DIGIT RECOGNITION

USING

CONVOLUTIONAL NEURAL NETWORK

(CNN)

BY

Er. SEASON MAHARJAN

Table of Contents

INTRODUCTION

Background

This project deals with the 5 layers Sequential Convolutional Neural Network for digits recognition trained on MNIST (Modified National Institute of Standards and Technology) dataset from kaggle. It is a concrete case of Deep Learning neural networks, which is popular when dealing with achieving very accurate results regarding image recognition. Here we take the MNIST dataset and perform the processing tasks, and perform some explorations to the dataset. Now we train the models using the processed data set and apply convolution neural network processes for digit recognition and also the implementation of Keras using TensorFlow backend.

Why Convolutional Neural Networks Are So Important

- Because when it comes to Image Recognition, then CNN's are the best.
- It became successful in the late 90's after Yann LeCun used it on MNIST and achieved 99.5% accuracy.
- You can try other Models like Support Vector Machines, Decision Tree, K-Nearest Neighbour, Random Forest but the accuracy achieved is 96-97%, which is not that good.
- The Biggest Challenge is picking the Right model by understanding the Data rather than Tuning parameters of other models.
- And the last point, a large Training data really helps in improving Neural Networks Accuracy.

Libraries used

1. **Numpy**: The fundamental package for scientific computing with python. Working with Numpy arrays.
2. **Pandas**: Library for python programming language for data manipulation and analysis. Working with csv files and data frames.
3. **Seaborn**: Python data visualization library based on Matplotlib. Working with informative statistical graphics.
4. **Matplotlib**: A plotting library for python, it's a numerical mathematics extension Numpy. Working with pyplot.
5. **Sklearn** (**Scikit**): Machine learning library for python. Working with data analysis.
6. **Keras**: Neural network library. Working on top of TensorFlow backend.
7. **TensorFlow**: Symbolic math library for dataflow and differentiable programming across a range. Working with neural networks.

Import Libraries

```
In [2]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import matplotlib.image as mpimg
        import seaborn as sns
        %matplotlib inline
```

```
In [3]: np.random.seed(2)
```

sklearn, keras

```
In [10]: from sklearn.model_selection import train_test_split
         from sklearn.metrics import confusion_matrix
         import itertools
         from keras.utils.np_utils import to_categorical # convert to one-hot-encoding
         from keras.models import Sequential
         from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D
         from keras.optimizers import RMSprop
         from keras.preprocessing.image import ImageDataGenerator
         from keras.callbacks import ReduceLROnPlateau


         sns.set(style='white', context='notebook', palette='deep')
```

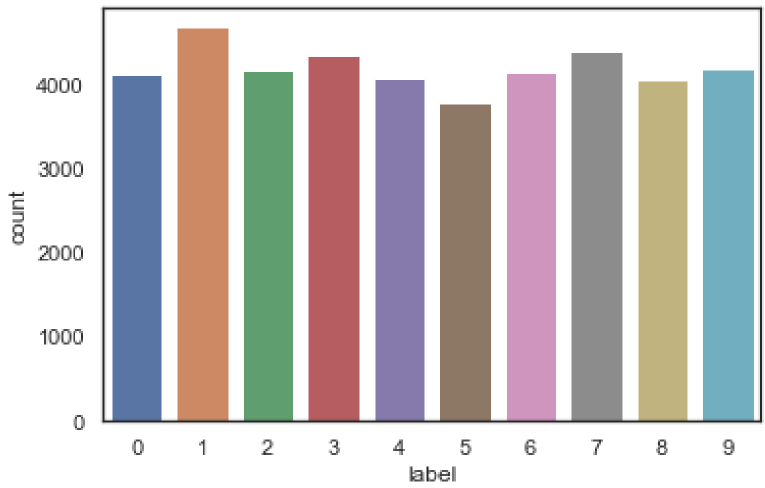
```
Using TensorFlow backend.
```

DATA PREPARATION

In [ ]:
```
# Loading train and test data
train = pd.read_csv("train.csv")
test = pd.read_csv("test.csv")
```

In [12]:
```
Y_train = train["label"]

# Drop 'label' column
X_train = train.drop(labels = ["label"],axis = 1)

# free some space
del train

g = sns.countplot(Y_train)

Y_train.value_counts()
```

Out[12]:
```
1    4684
7    4401
3    4351
9    4188
2    4177
6    4137
0    4132
4    4072
8    4063
5    3795
Name: label, dtype: int64
```



