CNN_MNIST

```
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.layers import Dropout
from keras.layers.normalization import BatchNormalization
```

```
Using TensorFlow backend.
```

CNN with 3 layers

```
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()

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)

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Conv2D(64,kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
model.add(Dropout(0.5))
```

```
model.add(Conv2D(64,kernel_size=(3,3), activation='relu',padding = 'same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
model.add(Dropout(0.5))


model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

history_1 = model.fit(x_train, y_train,batch_size=batch_size,epochs=epochs,verbose=1,validation_data
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
WARNING: Logging before flag parsing goes to stderr.
W0709 04:23:51.060149 140694541633408 deprecation_wrapper.py:119] From /usr/local/lib/py

W0709 04:23:51.095492 140694541633408 deprecation_wrapper.py:119] From /usr/local/lib/py

W0709 04:23:51.102290 140694541633408 deprecation_wrapper.py:119] From /usr/local/lib/py

W0709 04:23:51.159436 140694541633408 deprecation_wrapper.py:119] From /usr/local/lib/py

W0709 04:23:51.160586 140694541633408 deprecation_wrapper.py:119] From /usr/local/lib/py

x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
W0709 04:23:51.479045 140694541633408 deprecation_wrapper.py:119] From /usr/local/lib/py

W0709 04:23:51.570933 140694541633408 deprecation.py:506] From /usr/local/lib/python3.6/
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
W0709 04:23:51.604393 140694541633408 deprecation_wrapper.py:119] From /usr/local/lib/py

W0709 04:23:52.012918 140694541633408 deprecation_wrapper.py:119] From /usr/local/lib/py

W0709 04:23:52.135548 140694541633408 deprecation.py:323] From /usr/local/lib/python3.6/
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 267s 4ms/step - loss: 0.4379 - acc: 0.865
Epoch 2/12
60000/60000 [==============================] - 265s 4ms/step - loss: 0.1563 - acc: 0.955
Epoch 3/12
60000/60000 [==============================] - 269s 4ms/step - loss: 0.1162 - acc: 0.966
Epoch 4/12
60000/60000 [==============================] - 268s 4ms/step - loss: 0.0990 - acc: 0.971
Epoch 5/12
60000/60000 [==============================] - 268s 4ms/step - loss: 0.0858 - acc: 0.975
Epoch 6/12
60000/60000 [==============================] - 267s 4ms/step - loss: 0.0778 - acc: 0.977
Epoch 7/12
60000/60000 [==============================] - 264s 4ms/step - loss: 0.0768 - acc: 0.978
Epoch 8/12
60000/60000 [==============================] - 267s 4ms/step - loss: 0.0705 - acc: 0.980
Epoch 9/12
60000/60000 [==============================] - 266s 4ms/step - loss: 0.0683 - acc: 0.980
Epoch 10/12
60000/60000 [==============================] - 265s 4ms/step - loss: 0.0635 - acc: 0.981
Epoch 11/12
60000/60000 [==============================] - 264s 4ms/step - loss: 0.0624 - acc: 0.982
```

```
import matplotlib.pyplot as plt
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')
```

