HANDWRITTEN DIGIT CLASSIFICATION APP

CREATED USING CNN ALONG WITH MNIST DATASET

RISHABH ARYAN DAS
rishabh.das@ufl.edu

OVERVIEW

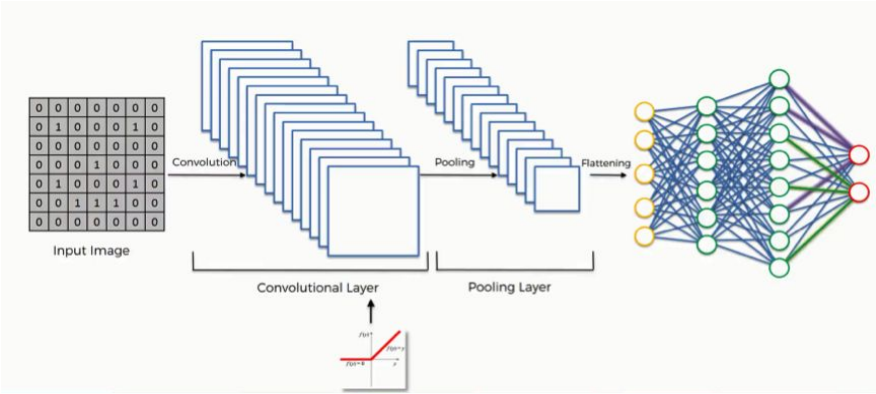
- Why is handwritten digit recognition required?
- MNIST dataset, it’s modelling and preprocessing
- Handwritten digit recognition using deep learning: CNN(Keras)
- References

WHY HANDWRITTEN DIGIT CLASSIFICATION?

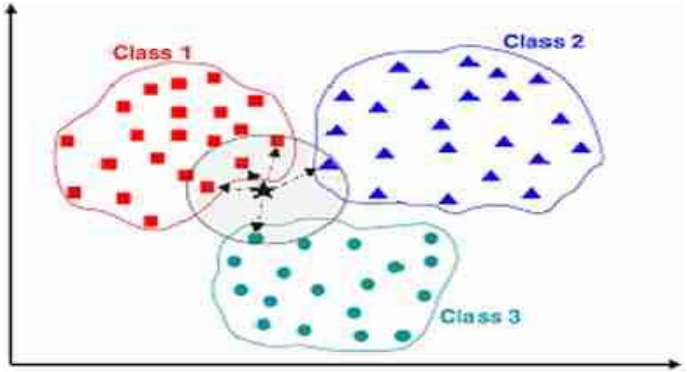
- Data entry for business documents like check, passport, invoice, bank statement and receipt
- Automatic number plate identification
- Insurance document key information extraction
- Extracting business card information into contact list
- Creating textual version of printed documents, eg Book Scanning
- To search electronic images of printed documents, eg Google Books
- Conversion of handwritten documents into digital data (Pen computing)
- Overcoming CAPTCHA anti-bot systems
- Assistive technology for blind/visually impaired users

METHODS AVAILABLE FOR HANDWRITTEN DIGIT CLASSIFICATION

- Machine Learning
  1. Supervised ML
  2. K nearest neighbours
- Deep Learning
  1. Convolutional Neural Networks

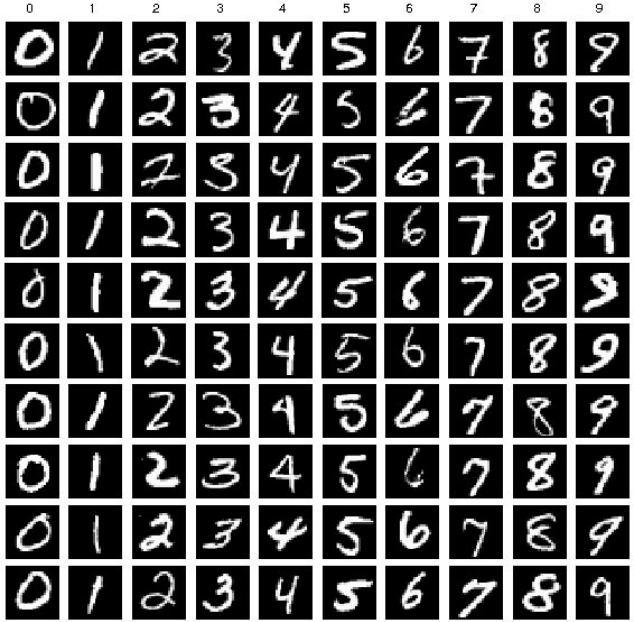


Tools used for implementation are Keras, Tensorflow, Android Studio.