Checking for missing values

Here I checked for the corrupted images i.e. missing values inside but there were no any missing values found in both test and train datasets.

```
In [13]: # Check the data
         X_train.isnull().any().describe()

Out[13]: count      784
         unique       1
         top      False
         freq       784
         dtype: object

In [14]: test.isnull().any().describe()

Out[14]: count      784
         unique       1
         top      False
         freq       784
         dtype: object
```

Normalization

To reduce the effect of illumination's differences, I have performed normalization. By looking at the CNN converging faster to [0..1] than to [0..255], here we divide both train and test with 255.

```
In [15]: # Normalize the data
         X_train = X_train / 255.0
         test = test / 255.0
```

Reshaping

Train and test images (28px x 28px) has been stock into pandas. Dataframe as 1D vectors of 784 values. We reshape all data to 28x28x1 3D matrices.

MNIST images are grey scaled so it use only one channel. For RGB images, there is 3 channels, we would have reshaped 784px vectors to 28x28x3 3D matrices.

```
# Reshape image in 3 dimensions (height = 28px, width = 28px , canal = 1)
X_train = X_train.values.reshape(-1,28,28,1)
test = test.values.reshape(-1,28,28,1)
```

Label Encoding

Here we use one hot encoding for labels as labels are numbers from 0 to 9.

```
# Encoding labels to one hot vectors
Y_train = to_categorical(Y_train, num_classes = 10)
```

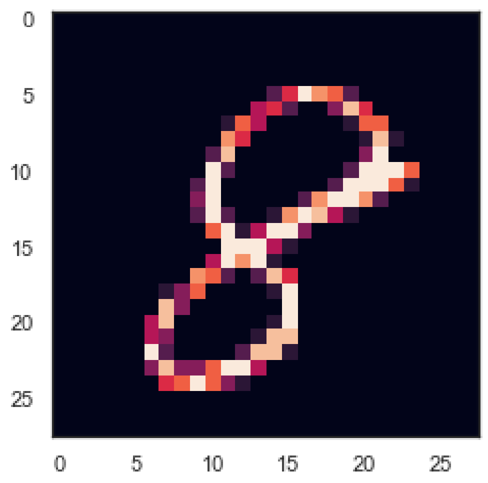
Training and validation sets

Here I have split the train set into two parts for validation and train sets i.e. 10% of train data for validation and rest for training the model.

```
# Split the train and the validation set for the fitting
X_train, X_val, Y_train, Y_val = train_test_split(X_train, Y_train, test_size = 0.1, random_state=random_seed)
```

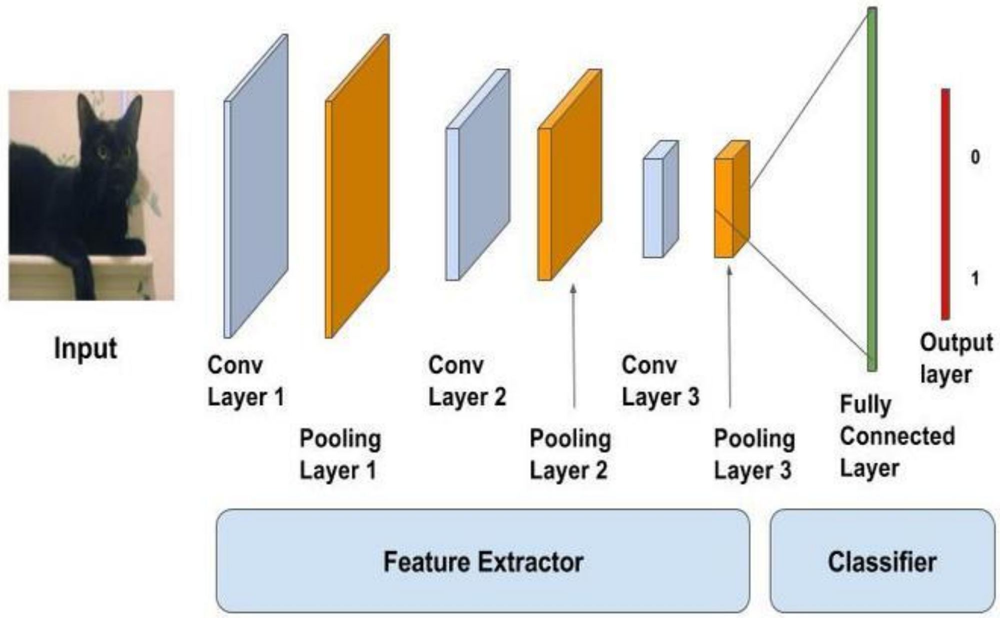
In [20]:

```
g = plt.imshow(X_train[0][:,:,0])
```

