

# Multi Character Recognition On Extended MNIST

A report submitted to the Cluster Innovation Center in partial  
fulfillment of the requirements for paper

**V1.1 Linear Programming and Game Theory**

**(Semester VI)**



**Ankit (Roll No. 11706)**

**Prabhakar Deep Tirkey (Roll No. 11730)**

**Yatharth Rai (Roll No. 11745)**

**Cluster Innovation Centre**

**University of Delhi**

**Delhi-110007**

## **Abstract**

Computer Vision has become ubiquitous in our society, with applications in several fields. In this project, we focus on handwritten multi-character recognition. We make use of the recently released EMNIST (extended MNIST) dataset. We first apply some data preprocessing to the EMNIST dataset and then use Convolutional Neural Networks (CNNs) for the classification task, which currently yields an accuracy of 87.77% after training it for 70 epochs.

An added aim of the project is to improve the current standard architectures available for the task of handwritten character recognition and eventually integrate the model with an Android Application so as to make an app that can work as a real-time classifier.

## **Keywords**

Convolutional Neural Network, Canny Edge Detection, Contours

## Table of Contents

Abstract	2
Introduction	4
Data	5
Data Preprocessing	6
Methodology	8
Convolutional Neural Network	8
CNN Architecture	10
Mathematical Model	11
Results	13
Implementation of CNN with its Model Accuracy and Loss	14
Conclusion	16
References	17
<i>Appendix</i>	18

## Introduction

With a large number of hand-written documents, there is a great demand to convert the handwritten documents into digital record copies, which are more accessible through digital systems, such as digital forms and databases. To automatically accomplish the transformation from handwritten numbers to their digital version, multi-character recognition is inevitable and useful. Convolutional neural network (CNN) is a state-of-the-art image recognition technology in the community. Prior researches have shown single digit recognition can achieve less than a 1% error rate using CNN. For Multi-Character recognition, there are studies about recognizing house numbers, vehicle VIN numbers, and so on.

The following are the three major steps required to generate a multi-character recognition system-

- **Segmenting characters in the image**

We reduce the computation in the recognition step by preprocessing and segmenting the digit from the original images and only feed the digit patches to CNN for recognition.

- **Simple CNN architecture**

Compared with the state-of-the-art deep CNNs which are designed for complex object recognition, we build a shallow network with fewer kernels. The simple CNN requires much less memory and runs fast, and yet our result shows that it can still achieve decent accuracy

- **Batching fully-connected layers**

Batching the computation of the fully-connected layer for several images trades off the computation efficiency against the single image latency. The latter is not important for our multi-digit recognition task, where we aim at the low latency of recognizing all the digits in the frame. Therefore, we implement a batched fully-connected layer for our CNN, which has more parameter reuse and better cache locality.

## Data

The Extended-MNIST dataset has become a standard benchmark for learning, classification, and computer vision systems. In the following sections, we explore this dataset in detail and describe the steps undertaken for data preprocessing.

### *DataSet*

The dataset used is EMNIST. You can find the data [here](#). I have used the EMNIST gby merge dataset.

### *Requirements*

1. Tensorflow
2. Keras
3. Numpy
4. h5py
5. Pandas
6. OpenCV

## **Extended MNIST ( EMNIST)**

The MNIST database is a subset of a much larger dataset known as the NIST Special Database 19. This NIST dataset contains both handwritten numerals and letters and is appropriate for larger and usually more complex classification. However, the difficulties inaccessibility of the NIST dataset, coupled with its higher storage requirements led to the widespread adoption of the Modified-NIST (MNIST) database instead. The Extended-MNIST (EMNIST) dataset is a subset of the full NIST database and uses the same conversion paradigm as the MNIST dataset. It consists of both digits and characters and is easy to store use and access. It is a relatively small dataset, and multiple research groups have published classifier accuracies above 99.7% using the same, which explains its increasing popularity. The following steps are used to convert images in the NIST dataset to the EMNIST format (see Fig. 1)-

1. The original image in the NIST dataset is a 128 x 128-pixel binary image.
2. Next, a Gaussian filter with  $\sigma = 1$  is applied to the image to soften the edges. Next, the image is cropped and only the region containing the actual digit/character is retained.
3. The digit is then placed and centered into a square image
4. The image is then centered.
5. Next, the region of interest is padded with a 2-pixel border when placed into the square image, in order to match all the digits in the MNIST dataset. Finally, the image is down-sampled to an 8-bit  $28 \times 28$  pixels using bi-cubic interpolation. The range of intensity values is then scaled to  $[0, 255]$ , resulting in the  $28 \times 28$  pixels grayscale image.

