



Analysis Of Different ML Prediction Techniques For Handwritten Digit Recognition

A project report submitted for the course
Digital Image Processing (EEE3009)

Sailesh M	18BEE0158
Advait Marathe	18BEE0164

Under the Guidance of
Prof. Abhishek G
SELECT, VIT Vellore

Abstract

The digital image processing techniques are becoming extremely prominent in the field of Artificial Intelligence and Neural Network Processing. This is mainly due to the fact that such techniques simplify the space and time complexity of the process. The image processing techniques along with AI and NN provide solutions to a vast number of applications such as Biomedical Imaging, Security, Quality Assurance and other innovative solutions which do greater good for society. In this project, we are going to implement one of the solutions, “Handwritten Digit Recognition”. Now, this solution has vast applications. For instance, this can be implemented as one of the text-to-speech apps, so that visually challenged people can perceive the handwritten digits, such as important contacts saved in a diary in a situation of an emergency, or say the amount written on a cheque while they receive any sort of aid. This solution can also help to decrypt ancient scriptures or inscriptions, which can help us to make sense out of the given data and hence take a deeper dive into our past and the origin. Also, handwriting is one of the parameters which can be used to detect dyslexia and Parkinson’s in early stages. So this solution can be used to provide a better healthcare solution for such patients.

Contents

Sr. No.	Title	Page
1.	Introduction	3
2.	Literature Survey	4
3.	Background	6
4.	Procedure	10
5.	Results	13
6.	Conclusion	16
7.	References	17
8.	Appendix - Software Codes	19

Introduction

Optical Character Recognition (OCR) is an important application which has widespread uses like reading an image or a PDF file and converting it into editable and searchable form and can also be used in future by automated cars and robots to read and understand the human environment and convert the data into machine usable formats. Many tech giants like Google have developed software for this purpose (Google AI Vision)[1]. Another interesting application for OCR is print inspection, i.e checking if a particular printed material is up to its standard without any spelling errors and such [2]. It also finds its use in automated license plate recognition [3] for traffic offenders, health care, and for automated passport scanning purposes during travelling. The handwritten digit recognition problem is an extension of OCR, which generally only involves printed characters. Dealing with human handwriting poses a challenge as everyone has an unique way and style of writing. This is a relatively huge task to develop an algorithm to detect every human writing across several languages. To simplify the work, the problem statement of this project is creating an algorithm to detect handwritten digits from 0-9 alone. Usually for many image recognition tasks, Machine Learning (ML) technique proves to be much easier and also gives much better results when compared with using proper digital image processing techniques provided that a good dataset is available. So, in this project algorithms like Convolutional Neural Networks (CNN), random forest and Support Vector Machines (SVM) are implemented in tensorflow and the results obtained are then compared and a clear analysis is provided. Also, the need for image augmentation and how it affects the model are also studied in detail.

Literature Survey

Sr. No.	Name Of The Paper	Author	Description	Year of publication
1.	Automated number plate recognition system	Emina E. Etomi and Donatus. U. Onyishi	The paper proposes a system such as the number plate is detected via the means of contour detection and then the digits are recognised and obtained as a text.	2021
2.	Handwritten digit recognition using Bayesian ResNet	Purva Mhasakar, Prapti Trivedi, Srimanta Mandal and Suman K. Mitra	The paper implements ResNet architecture to identify the characters of various scripts.	2021
3.	Comparison of learning algorithms for handwritten digit recognition	Y. LeCun, and 7 others.	The paper discusses in length of the various learning algorithms used to implement the problem statement	1995
4.	Deep big simple neural nets excel on handwritten digit recognition	Dan Claudiu Cires, Ueli Meier, Luca Maria Gambardella and Jürgen Schmidhuber.	The paper implements the solution for given problem statement using backpropagation	2010

5.	Multi-script handwritten digit recognition using Multi-task learning	Mesay Samuel Gondere, Lars Schmidt-Thieme , Durga Prasad Sharma and Randolph Scholz	The paper proposes the implementation of the problem statement by the means of Multi-task learning.	2021
6.	Handwritten digit recognition using machine and deep learning algorithms	Ritik Dixit, Rishika Kushwaha and Samay Pashine	The paper proposes the solution using the Multi-layer Perceptron and SVM.	2021
7.	Handwritten digit recognition using deep learning	Gaganashree J. S. Padmashali, Diksha Kumari.	The paper proposes the solution of the problem statement using the CNN.	2021

Background

a) Understanding the dataset

A good dataset is the soul of any application which uses ML techniques. It doesn't matter how good the ML engineer is, it is impossible to get the application working in a real world scenario if the variety of data available is less. For example, in an OCR application, illumination plays an important role. If all the images available in the dataset contains proper illumination, then the system won't work if it is deployed in a scenario with different lightings. Therefore, it is imperative to have a variety of data in different illuminations, orientations along with the sheer quantity. Before moving on to applying ML, an understanding of the dataset and what formats the images are in is important. The dataset used in this project is the MNIST dataset for handwritten digit recognition. The data are in a CSV format and contain around 42000 images in binarized format containing digits as classes from 0-9 (total 10 classes). The distribution of the dataset for different classes is shown in figure 1. The x-axis tells the label and the y-axis shows how many images are there for that label.

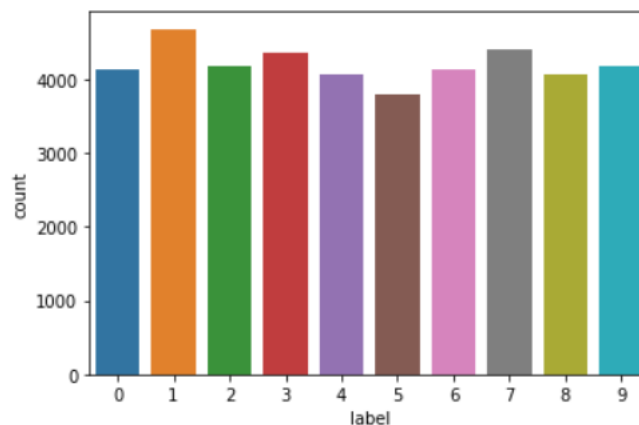


Figure 1: Histogram of dataset distribution

The shape of the image is (28,28), which is rolled out into a single row containing 784 pixels, which is to be reshaped accordingly by the ML engineer while training the model. A sample image is given below for illustration.

