

Handwritten Digit Recognizer using Deep Neural Network

I. Definition

Project Overview

Handwriting recognition is the ability of a computer to receive and interpret intelligible handwritten input from sources such as paper documents, photographs, touch-screens and other devices. The image of the written text may be sensed "off line" from a piece of paper by optical scanning (optical character recognition) or intelligent word recognition. Alternatively, the movements of the pen tip may be sensed "on line", for example by a pen-based computer screen surface, a generally easier task as there are more clues available [1].

Handwriting Recognition plays an important role in storage and retrieval of typed and handwritten information. A wealth of historical papers exists in a physical format. This includes genealogical information, old family records, written manuscripts, personal diaries and many other pieces of a shared past. Consistent review of this information damages the original paper and can lead to physical data corruption. Handwriting recognition allows people to translate this information into easily readable electronic formats [2].

Since 2009, the recurrent neural networks and deep feedforward neural networks developed in the research group of Jürgen Schmidhuber at the Swiss AI Lab IDSIA have won several international handwriting competitions. In particular, the bi-directional and multi-dimensional Long short-term memory (LSTM) of Alex Graves et al. won three competitions in connected handwriting recognition at the 2009 International Conference on Document Analysis and Recognition (ICDAR), without any prior knowledge about the three different languages (French, Arabic, Persian) to be learned. Recent GPU-based deep learning methods for feedforward networks by Dan Ciresan and colleagues at IDSIA won the ICDAR 2011 offline Chinese handwriting recognition contest; their neural networks also were the first artificial pattern recognizers to achieve human-competitive performance on the famous MNIST handwritten digit problem of Yann LeCun and colleagues at NYU [1].

Most modern tablet pc and hybrid laptops includes an active stylus and display which allows the user to write and draw on the screen. It would be very useful for the users if they can recognize, derive meaning and store the handwritten text into digital text which can be used within computer and text-processing applications. This motivated me to take the first step of implementing a Handwritten Digit Recognizer.

Problem Statement

The goal is to predict the digit handwritten in an image. The dataset contains images of handwritten digits and their respective numbers which can be used to train and test the model. This is supervised learning problem – more specifically a classification problem. The model should be able to classify which number (0 - 9) is handwritten in the image. The model can be scored for its ability to predict the number correctly over large different test data and real data.

The dataset is loaded into python using keras library as discussed in the Data exploration. The loaded data will be first explored and visualized using numpy and matplotlib library to understand the nature of the data. Exploring the data will help us in deciding how to approach and whether any preprocessing of the data is needed. Preprocessing of the data is done as required. Once the data is ready, the Deep neural network(DNN) will be built based on the architecture(ConvNet) as discussed in the Implementation Section. Once the model is built, it will be compiled to check if the architecture has any error. Then the compiled model will be trained on the training data and evaluated using accuracy score against the testing data. Then the results can be analyzed and compared with respect to the benchmark model to know the overall performance of the model. Now we have a trained model, a test is conducted against the model by loading images of digits which are not from MNIST dataset to evaluate the performance of the final model. The images are loaded and then preprocessed to match the MNIST dataset format so that we can test it. The model is then made to predict the digit in the preprocessed image. Thus, we can evaluate our model's performance over real time data.

Metrics

Accuracy is measured as ratio between the sum of true positives and true negatives and dataset size. True Positive and True Negative here are the same because both are the sum of all truly classified handwritten digit image samples. In general, it's the measure of correctness of the model. It's just the ratio between correctly classified samples to the overall samples. It ranges from 0 – 1, where 0 means poor performance and 1 means good performance.

The cross-entropy (loss function) measure has been used as an alternative to squared error. Cross-entropy can be used as an error measure when a network's outputs can be thought of as representing independent hypotheses (e.g. each node stands for a different concept), and the node activations can be understood as representing the probability (or confidence) that each hypothesis might be true. In that case, the output vector represents a probability distribution, and our error measure - cross-entropy - indicates the distance between what the network believes this distribution should be, and what the teacher says it should be. There is a practical reason to use cross-entropy as well. It may be more useful in problems in which the targets are 0 and 1 (thought the outputs obviously may assume values in between.) Cross-entropy tends to allow errors to change weights even when nodes saturate (which means that their derivatives are asymptotically close to 0) [3]

II. Analysis

Data Exploration

The MNIST database [4] of handwritten digits has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image.

