```
# Importing libraries
from __future__ import print_function
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K
from keras.initializers import he_normal
from keras.layers.normalization import BatchNormalization
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

batch_size = 128
num_classes = 10
epochs = 12

# input image dimensions
img_rows, img_cols = 28, 28

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
Using TensorFlow backend.
Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
```

```
if K.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

```
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
```

```
# this function is used draw Categorical Crossentropy Loss VS No. of epochs plot
def plt_dynamic(x, vy, ty):
  plt.figure(figsize=(10,5))
  plt.plot(x, vy, 'b', label="Validation Loss")
  plt.plot(x, ty, 'r', label="Train Loss")
  plt.xlabel('Epochs')
  plt.ylabel('Categorical Crossentropy Loss')
  plt.title('\nCategorical Crossentropy Loss VS Epochs')
  plt.legend()
  plt.grid()
  plt.show()
```

CNN with kernel [3x3] and 3 layers.

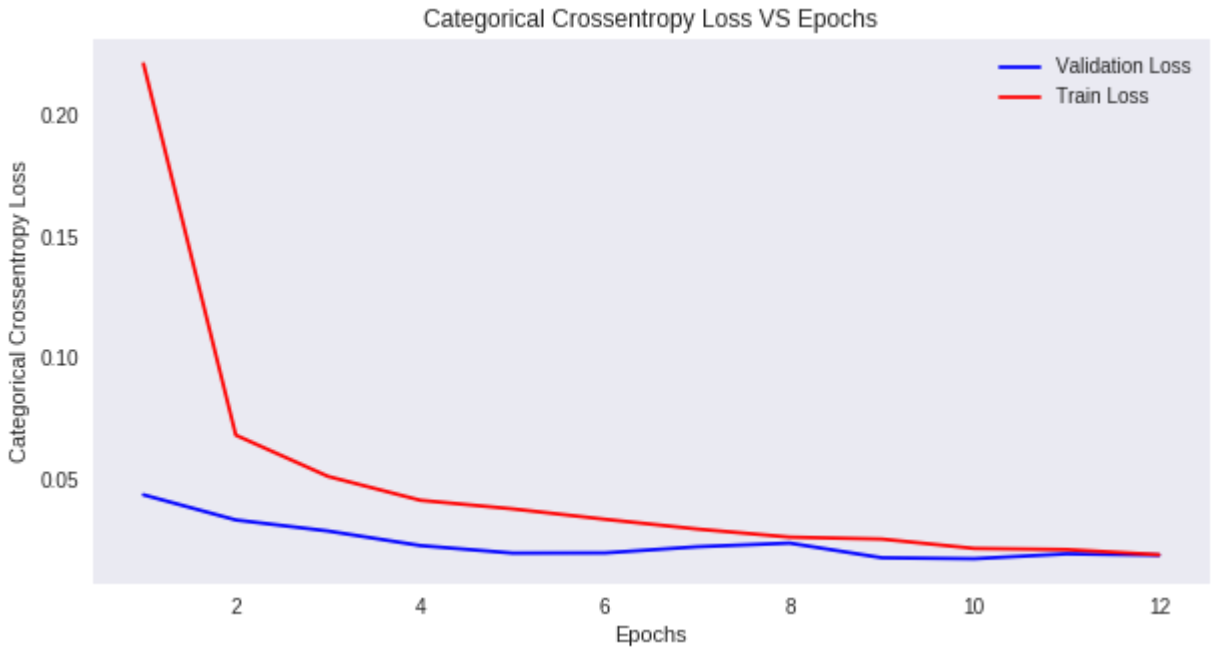
```
# Initialising the model
model_3 = Sequential()

# Adding first conv layer
model_3.add(Conv2D(32, kernel_size=(3, 3),activation='relu',input_shape=input_shape))

# Adding second conv layer
model_3.add(Conv2D(64, (3, 3), activation='relu'))

# Adding Maxpooling layer
model_3.add(MaxPooling2D(pool_size=(2, 2)))

# Adding Dropout
model_3.add(Dropout(0.25))

# Adding third conv layer
model_3.add(Conv2D(128, (3, 3), activation='relu'))

# Adding Maxpooling layer
model_3.add(MaxPooling2D(pool_size=(2, 2)))

# Adding Dropout
model_3.add(Dropout(0.25))

# Adding flatten layer
model_3.add(Flatten())

# Adding first hidden layer
model_3.add(Dense(256, activation='relu',kernel_initializer=he_normal(seed=None)))
```

```
# Adding Dropout
model_3.add(Dropout(0.5))

