```python
# Credits: https://github.com/keras-team/keras/blob/master/examples/mnist_cnn.py

%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings("ignore")

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

batch_size = 128
num_classes = 10
epochs = 10

# 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)

```

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

```python
import numpy as np
import time
# https://gist.github.com/greydanus/f6eee59eaf1d90fcb3b534a25362cea4
# https://stackoverflow.com/a/14434334
# this function is used to update the plots for each epoch and error
def plt_dynamic(x, vy, ty, fig, ax, colors=['b']):
    ax.plot(x, vy, 'b', label="Validation Loss")
    ax.plot(x, ty, 'r', label="Train Loss")
    plt.legend()
    plt.grid()
    fig.canvas.draw()
```

```python
def loss_plot(model_name):
    score = model_name.evaluate(x_test, y_test, verbose=0)
    print('Test score:', score[0])
    print('Test accuracy:', score[1])

    fig, ax = plt.subplots(1,1, figsize=(10,6))
    ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')
    ax.set_title('Variation of Loss with epochs')

    # list of epoch numbers
    x = list(range(1,epochs+1))

    # print(history.history.keys())
    # dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
    # history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))

    # we will get val_loss and val_acc only when you pass the paramter validation_data
    # val_loss : validation loss
    # val_acc : validation accuracy

    # loss : training loss
    # acc : train accuracy
    # for each key in histrory.histrory we will have a list of length equal to number of epochs


    vy = history.history['val_loss']
    ty = history.history['loss']
    plt_dynamic(x, vy, ty, fig, ax)
```

```python
def weight_plot(model_name):
    w_after = model_name.get_weights()

    h1_w = w_after[0].flatten().reshape(-1,1)
```

```
    h2_w = w_after[2].flatten().reshape(-1,1)
    out_w = w_after[4].flatten().reshape(-1,1)

    fig = plt.figure(figsize=(15,7))
    fig.suptitle("Weight matrices after model trained")
    plt.subplot(1, 3, 1)
    ax = sns.violinplot(y=h1_w,color='b')
    plt.xlabel('Hidden Layer 1')

    plt.subplot(1, 3, 2)
    ax = sns.violinplot(y=h2_w, color='r')
    plt.xlabel('Hidden Layer 2 ')

    plt.subplot(1, 3, 3)
    ax = sns.violinplot(y=out_w,color='y')
    plt.xlabel('Output Layer ')
    plt.show()
```

Architecture 1:

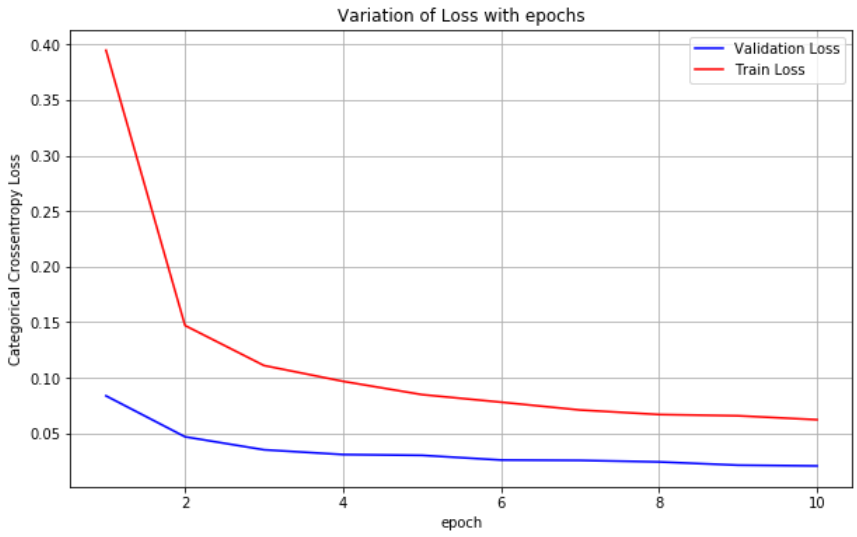
no. of Layers = 3; Kernel_size = 3x3

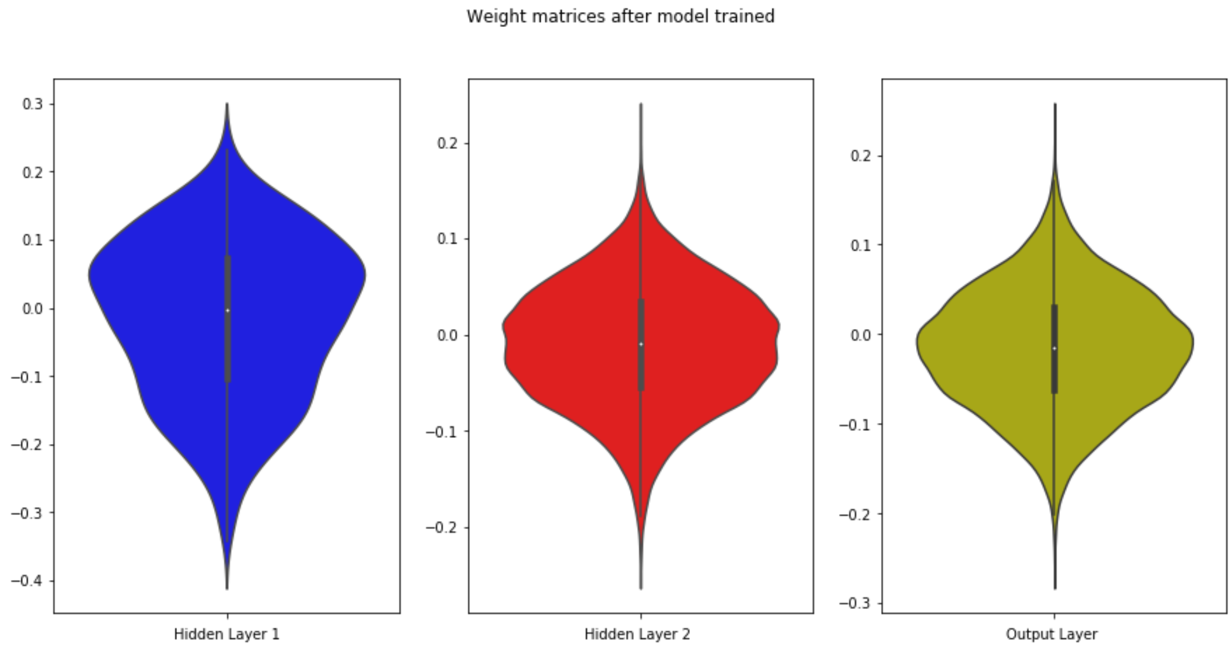
```
model1 = Sequential()
#Layer 1
model1.add(Conv2D(48, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
#Layer 2
model1.add(Conv2D(64, (3, 3), activation='relu'))
model1.add(MaxPooling2D(pool_size=(2, 2)))
model1.add(Dropout(0.5))

#Layer 3
model1.add(Conv2D(50, (3, 3), activation='relu'))
model1.add(MaxPooling2D(pool_size=(2, 2)))
model1.add(Dropout(0.5))

model1.add(Flatten())
model1.add(Dense(128, activation='relu'))
model1.add(Dropout(0.5))
model1.add(Dense(num_classes, activation='softmax'))
print(model1.summary())

model1.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

history = model1.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, verbose=1, validation_data=(x_test, y_test))
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_18 (Conv2D)           (None, 26, 26, 48)        480       
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 24, 24, 64)        27712     
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 12, 12, 64)        0         
_________________________________________________________________
dropout_16 (Dropout)         (None, 12, 12, 64)        0         
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 10, 10, 50)        28850     
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 5, 5, 50)          0         
_________________________________________________________________
dropout_17 (Dropout)         (None, 5, 5, 50)          0         
_________________________________________________________________
flatten_4 (Flatten)          (None, 1250)              0         
_________________________________________________________________
dense_7 (Dense)              (None, 128)               160128    
_________________________________________________________________
dropout_18 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_8 (Dense)              (None, 10)                1290      
=================================================================
Total params: 218,460
Trainable params: 218,460
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 5s 88us/step - loss: 0.3946 - acc: 0.8724 - val_loss: 0.0837 - val_acc: 0.9743
Epoch 2/10
60000/60000 [==============================] - 4s 71us/step - loss: 0.1470 - acc: 0.9553 - val_loss: 0.0469 - val_acc: 0.9843
Epoch 3/10
60000/60000 [==============================] - 4s 72us/step - loss: 0.1110 - acc: 0.9662 - val_loss: 0.0352 - val_acc: 0.9877
Epoch 4/10
60000/60000 [==============================] - 4s 72us/step - loss: 0.0968 - acc: 0.9710 - val_loss: 0.0309 - val_acc: 0.9887
Epoch 5/10
60000/60000 [==============================] - 4s 72us/step - loss: 0.0848 - acc: 0.9741 - val_loss: 0.0302 - val_acc: 0.9901
Epoch 6/10
60000/60000 [==============================] - 4s 72us/step - loss: 0.0780 - acc: 0.9770 - val_loss: 0.0260 - val_acc: 0.9916
Epoch 7/10
60000/60000 [==============================] - 4s 72us/step - loss: 0.0710 - acc: 0.9791 - val_loss: 0.0257 - val_acc: 0.9910
Epoch 8/10
60000/60000 [==============================] - 4s 72us/step - loss: 0.0669 - acc: 0.9801 - val_loss: 0.0243 - val_acc: 0.9916
Epoch 9/10
60000/60000 [==============================] - 4s 72us/step - loss: 0.0658 - acc: 0.9804 - val_loss: 0.0214 - val_acc: 0.9940
Epoch 10/10
60000/60000 [==============================] - 4s 72us/step - loss: 0.0623 - acc: 0.9818 - val_loss: 0.0207 - val_acc: 0.9935
```

```
loss_plot(model1)
weight_plot(model1)
```

```
Test score: 0.02070148353522236
Test accuracy: 0.9935
```