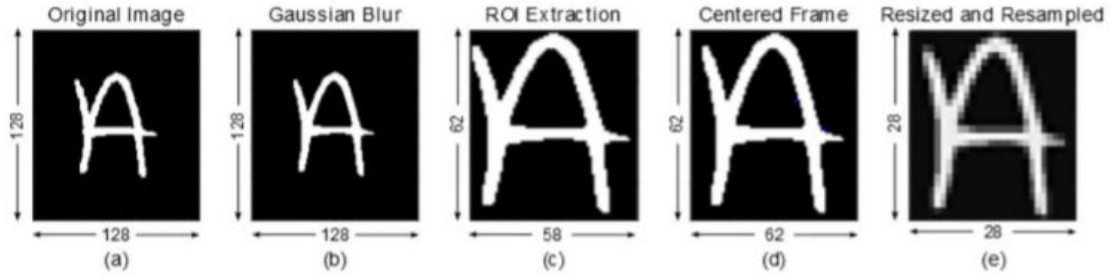


Fig. 1: Conversion of NIST images to EMNIST

After every image from NIST is converted into the EMNIST format, we have the following breakdown of testing and training samples :

Name	Classes	# of training	# of testing	Validation	Total
By_Class	62	697,932	116,323	No	814,255
By_Merge	47	697,932	116,323	No	814,255
Balanced	47	112,800	18,800	Yes	131,600
Digits	10	240,000	40,000	Yes	280,000
Letters	37	88,800	14,800	Yes	103,600
MNIST	10	60,000	10,000	Yes	70,000

Table 1: Breakdown of EMNIST

Here, **By\_Class** denotes **62 unique classes** split into [0-9], [a-z] and [A-Z] **By\_Merge** is a subset of By\_Class with those classes removed where the uppercase and lowercase letters look the same (eg- for the letters C, I, J, K, L, M, O, P, S, U, V, W, X, Y and Z their corresponding lowercase c,i,j,k,m,o,p,s,u,v,w,x,y, and z look the same when handwritten). Hence, **By\_Merge** comprises **47 unique classes**.

## Data Preprocessing

The task of Multiple character recognition requires the image to be segmented and then fed to the Convolutional Neural Network. In order to segment the image, we need to perform a few modifications to the input image. We deploy the following techniques to preprocess the input image -