MNIST DATASET, DATA MODELLING AND PRE-PROCESSING

- MNIST dataset is a subset of larger dataset NIST
- MNIST contains 70k examples of handwritten digits divided as training set of 60k examples and a test set of 10k examples
- MNIST dataset is available in four files:
  1. train-images-idx3-ubyte: Training set images
  2. train-labels-idx1-ubyte: Training set labels
  3. t10k-images-idx3-ubyte: Test set images
  4. t10k-labels-idx2-ubyte: Test set labels



READING MNIST DATASETS

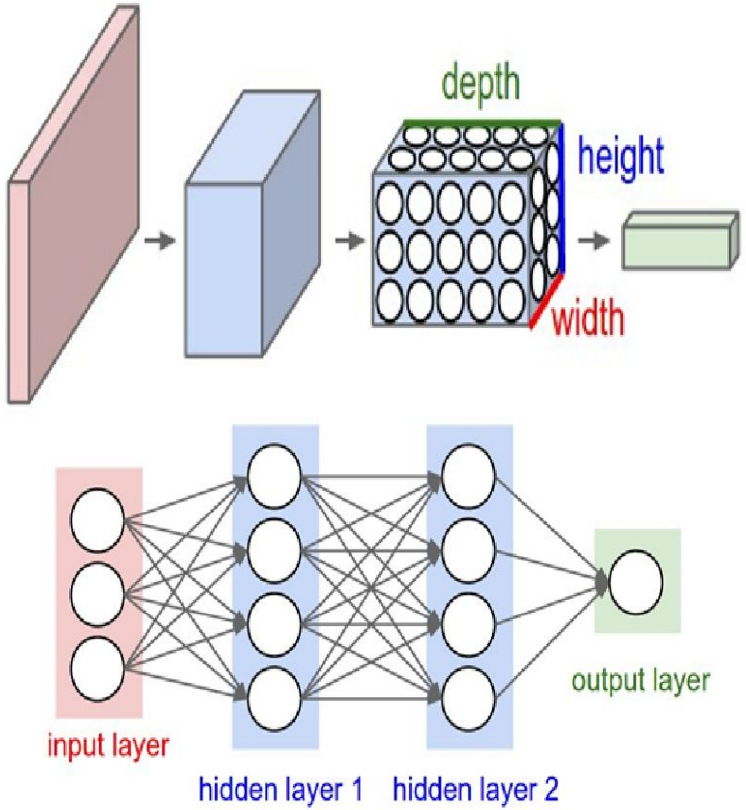
- MNIST dataset is provided in a specific format
- Each file has a specific magic number value that identifies the file
- The second row indicates the total number of images that are present in the file
- The third and fourth rows tell the number of rows and columns in dataset file
- The digits are available as an array of 784 elements/pixels corresponding to each label
- These elements/pixels are read into 28x28 matrix to be converted into image
- The digit itself occupies the central 20x20 pixels and the center of mass lies at the center of the box

HANDWRITTEN DIGIT CLASSIFICATION USING DEEP LEARNING

- One of the most commonly used Deep Learning algorithms is the Convolutional Neural Network
- Several tools are available for creating deep learning models such as Tensorflow, Keras, PyTorch, Theano, PyBrains, TFLearn, etc
- Here CNN is implemented using tensorflow with Keras

CONVOLUTIONAL NEURAL NETWORK

- CNN are made up of neurons that have learnable weights and biases
- Each neurons receives some input performs a dot product and optionally follows it with a non-linearity
- The whole network expresses a single differentiable score function from the raw image pixels on one end to class scores at the other. It uses a loss function (eg Softmax) on the last layer.
- The layers of a ConvNet have neurons arranged in 3 dimensions: width, height and depth