The original black and white (bi-level) 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.

The dataset can be downloaded and loaded using Keras, a high-level neural networks python library using the following code

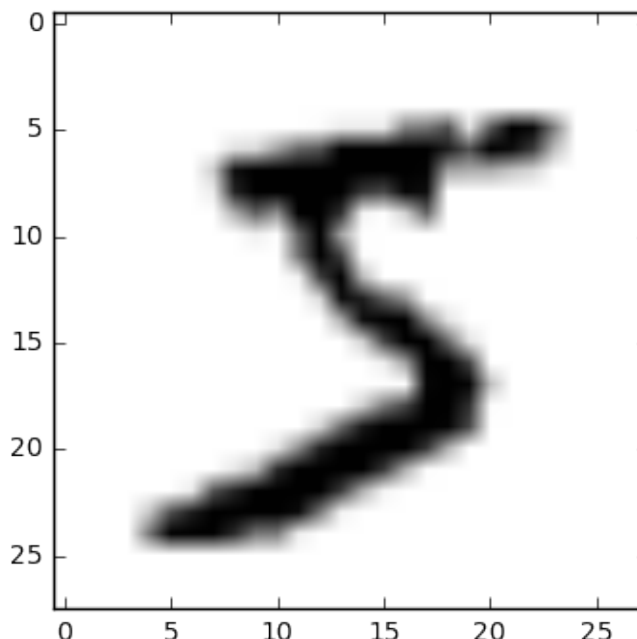
```
from keras.datasets import mnist
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

- X_train, X_test: uint8 array of grayscale image split as train and test data
- y_train, y_test: uint8 array of digit labels (integers in range 0-9) split as train and test data

Thus we can use the above variables to train the model with pixel-values of the handwritten image as features(X_train) and its respective label(y_train) as target. We can also test the model by evaluating it against the test variables X_test and y_test.

Exploratory Visualization

Let's visualize an image of a digit in the training set of MNIST dataset. The image contains 28x28 pixels as represented in the below. The image is a grayscale image i.e. it contains only one color channel whose values ranging from 0 – 255, where 0 represents white and 255 represents black in this case.

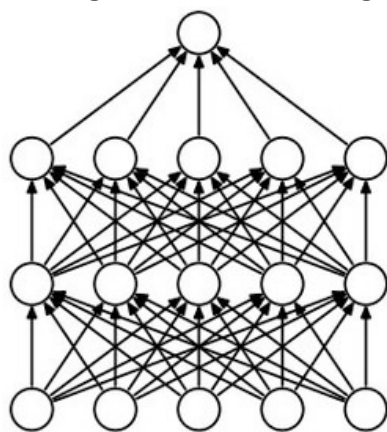


Algorithms and Techniques

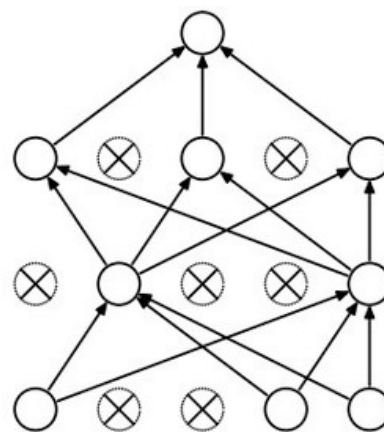
Given that the problem is a supervised learning problem and more specifically a classification problem, there are a lot of algorithms available for training a classifier to learn from the data. The algorithm chosen for this project is Deep Neural Network(DNN) using Convolutional Neural Network(ConvNet).

Convolutional neural network (CNN, or ConvNet) is a type of feed-forward artificial neural network in which the connectivity pattern between its neurons is inspired by the organization of the animal visual cortex [5]. Since it models animal visual perception, it can be applied to visual recognition tasks such as handwritten digit recognition from images. The below are the common types of layers to build ConvNet or DNN architectures:

- **Convolutional Layer** - The input to a convolutional layer is a $m \times m \times r$ image where m is the height and width of the image and r is the number of channels, e.g. an RGB image has $r=3$. The convolutional layer will have k filters (or kernels) of size $n \times n \times q$ where n is smaller than the dimension of the image and q can either be the same as the number of channels r or smaller and may vary for each kernel. The size of the filters gives rise to the locally connected structure which are each convolved with the image to produce k feature maps of size $m-n+1$. [6]
- **ReLU Layer** - The Rectified Linear Unit apply an elementwise activation function, such as the $\max(0, x)$ thresholding at zero.
- **Pooling Layer**-perform a down sampling operation along the spatial dimensions (width, height)
- **Fully-Connected or Dense Layer** - compute the class scores (between 0-9 digits). As with ordinary Neural Networks and as the name implies, each neuron in this layer will be connected to all the numbers in the previous volume. [7]
- **Dropout** is a technique for dealing with overfitting in neural networks. The key idea is to randomly drop units (along with their connections) from the neural network during training as shown in the diagram below. [8]



(a) Standard Neural Net



(b) After applying dropout.

- **Flatten** - Flattens the input. Does not affect the batch size. Consider the example code below.[9]