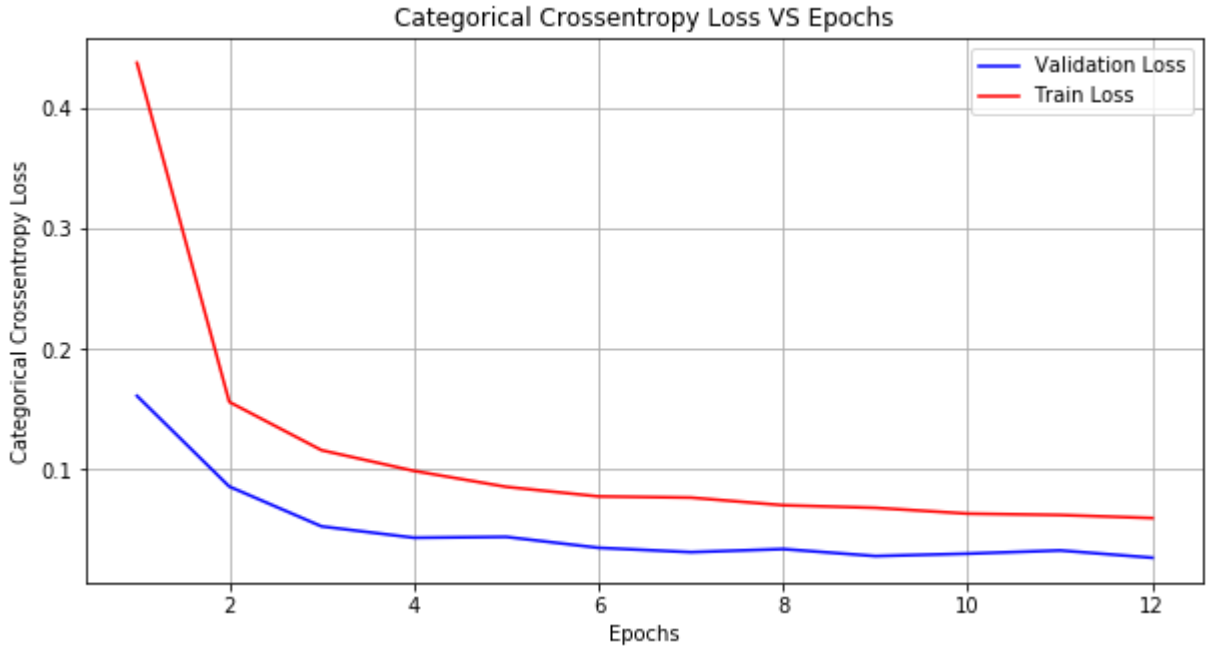
```
    plt.title('\nCategorical Crossentropy Loss VS Epochs')
    plt.legend()
    plt.grid()
    plt.show()


# Test and train accuracy of the model
model_test = score[1]
model_train = max(history_1.history['acc'])

# Plotting Train and Test Loss VS no. of epochs
# list of epoch numbers
x = list(range(1,13))

# Validation loss
vy = history_1.history['val_loss']
# Training loss
ty = history_1.history['loss']

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



CNN with 5 layers

```
model=Sequential()

model.add(Conv2D(64, kernel_size=(2, 2),activation='relu',input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(1, 1)))

#Second layer
model.add(Conv2D(32,kernel_size=(2,2), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.5))


#third layer
model.add(Conv2D(64,kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(1, 1)))
model.add(BatchNormalization())
```

```
model.add(Dropout(0.5))



#Fourth Layer
model.add(Conv2D(64,kernel_size=(2,2), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
model.add(Dropout(0.5))


#Fifth Layer
model.add(Conv2D(128,kernel_size=(5,5), activation='relu',padding = 'same'))
model.add(MaxPooling2D(pool_size=(3, 3)))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

history_2 = model.fit(x_train, y_train,batch_size=batch_size,epochs=epochs,verbose=1,validation_data
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