1. Canny Edge detection- The working of Canny Edge Detection can be explained by the following terms-

- Gaussian Filter
- Finding the intensity gradient of the image-

$$G = \sqrt{G_x^2 + G_y^2}$$

$$\theta = \text{atan2}(G_y, G_x)$$

Here,  $G_x$  = First derivative in the horizontal direction,

$G_y$  = First derivative in the vertical direction

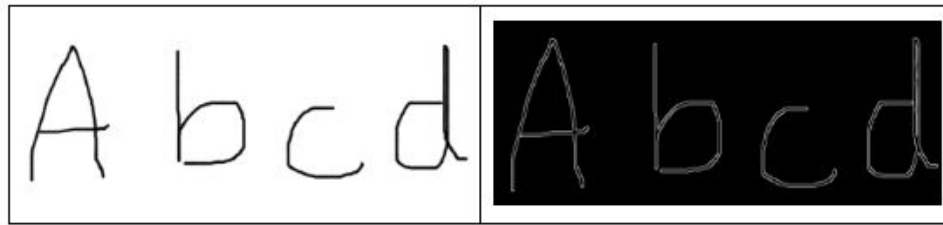


Fig. 2: Canny edge detection of handwritten text

- Non- Max Suppression
- Double Threshold
- Edge Tracking by Hysteresis

2. Dilation



Fig. 3: Output of Dilation operator

3. Finding contours for segmentation

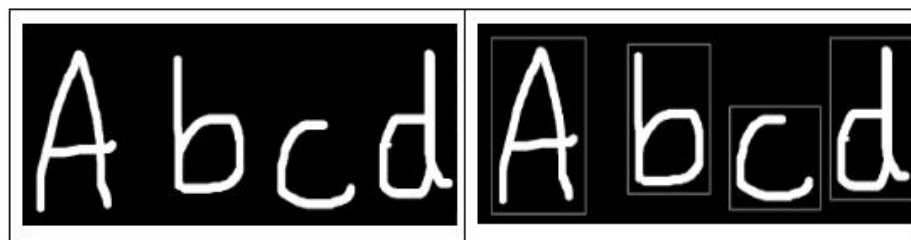


Fig. 4: Output of findcontour in OpenCV

## Methodology

### Convolutional Neural Network

We aim to complete this by using the concepts of a convolution neural network. The proposed CNN framework is well equipped with suitable parameters for the high accuracy of MNIST digit classification. The time factor is also considered for training the system. Furthermore, high accuracy is counter verified by changing the amount of CNN layers. Employment of additional pooling layers removes discretionary details in images and implants other higher-level characteristics. The MNIST dataset was used to train the network in experiments. These algorithms are employed to determine the accuracy with which these digits are classified. CNN classification proposed for HDR seems to be superior to other approaches used for handwritten characters'/pattern identification in terms of high accuracy and low computational time. It was noticed that DL4J train-ing and prediction speed was efficient and quite good. By implementing this approach (CNN-based framework reinforced by DL4J for HDR), higher and precise results were obtained. Though the goal is to just create a model that can recognize the digits, alphabets, etc it can be extended to letters and then a person's handwriting.

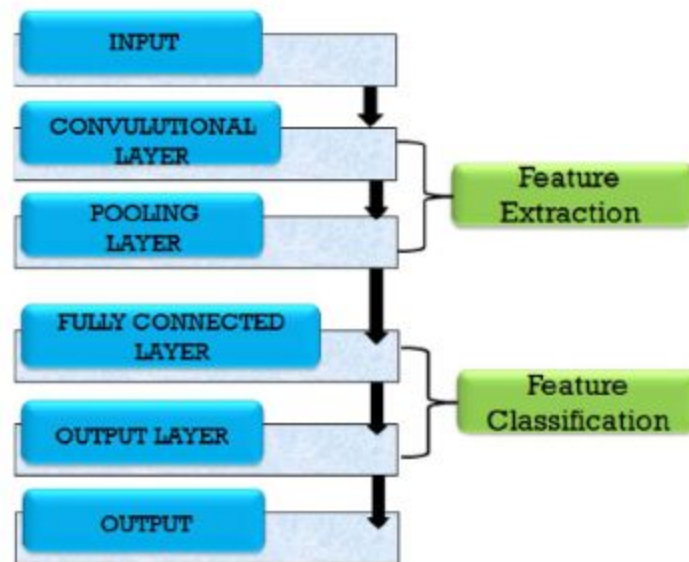


Fig. 5: Overview of CNN framework



The entire architecture of a convnet can be explained using four main operations namely-

1. Convolution
2. Non- Linearity (ReLU)
3. Pooling or Sub Sampling
4. Classification (Fully Connected Layer)

These operations are the basic building blocks of every Convolutional Neural Network, so understanding how these work is an important step to developing a sound understanding of ConvNets. We will discuss each of these operations in detail below. Essentially, every image can be represented as a matrix of pixel values. An image from a standard digital camera will have three channels – red, green and blue – you can imagine those as three 2d-matrices stacked over each other (one for each color), each having pixel values in the range 0 to 255.

### Convolutional layer (CNL)

CNL is the rst layer in CNN which memorizes the features of the input image covering its entire region during scanning through vertical and horizontal sliding filters. It adds a bias for every region followed by evaluation of scalar product of both lter values and image regions. For thresholding element-wise activation functions, such as  $\max(0, x)$ , sigmoid and tanh, are applied to output of this layer via rectified linear units.

