1.) Data Preprocessing

```
#importing libraries
from keras.datasets import mnist
import numpy as np

# Importing the dataset
(X_train,y_train),(X_test,y_test) = mnist.load_data()

#fixing shapes
X_train=np.expand_dims(X_train,3)
X_test=np.expand_dims(X_test,3)
y_train=y_train.reshape((y_train.shape[0],1))
y_test=y_test.reshape((y_test.shape[0],1))

#Encoding of categorical data to one-hot format
from sklearn.preprocessing import OneHotEncoder
onehotencoder=OneHotEncoder(sparse=False)
y_train=onehotencoder.fit_transform(y_train)
y_test=onehotencoder.transform(y_test)

# Splitting the Training into the Training set and Validation set
from sklearn.model_selection import train_test_split
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,
test_size = 0.175, random_state = 0)

#Feature Scaling
X_train = X_train / 255
X_val = X_val / 255
X_test = X_test / 255

#Creating generator for Real-time Data Augmentation
from keras.preprocessing.image import ImageDataGenerator

train_datagen=ImageDataGenerator(rotation_range=5,shear_range=0.05,zoo
m_range=0.05)

train_generator=train_datagen.flow(X_train,y_train,batch_size=128)
```

2.) Build and Train CNN

```
#CNN

#import libraries
import keras
```

```
from keras.models import Sequential
from keras.layers import
Dense,Activation,Dropout,Conv2D,MaxPool2D,Flatten

#Build CNN model
model=Sequential()
model.add(Conv2D(32,kernel_size=(3,3),padding='same',strides=1,activat
ion='relu',kernel_initializer='he_normal',input_shape=(28,28,1)))
model.add(Conv2D(32,kernel_size=(3,3),padding='same',strides=1,activat
ion='relu',kernel_initializer='he_normal'))
model.add(MaxPool2D(pool_size=(2,2),strides=2,padding='valid'))
model.add(Dropout(0.25))

model.add(Conv2D(64,kernel_size=(3,3),strides=1,padding='same',activat
ion='relu',kernel_initializer='he_normal'))
model.add(Conv2D(64,kernel_size=(3,3),strides=1,padding='same',activat
ion='relu',kernel_initializer='he_normal'))
model.add(MaxPool2D(pool_size=(2,2),strides=2,padding='valid'))
model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(256,activation='relu',kernel_initializer='he_normal'))
model.add(Dropout(0.4))
model.add(Dense(10,activation='softmax',kernel_initializer='glorot_nor
mal'))

#Compiling model
model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.001),los
s='categorical_crossentropy',metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 28, 28, 32)        320
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 28, 28, 32)        9248
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