# Adding output layer
model_3.add(Dense(num_classes, activation='softmax'))

# Printing model Summary
print(model_3.summary())

# Compiling the model
model_3.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Fitting the data to the model
history_3 = model_3.fit(x_train, y_train,batch_size=batch_size,epochs=epochs,verbose=1,validation_data=(x_test, y_test))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320       
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 24, 24, 64)        18496     
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 64)        0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 12, 12, 64)        0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 10, 10, 128)       73856     
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 128)         0         
_________________________________________________________________
dropout_2 (Dropout)          (None, 5, 5, 128)         0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 3200)              0         
_________________________________________________________________
dense_1 (Dense)              (None, 256)               819456    
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 10)                2570      
=================================================================
Total params: 914,698
```

```
# Evaluating the model
score = model_3.evaluate(x_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

# Test and train accuracy of the model
model_3_test = score[1]
model_3_train = max(history_3.history['acc'])

# Plotting Train and Test Loss VS no. of epochs
# list of epoch numbers
x = list(range(1,epochs+1))

# Validation loss
vy = history_3.history['val_loss']
# Training loss
ty = history_3.history['loss']

# Calling the function to draw the plot
plt_dynamic(x, vy, ty)
```

```
Test score: 0.018467119540157728
Test accuracy: 0.9945
```



CNN with kernel [4x4] and 4 layers.

```
# Initialising the model
model_4 = Sequential()

# Adding first conv layer
model_4.add(Conv2D(8, kernel_size=(5, 5),padding='same',activation='relu',input_shape=input_shape))

# Adding second conv layer
model_4.add(Conv2D(16, (5, 5), activation='relu'))

# Adding Maxpooling layer
model_4.add(MaxPooling2D(pool_size=(2, 2),padding='same'))

# Adding Dropout
model_4.add(Dropout(0.25))

# Adding third conv layer
model_4.add(Conv2D(32, (5, 5),padding='same', activation='relu'))
```

```
# Adding Maxpooling layer
model_4.add(MaxPooling2D(pool_size=(2, 2),padding='same'))

# Adding Dropout
model_4.add(Dropout(0.25))

# Adding fourth conv layer
model_4.add(Conv2D(64, (5, 5),padding='same',activation='relu'))

# Adding Maxpooling layer
model_4.add(MaxPooling2D(pool_size=(2, 2),padding='same'))

# Adding Dropout
model_4.add(Dropout(0.25))

# Adding flatten layer
model_4.add(Flatten())

# Adding first hidden layer
model_4.add(Dense(256, activation='relu',kernel_initializer=he_normal(seed=None)))

# Adding Batch Normalization
model_4.add(BatchNormalization())

# Adding Dropout
model_4.add(Dropout(0.5))

# Adding output layer
model_4.add(Dense(num_classes, activation='softmax'))

# Printing model Summary
print(model_4.summary())

# Compiling the model
model_4.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Fitting the data to the model
history_4 = model_4.fit(x_train, y_train,batch_size=batch_size,epochs=epochs,verbose=1,validation_data=(x_test, y_test))
```

```
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 24, 24, 16)        3216      
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 12, 12, 16)        0         
_________________________________________________________________
dropout_4 (Dropout)          (None, 12, 12, 16)        0         
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 12, 12, 32)        12832     
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 6, 6, 32)          0         
_________________________________________________________________
dropout_5 (Dropout)          (None, 6, 6, 32)          0         
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 6, 6, 64)          51264     
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 3, 3, 64)          0         
_________________________________________________________________
dropout_6 (Dropout)          (None, 3, 3, 64)          0         
_________________________________________________________________
flatten_2 (Flatten)          (None, 576)               0         
_________________________________________________________________
dense_3 (Dense)              (None, 256)               147712    
_________________________________________________________________
batch_normalization_1 (Batch (None, 256)               1024      
_________________________________________________________________
dropout_7 (Dropout)          (None, 256)               0         
_________________________________________________________________
dense_4 (Dense)              (None, 10)                2570      
=================================================================
Total params: 218,826
Trainable params: 218,314
Non-trainable params: 512
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 136s 2ms/step - loss: 0.2902 - acc: 0.9069 - val_loss: 0.0411 - val_
Epoch 2/12
60000/60000 [==============================] - 135s 2ms/step - loss: 0.0797 - acc: 0.9754 - val_loss: 0.0278 - val_
Epoch 3/12
60000/60000 [==============================] - 130s 2ms/step - loss: 0.0615 - acc: 0.9811 - val_loss: 0.0222 - val_
Epoch 4/12
60000/60000 [==============================] - 130s 2ms/step - loss: 0.0528 - acc: 0.9837 - val_loss: 0.0211 - val_
Epoch 5/12
```