### Pooling layer (PL)

Then, there comes a pooling layer which is also called a max pooling layer or subsampling. In the pooling layer (PL), shrinkage in the volume of data takes place for the easier and faster network computation. Max pooling and average pooling are main tools for implementing pool-ing. This layer obtains maximum value or average value for each region of the input data by applying vertical and horizontal sliding filters through input image and reduces the volume of data as shown in Fig. 6

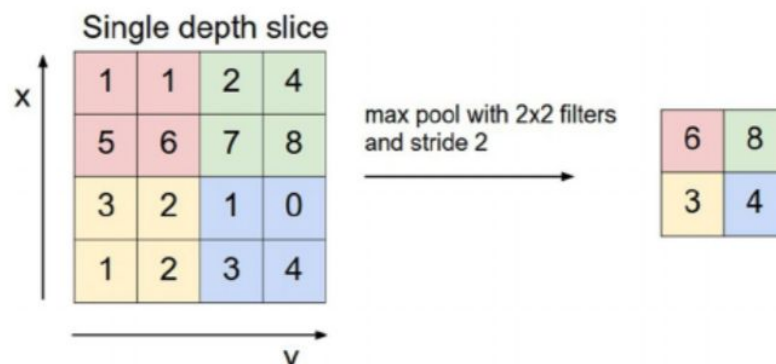


Fig. 6: Data Reduction in pooling layer

## CNN Architecture

The network architecture we discuss in this section has been trained on the EMNIST dataset for 70 epochs. The network consists of the following layers -

- *Input Layer*

The input accepted by the network architecture is a grayscale image with dimensions as  $28 \times 28$ . Since the image is grayscale thus the channel in the input shape is taken as 1.

- *Convolution 1*

The first hidden layer is a convolutional layer. We use 32 filters with stride length as 1 and the dimension of the filter are  $3 \times 3$ . The filters slide over the entire image to generate an output vector of size  $26 \times 26$  and the thickness of this output is 32 since 32 filters are used.

- *Convolution 2*

The second convolution is performed with 64 filters of size  $3 \times 3$ . The output of the second convolution is  $24 \times 24 \times 64$  dimensional tensor.

- *Maxpool 1*

The max pooling operation is applied on the output tensor generated by the convolution layer. We use the stride length of 2 and the pooling is performed with  $2 \times 2$  dimensions. The output of the pooling operation is a  $12 \times 12 \times 64$  dimensional vector

- *Dropout*

Since the model needs to learn new pathways to the correct classification, thus we introduce a dropout of 0.25 which means that a neuron is turned off with a probability of 0.25 so that the model can find new pathways to reach the correct solution

- *Flatten*

The flatten operation stretches the output of the previous layer. Thus the output of this layer is  $12 \times 12 \times 64 = 9216$ . We then connect a fully connected layer to this output to generate predictions.

- *Dense*

We use a fully connected layer first to compress the output of previous layer to dimension  $128 \times 1$  and then append another fully connected layer with 47 classes so that the network is able to make the predictions. A dropout rate of 0.5 is provided between these two fully connected layers.

## Mathematical Model

Each input node on convolution operation in convolution layer is meant for extracting features from input images, whereas the input nodes on average or maximum operation in max pooling layer abstract the features from image. The outputs of S-1th layer are utilized as input for the nth layer, and then, the inputs pass through a set of kernels followed by nonlinear function ReLU (f). Suppose, if  $x_i^{s-1}$  input from S-1th layer,  $k_{i,j}^s$  are the kernels of 1th layer.  $b_j^s$  represents the biases of 1th layer. Then, the convolution operation can be stated as follows:

