**DIGIT RECOGNITION**

**USING**

**CONVOLUTIONAL NEURAL NETWORK**

**(CNN)**

**BY**

**Er. SEASON MAHARJAN**

Table of Contents

[**INTRODUCTION** 3](#_Toc9863346)

[**Background** 3](#_Toc9863347)

[**Why Convolutional Neural Networks Are So Important** 3](#_Toc9863348)

[**Libraries used** 4](#_Toc9863349)

[**Import Libraries** 5](#_Toc9863350)

[**DATA PREPARATION** 6](#_Toc9863351)

[**Checking for missing values** 7](#_Toc9863352)

[**Normalization** 7](#_Toc9863353)

[**Reshaping** 8](#_Toc9863354)

[**Label Encoding** 8](#_Toc9863355)

[**Training and validation sets** 9](#_Toc9863356)

[**CNN (Convolutional Neural Network)** 10](#_Toc9863357)

[**Process of CNN** 10](#_Toc9863358)

[**Defining the model** 12](#_Toc9863359)

[**Set the optimizer and annealer** 13](#_Toc9863360)

[**Data Augmentation** 15](#_Toc9863361)

[**EVALUATE THE MODEL** 16](#_Toc9863362)

[**Confusion matrix** 16](#_Toc9863363)

[**Displaying Errors** 18](#_Toc9863364)

[**PREDICTION AND SUBMISSION** 19](#_Toc9863365)

[**OTHER MODELS** 20](#_Toc9863366)

[**1. Decision Tree Classifier** 20](#_Toc9863367)

[**2. SVM (Support Vector Machine)** 23](#_Toc9863368)

[**3. Random Forest** 25](#_Toc9863369)

[**COMPARISION AMONG DIFFERENT MODEL** 28](#_Toc9863370)

[**CONCLUSION** 28](#_Toc9863371)

[**REFERENCES** 28](#_Toc9863372)

# **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.

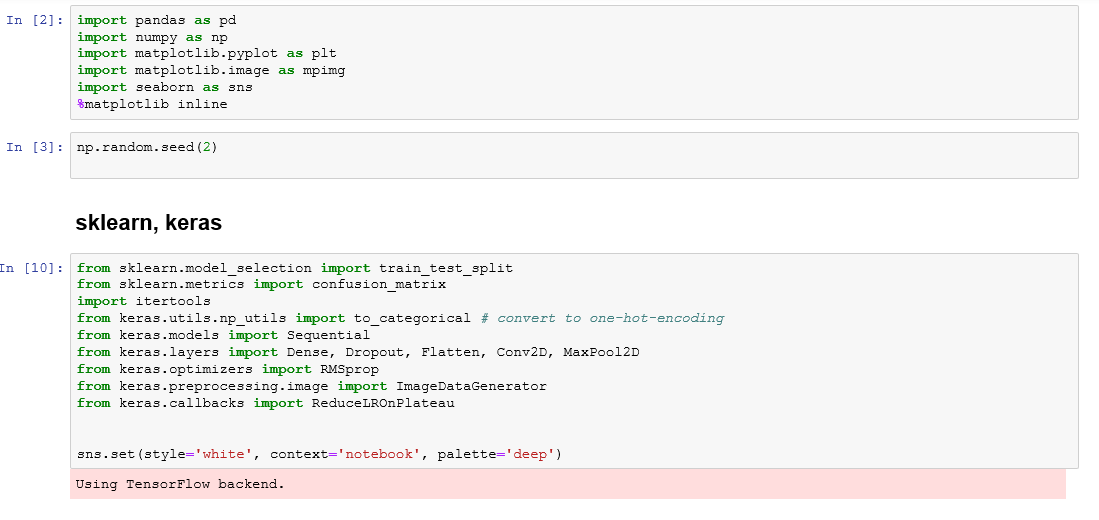
## **Why Convolutional Neural Networks Are So Important**

* 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.