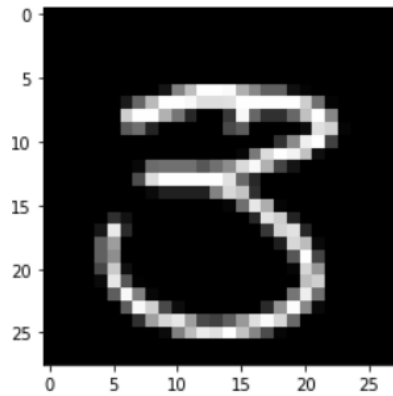


Figure 2: Sample image from the dataset

There are 42000 rows in the CSV file, each row corresponding to one image. Every row has 785 columns, with the first column being the label for the particular row and the remaining 784 columns correspond to the image pixels in binary format. For this application, the dataset is split into training and testing in the ratio of 80% : 20%.

b) Overview of ML Algorithms Used

1. Convolutional Neural Network

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics. The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area.

2. Random Forest Classification

Random forest is a flexible, easy to use machine learning algorithm that produces, even without hyper-parameter tuning, a great result most of the time. It is also one of the most used algorithms, because of its simplicity and diversity (it can be used for both classification and

regression tasks). Random forest is a supervised learning algorithm. The "forest" it builds is an ensemble of decision trees, usually trained with the “bagging” method. The general idea of the bagging method is that a combination of learning models increases the overall result.

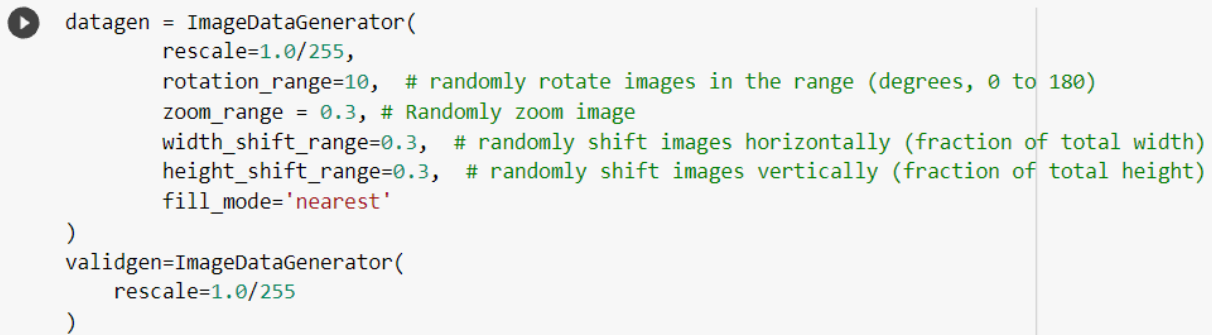
3. Support Vector Machines

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning. The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane. SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called support vectors, and hence the algorithm is termed as Support Vector Machine.

c) Need for Image Augmentation

In the previous section, it was shown why it is important to have a variety of data along with the quantity. When we personally collect the dataset, it'll be known how much variety is there, and if it is enough or not. Mostly everyone uses existing datasets only. So, there may or may not be different orientations of the image. But this is needed to get a good model and to get in working in real-time applications under all scenarios. To overcome this issue, image augmentation can be incorporated. This can be done in tensorflow using the Image Generator API [4]. This plays a key role in developing a better model. For example, if all the images in the dataset are located in the center, then the model will have difficulty in predicting when the images are located on corners, or if they are tilted to some degrees or if the size of the image is different. Using the Image Generator API, we can modify our existing images and create new ones which can be tilted, zoomed in or out, shifted to corners etc. The main advantage of this API is that it does all these necessary and specified modifications to the dataset while training, and it doesn't modify the existing data, nor creates new data, which can lead to memory overflow. It reads an image from the memory, creates a copy and modifies it in run time and deletes once the task is completed. This prevents unnecessary modification to the existing dataset. The user, too, doesn't

have to write any code to modify several thousand images by themselves. It also plays an important role in avoiding overfitting. So, this is indeed a handy tool in image processing and machine learning. The effects of using this image augmentation technique is discussed in the results section. An example of using this API is given in figure 3.



```
▶ datagen = ImageDataGenerator(  
    rescale=1.0/255,  
    rotation_range=10, # randomly rotate images in the range (degrees, 0 to 180)  
    zoom_range = 0.3, # Randomly zoom image  
    width_shift_range=0.3, # randomly shift images horizontally (fraction of total width)  
    height_shift_range=0.3, # randomly shift images vertically (fraction of total height)  
    fill_mode='nearest'  
)  
validgen=ImageDataGenerator(  
    rescale=1.0/255  
)
```

Figure 3: Using the Image Generator API

Procedure

This section deals with the implementation strategy of this project.

Every machine learning model needs a sufficient amount of data. So, one of the important steps in implementing an ML model is collection of a good dataset, or in some cases creating it. Next, we have to analyse the data, see if there is enough variety in it, so we get an idea of what kind of data augmentation has to be performed. Then comes the selection of an appropriate ML model. Though in many cases, many algorithms like SVM, CNN, Regression, Random Forests, LSTM show similar performance, appropriate technique has to be selected based on the type of data which is involved. For example, if it is a time series or a Natural Language Processing problem, it would be a better choice to choose RNNs or LSTM. As this project involves image, CNN will be the ideal choice. Next we have to train the model and use techniques like batch normalisation, dropouts, augmentation to avoid overfitting. Lastly, the validation of results has to be done properly. The following figure shows the procedure to be followed clearly.

Methodology for Handwritten Digit Recognition

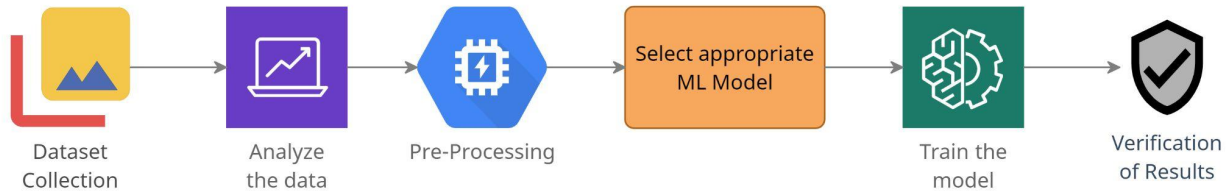


Figure 4: Methodology for project implementation

Performance of the handwritten digit recognition algorithm was compared in 4 ways.

a) Using convolutional neural network

This method consists of using 4 convolutional layers, one fully connected layer followed by a softmax layer. CNN was used instead of a Deep Neural Network (DNN) because of its high efficiency in dealing with images and 2 Dimensional inputs. It will be better for the algorithm to look at a 2D image and determine what digit it is, instead of looking at every single pixel by pixel and trying to determine as it is the case in DNNs. Though this method provided decent results (discussed under result and analysis section), it showed signs of overfitting, which is a common problem in Machine Learning. To avoid this, we perform image augmentation to get much better results. The structure of the CNN is summarised in the given figure.