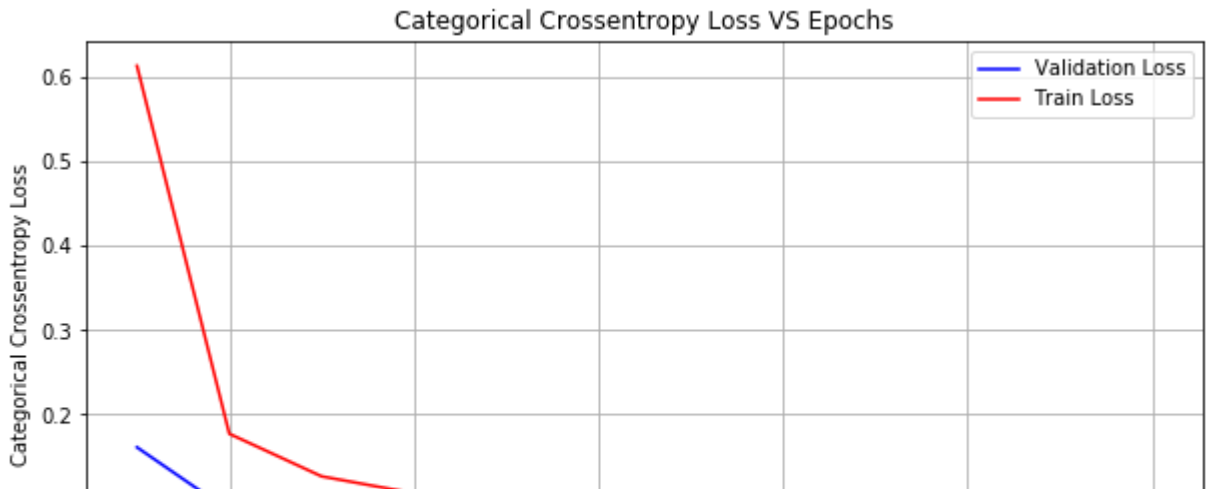
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 262s 4ms/step - loss: 0.6133 - acc: 0.803
Epoch 2/12
60000/60000 [==============================] - 260s 4ms/step - loss: 0.1773 - acc: 0.946
Epoch 3/12
60000/60000 [==============================] - 262s 4ms/step - loss: 0.1269 - acc: 0.961
Epoch 4/12
60000/60000 [==============================] - 262s 4ms/step - loss: 0.1068 - acc: 0.966
Epoch 5/12
60000/60000 [==============================] - 263s 4ms/step - loss: 0.0975 - acc: 0.971
Epoch 6/12
60000/60000 [==============================] - 265s 4ms/step - loss: 0.0878 - acc: 0.973
Epoch 7/12
60000/60000 [==============================] - 265s 4ms/step - loss: 0.0799 - acc: 0.976
Epoch 8/12
60000/60000 [==============================] - 265s 4ms/step - loss: 0.0779 - acc: 0.976
Epoch 9/12
60000/60000 [==============================] - 264s 4ms/step - loss: 0.0744 - acc: 0.978
Epoch 10/12
60000/60000 [==============================] - 264s 4ms/step - loss: 0.0696 - acc: 0.978
Epoch 11/12
60000/60000 [==============================] - 265s 4ms/step - loss: 0.0683 - acc: 0.979
Epoch 12/12
60000/60000 [==============================] - 267s 4ms/step - loss: 0.0647 - acc: 0.981
Test loss: 0.02990754226117715
Test accuracy: 0.9912
```

```
# Test and train accuracy of the model
model_test = score[1]
model_train = max(history_1.history['acc'])