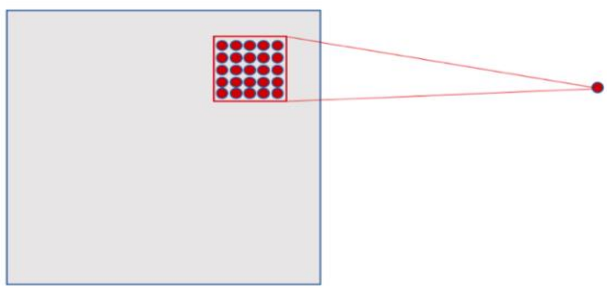


CNN (Convolutional Neural Network)

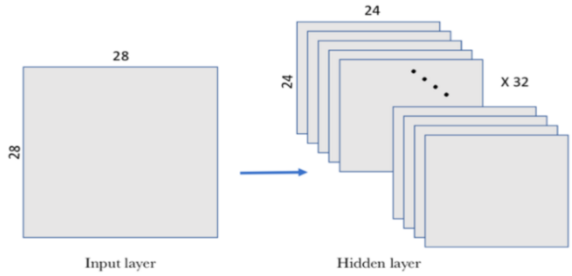
Process of CNN: following diagram is an example shown where input is the image of a cat and various layers on convolutions and pooling is shown.



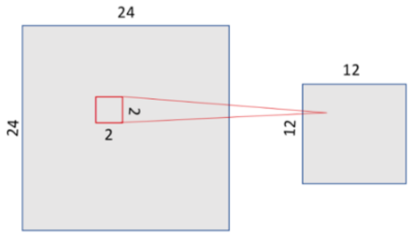
In the case of MNIST, as input to our neural network we can think of a space of two-dimensional neurons 28×28 (height = 28, width = 28, depth = 1). A first layer of hidden neurons connected to the neurons of the input layer that we have discussed will perform the convolutional operations that we have just described.



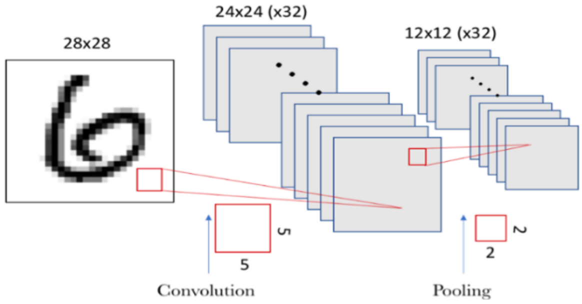
In this example, the first convolutional layer receives a size input tensor (28, 28, 1) and generates a size output (24, 24, 32), a 3D tensor containing the 32 outputs of 24×24 pixel result of computing the 32 filters on the input.



Max pooling: In our example, we are going to choose a 2×2 window of the convolutional layer and we are going to synthesize the information in a point in the pooling layer. Visually, it can be expressed as follows:



As mentioned above, the convolutional layer hosts more than one filter and, therefore, as we apply the max-pooling to each of them separately, the pooling layer will contain as many pooling filters as there are convolutional filters:



The result is, since we had a space of 24×24 neurons in each convolutional filter, after doing the pooling we have 12×12 neurons which corresponds to the 12×12 regions (of size 2×2 each region) that appear when dividing the filter space.

Defining the model

```python
# Set the CNN model
# my CNN architechture is In -> [[Conv2D->relu]*2 -> MaxPool2D -> Dropout]*2 -> Flatten -> Dense -> Dropout -> Out

model = Sequential()

model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',
                 activation ='relu', input_shape = (28,28,1)))
model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',
                 activation ='relu'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.25))


model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',
                 activation ='relu'))
model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',
                 activation ='relu'))
model.add(MaxPool2D(pool_size=(2,2), strides=(2,2)))
model.add(Dropout(0.25))


model.add(Flatten())
model.add(Dense(256, activation = "relu"))
model.add(Dropout(0.5))
model.add(Dense(10, activation = "softmax"))
```