Model: "sequential_9"

Layer (type)	Output Shape	Param #
conv2d_36 (Conv2D)	(None, 28, 28, 32)	832
conv2d_37 (Conv2D)	(None, 28, 28, 32)	25632
max_pooling2d_34 (MaxPooling)	(None, 14, 14, 32)	0
dropout_27 (Dropout)	(None, 14, 14, 32)	0
conv2d_38 (Conv2D)	(None, 14, 14, 64)	18496
conv2d_39 (Conv2D)	(None, 14, 14, 64)	36928
max_pooling2d_35 (MaxPooling)	(None, 7, 7, 64)	0
dropout_28 (Dropout)	(None, 7, 7, 64)	0
flatten_9 (Flatten)	(None, 3136)	0
dense_18 (Dense)	(None, 256)	803072
dropout_29 (Dropout)	(None, 256)	0
dense_19 (Dense)	(None, 10)	2570

=====
 Total params: 887,530
 Trainable params: 887,530
 Non-trainable params: 0

Figure 5: Structure of CNN Model

b) CNN With Image Augmentation

The types of augmentation performed on the images is shown in figure 3. The images are rotated, normalised, zoomed, and shifted towards corners. This is done to increase the variety of images to the training set. During shifting and zooming, there will be some pixels whose values are undefined. These are solved using the fillmode parameter. Results show that overfitting has been reduced (in next section).

c) Random Forest and SVM

Though CNNs are often used for image related applications, other algorithms also yield decent results. Here, the performance of the digit recognition model is tested by using random forest and SVM whose details are covered in the previous section. For random forest, 200 estimators are used and the one versus one classification technique has been implemented in the SVM.

Results

a) Impact of Image Augmentation on CNN

From the figure 3, we can see how the image augmentation has been implemented in tensorflow and how it makes programming easier. To show the importance of this technique, the handwritten digit recognition was implemented using CNN in two different ways, with and without using the image augmentation. First, we'll see how the result is without the augmentation. In this case, the graph of training and validation accuracy can be seen in the given figure.

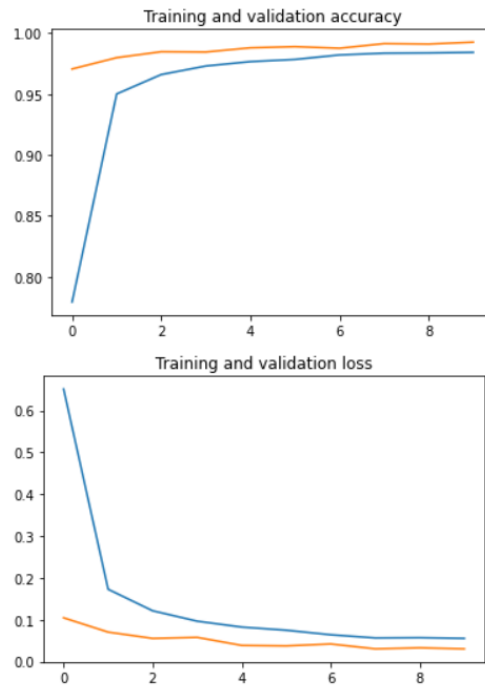


Figure 6: Depiction of training and validation accuracy and losses

The next figure shows the accuracy of the model and its corresponding confusion matrix.

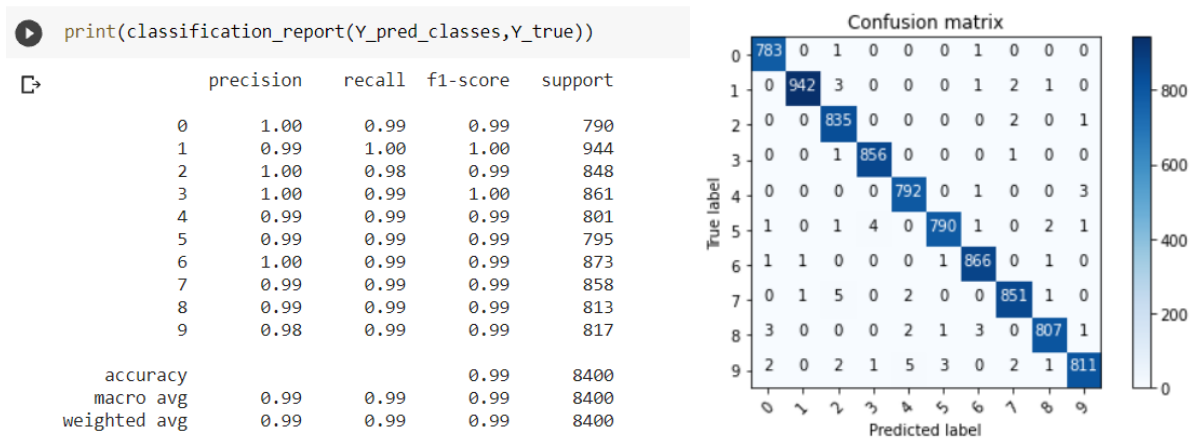
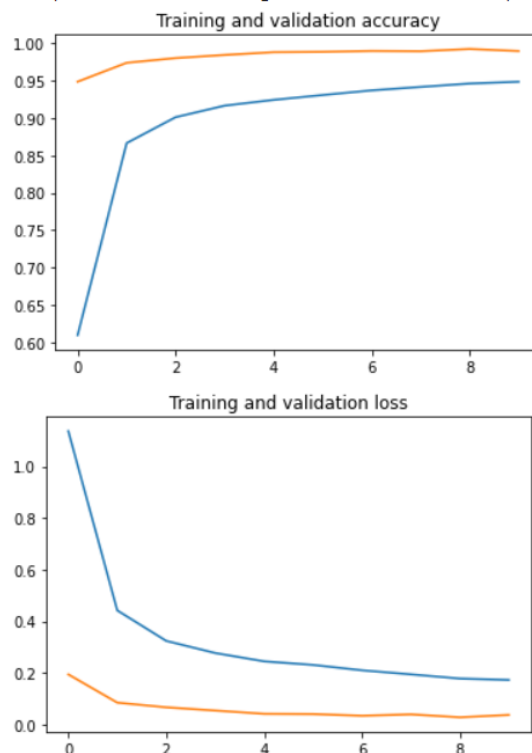


Figure 7: Accuracy of the model without augmentation

From the above figures, we can clearly see a sign that we are overfitting our model, and that the model will not perform well in real time because of a lack of variety of datas. Next, the following figures show the performance of the model trained with augmented data. The first important thing to notice is the increase in training time. Without augmentation, it took around 70 seconds per epoch, and due to augmentation the training time increased to 240 seconds per epoch. The model was trained for 10 epochs and the results are shown in the figures below.