$$x_j^s = f(x_i^{s-1} * k_{i,j}^s) + b_j^s$$

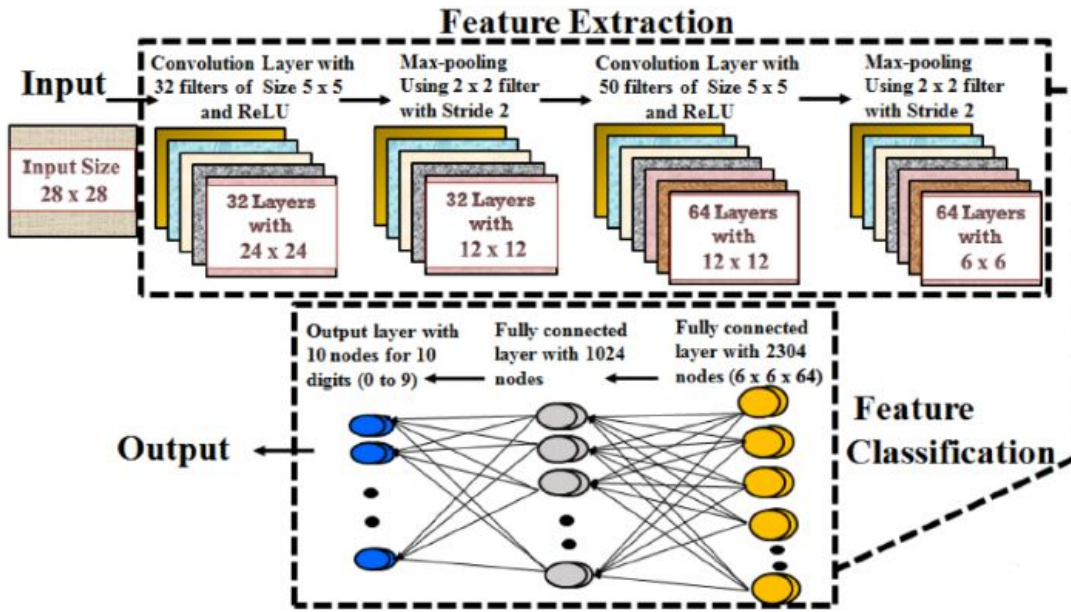


Fig. 7: Proposed design of CNN Architecture

The input nodes on average or maximum operation in the max pooling layer abstract the features from the image. Here,  $2 \times 2$  down-sampling kernel is applied and each output dimension will be the half of the corresponding input dimension for all the inputs. The pooling operation can be expressed as follows:

$$x_j^s = \text{down}(x_i^{s-1})$$

CNN propagates features of lower-level layers to produce features of higher-level layers contrary to traditional neural networks (NN). With the feature propagation, dimension of feature reduced depending upon the size of convolution and pooling masks. However, for the enhanced accuracy for classification, one can select the feature mapping for extreme suitable features of the input images. Fully connected layer uses the output of the last layer of CNN as inputs. Softmax operation is usually utilized to get the classification outputs. Softmax operation for the  $i$ th class of input sample( $x$ ), weight vector ( $w$ ) and distinct linear functions ( $K$ ) can be stated as follows-

$$P(y = i|x) = \frac{\exp(x^T w_i)}{\sum_{k=1}^K \exp(x^T w_k)}$$

## Results

The neural network architecture proposed for the task of character classification of the images reaches an accuracy of 87.77 % on training it for 70 epochs with a batch size of 128. The testing sample size was 18,800 images corresponding to which the model produces decent results. The model when coupled with the segmentation for multiple digit recognition also generates good results. Following are a few examples of the output generated by the model-




	<b>Detected Text - AbCd</b>
	<b>Detected Text - dab</b>
	<b>Detected Text - 235</b>

Fig. 8: Output of the CNN-Model

## Implementation of CNN with its Model Accuracy and Loss

```
Model: "model"
-----
Layer (type)                 Output Shape              Param #
=====
input_1 (InputLayer)         [(None, 28, 28, 1)]      0
-----
conv2d (Conv2D)              (None, 28, 28, 32)       832
-----
max_pooling2d (MaxPooling2D) (None, 14, 14, 32)       0
-----
conv2d_1 (Conv2D)            (None, 14, 14, 64)       18496
-----
max_pooling2d_1 (MaxPooling2 (None, 7, 7, 64)        0
-----
conv2d_2 (Conv2D)            (None, 7, 7, 128)        73856
-----
max_pooling2d_2 (MaxPooling2 (None, 3, 3, 128)        0
-----
flatten (Flatten)            (None, 1152)             0
-----
dense (Dense)                 (None, 256)              295168
-----
dense_1 (Dense)               (None, 128)              32896
-----
dense_2 (Dense)               (None, 47)               6063
=====
Total params: 427,311
Trainable params: 427,311
Non-trainable params: 0
-----
```