- Here I have used the keras sequential API, the first is the convolutional 2D layer which are like the set of learnable filters and set the first two conv2D layers with 32 filters and last two with 64 filters. Each filter transforms a part of the image using the kernel filter. The kernel filter matrix is applied on the whole image. Filters can be seen as a transformation of the image. The CNN can isolate features that are useful everywhere from these transformed images.
- The second important layer in CNN is the pooling layer (MaxPool2D). This layer simply acts as a downsampling filter. It works at the 2 neighbouring pixels and picks the maximum value. These are used to reduce computation cost and also reduce overfitting. We have to choose the pooling size more the pooling dimension is high more the downsampling is important.
- Combining conv2D and Maxpool2D layers, CNN are able to combine local features and learn more global features to the image. Dropout is a regularization method, where a proportion of nodes in the layer are randomly ignored i.e. setting weights to zero for each training sample. This drops randomly a proportion of the network and forces the network to learn features in a distributed way. This technique also improves generalization and reduces the overfitting.

- ‘relu’ is the rectifier, a rectifier activation function is used to add non linearity to the network. The flatten layer is use to convert the final feature maps into one single 1D vector. This flattering step is needed so that you can make use of fully connected layers after some convolutional/maxpool layers. It combines all the found local features of the previous convolutional layers.
- In the end, I used the features in two fully connected i.e. dense layers which is just an artificial neural network classifier (ANN). In the last layer (Dense (10, activation =”softmax”)) the net outputs distribution of probability of each class.

Set the optimizer and annealer

- Once our layers are added to the model, we need to set up a score function, a loss function and an optimisation algorithm. We define the loss function to measure how poorly our model performs on images with known labels. It is the error rate between the observed labels and the predicted ones. We use a specific form for categorical classifications (>2 classes) called the "categorical_crossentropy".

- The most important function is the optimizer. This function will iteratively improve parameters (filters kernel values, weights and bias of neurons ...) in order to minimise the loss.

- I choose RMSprop (with default values), it is a very effective optimizer. The RMSprop update adjusts the Adagrad method in a very simple way in an attempt to reduce its aggressive, monotonically decreasing learning rate. We could also have used Stochastic Gradient Descent ('sgd') optimizer, but it is slower than RMSprop. The metric function "accuracy" is used is to evaluate the performance our model. This metric function is similar to the loss function, except that the results from the metric evaluation are not used when training the model (only for evaluation).

In [13]:
```
# Define the optimizer
optimizer = RMSprop(lr=0.001, rho=0.9, epsilon=1e-08, decay=0.0)
```

In [14]:
```
# Compile the model
model.compile(optimizer = optimizer , loss = "categorical_crossentropy", metrics=["accuracy"])
```

In order to make the optimizer converge faster and closest to the global minimum of the loss function, i used an annealing method of the learning rate (LR).

The LR is the step by which the optimizer walks through the 'loss landscape'. The higher LR, the bigger are the steps and the quicker is the convergence. However the sampling is very poor with a high LR and the optimizer could probably fall into a local minima.

It’s better to have a decreasing learning rate during the training to reach efficiently the global minimum of the loss function.

To keep the advantage of the fast computation time with a high LR, i decreased the LR dynamically every X steps (epochs) depending if it is necessary (when accuracy is not improved).

With the ReduceLROnPlateau function from Keras.callbacks, I choose to reduce the LR by half if the accuracy is not improved after 3 epochs.

```
# Set a learning rate annealer
learning_rate_reduction = ReduceLROnPlateau(monitor='val_acc',
                                            patience=3,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=0.00001)
```

```
epochs = 1
batch_size = 86
```

Data Augmentation

Approaches that alter the training data in ways that change the array representation while keeping the label same are known as data augmentation techniques. Some popular augmentations people use are grayscales, horizontal flips, vertical flips, random crops, colour jitters, translations, rotations, and much more.

By applying just a couple of these transformations to our training data, we can easily double or triple the number of training examples and create a very robust model.

```
datagen = ImageDataGenerator(
        featurewise_center=False,  # set input mean to 0 over the dataset
        samplewise_center=False,  # set each sample mean to 0
        featurewise_std_normalization=False,  # divide inputs by std of the dataset
        samplewise_std_normalization=False,  # divide each input by its std
        zca_whitening=False,  # apply ZCA whitening
        rotation_range=10,  # randomly rotate images in the range (degrees, 0 to 180)
        zoom_range = 0.1, # Randomly zoom image
        width_shift_range=0.1,  # randomly shift images horizontally (fraction of total width)
        height_shift_range=0.1,  # randomly shift images vertically (fraction of total height)
        horizontal_flip=False,  # randomly flip images
        vertical_flip=False)  # randomly flip images


datagen.fit(X_train)
```

For the data augmentation, I choose to:

- Randomly rotate some training images by 10 degrees
- Randomly Zoom by 10% of some training images
- Randomly shift images horizontally by 10% of the width
- Randomly shift images vertically by 10% of the height

I did not apply a vertical flip nor horizontal flip since it could have led to misclassify symmetrical numbers such as 6 and 9.