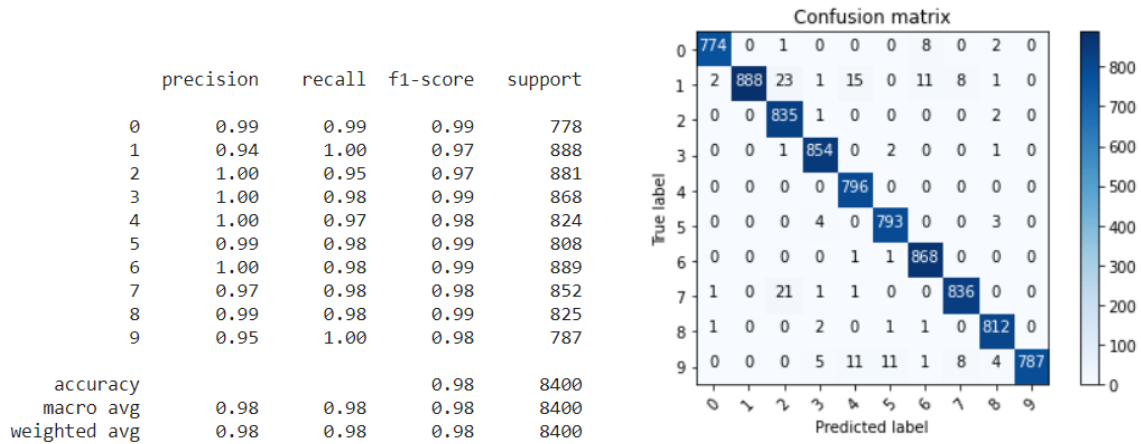


Figure 8: Performance analysis of model trained with augmentation

We can clearly see that we have now prevented overfitting, and our model can perform well for many different scenarios.

b) Performance on Random Forest and SVM

It is evident that the accuracy of the model trained using CNN will be higher than other methods. This is in concurrence with the results obtained. The accuracy obtained using random forest is given in figure.

```
clf = RandomForestClassifier(n_estimators=200,max_samples=0.5)
clf.fit(X_train, Y_train)
print(clf.score(X_test, Y_test))
```

0.9570668994524244

Figure 9: Accuracy Obtained using Random Forest

We see that the accuracy obtained is around 95.7% which is slightly lower than CNN. Note that no image augmentation has been performed. For SVM, an accuracy of around 97% is obtained as given in figure.

```
clf2 = svm.SVC(decision_function_shape='ovo')  
  
clf2.fit(X_train, Y_train)  
  
SVC(decision_function_shape='ovo')  
  
clf2.score(X_test, Y_test)  
  
0.9741290373779858
```

Figure 10: Accuracy using SVM

Conclusion

From the performance analysis of various methods used in the project, we can see the importance of the data augmentation and how it helped in creating a better model and avoiding overfitting. We also see that CNN provides a good solution when compared with random forest and SVM. The number of parameters involved in the CNN model is around 888,000. All the weights and biases will be in floating point, so it approximately consumes a memory of around 3.4MB. The goal of every machine learning engineer is to build an accurate model which also consumes less space. If the project is to be implemented in a bare metal microcontroller or an FPGA board, it will occupy 3.4 MB of memory, which is very high. Also, all the computations will be in floating point, which makes the convolution layers computationally expensive especially when using an FPGA board. So, in future work more emphasis can be given on developing a smaller, efficient model with weights and biases in 8 bit integer format using a technique called post training quantisation [5]. This can be easily achieved using tensorflow lite.

References

- [1] Google Cloud, Published On - 07 April 2008, Accessed on - 04 November 2021, Available at - <https://cloud.google.com/vision>, Cloud Vision API.
- [2] Yantra Vision, Published On - 14 December 2014, Accessed On - 07 November 2021, Available at - <https://yantravision.com/print-inspection/>, High-speed Print Quality inspection System for Inline, Offline & Web Applications
- [3] R. Panahi and I. Gholampour, "Accurate Detection and Recognition of Dirty Vehicle Plate Numbers for High-Speed Applications," in IEEE Transactions on Intelligent Transportation Systems, vol. 18, no. 4, pp. 767-779, April 2017, doi: 10.1109/TITS.2016.2586520.
- [4] Tensor Flow, Published On - 12 December 2012, Accessed On - 19 October 2021, Available at - https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator, `tf.keras.preprocessing.image.ImageDataGenerator`
- [5] J. -I. Guo, C. -C. Tsai, J. -L. Zeng, S. -W. Peng and E. -C. Chang, "Hybrid Fixed-Point/Binary Deep Neural Network Design Methodology for Low-Power Object Detection," in IEEE Journal on Emerging and Selected Topics in Circuits and Systems, vol. 10, no. 3, pp. 388-400, Sept. 2020, doi: 10.1109/JETCAS.2020.3015753.
- [6] Etomi, E.E. and Onyishi, D.U., 2021. Automated number plate recognition system. Tropical Journal of Science and Technology, 2(1), pp.38-48.
- [7] Mhasakar, P., Trivedi, P., Mandal, S. and Mitra, S.K., 2021. Handwritten Digit Recognition Using Bayesian ResNet. SN Computer Science, 2(5), pp.1-10.
- [8] LeCun, Y., Jackel, L.D., Bottou, L., Brunot, A., Cortes, C., Denker, J., Drucker, H., Guyon, I., Muller, U.A., Sackinger, E. and Simard, P., 1995, October. Comparison of learning algorithms for handwritten digit recognition. In International conference on artificial neural networks (Vol. 60, pp. 53-60).