dropout_4 (Dropout)          (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 14, 14, 64)        18496
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 14, 14, 64)        36928
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
```

```
dropout_5 (Dropout)          (None, 7, 7, 64)          0

_________________________________________________________________
flatten_2 (Flatten)          (None, 3136)              0

_________________________________________________________________
dense_3 (Dense)              (None, 256)               803072

_________________________________________________________________
dropout_6 (Dropout)          (None, 256)               0

_________________________________________________________________
dense_4 (Dense)              (None, 10)                2570
=================================================================
Total params: 870,634
Trainable params: 870,634
Non-trainable params: 0

_________________________________________________________________
```

```
#Training model
def scheduler(epoch):
  if epoch < 10:
    return 0.001
  else:
    return 0.001 * np.exp(0.1 * (10 - epoch))
callback = keras.callbacks.LearningRateScheduler(scheduler)
history=model.fit(train_generator,epochs=50,validation_data=(X_val,y_val),callbacks=[callback],workers=5,use_multiprocessing=True)
```

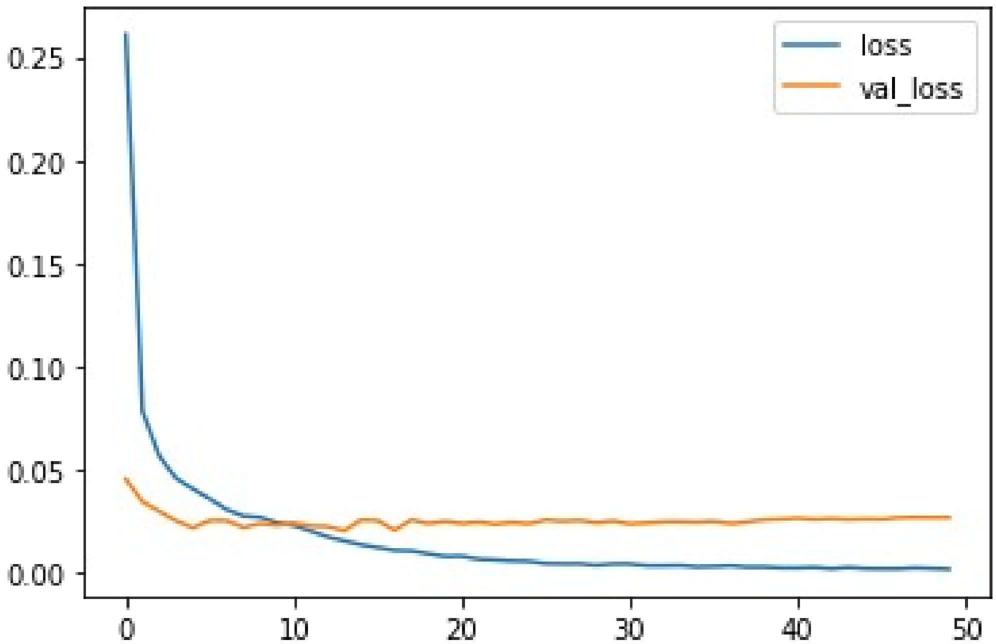
```
Epoch 1/50
386/387 [============================>.] - ETA: 0s - loss: 0.2614 - accuracy: 0.9170
387/387 [==============================] - 11s 28ms/step - loss: 0.2608 - accuracy: 0.9171 - val_loss: 0.0453 - val_accuracy: 0.9860
Epoch 2/50
387/387 [==============================] - 10s 27ms/step - loss: 0.0780 - accuracy: 0.9762 - val_loss: 0.0341 - val_accuracy: 0.9900
Epoch 3/50
387/387 [==============================] - 10s 26ms/step - loss: 0.0563 - accuracy: 0.9827 - val_loss: 0.0296 - val_accuracy: 0.9909
Epoch 4/50
387/387 [==============================] - 10s 27ms/step - loss: 0.0456 - accuracy: 0.9857 - val_loss: 0.0250 - val_accuracy: 0.9918
Epoch 5/50
387/387 [==============================] - 11s 27ms/step - loss: 0.0405 - accuracy: 0.9878 - val_loss: 0.0217 - val_accuracy: 0.9930
Epoch 6/50
387/387 [==============================] - 10s 27ms/step - loss: 0.0356 - accuracy: 0.9891 - val_loss: 0.0251 - val_accuracy: 0.9925
Epoch 7/50
387/387 [==============================] - 10s 27ms/step - loss: 0.0305 - accuracy: 0.9907 - val_loss: 0.0251 - val_accuracy: 0.9926