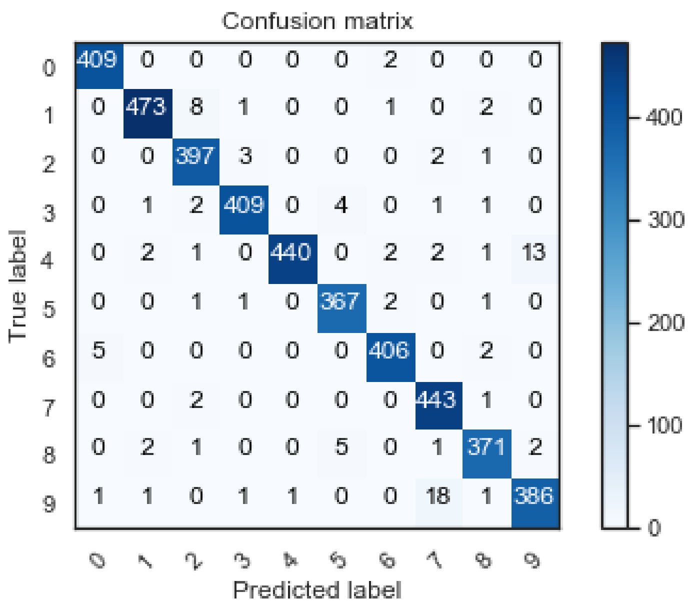
```
# Fit the model
history = model.fit_generator(datagen.flow(X_train,Y_train, batch_size=batch_size),
                              epochs = epochs, validation_data = (X_val,Y_val),
                              verbose = 2, steps_per_epoch=X_train.shape[0] // batch_size
                              , callbacks=[learning_rate_reduction])
```

EVALUATE THE MODEL

Confusion matrix

Confusion matrix can be very helpful to see your model drawbacks. I plot the confusion matrix of the validation results.

```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

# Predict the values from the validation dataset
Y_pred = model.predict(X_val)
# Convert predictions classes to one hot vectors
Y_pred_classes = np.argmax(Y_pred,axis = 1)
# Convert validation observations to one hot vectors
Y_true = np.argmax(Y_val,axis = 1)
# compute the confusion matrix
confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)
# plot the confusion matrix
plot_confusion_matrix(confusion_mtx, classes = range(10))
```

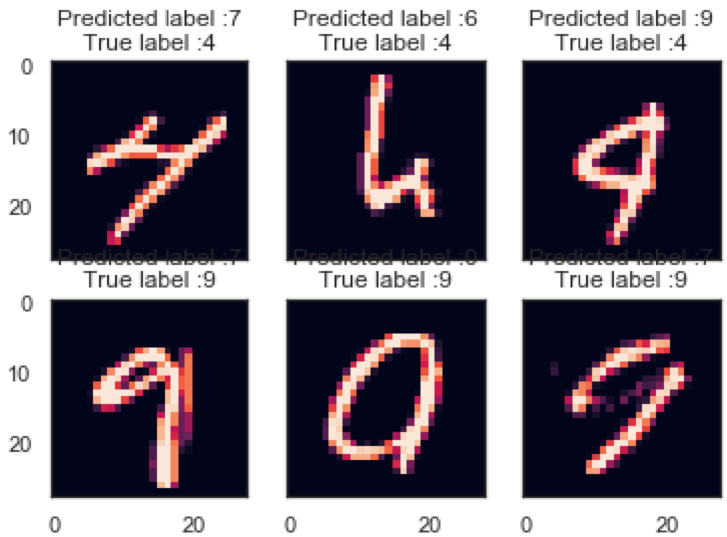


Here we can see that our CNN performs very well on all digits with few errors considering the size of the validation set (4200 images).

However, it seems that our CNN has some little troubles with the 4 digits, hey are misclassified as 9. Sometime it is very difficult to catch the difference between 4 and 9 when curves are smooth.