- [9] Alwazwazy, H.A., Albehadili, H.M., Alwan, Y.S. and Islam, N.E., 2016. Handwritten digit recognition using convolutional neural networks. *International Journal of Innovative Research in Computer and Communication Engineering*, 4(2), pp.1101-1106.
- [10] Gondere, M.S., Schmidt-Thieme, L., Sharma, D.P. and Scholz, R., 2021. Multi-script Handwritten Digit Recognition Using Multi-task Learning. *arXiv preprint arXiv:2106.08267*.
- [11] Gondere, M.S., Schmidt-Thieme, L., Sharma, D.P. and Scholz, R., 2021. Multi-script Handwritten Digit Recognition Using Multi-task Learning. *arXiv preprint arXiv:2106.08267*.
- [12] Pashine, S., Dixit, R. and Kushwah, R., 2021. Handwritten Digit Recognition using Machine and Deep Learning Algorithms. *arXiv preprint arXiv:2106.12614*.
- [13] Ashiquzzaman, A. and Tushar, A.K., 2017, February. Handwritten Arabic numeral recognition using deep learning neural networks. In *2017 IEEE International Conference on Imaging, Vision & Pattern Recognition (icIVPR)* (pp. 1-4). IEEE.

```


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
import itertools
import tensorflow as tf
from tensorflow import keras
from keras.utils.np_utils import to_categorical #one-hot-encoding
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ReduceLROnPlateau

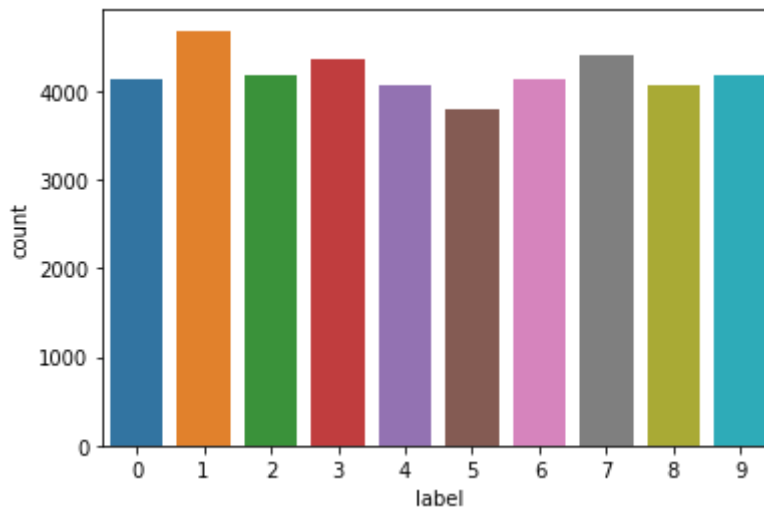
```

```

train = pd.read_csv("train.csv")
Y_train = train["label"]
X_train = train.drop(labels = ["label"],axis = 1)
sns.countplot(Y_train)

```

 /usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass
FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7f2b1cb97dd0>



```

X_train = X_train.values.reshape(-1,28,28,1)
Y_train = to_categorical(Y_train, num_classes = 10)
X_train, X_val, Y_train, Y_val = train_test_split(X_train, Y_train, test_size = 0.2, random_state = 42)

```

```

model = Sequential()

```

```

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

```

```

model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.3))

model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',
                  activation = 'relu'))
model.add(MaxPool2D(pool_size=(2,2)))
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(0.3))

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

model.compile(optimizer = 'adam' , loss = "categorical_crossentropy", metrics=["accuracy"])

model.summary()

```

Model: "sequential"

Layer (type)	Output Shape	Param #
=====		
conv2d (Conv2D)	(None, 28, 28, 32)	832
max_pooling2d (MaxPooling2D)	(None, 14, 14, 32)	0
conv2d_1 (Conv2D)	(None, 14, 14, 32)	25632
max_pooling2d_1 (MaxPooling2	(None, 7, 7, 32)	0
dropout (Dropout)	(None, 7, 7, 32)	0
conv2d_2 (Conv2D)	(None, 7, 7, 64)	18496
max_pooling2d_2 (MaxPooling2	(None, 3, 3, 64)	0
conv2d_3 (Conv2D)	(None, 3, 3, 64)	36928
max_pooling2d_3 (MaxPooling2	(None, 1, 1, 64)	0
dropout_1 (Dropout)	(None, 1, 1, 64)	0
flatten (Flatten)	(None, 64)	0
dense (Dense)	(None, 256)	16640
dropout_2 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 10)	2570
=====		
Total params: 101,098		
Trainable params: 101,098		

Non-trainable params: 0

```

datagen = ImageDataGenerator(
    rescale=1.0/255,
)
validgen=ImageDataGenerator(
    rescale=1.0/255
)

epochs = 10
batch_size = 86

history = model.fit_generator(datagen.flow(X_train,Y_train, batch_size=batch_size),
                             epochs = epochs,
                             validation_data=validgen.flow(X_val,Y_val),
                             verbose = 2, steps_per_epoch=X_train.shape[0] // batch_size
                             )

/usr/local/lib/python3.7/dist-packages/keras/engine/training.py:1972: UserWarning: `Model.fit_generator` is deprecated and
warnings.warn("`Model.fit_generator` is deprecated and ")
Epoch 1/10
390/390 - 72s - loss: 0.6515 - accuracy: 0.7793 - val_loss: 0.1047 - val_accuracy: 0
Epoch 2/10
390/390 - 71s - loss: 0.1729 - accuracy: 0.9501 - val_loss: 0.0705 - val_accuracy: 0
Epoch 3/10
390/390 - 71s - loss: 0.1214 - accuracy: 0.9660 - val_loss: 0.0554 - val_accuracy: 0
Epoch 4/10
390/390 - 70s - loss: 0.0966 - accuracy: 0.9730 - val_loss: 0.0579 - val_accuracy: 0
Epoch 5/10
390/390 - 69s - loss: 0.0825 - accuracy: 0.9766 - val_loss: 0.0387 - val_accuracy: 0
Epoch 6/10
390/390 - 69s - loss: 0.0750 - accuracy: 0.9784 - val_loss: 0.0376 - val_accuracy: 0
Epoch 7/10
390/390 - 68s - loss: 0.0641 - accuracy: 0.9821 - val_loss: 0.0421 - val_accuracy: 0
Epoch 8/10
390/390 - 68s - loss: 0.0565 - accuracy: 0.9836 - val_loss: 0.0305 - val_accuracy: 0
Epoch 9/10
390/390 - 68s - loss: 0.0571 - accuracy: 0.9838 - val_loss: 0.0329 - val_accuracy: 0
Epoch 10/10
390/390 - 68s - loss: 0.0555 - accuracy: 0.9842 - val_loss: 0.0305 - val_accuracy: 0