# Plotting Train and Test Loss VS no. of epochs
# list of epoch numbers
x = list(range(1,13))

# Validation loss
vy = history_1.history['val_loss']
# Training loss
ty = history_2.history['loss']

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



CNN with 7 layers

```
model=Sequential()


#first layer
model.add(Conv2D(64, kernel_size=(2, 2),activation='relu',input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(1, 1)))

#Second layer
model.add(Conv2D(64,kernel_size=(2,2), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.5))


#third layer
model.add(Conv2D(32,kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(1, 1)))
model.add(BatchNormalization())
model.add(Dropout(0.5))



#Fourth Layer
model.add(Conv2D(32,kernel_size=(2,2), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
model.add(Dropout(0.5))

#Fifth layer
model.add(Conv2D(16, kernel_size=(2, 2), activation='relu'))
model.add(BatchNormalization(axis=1))
model.add(MaxPooling2D(pool_size=(1,1)))

#Sixth Layer
model.add(Conv2D(16,kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(1, 1)))
model.add(BatchNormalization())
model.add(Dropout(0.5))

#Seventh Layer
model.add(Conv2D(8,kernel_size=(3,3), activation='relu',padding = 'same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Flatten())
```

```
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

history_2 = model.fit(x_train, y_train,batch_size=batch_size,epochs=epochs,verbose=1,validation_data
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

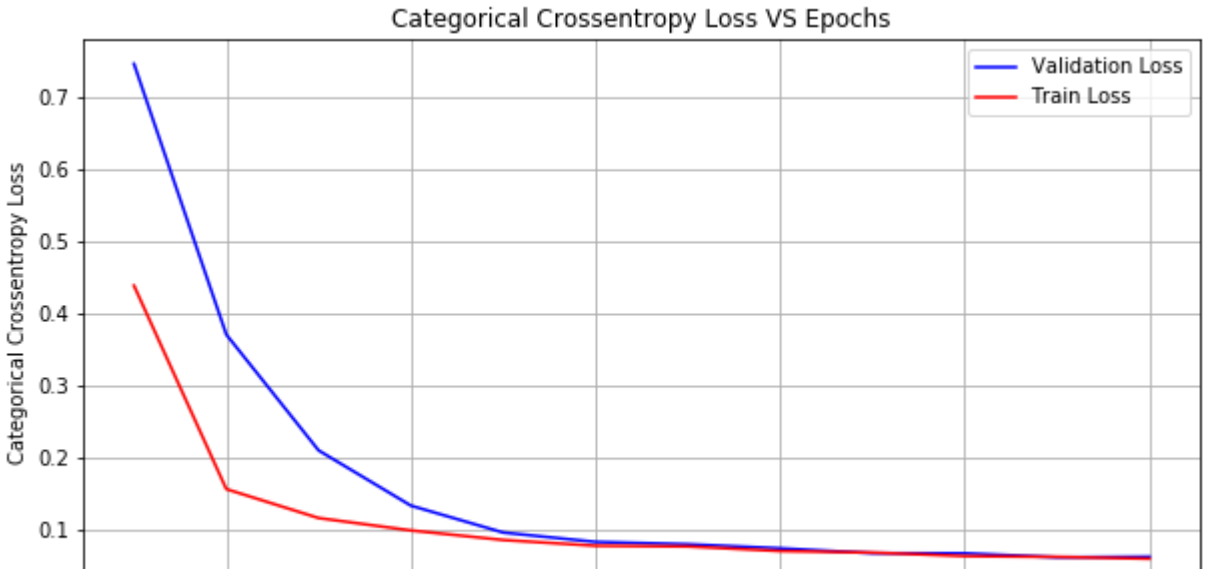
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 227s 4ms/step - loss: 1.7556 - acc: 0.346
Epoch 2/12
60000/60000 [==============================] - 224s 4ms/step - loss: 1.0674 - acc: 0.583
Epoch 3/12
60000/60000 [==============================] - 225s 4ms/step - loss: 0.8326 - acc: 0.673
Epoch 4/12
60000/60000 [==============================] - 224s 4ms/step - loss: 0.7163 - acc: 0.723
Epoch 5/12
60000/60000 [==============================] - 225s 4ms/step - loss: 0.6444 - acc: 0.750
Epoch 6/12
60000/60000 [==============================] - 224s 4ms/step - loss: 0.5991 - acc: 0.772
Epoch 7/12
60000/60000 [==============================] - 225s 4ms/step - loss: 0.5623 - acc: 0.790
Epoch 8/12
60000/60000 [==============================] - 225s 4ms/step - loss: 0.5326 - acc: 0.804
Epoch 9/12
60000/60000 [==============================] - 224s 4ms/step - loss: 0.5149 - acc: 0.812
Epoch 10/12
60000/60000 [==============================] - 222s 4ms/step - loss: 0.4960 - acc: 0.823
Epoch 11/12
60000/60000 [==============================] - 221s 4ms/step - loss: 0.4749 - acc: 0.831
Epoch 12/12
60000/60000 [==============================] - 220s 4ms/step - loss: 0.4674 - acc: 0.836
Test loss: 0.06263806874752045
Test accuracy: 0.9818
```

```
# Test and train accuracy of the model
model_test = score[1]
model_train = max(history_1.history['acc'])

# Plotting Train and Test Loss VS no. of epochs
# list of epoch numbers
x = list(range(1,13))

# Validation loss
vy = history_2.history['val_loss']
# Training loss
ty = history_1.history['loss']

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



Performance Table

```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["No of layers", "Accuracy %"]

x.add_row(["3","99.22"])
x.add_row(["5","99.12"])
x.add_row(["7","98.18"])


print(x)
```

```
+--------------+------------+
| No of layers | Accuracy % |
+--------------+------------+
|      3       |   99.22    |
|      5       |   99.12    |
|      7       |   98.18    |
+--------------+------------+
```

1. We applied 3 different convolutional neural network
2. We used different kernel_size, max_pooling_size, batch narmalization and drop out technique
3. we can observe that after increasing number of layers our accuracy decreases.