorithm for handwritten numeral recognition of four popular South Indian scripts, Journal of Theoretical and Applied Information Technology 4 (2008), 1171–1181.

6. N. Das, S. Basu, R. Sarkar, M. Kundu, M. Nasipuri, D.K. Basu, A soft computing paradigm for handwritten digit recognition with application to Bangla digits, in: International Conference on Modeling and Simulation, MS-07, Kolkata, India, 2007, pp. 771–774.

7. N. Das, R. Sarkar, S. Basu, M. Kundu, M. Nasipuri, D.K. Basu, A genetic algorithm based region sampling for selection of local features in handwritten digit recognition application, Applied Soft Computing 12 (2012) 1592-1606.

8. S. Basu, M. Kundu, M. Nasipuri, D.K. Basu, A two-pass fuzzy-geno approach to pattern classification, in: International Conference on Computer Processing of Bangla, Dhaka, Bangladesh, 2006, pp. 130–134.

9. N. Das, J.M. Reddy, R. Sarkar, S. Basu, M.Kundu, M. Nasipuri, D.K. Basu, A statistical-topological feature combination for recognition of handwritten numerals, Applied Soft Computing 12 (2012) 2486-2495.

10. S. Basu, N. Das, R. Sarkar, M. Kundu, M. Nasipuri, D.K. Basu, A hierarchical approach to recognition of handwritten Bangla characters, Pattern Recognition 42 (2009) 1467–1484.

11. S. Basu, R. Sarkar, N. Das, M. Kundu, M. Nasipuri, D. Basu, Handwritten Bangla digit recognition using classifier combination through DS technique, in: S. Pal, S. Bandyopadhyay, S. Biswas (Eds.), Pattern Recognition and Machine Intelligence, Springer, Berlin/Heidelberg, 2005, pp. 236–241.

12. N. Das, B. Mandal, S. Basu, R. Sarkar, M. Kundu, M. Nasipuri, An SVM-MLP classifier combination scheme for recognition of handwritten Bangla digits, in: K.V. Kale, S.C. Malhrota, R.R. Manza (Eds.), 2nd International Conference on Advances in Computer Vision and Information Technology, vol. I, K. International Publishing House Pvt. Ltd., Aurangabad, India, 2009, pp. 615–623.

13. Y. LeCun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard and L.D. Jackel, Backpropagation applied to handwritten zip code recognition, Neural Computation 1 (1989) 541-551.

14. S. Lawrence, C.L. Giles, A.C. Tsoi and A.D. Back, Face Recognition: A Convolutional Neural-Network Approach, IEEE Transactions on Neural Networks, 8 (1997) 98-113.

15. Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, Gradient-Based Learning Applied to Document Recognition, in: Proc of the IEEE, 86 (1998), pp. 2278-2324.

16. Y. LeCun and Y. Bengio, Convolutional networks for images, speech, and time series, The Handbook of Brain Theory and Neural Networks, M. A. Arbib, Ed. Cambridge, MA: MIT Press, 1995, pp. 255–258.

17. H. Cecotti, A Time-Frequency Convolutional Neural Network for the Offline Classification of Steady-State Visual Evoked Potential Responses, Pattern Recognition Letters, 32 (2011) 1145-1153.

18. H. A. Perlin and H. S. Lopes, Extracting Human Attributes using a Convolutional Neural Network Approach, Pattern Recognition Letters, 68 (2015) 250-259.

19. G. Antipov, S.A. Berrani and J.L. Dugelay, Minimalistic CNN-based Ensemble Model for Gender Prediction from Face Images, Pattern Recognition Letters, 70 (2016) 59-65.

20. S. Zhu, Z. Shi, C. Sun and S. Shen, Deep Neural Network Based Image Annotation, Pattern Recognition Letters, 65 (2015) 103-108.

21. J. Schmidhuber, Deep Learning in Neural Networks: An Overview, Elsevier Neural Networks, 61 (2015) 85-117.

22. C. Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu., Deeply-Supervised Nets. arXiv:1409.5185, 2014.

23. Off-Line Handwritten Bangla Numeral Database,
http://www.isical.ac.in/~ujjwal/download/database.html.

24. CMATERdb 3.1.1: Handwritten Bangla Numeral Database, http://code.google.com/p/cmaterdb/

25. Dataset: http://code.google.com/p/cmaterdb/as CMATERdb 3.1.1

26. D. Scherer, A. Muller, and S. Behnke, Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition in International Conference on Artificial Neural Networks, 2010.

27. Y. Jia, Caffe: An open source convolutional architecture for fast feature embedding, http://caffe.berkeleyvision. org/, 2013.

28. S. Basu, N. Das, R. Sarkar, M. Kundu, M. Nasipuri, D. Kumar Basu, A novel framework for automatic sorting of postal documents with multi-script address blocks, Pattern Recognition 43 (2010) 3507–3521.

29. S. Basu, N. Das, R. Sarkar, M. Kundu, M. Nasipuri, D. Basu, Recognition of numeric postal codes from multi-script postal address blocks, in: S. Chaudhury, S. Mitra, C. Murthy, P. Sastry, S. Pal (Eds.), Pattern Recognition and Machine Intelligence, Springer, Berlin/Heidelberg, 2009, pp. 381–386.