```
60000/60000 [==============================] - 131s 2ms/step - loss: 0.0461 - acc: 0.9856 - val_loss: 0.0227 - val_
Epoch 6/12
60000/60000 [==============================] - 130s 2ms/step - loss: 0.0403 - acc: 0.9871 - val_loss: 0.0229 - val_
Epoch 7/12
60000/60000 [==============================] - 129s 2ms/step - loss: 0.0387 - acc: 0.9883 - val_loss: 0.0262 - val_
```

```
# Evaluating the model
score = model_4.evaluate(x_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

# Test and train accuracy of the model
model_4_test = score[1]
model_4_train = max(history_4.history['acc'])

# Plotting Train and Test Loss VS no. of epochs
# list of epoch numbers
x = list(range(1,epochs+1))

# Validation loss
vy = history_4.history['val_loss']
# Training loss
ty = history_4.history['loss']

# Calling the function to draw the plot
plt_dynamic(x, vy, ty)
```

```
Test score: 0.024837667104176946
Test accuracy: 0.993
```

CNN with kernel [5x5] and 5 layers.


```
# Initialising the model
model_5 = Sequential()

# Adding first conv layer
model_5.add(Conv2D(8, kernel_size=(5, 5),padding='same',activation='relu',input_shape=input_shape))

# Adding second conv layer
model_5.add(Conv2D(16, (5, 5), activation='relu'))

# Adding Maxpooling layer
model_5.add(MaxPooling2D(pool_size=(2, 2),padding='same'))

# Adding Dropout
model_5.add(Dropout(0.25))

# Adding third conv layer
model_5.add(Conv2D(32, (5, 5),padding='same', activation='relu'))

# Adding Maxpooling layer
model_5.add(MaxPooling2D(pool_size=(2, 2),padding='same'))

# Adding Dropout
model_5.add(Dropout(0.25))

# Adding fourth conv layer
model_5.add(Conv2D(64, (5, 5),padding='same',activation='relu'))

# Adding fifth conv layer
model_5.add(Conv2D(64, (5, 5), activation='relu'))

# Adding Maxpooling layer
model_5.add(MaxPooling2D(pool_size=(2, 2),padding='same'))

# Adding Dropout
model_5.add(Dropout(0.25))

# Adding flatten layer
model_5.add(Flatten())

# Adding first hidden layer
model_5.add(Dense(256, activation='relu',kernel_initializer=he_normal(seed=None)))

# Adding Batch Normalization
model_5.add(BatchNormalization())
```

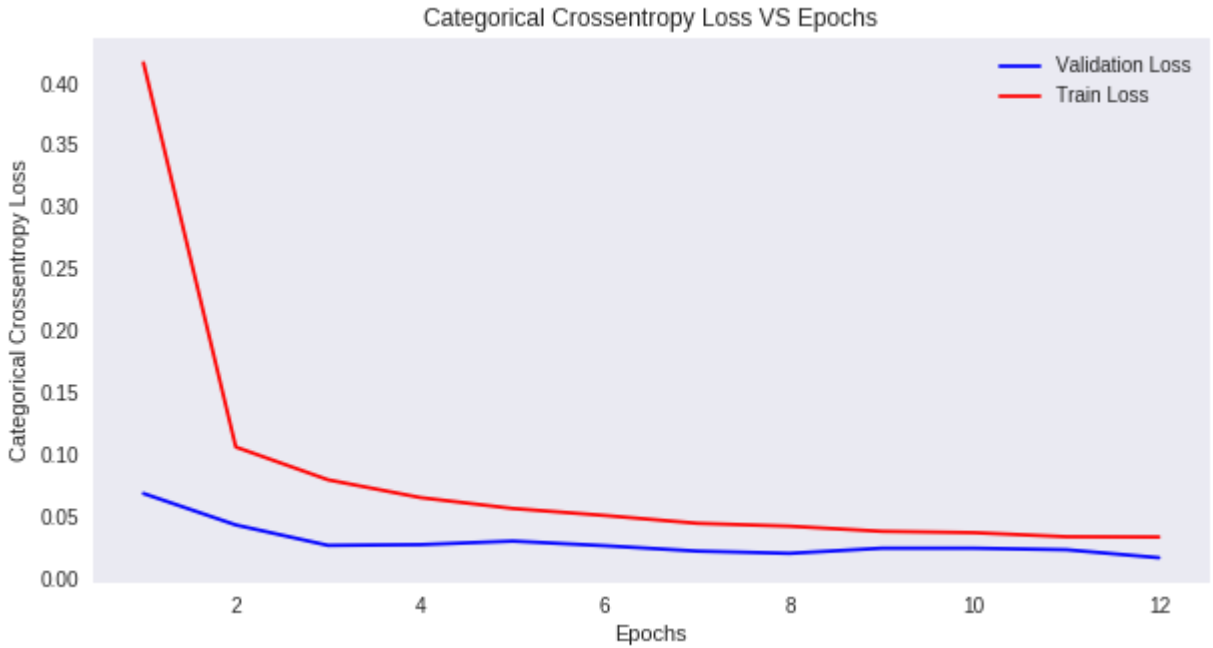
```
# Adding Dropout
model_5.add(Dropout(0.5))

# Adding output layer
model_5.add(Dense(num_classes, activation='softmax'))

# Printing model Summary
print(model_5.summary())

# Compiling the model
model_5.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Fitting the data to the model
history_5 = model_5.fit(x_train, y_train,batch_size=batch_size,epochs=epochs,verbose=1,validation_data=(x_test, y_test))
```

```