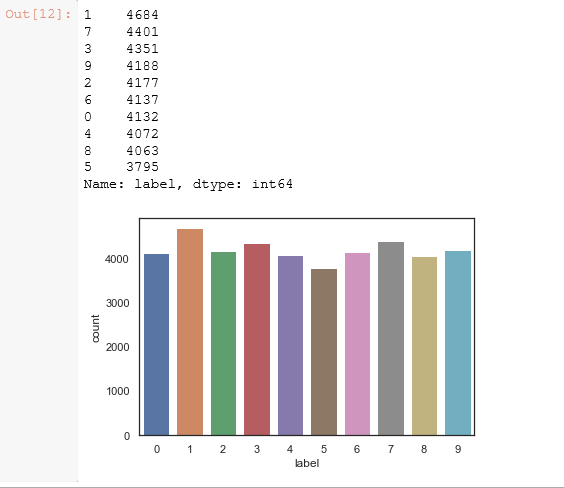
## **Libraries used**

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**

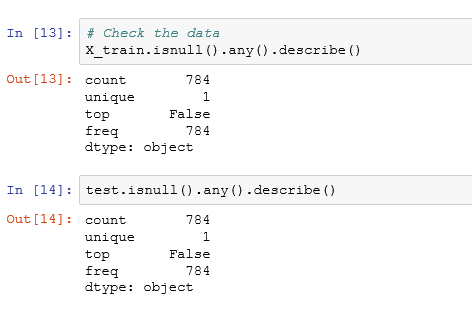


# **DATA PREPARATION**



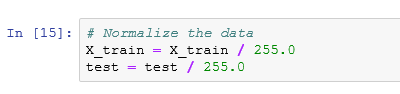
## **Checking for missing values**

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.



## **Normalization**

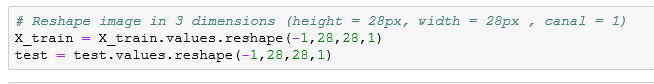
To reduce the effect of illumination’s differences, I have performed normalization. By looking at the CNN converging faster to [0..1] than to [0..255], here we divide both train and test with 255.



## **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.



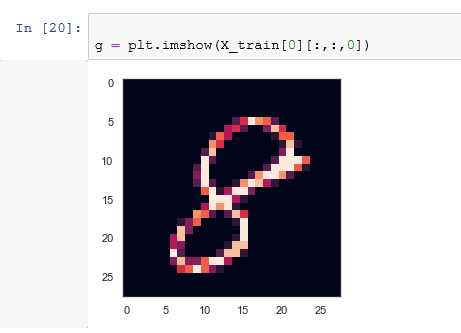
## **Label Encoding**

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



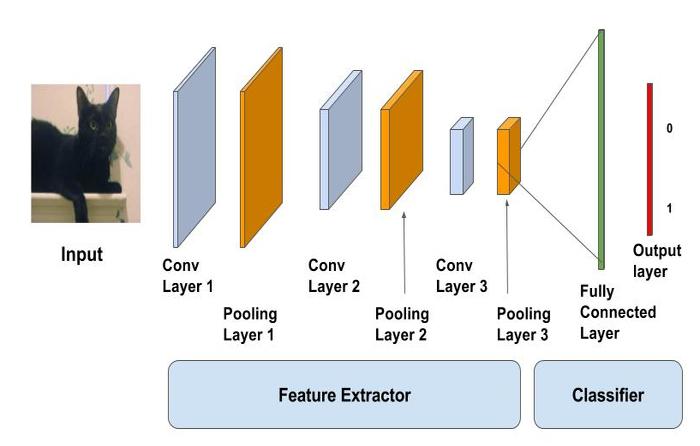
## **Training and validation sets**

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.