Fig. 9: Implementation of CNN

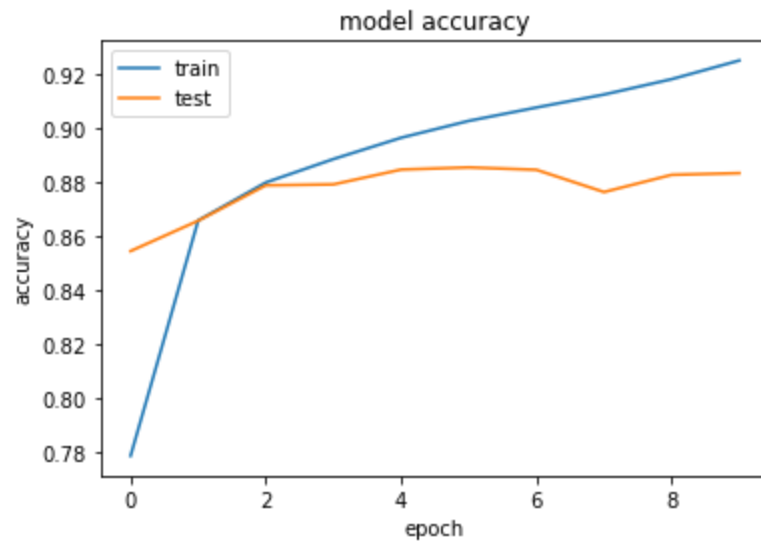


Fig. 10: Model Accuracy

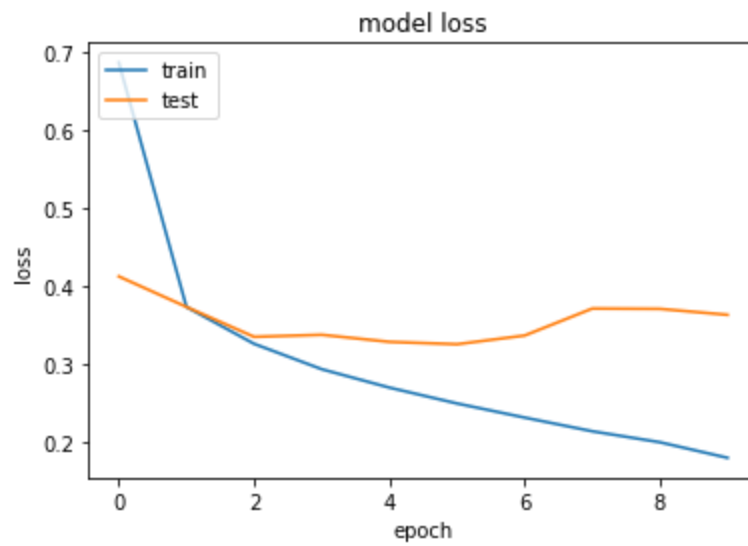


Fig. 10: Model Loss

## **Conclusion**

Character recognition is a key step toward artificial intelligence and computer vision. To test the classification performance, pattern classification and machine learning communities use the problem of handwritten digit recognition as a model.