max_pooling2d_6 (MaxPooling2 (None, 12, 12, 16)        0
_________________________________________________________________
dropout_8 (Dropout)          (None, 12, 12, 16)        0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 12, 12, 32)        12832
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 6, 6, 32)          0
_________________________________________________________________
dropout_9 (Dropout)          (None, 6, 6, 32)          0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 6, 6, 64)          51264
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 2, 2, 64)          102464
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 1, 1, 64)          0
_________________________________________________________________
dropout_10 (Dropout)         (None, 1, 1, 64)          0
_________________________________________________________________
flatten_3 (Flatten)          (None, 64)                0
_________________________________________________________________
dense_5 (Dense)              (None, 256)               16640
_________________________________________________________________
batch_normalization_2 (Batch (None, 256)               1024
_________________________________________________________________
dropout_11 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_6 (Dense)              (None, 10)                2570
=================================================================
```

```
# Evaluating the model
score = model_5.evaluate(x_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

# Test and train accuracy of the model
model_5_test = score[1]
model_5_train = max(history_5.history['acc'])

# Plotting Train and Test Loss VS no. of epochs
# list of epoch numbers
x = list(range(1,epochs+1))

# Validation loss
vy = history_5.history['val_loss']
# Training loss
ty = history_5.history['loss']
```

```
# Calling the function to draw the plot
plt_dynamic(x, vy, ty)
```

```
Test score: 0.01672099802085286
Test accuracy: 0.9953
```



SUMMARY

```
print("Activation function= Relu")
print("Epochs= 12")
print("Batch size= 128")

from prettytable import PrettyTable
x=PrettyTable()
x.field_names = ["Kernel","CNN layers","Test accuracy","Optimal Epochs"]
x.add_row(["3x3","3 layers","0.9945","8"])
x.add_row(["4x4","4 layers","0.9930","12"])
x.add_row(["5x5","5 layers","0.9953","Didn't merge till 12 epochs"])

print(x)
```

```
Activation function= Relu
Epochs= 12
Batch size= 128
```

Kernel	CNN layers	Test accuracy	Optimal Epochs
3x3	3 layers	0.9945	8
4x4	4 layers	0.9930	12
5x5	5 layers	0.9953	Didn't merge till 12 epochs