Displaying Errors

```
In [31]: errors = (Y_pred_classes - Y_true != 0)
Y_pred_classes_errors = Y_pred_classes[errors]
Y_pred_errors = Y_pred[errors]
Y_true_errors = Y_true[errors]
X_val_errors = X_val[errors]
def display_errors(errors_index,img_errors,pred_errors, obs_errors):
    """ This function shows 6 images with their predicted and real labels"""
    n = 0
    nrows = 2
    ncols = 3
    fig, ax = plt.subplots(nrows,ncols,sharex=True,sharey=True)
    for row in range(nrows):
        for col in range(ncols):
            error = errors_index[n]
            ax[row,col].imshow((img_errors[error]).reshape((28,28)))
            ax[row,col].set_title("Predicted label :{}\nTrue label :{}".format(pred_errors[error],obs_errors[error]))
            n += 1
# Probabilities of the wrong predicted numbers
Y_pred_errors_prob = np.max(Y_pred_errors,axis = 1)
# Predicted probabilities of the true values in the error set
true_prob_errors = np.diagonal(np.take(Y_pred_errors, Y_true_errors, axis=1))
# Difference between the probability of the predicted label and the true label
delta_pred_true_errors = Y_pred_errors_prob - true_prob_errors
# Sorted list of the delta prob errors
sorted_dela_errors = np.argsort(delta_pred_true_errors)
# Top 6 errors
most_important_errors = sorted_dela_errors[-6:]
# Show the top 6 errors
display_errors(most_important_errors, X_val_errors, Y_pred_classes_errors, Y_true_errors)
```



For those six case, the model is not ridiculous. Some of these errors can also be made by humans, especially for one the 9 that is very close to a 4. The last 9 is also very misleading, it seems for me that is a 0.

PREDICTION AND SUBMISSION

```
# predict results
results = model.predict(test)

# select the indix with the maximum probability
results = np.argmax(results,axis = 1)

results = pd.Series(results,name="Label")
```

```
submission = pd.concat([pd.Series(range(1,28001),name = "ImageId"),results],axis = 1)

submission.to_csv("cnn_mnist_datagen.csv",index=False)
```

OTHER MODELS

1. Decision Tree Classifier

I had performed decision tree classifier for the same MNIST dataset and observed the differences in results.

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier

from sklearn.model_selection import train_test_split
#constants

IMG_HEIGHT=28
IMG_WIDTH=28

# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list the files in the input directory
```

```
loaded_images=pd.read_csv('train.csv')
loaded_images.head()
```

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781	pix
0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	
2	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	
3	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	

5 rows × 785 columns

```
images=loaded_images.iloc[:,1:]
labels=loaded_images.iloc[:,:1]   # for the labels to be a dataframe . iloc[:,0] returns a Series  iloc[:,:1] returns a
labels.head()
```

	label
0	1
1	0
2	1
3	4
4	0

```
train_images,test_images,train_labels,test_labels=train_test_split(images,labels,test_size=0.2,random_state=13)
```

```
train_images.describe()
```

	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel774	pixel775	pixel776	pixel777	
count	33600.0	33600.0	33600.0	33600.0	33600.0	33600.0	33600.0	33600.0	33600.0	33600.0	...	33600.000000	33600.000000	33600.000000	33600.000000	33
mean	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.217321	0.115327	0.056161	0.024405	
std	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	6.336594	4.529554	3.279590	1.964475	
min	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.000000	0.000000	0.000000	
25%	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.000000	0.000000	0.000000	
50%	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.000000	0.000000	0.000000	
75%	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.000000	0.000000	0.000000	
max	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	254.000000	254.000000	253.000000	253.000000	

8 rows × 784 columns

```
tree=DecisionTreeClassifier(criterion='gini',random_state=1)
tree.fit(train_images,train_labels)
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=1,
            splitter='best')
```

```
tree.score(train_images,train_labels.values.ravel())
```

```
1.0
```

```
tree.score(test_images,test_labels.values.ravel())
```

```
0.854047619047619
```

```python
new_data=pd.read_csv('test.csv')
new_data.head(n=3)
```

	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel774	pixel775	pixel776	pixel777	pixel778
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0

3 rows × 784 columns

```python
y_pred=tree.predict(new_data)
```

```python
y_pred.shape
```

```
(28000,)
```

```python
submissions=pd.DataFrame({"ImageId":list(range(1,len(y_pred)+1)), "Label":y_pred})
submissions.head()
```

	ImageId	Label
0	1	2
1	2	2
2	3	9
3	4	9
4	5	8