```

```

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    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)

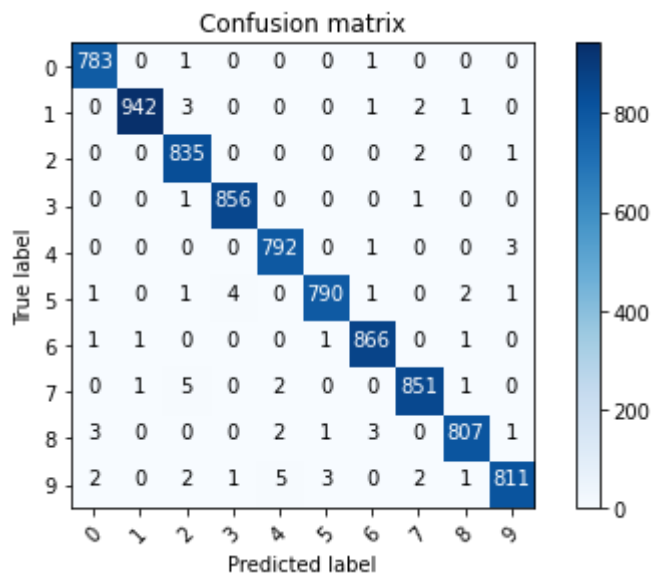
```

```

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')
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)
confusion_mtx =confusion_matrix(Y_true, Y_pred_classes)
plot_confusion_matrix(confusion_mtx, classes = range(10))

```



```
print(classification_report(Y_pred_classes,Y_true))
```

	precision	recall	f1-score	support
0	1.00	0.99	0.99	790
1	0.99	1.00	1.00	944
2	1.00	0.98	0.99	848
3	1.00	0.99	1.00	861
4	0.99	0.99	0.99	801
5	0.99	0.99	0.99	795
6	1.00	0.99	0.99	873
7	0.99	0.99	0.99	858
8	0.99	0.99	0.99	813
9	0.98	0.99	0.99	817
accuracy			0.99	8400
macro avg	0.99	0.99	0.99	8400
weighted avg	0.99	0.99	0.99	8400

```

import matplotlib.image as mpimg
import matplotlib.pyplot as plt
acc      = history.history[ 'accuracy' ]
val_acc  = history.history[ 'val_accuracy' ]
loss     = history.history[ 'loss' ]
val_loss = history.history[ 'val_loss' ]

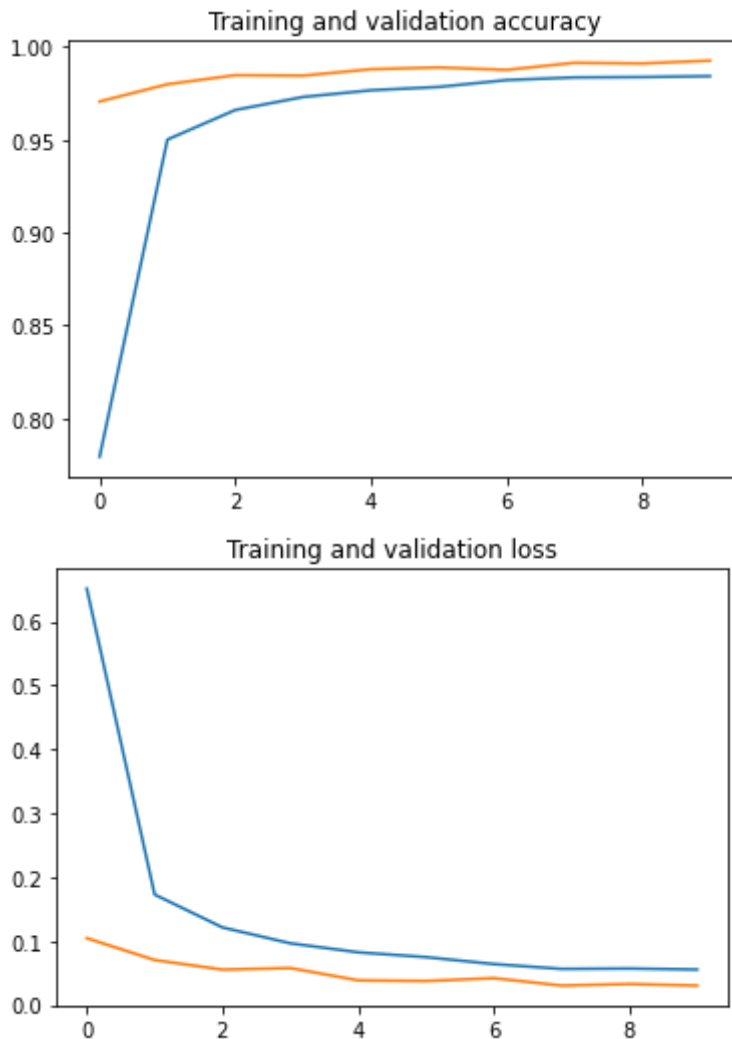
epochs   = range(len(acc)) # Get number of epochs

#-----
# Plot training and validation accuracy per epoch
#-----
plt.plot ( epochs,    acc )
plt.plot ( epochs, val_acc )
plt.title ('Training and validation accuracy')
plt.figure()

#-----
# Plot training and validation loss per epoch
#-----
plt.plot ( epochs,    loss )
plt.plot ( epochs, val_loss )
plt.title ('Training and validation loss' )

```

Text(0.5, 1.0, 'Training and validation loss')






```


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
import itertools
import tensorflow as tf
from tensorflow import keras
from keras.utils.np_utils import to_categorical #one-hot-encoding
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ReduceLROnPlateau

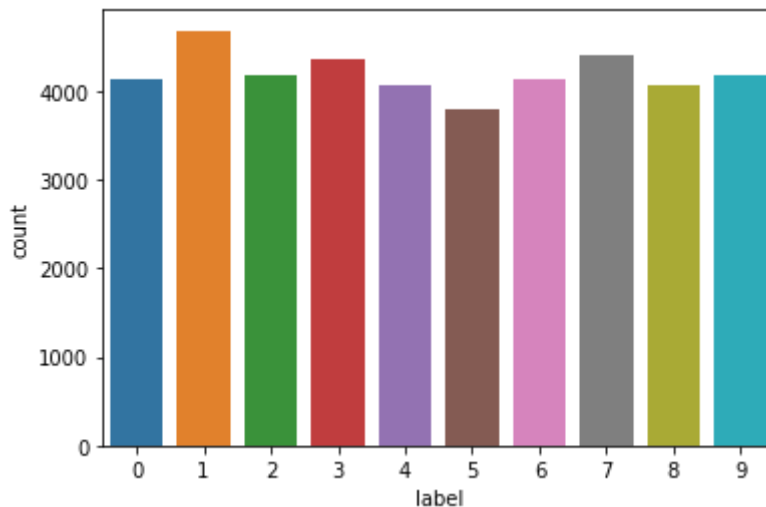
```

```

train = pd.read_csv("train.csv")
Y_train = train["label"]
X_train = train.drop(labels = ["label"],axis = 1)
sns.countplot(Y_train)