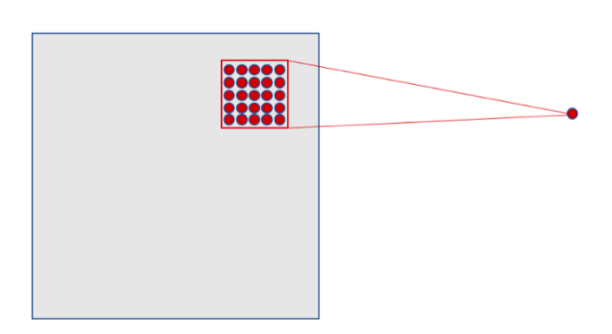


# **CNN (Convolutional Neural Network)**

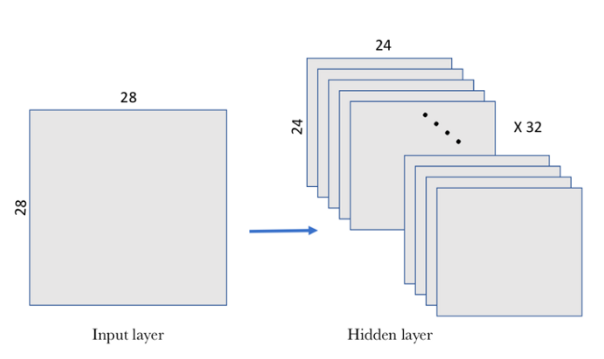
**Process of CNN**: following diagram is an example shown where input is the image of a cat and various layers on convolutions and pooling is shown.



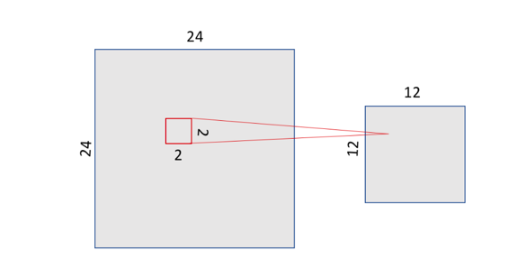
In the case of MNIST, as input to our neural network we can think of a space of two-dimensional neurons 28×28 (height = 28, width = 28, depth = 1). A first layer of hidden neurons connected to the neurons of the input layer that we have discussed will perform the convolutional operations that we have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



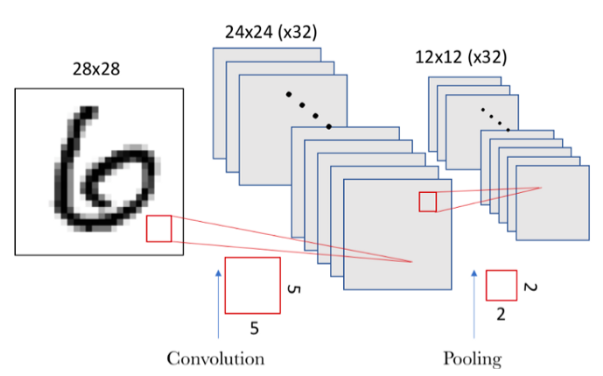
In this example, the first convolutional layer receives a size input tensor (28, 28, 1) and generates a size output (24, 24, 32), a 3D tensor containing the 32 outputs of 24×24 pixel result of computing the 32 filters on the input.



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**



* 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).



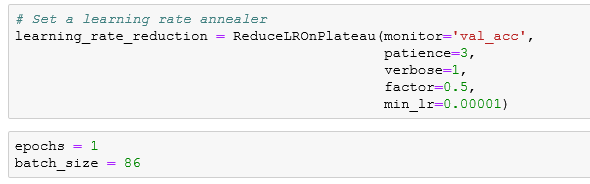
In order to make the optimizer converge faster and closest to the global minimum of the loss function, i used an annealing method of the learning rate (LR).

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).

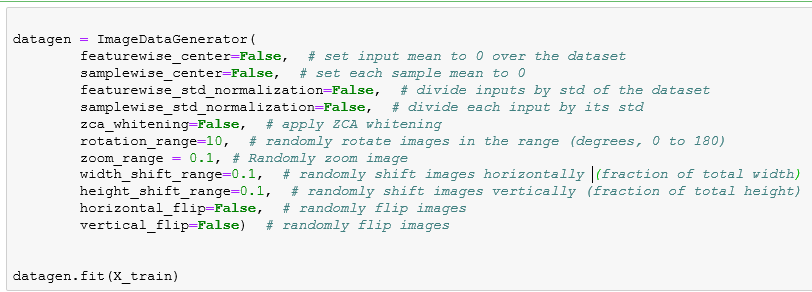
With the ReduceLROnPlateau function from Keras.callbacks, I choose to reduce the LR by half if the accuracy is not improved after 3 epochs.