Epoch 8/50
387/387 [==============================] - 10s 27ms/step - loss:
```

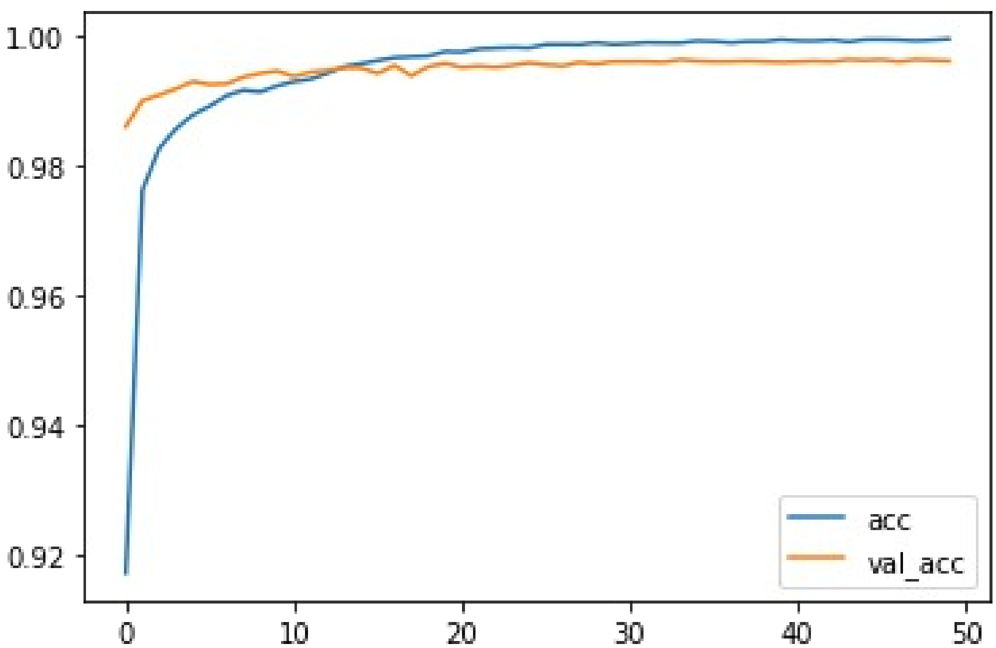
```
0.0273 - accuracy: 0.9916 - val_loss: 0.0218 - val_accuracy: 0.9936
Epoch 9/50
387/387 [==============================] - 10s 27ms/step - loss:
0.0267 - accuracy: 0.9913 - val_loss: 0.0237 - val_accuracy: 0.9942
Epoch 10/50
387/387 [==============================] - 10s 27ms/step - loss:
0.0240 - accuracy: 0.9923 - val_loss: 0.0228 - val_accuracy: 0.9946
Epoch 11/50
387/387 [==============================] - 10s 27ms/step - loss:
0.0228 - accuracy: 0.9929 - val_loss: 0.0239 - val_accuracy: 0.9937
Epoch 12/50
387/387 [==============================] - 10s 27ms/step - loss:
0.0201 - accuracy: 0.9933 - val_loss: 0.0225 - val_accuracy: 0.9944
Epoch 13/50
387/387 [==============================] - 10s 27ms/step - loss:
0.0172 - accuracy: 0.9942 - val_loss: 0.0223 - val_accuracy: 0.9947
Epoch 14/50
387/387 [==============================] - 11s 27ms/step - loss:
0.0153 - accuracy: 0.9952 - val_loss: 0.0202 - val_accuracy: 0.9950
Epoch 15/50
387/387 [==============================] - 10s 27ms/step - loss:
0.0134 - accuracy: 0.9957 - val_loss: 0.0254 - val_accuracy: 0.9950
Epoch 16/50
387/387 [==============================] - 10s 27ms/step - loss:
0.0120 - accuracy: 0.9962 - val_loss: 0.0251 - val_accuracy: 0.9942
Epoch 17/50
387/387 [==============================] - 10s 27ms/step - loss:
0.0107 - accuracy: 0.9966 - val_loss: 0.0204 - val_accuracy: 0.9954
Epoch 18/50
387/387 [==============================] - 10s 27ms/step - loss:
0.0104 - accuracy: 0.9967 - val_loss: 0.0255 - val_accuracy: 0.9938
Epoch 19/50
387/387 [==============================] - 10s 27ms/step - loss:
0.0090 - accuracy: 0.9968 - val_loss: 0.0237 - val_accuracy: 0.9952
Epoch 20/50
387/387 [==============================] - 10s 27ms/step - loss:
0.0077 - accuracy: 0.9975 - val_loss: 0.0248 - val_accuracy: 0.9957
Epoch 21/50
387/387 [==============================] - 10s 27ms/step - loss:
0.0079 - accuracy: 0.9975 - val_loss: 0.0237 - val_accuracy: 0.9951
Epoch 22/50
387/387 [==============================] - 10s 27ms/step - loss:
0.0064 - accuracy: 0.9980 - val_loss: 0.0244 - val_accuracy: 0.9953
Epoch 23/50
387/387 [==============================] - 10s 27ms/step - loss:
0.0061 - accuracy: 0.9981 - val_loss: 0.0234 - val_accuracy: 0.9951
Epoch 24/50
387/387 [==============================] - 10s 26ms/step - loss:
0.0055 - accuracy: 0.9982 - val_loss: 0.0242 - val_accuracy: 0.9954
```