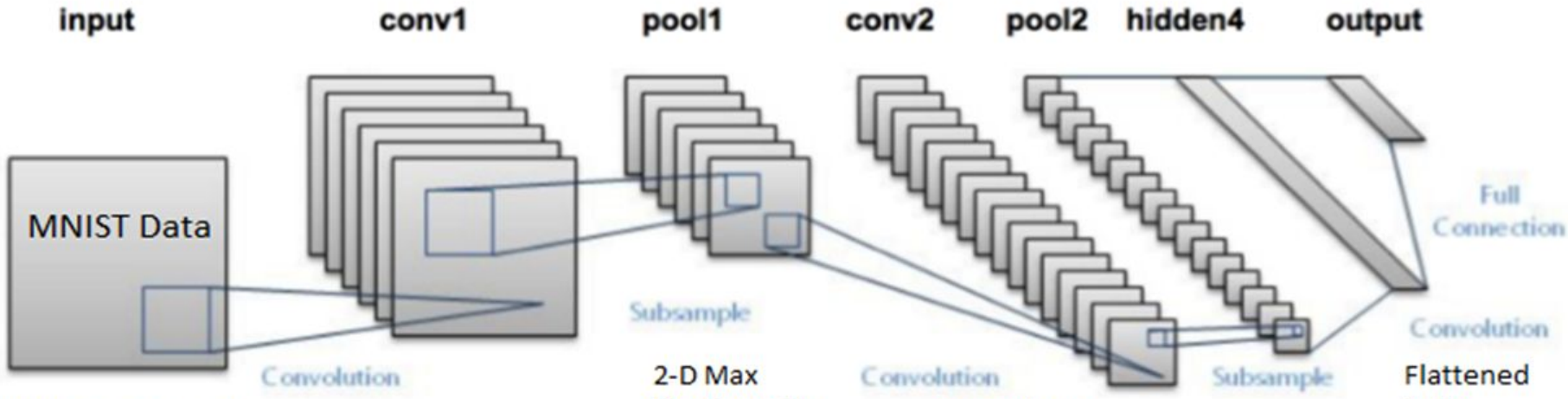


LAYERS OF CONVOLUTIONAL NEURAL NETWORK

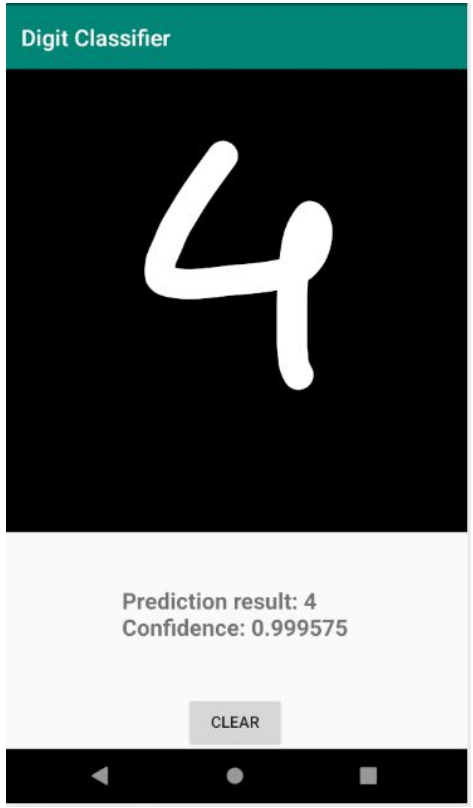
Types of layers used to build a CNN are as follows:

- **Input**: holds the raw pixel values of image [28x28x3]
- **Conv2D**: computes the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. This may result in volume such as [28x28x64] where 64 is the number of filters we are using in the code
- **Rectified Linear Unit (ReLU)**: applies an element wise activation function such as the max(0,x)max(0,x) thresholding at zero which leaves the size of the volume unchanged
- **Pooling layer**: perform a down-sampling operation along the spatial dimensions like width and height resulting in volume such as [14x14x64]
- **Fully Connected Layer**: compute the class scores, resulting in volume of size [1x1x10] where each of the 10 numbers correspond to a class score such as among the 10 classes of MNIST.

CNN LAYERS FOR HANDWRITTEN DIGIT CLASSIFICATION



SAMPLE PREDICTION



MODEL PARAMETERS

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 352s 6ms/step - loss: 0.1338 - accuracy: 0.9573 - val_loss: 0.0324 - val_accuracy: 0.9891
Epoch 2/10
60000/60000 [==============================] - 346s 6ms/step - loss: 0.0352 - accuracy: 0.9886 - val_loss: 0.0285 - val_accuracy: 0.9905
Epoch 3/10
60000/60000 [==============================] - 351s 6ms/step - loss: 0.0244 - accuracy: 0.9920 - val_loss: 0.0204 - val_accuracy: 0.9932
Epoch 4/10
60000/60000 [==============================] - 345s 6ms/step - loss: 0.0181 - accuracy: 0.9945 - val_loss: 0.0246 - val_accuracy: 0.9920
Epoch 5/10
60000/60000 [==============================] - 344s 6ms/step - loss: 0.0133 - accuracy: 0.9958 - val_loss: 0.0264 - val_accuracy: 0.9927
Epoch 6/10
60000/60000 [==============================] - 344s 6ms/step - loss: 0.0121 - accuracy: 0.9960 - val_loss: 0.0269 - val_accuracy: 0.9917
Epoch 7/10
60000/60000 [==============================] - 348s 6ms/step - loss: 0.0097 - accuracy: 0.9970 - val_loss: 0.0239 - val_accuracy: 0.9927
Epoch 8/10
60000/60000 [==============================] - 343s 6ms/step - loss: 0.0101 - accuracy: 0.9966 - val_loss: 0.0290 - val_accuracy: 0.9902
Epoch 9/10
60000/60000 [==============================] - 348s 6ms/step - loss: 0.0079 - accuracy: 0.9974 - val_loss: 0.0246 - val_accuracy: 0.9927
Epoch 10/10
60000/60000 [==============================] - 343s 6ms/step - loss: 0.0048 - accuracy: 0.9983 - val_loss: 0.0322 - val_accuracy: 0.9923
Evaluating Accuracy and Loss Function...
10000/10000 [==============================] - 13s 1ms/step
Accuracy of Model: 99.23%
```

TECHNOLOGY USED FOR THIS PROJECT

- Language: Python
- IDE: Android Studio , Anaconda
- ML Library: Tensorflow , Keras