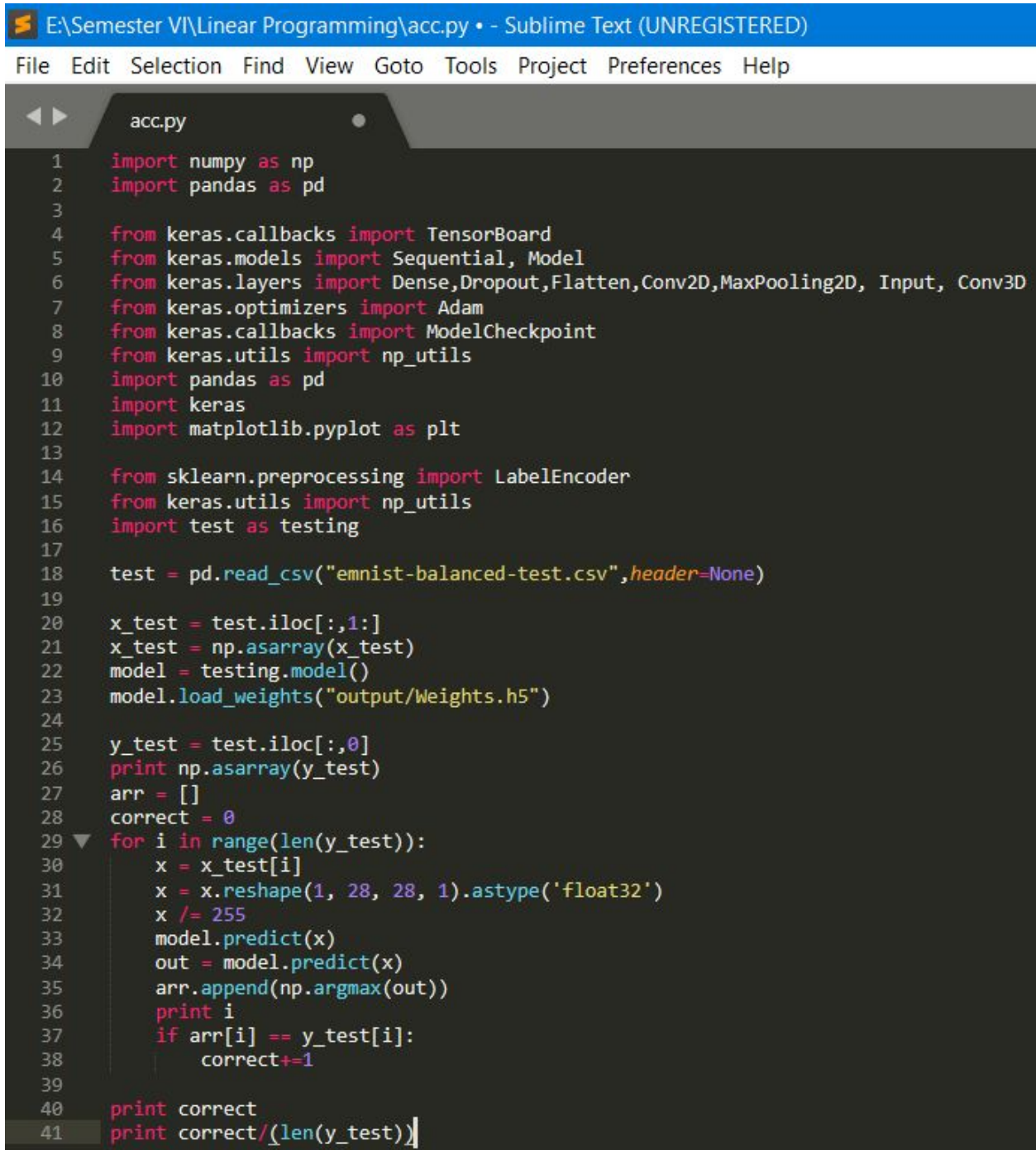
In this project, recognition of handwritten digit using convolutional neural network (CNN), incorporating a Deeplearning4j (DL4J) framework, with rectified linear units (ReLU) activation is implemented. The proposed CNN framework is well equipped with suitable parameters for high accuracy of MNIST digit classification. Time factor is also considered for training the system. Afterward, for further verification of accuracy, the system is also checked by changing the number of CNN layers. We make use of the recently released EMNIST (extended MNIST) dataset. We first apply some data preprocessing to the EMNIST dataset and then use Convolutional Neural Networks (CNNs) for the classification task, which currently yields an accuracy of 87.77% after training it for 70 epochs.



## References

1. Cohen, Gregory, et al. "Emnist: an extension of mnist to handwritten letters." arXiv preprint arXiv:1702.05373 (2017).
2. <https://web.stanford.edu/class/cs231m/projects/final-report-yang-pu.pdf>
3. Data : <https://www.kaggle.com/crawford/emnist/data>
4. Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. What is the best multi-stage architecture for object recognition? In Computer Vision, 2009 IEEE 12th International Conference on, pages 2146–2153. IEEE, 2009.
5. Dan Ciresan, Ueli Meier, and Jurgen Schmidhuber. Multi-column deep neural networks for " image classification. In Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on, pages 3642–3649. IEEE, 2012.
6. Marc Aurelio Ranzato, Fu Jie Huang, Y-Lan Boureau, and Yann LeCun. Unsupervised learning of invariant feature hierarchies with applications to object recognition. In Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on, pages 1–8. IEEE, 2007.