## **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.

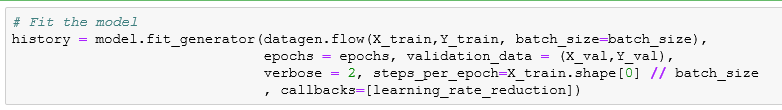
By applying just a couple of these transformations to our training data, we can easily double or triple the number of training examples and create a very robust model.



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

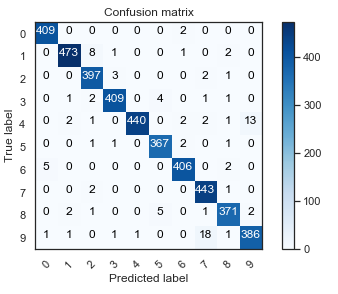
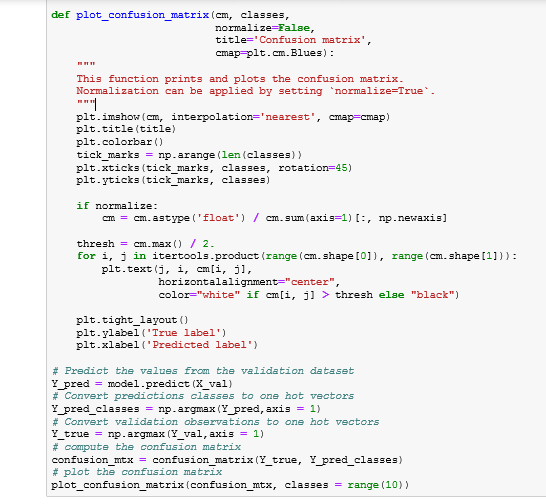
I did not apply a vertical flip nor horizontal flip since it could have led to misclassify symmetrical numbers such as 6 and 9.



# **EVALUATE THE MODEL**

## **Confusion matrix**

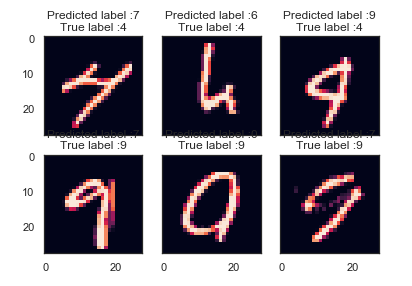
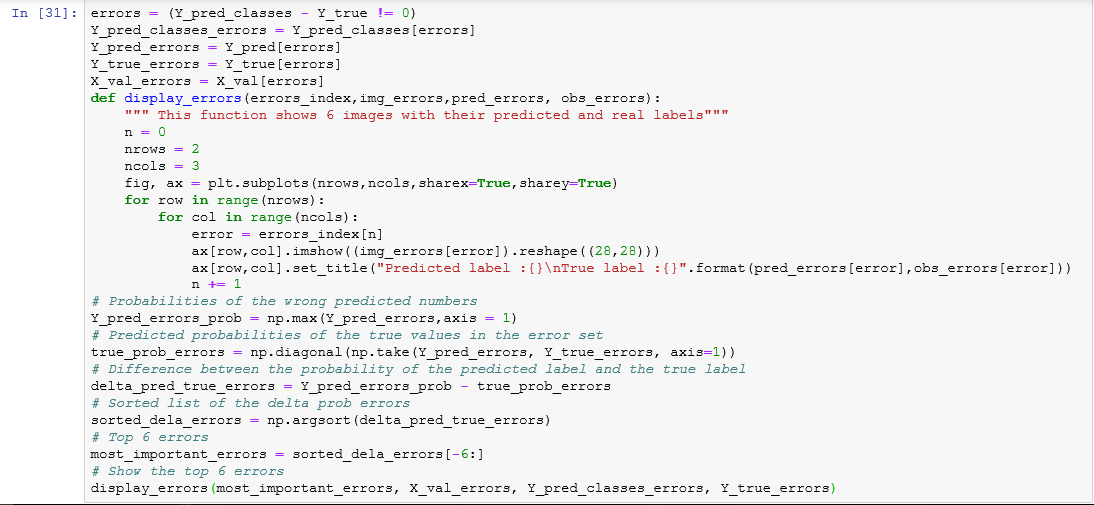
Confusion matrix can be very helpful to see your model drawbacks. I plot the confusion matrix of the validation results.



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).