```python
submissions.to_csv("mnist_decision_tree_submit.csv",index=False,header=True)
```

The score obtained after submission in kaggle was 85.142% from decision tree classifier

2. SVM (Support Vector Machine)

Using support vector machine for the same MNIST image dataset and observing the score by performing the kaggle submission.

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt, matplotlib.image as mpimg
from sklearn.model_selection import train_test_split
from sklearn import svm
%matplotlib inline
```

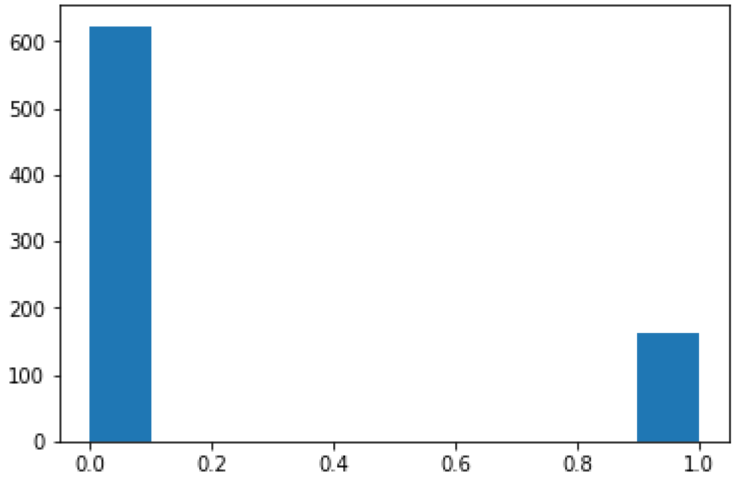
```
labeled_images = pd.read_csv('train.csv')
labels = labeled_images.iloc[:, 0]
images = labeled_images.iloc[:, 1:]
train_images, test_images,train_labels, test_labels = train_test_split(images, labels, train_size=0.8, random_state=0)
```

```
D:\pythonsnmn\lib\site-packages\sklearn\model_selection\_split.py:2179: FutureWarning: From version 0.21, test_size w
ill always complement train_size unless both are specified.
  FutureWarning)
```

```
train_images.iloc[train_images>0] = 1
test_images.iloc[test_images>0] = 1
plt.hist(train_images.iloc[5])
```

```
D:\pythonsnmn\lib\site-packages\pandas\core\indexing.py:189: SettingWith
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-dc
copy
  self._setitem_with_indexer(indexer, value)
D:\pythonsnmn\lib\site-packages\ipykernel_launcher.py:1: SettingWithCopy
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-dc
copy
  """Entry point for launching an IPython kernel.
D:\pythonsnmn\lib\site-packages\ipykernel_launcher.py:2: SettingWithCopy
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-dc
copy
```

```
(array([623.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0., 161.]),
 array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ]),
 <a list of 10 Patch objects>)
```




```
: clf = svm.SVC()
  clf.fit(train_images, train_labels.values.ravel())
  clf.score(test_images,test_labels)

  D:\pythonsnmn\lib\site-packages\sklearn\svm\base.py:196: Fu
  'auto' to 'scale' in version 0.22 to account better for uns
  to avoid this warning.
    "avoid this warning.", FutureWarning)

: 0.9428571428571428
```

```
: test_data=pd.read_csv('test.csv')
  test_data[test_data>0]=1
  results=clf.predict(test_data)
  results

: array([2, 0, 9, ..., 3, 9, 2], dtype=int64)
```

```
: df = pd.DataFrame(results)
  df.index.name='ImageId'
  df.index+=1
  df.columns=['Label']
  df.to_csv('results.csv', header=True)
```

The obtained score after kaggle submission using SVM was 93.7%.