```
model = Sequential()
model.add(Convolution2D(64, 3, 3, border_mode='same', input_shape=(3, 32, 32)))
# now: model.output_shape == (None, 64, 32, 32)
model.add(Flatten())
# now: model.output_shape == (None, 65536)
```

After the convolutional layer the output shape is (None,64,32,32). Now when flatten is applied to the layer, the output shape is flattened by multiplying $64 \times 32 \times 32 = 65536$.

- **Softmax layer** is typically the final output layer in a neural network that performs multi-class classification (for example: object recognition). The name comes from the softmax function that takes as input a number of scores values ($z_k, k = 1 \dots K$) and squashes them into values in the range between 0, and 1 whose sum is 1. Therefore, they represent a true probability distribution. [10]

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}}$$

An example of a Convolutional Neural Network is given below [7].

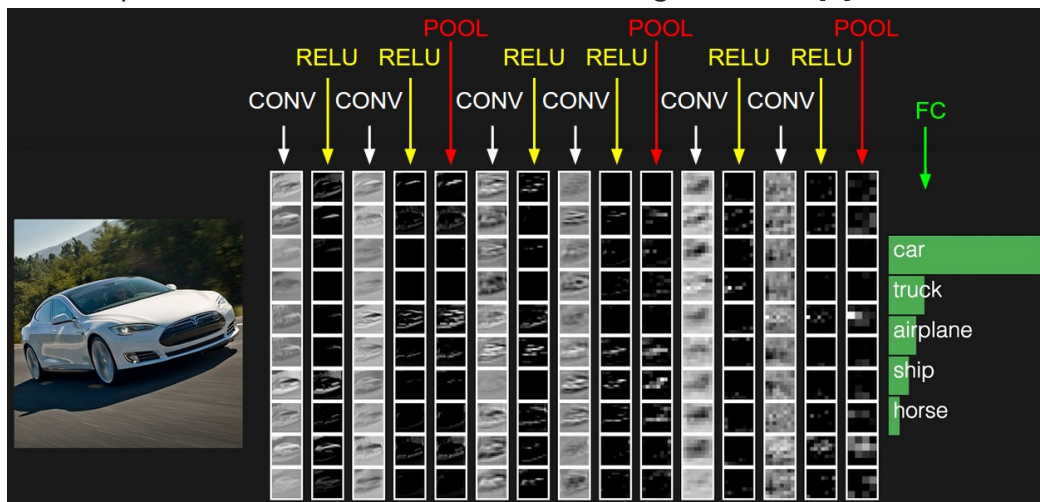


Fig 1 – An example of Convolutional Neural Network

Benchmark

The benchmark model is chosen from one of the kernels of the Kaggle Digit Recognizer competition [11]. The benchmark model solves the same handwritten digit recognition problem mentioned in this project by using a Random Forest classifier algorithm. It uses the same dataset used in this project. Thus making it a perfect benchmark model for this project. The result of the benchmark model is the accuracy score of the model. The same result can be

measured for our model by calculating the accuracy score. Accuracy score of the benchmark model is 0.872.

III. Methodology

Data Preprocessing

Since Keras library can be used to download and the load the MNIST dataset directly as numpy arrays, there is only one preprocessing step is needed. The MNIST dataset contains grayscale images where the color channel value of each pixels varies from 0 to 255. In order to reduce the computational load, we will map the values from 0 - 255 to 0 - 1 by dividing each pixel values by 255. Now the training and testing dataset

Implementation

Once the preprocessing of the data is done, the deep neural network is built. We are using Convolutional Neural Network(CNN) as the deep neural network for this project. The initial CNN model architecture and code used in this project is shown below.