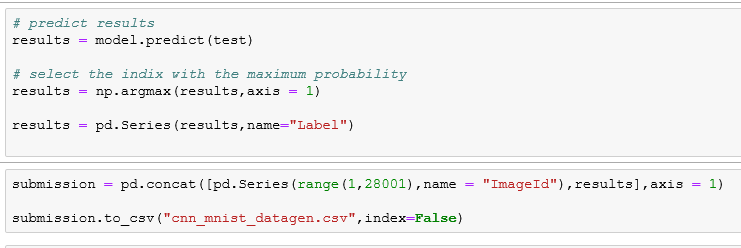
However, it seems that our CNN has some little troubles with the 4 digits, hey are misclassified as 9. Sometime it is very difficult to catch the difference between 4 and 9 when curves are smooth.

## **Displaying Errors**



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.

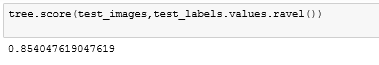
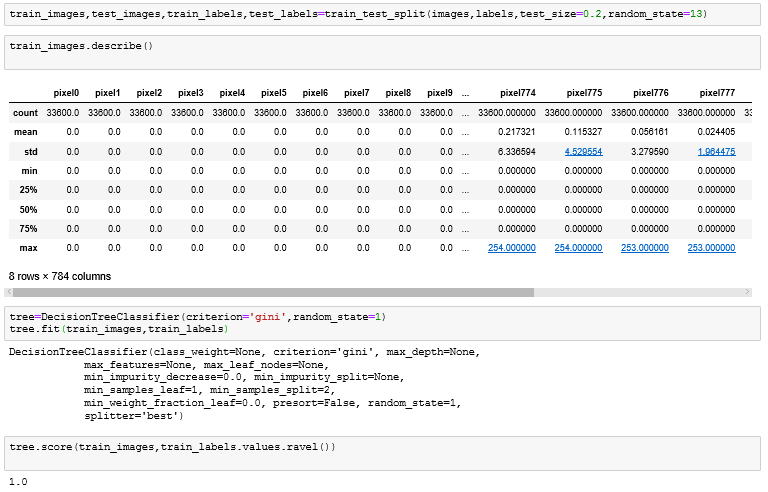
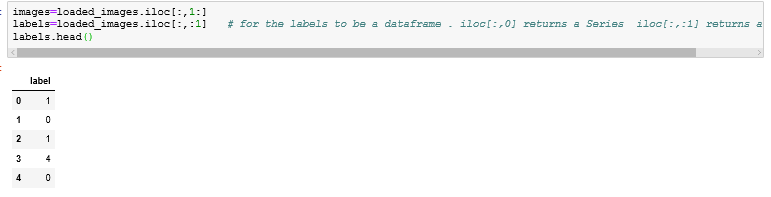
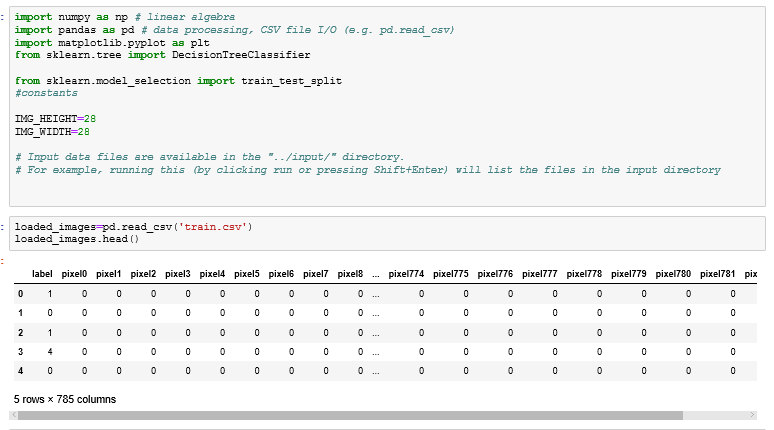
# **PREDICTION AND SUBMISSION**