3. Random Forest

Using Random Forest for the same MNIST image dataset and observing the score by performing the kaggle submission.

```
import numpy as np # linear algebra
import pandas as pd
```

```
%matplotlib notebook
import matplotlib.pyplot as plt
```

```
data = pd.read_csv("train.csv")
data.head()
```

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixe
0	1	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	
2	1	0	0	0	0	0	0	0	
3	4	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	

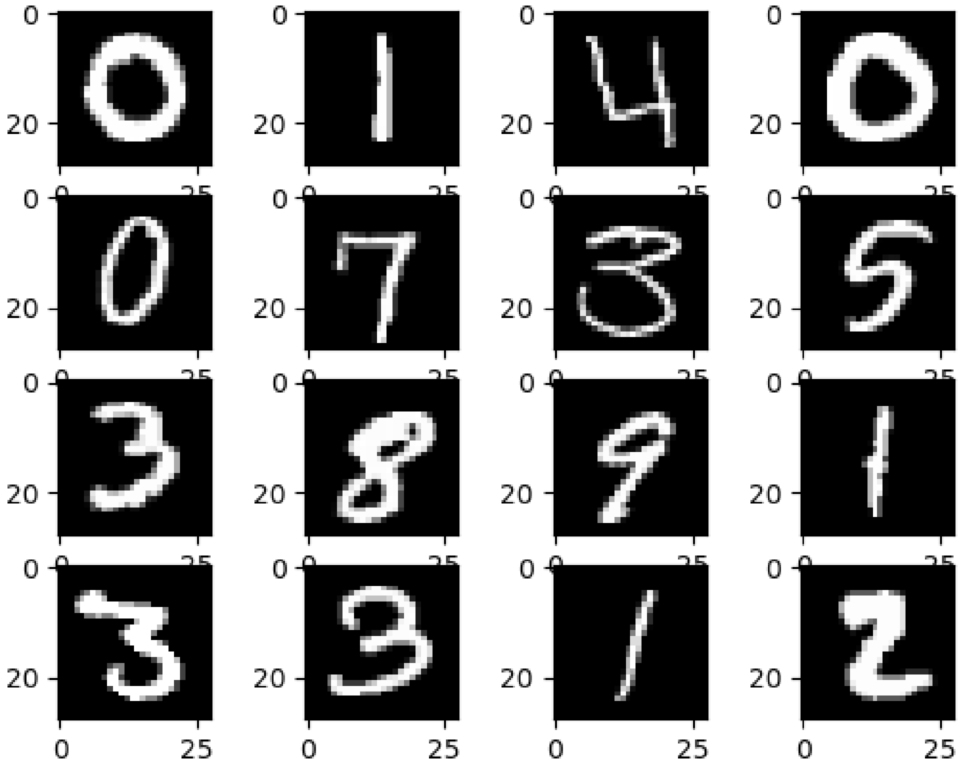
5 rows × 785 columns

```
L = np.sqrt(784)
L
```

```
28.0
```

```
def plotNum(ind):
    plt.imshow(np.reshape(np.array(data.iloc[ind,1:]), (28, 28)), cmap="gray")
```

```
plt.figure()
for ii in range(1,17):
    plt.subplot(4,4,ii)
    plotNum(ii)
```



```
X = data.iloc[:, 1:]
y = data['label']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
```

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(n_jobs=-1, n_estimators=10)
rfc.fit(X_train, y_train)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=-1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

```
rfc.score(X_test, y_test)
```

```
0.9377142857142857
```

```
unknown = pd.read_csv("test.csv")
```

```
y_out = rfc.predict(unknown)
y_out
```

```
array([2, 0, 4, ..., 3, 9, 2], dtype=int64)
```

```
Label = pd.Series(y_out,name = 'Label')
ImageId = pd.Series(range(1,28001),name = 'ImageId')
submission = pd.concat([ImageId,Label],axis = 1)
submission.to_csv('submissionRandom.csv',index = False)
```

The Score observed after the submission using Random Forest is 93.5%

COMPARISION AMONG DIFFERENT MODEL

Models	Scores
Decision Tree Classifier	85.14%
SVM(Support Vector Machines)	93.7%
Random Forest	93.5%
CNN(Convolutional Neural Network)	97.45%

The submission scores can be viewed on kaggle Digit Recognizer competition submissions. The link is https://www.kaggle.com/c/digit-recognizer/submissions. The accuracies can be observed by above table as the highest score obtained by CNN method.

CONCLUSION

This project involved analysis of image data's i.e. MNIST data's using various models. The most accurate model found out to be is CNN using tensor flow background for large datasets. The accuracy results is 97.45%.

REFERENCES

1. https://www.kaggle.com/fuzzywizard/beginners-guide-to-cnn-accuracy-99-7
2. https://pdfs.semanticscholar.org/5d79/11c93ddcb34cac088d99bd0cae9124e5dcd1.pdf
3. https://towardsdatascience.com/building-a-convolutional-neural-network-cnn-in-keras-329fbbadc5f5