```

 /usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass
FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7f9d12531e10>



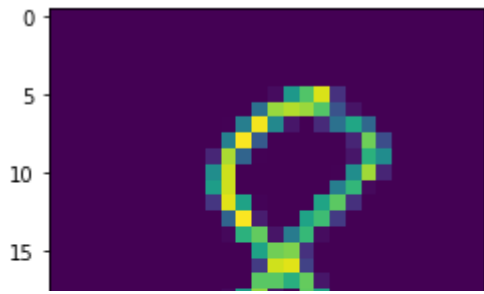
```

X_train = X_train.values.reshape(-1,28,28,1)
Y_train = to_categorical(Y_train, num_classes = 10)
X_train, X_val, Y_train, Y_val = train_test_split(X_train, Y_train, test_size = 0.2, random_state = 42)

plt.imshow(X_train[111][:,:,0])

```

<matplotlib.image.AxesImage at 0x7f9d13233dd0>



```
model = Sequential()
```

```
model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',
                  activation = 'relu', input_shape = (28,28,1)))
#model.add(MaxPool2D(pool_size=(2,2)))
model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',
                  activation = 'relu'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.3))
```

```
model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',
                  activation = 'relu'))
#model.add(MaxPool2D(pool_size=(2,2)))
model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',
                  activation = 'relu'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.3))
```

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

```
model.compile(optimizer = 'adam' , loss = "categorical_crossentropy", metrics=["accuracy"])
```

```
datagen = ImageDataGenerator(
    rescale=1.0/255,
    rotation_range=10, # randomly rotate images in the range (degrees, 0 to 180)
    zoom_range = 0.3, # Randomly zoom image
    width_shift_range=0.3, # randomly shift images horizontally (fraction of total wi
    height_shift_range=0.3, # randomly shift images vertically (fraction of total hei
    fill_mode='nearest'
)
validgen=ImageDataGenerator(
    rescale=1.0/255
)
```

```
epochs = 10
batch_size = 86
```

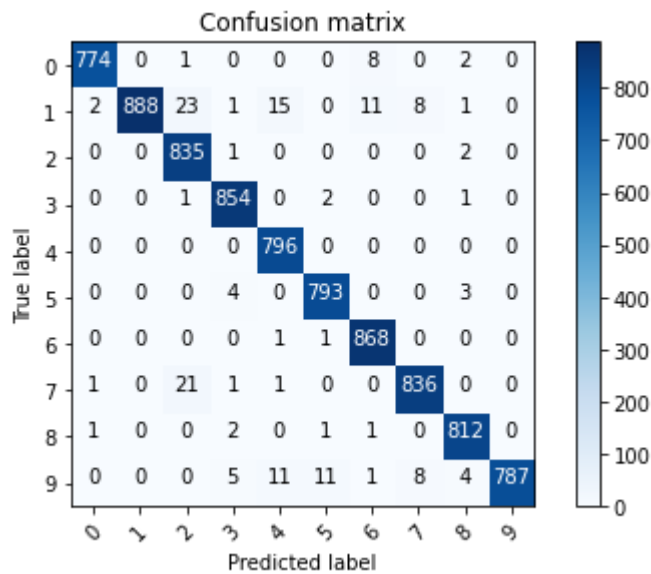
```
history = model.fit_generator(datagen.flow(X_train,Y_train, batch_size=batch_size),
                             epochs = epochs,
                             validation_data=validgen.flow(X_val,Y_val),
                             verbose = 2, steps_per_epoch=X_train.shape[0] // batch_size
    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

```
/usr/local/lib/python3.7/dist-packages/keras/engine/training.py:1972: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version.
warnings.warn("`Model.fit_generator` is deprecated and will be removed in a future version.")
Epoch 1/10
390/390 - 246s - loss: 1.1370 - accuracy: 0.6098 - val_loss: 0.1935 - val_accuracy: 0.6098
Epoch 2/10
390/390 - 243s - loss: 0.4416 - accuracy: 0.8664 - val_loss: 0.0840 - val_accuracy: 0.8664
Epoch 3/10
390/390 - 243s - loss: 0.3235 - accuracy: 0.9010 - val_loss: 0.0658 - val_accuracy: 0.9010
Epoch 4/10
390/390 - 242s - loss: 0.2764 - accuracy: 0.9164 - val_loss: 0.0532 - val_accuracy: 0.9164
Epoch 5/10
390/390 - 242s - loss: 0.2441 - accuracy: 0.9241 - val_loss: 0.0407 - val_accuracy: 0.9241
Epoch 6/10
390/390 - 242s - loss: 0.2303 - accuracy: 0.9306 - val_loss: 0.0396 - val_accuracy: 0.9306
Epoch 7/10
390/390 - 243s - loss: 0.2094 - accuracy: 0.9368 - val_loss: 0.0331 - val_accuracy: 0.9368
Epoch 8/10
390/390 - 245s - loss: 0.1936 - accuracy: 0.9415 - val_loss: 0.0384 - val_accuracy: 0.9415
Epoch 9/10
390/390 - 242s - loss: 0.1774 - accuracy: 0.9460 - val_loss: 0.0272 - val_accuracy: 0.9460
Epoch 10/10
390/390 - 243s - loss: 0.1720 - accuracy: 0.9483 - val_loss: 0.0363 - val_accuracy: 0.9483
```

```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    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')
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)
```

```
confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)
plot_confusion_matrix(confusion_mtx, classes = range(10))
```