```
pool_size = (2, 2)
kernel_size = (3, 3)

model = Sequential()

model.add(Convolution2D(32, kernel_size[0], kernel_size[1],
                        border_mode='valid',
                        input_shape=input_shape))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=pool_size))

model.add(Flatten())
model.add(Dense(128))
model.add(Activation('relu'))
model.add(Dense(nb_classes))
model.add(Activation('softmax'))

print("Successfully built the DNN Model!")
```

Successfully built the DNN Model!

Fig – 2 Initial Model Code

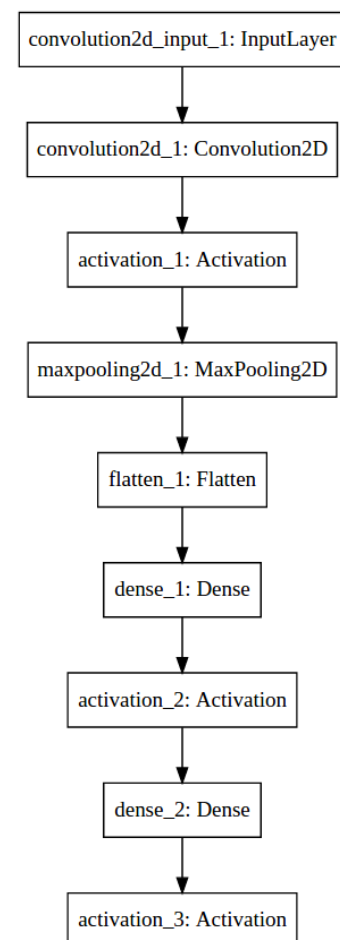


Fig – 3 Initial Model Block Diagram

The first layer of the model is the Input Layer which gets the input data as arrays during the training phase. The next layer is the convolutional layer with 32 filters and 3,3 kernel size which convolutes the input image. The next layer is an activation Layer made of Rectified Linear Unit (ReLU) which acts as an elementwise activation function. The next layer is the max pooling layer where down sampling is carried out. Then a flatten layer is used to flatten the data. Then A dense or fully connected layer of size 128 is applied followed by an activation layer. Afterwards a dropout layer is used to cut off some neurons in order to avoid overfitting of the model. The next layer is fully connected layer of size 10 i.e. it is the output size followed by a softmax activation layer. The output of the final activation layer is the final output.

Once the model is built using the above architecture, it is compiled by specifying the loss function as categorical cross entropy, optimizer as SGD(Stochastic Gradient Descent)[15], and metrics as accuracy. Once the model is compiled with no errors, it is trained over the training dataset X_train and Y_train. The model is then evaluated against the test dataset X_test and Y_test.

The accuracy score for this model is measured to be 0.9847 i.e. 98.47%.

Refinement

The initial model is improved by using the following technique

- It is made further deep by adding two more convolutional layer and one more extra dense (fully connected) layer.
- Two dropout layer is added to ensure that overfitting does not happen
- Using Nadam optimizer[15] instead of SGD(Stochastic Gradient Descent) to update the weights efficiently.

The refined CNN architecture and code of the model is shown below.

```
pool_size = (2, 2)
kernel_size = (3, 3)

rmodel = Sequential()

rmodel.add(Convolution2D(32, kernel_size[0], kernel_size[1],
                        border_mode='valid',
                        input_shape=input_shape))
rmodel.add(Activation('relu'))
rmodel.add(Convolution2D(64, kernel_size[0], kernel_size[1]))
rmodel.add(Activation('relu'))
rmodel.add(MaxPooling2D(pool_size=pool_size))
rmodel.add(Dropout(0.25))

rmodel.add(Flatten())
rmodel.add(Dense(128))
rmodel.add(Activation('relu'))
rmodel.add(Dropout(0.5))
rmodel.add(Dense(nb_classes))
rmodel.add(Activation('softmax'))