# **OTHER MODELS**

## **1. Decision Tree Classifier**

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

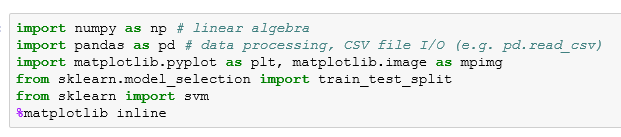


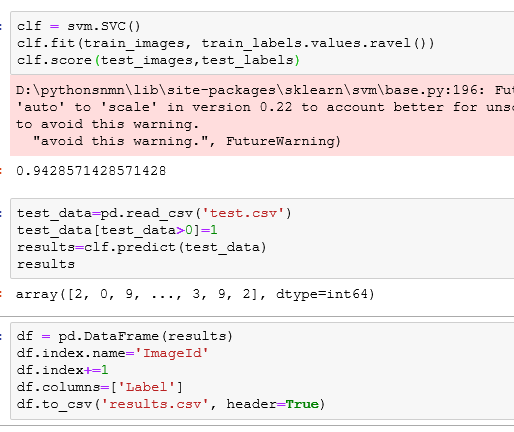
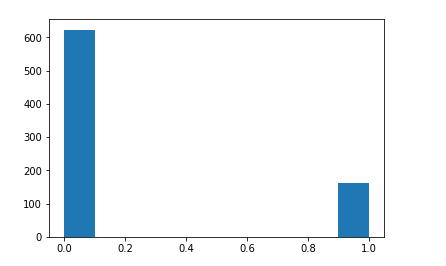
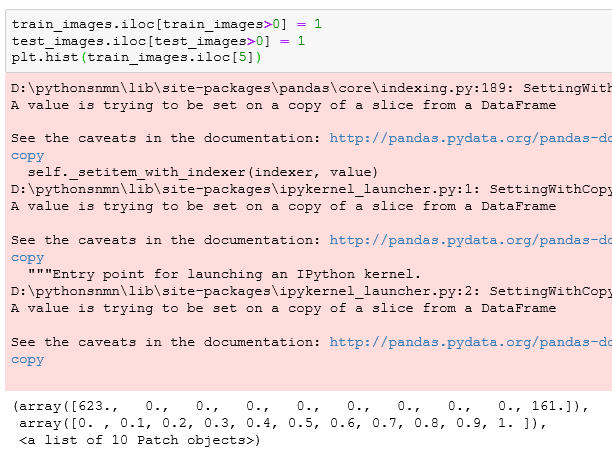
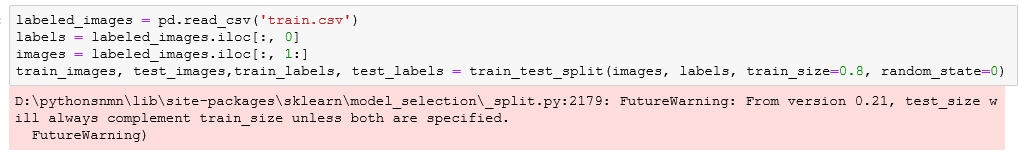


The score obtained after submission in kaggle was 85.142% from decision tree classifier

## **2. SVM (Support Vector Machine)**

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

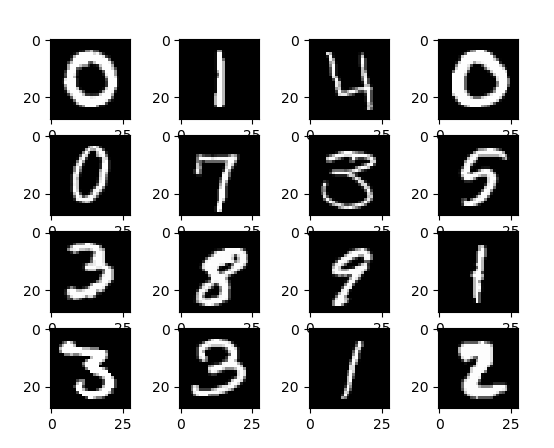
****



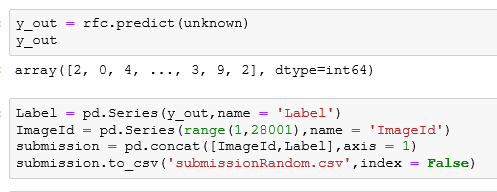
The obtained score after kaggle submission using SVM was 93.7%.

## **3. Random Forest**

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







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>