```
print(classification_report(Y_pred_classes, Y_true))
```

	precision	recall	f1-score	support
0	0.99	0.99	0.99	778
1	0.94	1.00	0.97	888
2	1.00	0.95	0.97	881
3	1.00	0.98	0.99	868
4	1.00	0.97	0.98	824
5	0.99	0.98	0.99	808
6	1.00	0.98	0.99	889
7	0.97	0.98	0.98	852
8	0.99	0.98	0.99	825
9	0.95	1.00	0.98	787
accuracy			0.98	8400
macro avg	0.98	0.98	0.98	8400
weighted avg	0.98	0.98	0.98	8400

```
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
acc      = history.history[ 'accuracy' ]
val_acc  = history.history[ 'val_accuracy' ]
loss     = history.history[ 'loss' ]
val_loss = history.history[ 'val_loss' ]

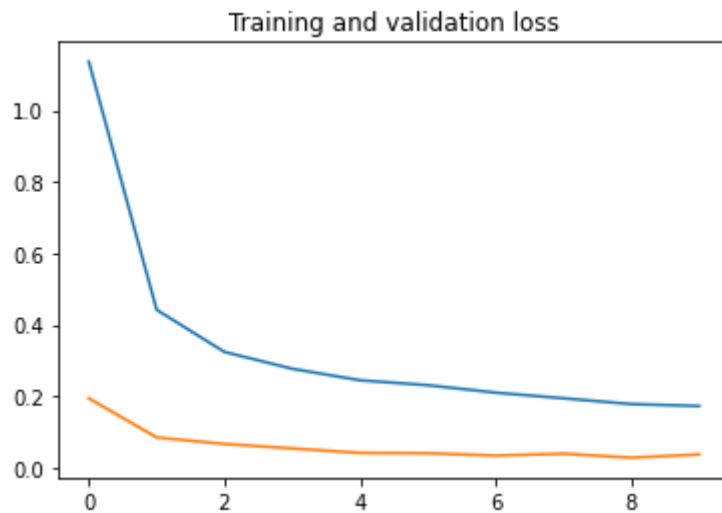
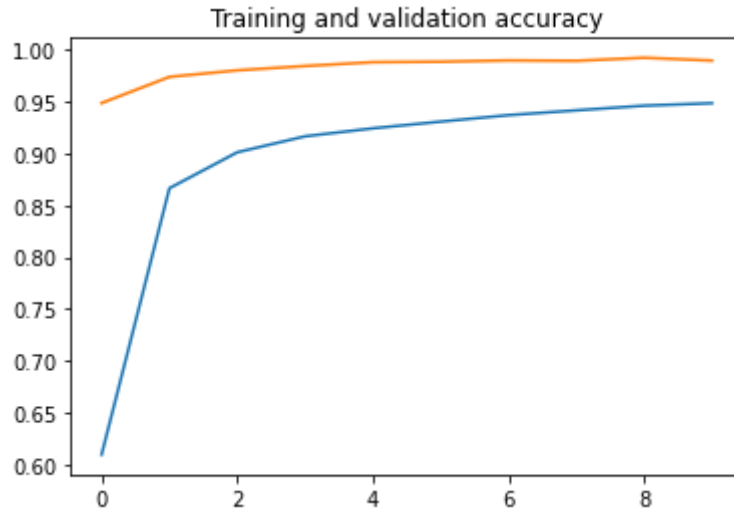
epochs   = range(len(acc)) # Get number of epochs

#-----
# Plot training and validation accuracy per epoch
#-----
plt.plot ( epochs, acc )
plt.plot ( epochs, val_acc )
```

```
plt.title('Training and validation accuracy')
plt.figure()
```

```
#-----
# Plot training and validation loss per epoch
#-----
plt.plot ( epochs,    loss )
plt.plot ( epochs, val_loss )
plt.title ('Training and validation loss'   )
```

```
Text(0.5, 1.0, 'Training and validation loss')
```



```
model.summary()
```

Model: "sequential_9"

Layer (type)	Output Shape	Param #
=====		
conv2d_36 (Conv2D)	(None, 28, 28, 32)	832
conv2d_37 (Conv2D)	(None, 28, 28, 32)	25632
max_pooling2d_34 (MaxPooling)	(None, 14, 14, 32)	0
dropout_27 (Dropout)	(None, 14, 14, 32)	0
conv2d_38 (Conv2D)	(None, 14, 14, 64)	18496

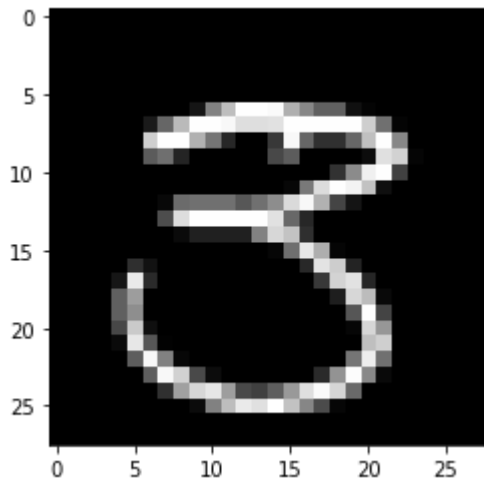
conv2d_39 (Conv2D)	(None, 14, 14, 64)	36928
max_pooling2d_35 (MaxPooling)	(None, 7, 7, 64)	0
dropout_28 (Dropout)	(None, 7, 7, 64)	0
flatten_9 (Flatten)	(None, 3136)	0
dense_18 (Dense)	(None, 256)	803072
dropout_29 (Dropout)	(None, 256)	0
dense_19 (Dense)	(None, 10)	2570
=====		
Total params: 887,530		
Trainable params: 887,530		
Non-trainable params: 0		

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import missingno
from sklearn.neural_network import MLPClassifier
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
```

```
df=pd.read_csv("train.csv")
```

```
label = df.iloc[7, 1:].values
label=label.reshape(28,28)
plt.imshow(label, cmap = 'gray')
```

```
↳ <matplotlib.image.AxesImage at 0x7f177fa50690>
```



```
X = df.iloc[:, 1:].values
Y = df.iloc[:, 0].values
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, train_size = 0.7, random_state =
clf = RandomForestClassifier(n_estimators=200,max_samples=0.5)
clf.fit(X_train, Y_train)
print(clf.score(X_test, Y_test))
```

```
0.9570668994524244
```

```
clf2 = svm.SVC(decision_function_shape='ovo')
```

```
clf2.fit(X_train, Y_train)
```

```
SVC(decision_function_shape='ovo')
```

```
clf2.score(X_test, Y_test)
```

```
0.9741290373779858
```