## Appendix



```
E:\Semester VI\Linear Programming\acc.py • - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

acc.py
1  import numpy as np
2  import pandas as pd
3
4  from keras.callbacks import TensorBoard
5  from keras.models import Sequential, Model
6  from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D, Input, Conv3D
7  from keras.optimizers import Adam
8  from keras.callbacks import ModelCheckpoint
9  from keras.utils import np_utils
10 import pandas as pd
11 import keras
12 import matplotlib.pyplot as plt
13
14 from sklearn.preprocessing import LabelEncoder
15 from keras.utils import np_utils
16 import test as testing
17
18 test = pd.read_csv("emnist-balanced-test.csv", header=None)
19
20 x_test = test.iloc[:, 1:]
21 x_test = np.asarray(x_test)
22 model = testing.model()
23 model.load_weights("output/Weights.h5")
24
25 y_test = test.iloc[:, 0]
26 print np.asarray(y_test)
27 arr = []
28 correct = 0
29 for i in range(len(y_test)):
30     x = x_test[i]
31     x = x.reshape(1, 28, 28, 1).astype('float32')
32     x /= 255
33     model.predict(x)
34     out = model.predict(x)
35     arr.append(np.argmax(out))
36     print i
37     if arr[i] == y_test[i]:
38         correct += 1
39
40 print correct
41 print correct / (len(y_test))
```

Fig. 11: *acc.py*

```
E:\Semester VI\Linear Programming\mnist.py • - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

mnist.py
1 import numpy as np
2 from keras.datasets import mnist
3 from keras.models import Sequential, Model
4 from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D, Input, BatchNormalization, Activation
5 from keras import optimizers
6 from keras.utils import np_utils
7
8 (x_train, y_train), (x_test, y_test) = mnist.load_data()
9 print x_train[1].shape
10 print len(x_test)
11 num_classes = 10
12 y_train = np_utils.to_categorical(y_train, num_classes)
13 y_test = np_utils.to_categorical(y_test, num_classes)
14
15 x_train = np.asarray(x_train)
16 x_test = np.asarray(x_test)
17 x_train = x_train.reshape(x_train.shape[0], 28, 28, 1).astype('float32')
18 x_test = x_test.reshape(x_test.shape[0], 28, 28, 1).astype('float32')
19 x_train /= 255
20 x_test /= 255
21 model5 = Sequential()
22 model5.add(Conv2D(32, 3, 3, border_mode='valid', input_shape=(28, 28, 1)))
23 model5.add(BatchNormalization())
24 model5.add(MaxPooling2D(pool_size=(2, 2)))
25 model5.add(Activation("relu"))
26 model5.add(Conv2D(32, 3, 3, border_mode='valid'))
27 model5.add(Dropout(0.25))
28 model5.add(BatchNormalization())
29 model5.add(Activation("relu"))
30 model5.add(MaxPooling2D(pool_size=(2, 2)))
31 model5.add(Dropout(0.3))
32 model5.add(Flatten())
33 model5.add(Dense(1024, activation='relu'))
34 model5.add(BatchNormalization())
35 model5.add(Dropout(0.4))
36 model5.add(Dense(128, activation='relu'))
37 model5.add(BatchNormalization())
38 model5.add(Dropout(0.4))
39 model5.add(Dense(num_classes))
40 model5.add(BatchNormalization())
41 model5.add(Activation("softmax"))
42 print model5.summary()
43
44 adam = optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=1e-6, amsgrad=False)
45 model5.compile(optimizer=adam, loss='categorical_crossentropy', metrics=['accuracy'])
46
47 model5_info = model5.fit(x_train, y_train, batch_size=128, \
48                          nb_epoch=50, verbose=1, validation_split=0.2)
49
50 model5.save_weights('output/Weights_mnist.h5', overwrite=True)
51 result = model5.predict(x_test)
52 predicted_class = np.argmax(result, axis=1)
53 true_class = np.argmax(y_test, axis=1)
54 num_correct = np.sum(predicted_class == true_class)
55 accuracy = float(num_correct)/result.shape[0]
56 print (accuracy * 100)
```

Fig. 12: *mnist.py*