print("Successfully built the Refined DNN Model!")
```

Successfully built the Refined DNN Model!

Fig – 4 Refined Model Code

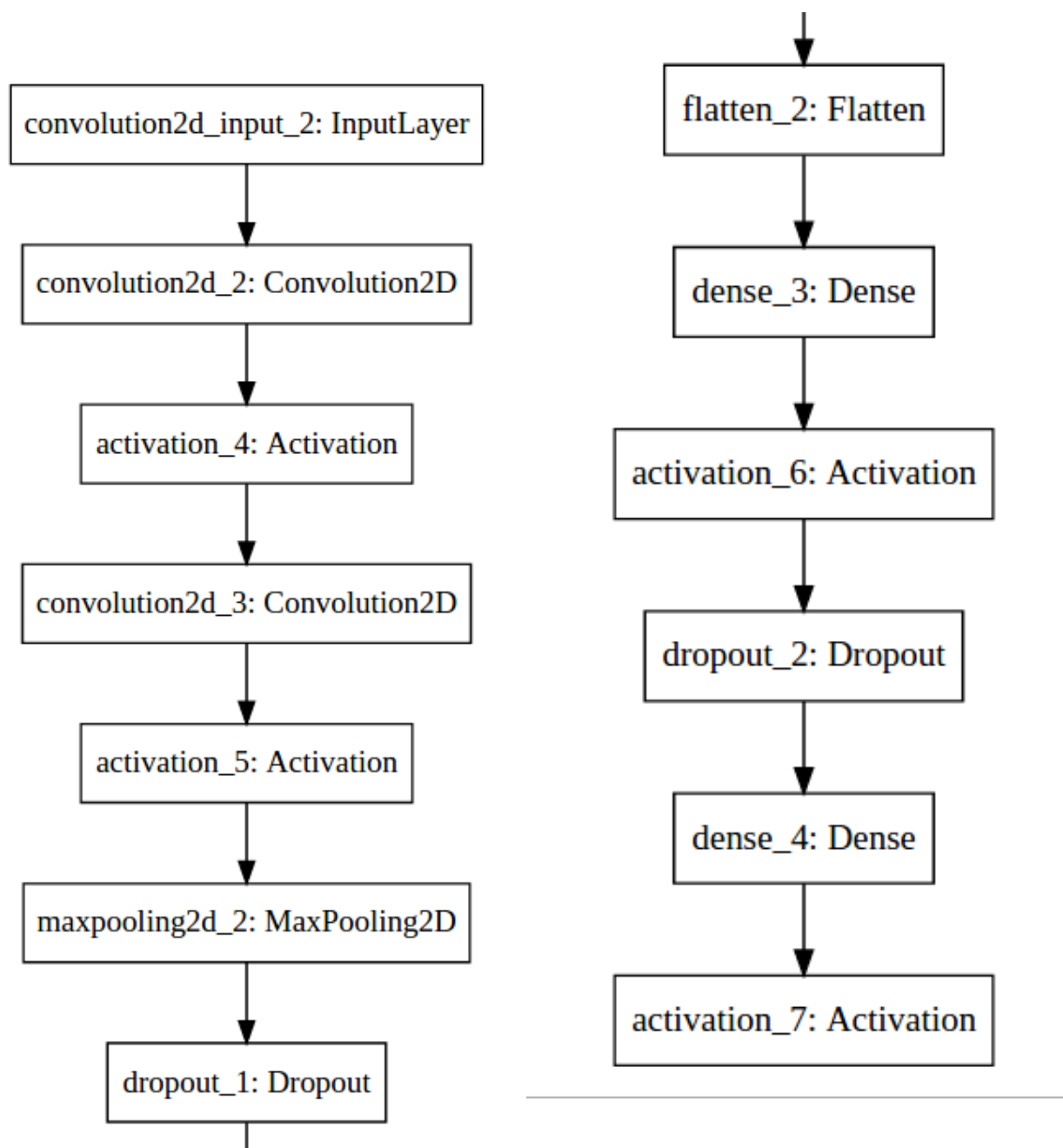


Fig – 5 Refined Model Block Diagram

Once the model is built using the above architecture, it is compiled by specifying the loss function as categorical cross entropy, optimizer as Nadam, and metrics as accuracy. Once the model is compiled with no errors, it is trained over the training dataset X_{train} and Y_{train} . The model is then evaluated against the test dataset X_{test} and Y_{test} .

The accuracy score for refined model is measured to be 0.9919 i.e. 99.19%.

IV. Results

Model Evaluation and Validation

The model is evaluated against the test set (X_{test} , Y_{Test}). The final architecture and hyper-parameters is chosen because they performed best out of the other models tried.

The robustness of the final model is verified by conducting a test against the final model using different images of digits other than MNIST dataset itself as shown in Fig 6.

The following observations are based on the results of the test:

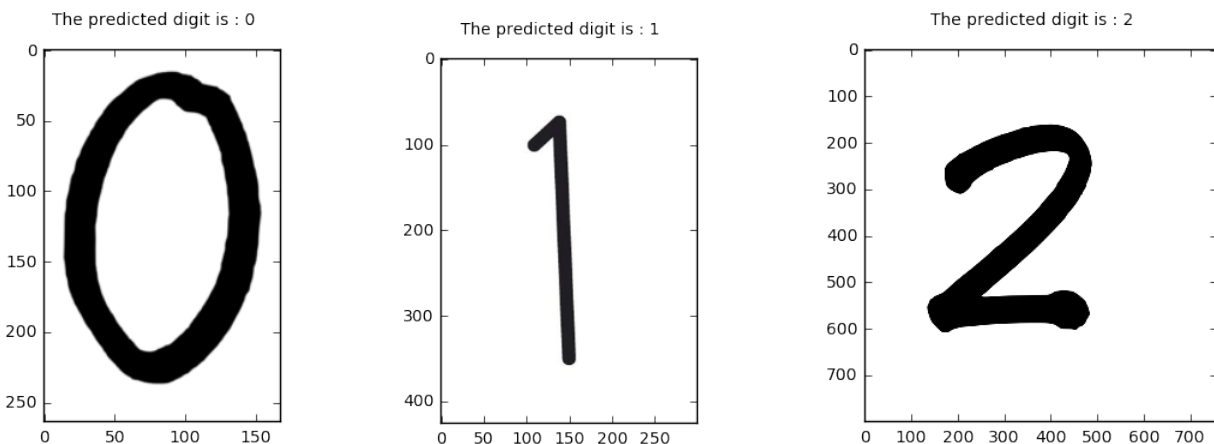
- The final model has predicted the digits of 10 out of 11 images correctly.
- Our model had problem in predicting digits that have thin strokes like the digit 4 with thin stroke in the test cases

Justification