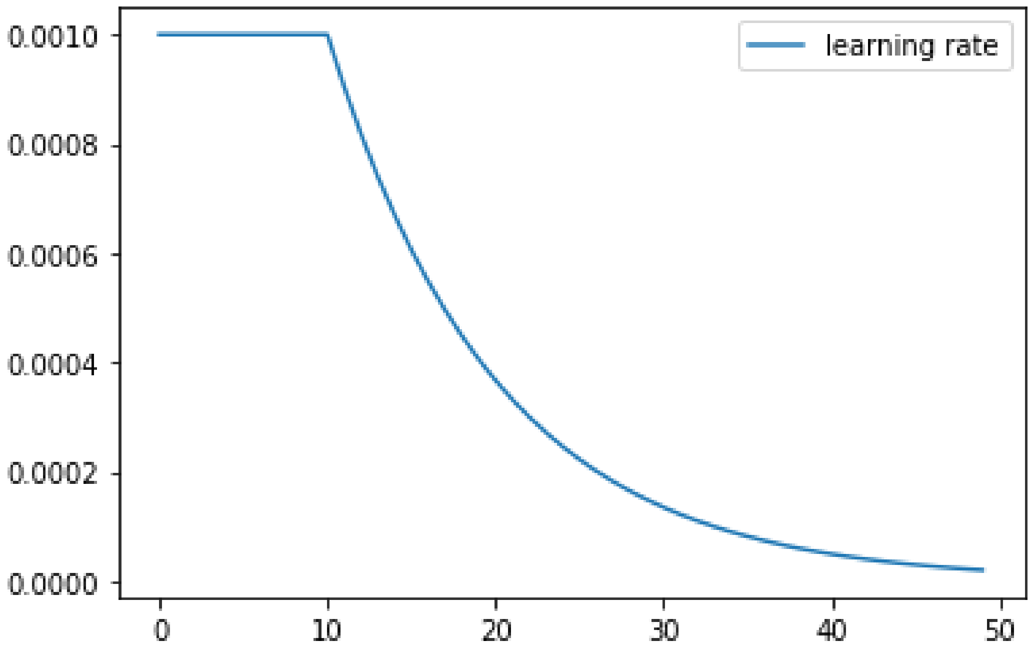
```
Epoch 25/50
387/387 [==============================] - 10s 27ms/step - loss:
0.0053 - accuracy: 0.9981 - val_loss: 0.0236 - val_accuracy: 0.9958
Epoch 26/50
387/387 [==============================] - 10s 26ms/step - loss:
0.0042 - accuracy: 0.9986 - val_loss: 0.0254 - val_accuracy: 0.9955
Epoch 27/50
387/387 [==============================] - 10s 26ms/step - loss:
0.0041 - accuracy: 0.9986 - val_loss: 0.0248 - val_accuracy: 0.9953
Epoch 28/50
387/387 [==============================] - 10s 27ms/step - loss:
0.0041 - accuracy: 0.9986 - val_loss: 0.0252 - val_accuracy: 0.9959
Epoch 29/50
387/387 [==============================] - 10s 27ms/step - loss:
0.0034 - accuracy: 0.9989 - val_loss: 0.0241 - val_accuracy: 0.9956
Epoch 30/50
387/387 [==============================] - 10s 26ms/step - loss:
0.0041 - accuracy: 0.9986 - val_loss: 0.0249 - val_accuracy: 0.9960
Epoch 31/50
387/387 [==============================] - 10s 26ms/step - loss:
0.0040 - accuracy: 0.9987 - val_loss: 0.0235 - val_accuracy: 0.9960
Epoch 32/50
387/387 [==============================] - 10s 26ms/step - loss:
0.0032 - accuracy: 0.9989 - val_loss: 0.0240 - val_accuracy: 0.9960
Epoch 33/50
387/387 [==============================] - 10s 26ms/step - loss:
0.0032 - accuracy: 0.9989 - val_loss: 0.0245 - val_accuracy: 0.9959
Epoch 34/50
387/387 [==============================] - 10s 27ms/step - loss:
0.0033 - accuracy: 0.9989 - val_loss: 0.0245 - val_accuracy: 0.9963
Epoch 35/50
387/387 [==============================] - 10s 26ms/step - loss:
0.0027 - accuracy: 0.9992 - val_loss: 0.0244 - val_accuracy: 0.9961
Epoch 36/50
387/387 [==============================] - 10s 26ms/step - loss:
0.0028 - accuracy: 0.9991 - val_loss: 0.0248 - val_accuracy: 0.9960
Epoch 37/50
387/387 [==============================] - 10s 26ms/step - loss:
0.0032 - accuracy: 0.9989 - val_loss: 0.0236 - val_accuracy: 0.9960
Epoch 38/50
387/387 [==============================] - 10s 26ms/step - loss:
0.0026 - accuracy: 0.9991 - val_loss: 0.0245 - val_accuracy: 0.9961
Epoch 39/50
387/387 [==============================] - 10s 26ms/step - loss:
0.0026 - accuracy: 0.9991 - val_loss: 0.0255 - val_accuracy: 0.9960
Epoch 40/50
387/387 [==============================] - 10s 26ms/step - loss:
0.0021 - accuracy: 0.9994 - val_loss: 0.0258 - val_accuracy: 0.9959
Epoch 41/50
```

```
387/387 [==============================] - 10s 26ms/step - loss:
0.0021 - accuracy: 0.9992 - val_loss: 0.0263 - val_accuracy: 0.9960
Epoch 42/50
387/387 [==============================] - 10s 26ms/step - loss:
0.0024 - accuracy: 0.9992 - val_loss: 0.0257 - val_accuracy: 0.9961
Epoch 43/50
387/387 [==============================] - 10s 26ms/step - loss:
0.0017 - accuracy: 0.9994 - val_loss: 0.0261 - val_accuracy: 0.9960
Epoch 44/50
387/387 [==============================] - 10s 26ms/step - loss:
0.0024 - accuracy: 0.9991 - val_loss: 0.0256 - val_accuracy: 0.9963
Epoch 45/50
387/387 [==============================] - 10s 26ms/step - loss:
0.0018 - accuracy: 0.9994 - val_loss: 0.0259 - val_accuracy: 0.9962
Epoch 46/50
387/387 [==============================] - 10s 27ms/step - loss:
0.0017 - accuracy: 0.9994 - val_loss: 0.0257 - val_accuracy: 0.9963
Epoch 47/50
387/387 [==============================] - 10s 26ms/step - loss:
0.0017 - accuracy: 0.9994 - val_loss: 0.0265 - val_accuracy: 0.9960
Epoch 48/50
387/387 [==============================] - 10s 26ms/step - loss:
0.0021 - accuracy: 0.9992 - val_loss: 0.0267 - val_accuracy: 0.9963
Epoch 49/50
387/387 [==============================] - 10s 26ms/step - loss:
0.0018 - accuracy: 0.9993 - val_loss: 0.0266 - val_accuracy: 0.9962
Epoch 50/50
387/387 [==============================] - 10s 26ms/step - loss:
0.0015 - accuracy: 0.9995 - val_loss: 0.0266 - val_accuracy: 0.9961
```

```python
#Some visualizations
import matplotlib.pyplot as plt
#Loss
plt.plot(history.history['loss'],label='loss')
plt.plot(history.history['val_loss'],label='val_loss')
plt.legend()
plt.show()
#Accuracy
plt.plot(history.history['accuracy'],label='acc')
plt.plot(history.history['val_accuracy'],label='val_acc')
plt.legend()
plt.show()
#Learning Rate
plt.plot(history.history['lr'],label='learning rate')
plt.legend()
plt.show()
```


loss
val_loss
0.25
0.20
0.15
0.10
0.05
0.00
0
10
20
30
40
50


acc
val_acc
1.00
0.98
0.96
0.94
0.92
0
10
20
30
40
50




```
train=model.evaluate(X_train,y_train)
print(train)
```

```
49500/49500 [==============================] - 4s 73us/step
[4.319644763384975e-05, 1.0]
```

```
val=model.evaluate(X_val,y_val)
print(val)
```

```
10500/10500 [==============================] - 1s 74us/step
[0.026639597142840775, 0.9960952401161194]
```

```
print(100*(1-train[1]))
print(100*(train[1]-val[1]))
```

```
0.0
0.3904759883880615
```

```
model.save('mnist-cnn.model')
```

3.) Evaluation on Test set

```
model.evaluate(X_test,y_test)
```

```
10000/10000 [==============================] - 1s 76us/step
```

```
[0.01838941933086469, 0.9961000084877014]
```