The accuracy of the final model is 99.19% whereas the benchmark model's accuracy is 87.2%. Our model has clearly outperformed the benchmark model. Although, in order to justify that our model has potentially solved the problem, we need to look at the results of the real time test conducted against model as shown in Fig 6. Our model can correctly predict 10 out of 11 tests. Thus, our model solved the problem with a good accuracy score and reliability in real time applications.

V. Conclusion

Free-Form Visualization



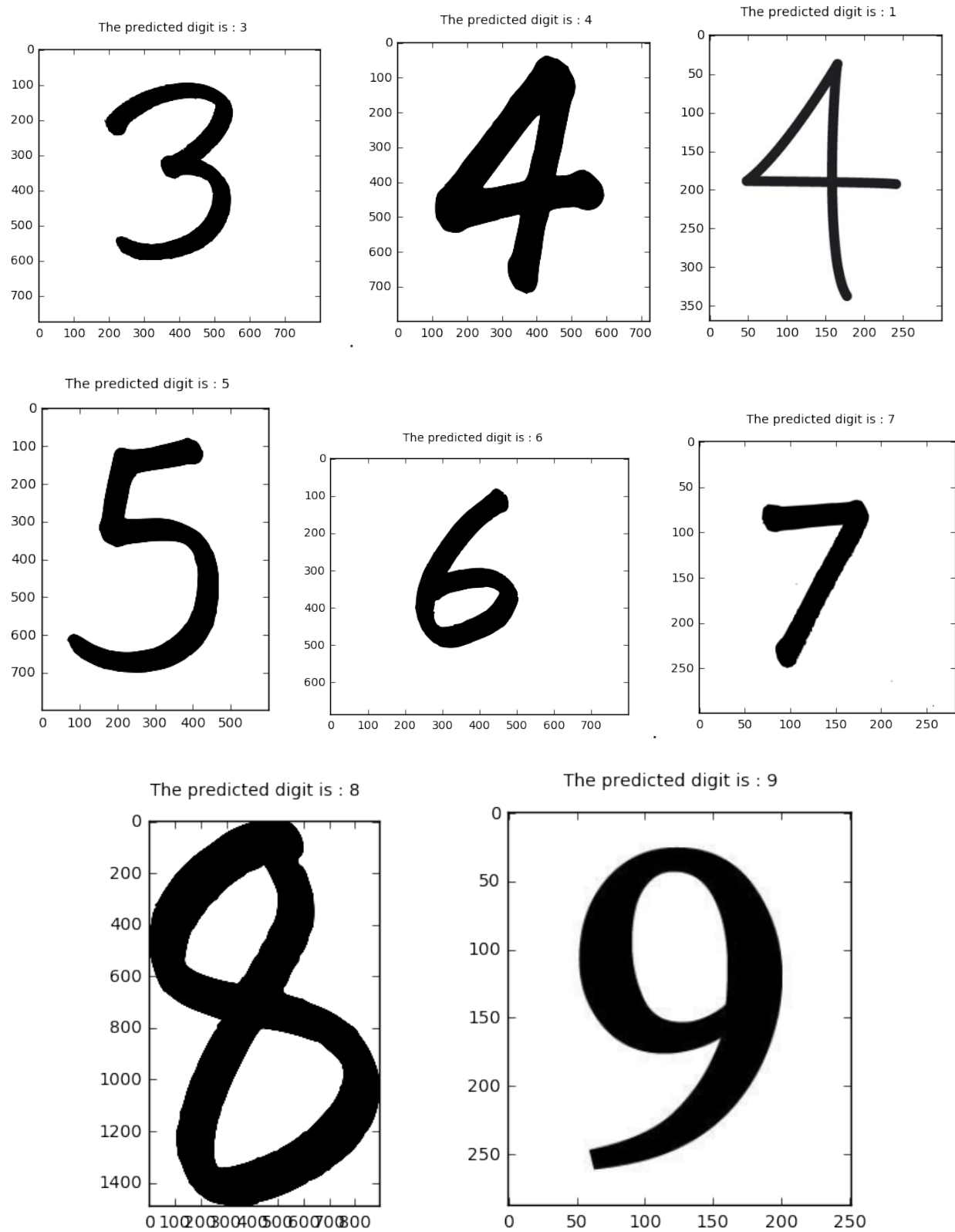


Fig – 6 Results of test conducted by loading images of digits which are not from MNIST dataset to evaluate the performance of the final model.

From the above results, we can see that our final model has predicted the digits of 10 out of 11 images correctly. Thus our model has a good accuracy score of 99.19% and good reliability as seen in the above test.

Reflection

The process involved in this project can be summarized using the below steps:

1. An initial problem and a relevant public dataset were found
2. A benchmark model is chosen for the problem
3. The dataset was downloaded and loaded in the code
4. The data is preprocessed as required
5. A deep neural network model using convolutional neural network is built
6. The model is trained using the training dataset and evaluated against the test dataset multiple times in order to find an optimum model architecture with good hyper-parameters
7. A test is conducted against the refined model by loading images of digits which are not from MNIST dataset to evaluate the performance of the final model.

I found steps 5 and 6 to be tough as I was new to Deep Learning and it took a while for me to grasp the concept. However using the MNIST dataset and keras library simplified my process a lot. MNIST Dataset was a good choice for a newbie in deep learning like me as there was a lot of tutorials and guides available using MNIST dataset which helped in understanding them easily. Also Keras helped me in kick-starting into deep learning without writing complex code which was really helpful in quick prototyping and experimenting in order to build a better model.

Improvement

The final model in this project uses only simple Convolutional Neural Network architecture. There are more advanced architectures like the Deep Residual Neural Network (ResNet)[12] and VGG-16[13] which could be used to improve the accuracy a little. Also we could use hyperas[14] library to fine tune the hyper-parameters and architecture of the model.

References

1. https://en.wikipedia.org/wiki/Handwriting_recognition
2. http://www.ehow.com/list_7353703_benefits-advantages-handwriting-recognition.html
3. <http://www.cse.unsw.edu.au/~billw/cs9444/crossentropy.html>
4. <http://yann.lecun.com/exdb/mnist/>
5. https://en.wikipedia.org/wiki/Convolutional_neural_network
6. <http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/>
7. <http://cs231n.github.io/convolutional-networks/>

8. <https://www.quora.com/What-is-dropout-in-deep-learning>
9. <https://keras.io/layers/core/#flatten>
10. <https://www.quora.com/What-are-softmax-layers>
11. <https://www.kaggle.com/romanroibu/digit-recognizer/random-forest-digit-classifier>
12. <https://arxiv.org/abs/1512.03385>
13. http://www.robots.ox.ac.uk/~vgg/research/very_deep/
14. <https://github.com/maxpumperla/hyperas>
15. <https://keras.io/optimizers/>