Architecture 2:

no. of Layers = 5; Kernel_size = 5x5

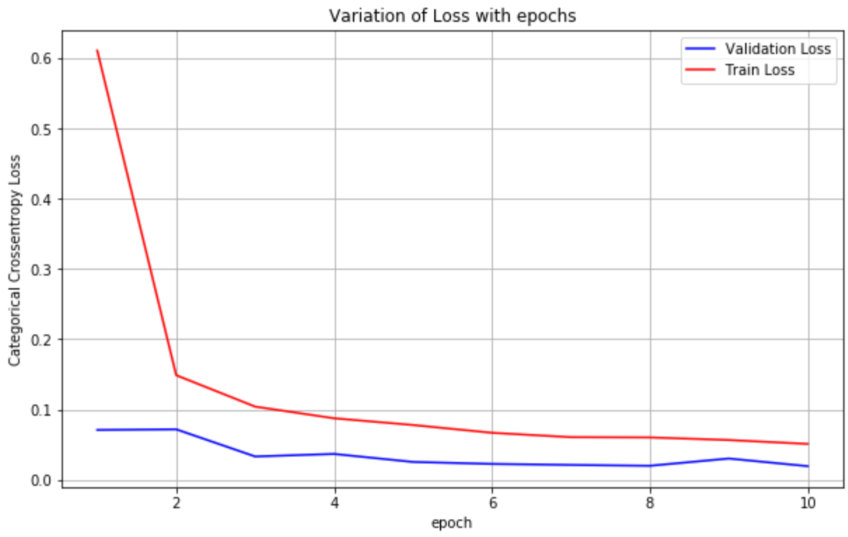
```
model2 = Sequential()
#Layer 1
model2.add(Conv2D(50, kernel_size=(5, 5), padding='same',
                  activation='relu',
                  input_shape=input_shape))

#Layer 2
model2.add(Conv2D(50, (5, 5), padding='same', activation='relu'))
model2.add(MaxPooling2D(pool_size=(2, 2)))
model2.add(Dropout(0.5))

#Layer 3
model2.add(Conv2D(60, (5, 5), padding='same', activation='relu'))
model2.add(Dropout(0.5))

#Layer 4
model2.add(Conv2D(40, (5, 5), padding='same', activation='relu'))
model2.add(MaxPooling2D(pool_size=(3, 3), padding='same'))
model2.add(Dropout(0.5))

#Layer 5
model2.add(Conv2D(30, (5, 5), padding='same', activation='relu'))
model2.add(Dropout(0.5))

model2.add(Flatten())
model2.add(Dense(128, activation='relu'))
model2.add(Dropout(0.5))
model2.add(Dense(num_classes, activation='softmax'))

print(model2.summary())

model2.compile(loss=keras.losses.categorical_crossentropy,
```

```
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

history = model2.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_70 (Conv2D)           (None, 28, 28, 50)        1300      
_________________________________________________________________
conv2d_71 (Conv2D)           (None, 28, 28, 50)        62550     
_________________________________________________________________
max_pooling2d_55 (MaxPooling (None, 14, 14, 50)        0         
_________________________________________________________________
dropout_70 (Dropout)         (None, 14, 14, 50)        0         
_________________________________________________________________
conv2d_72 (Conv2D)           (None, 14, 14, 60)        75060     
_________________________________________________________________
dropout_71 (Dropout)         (None, 14, 14, 60)        0         
_________________________________________________________________
conv2d_73 (Conv2D)           (None, 14, 14, 40)        60040     
_________________________________________________________________
max_pooling2d_56 (MaxPooling (None, 5, 5, 40)          0         
_________________________________________________________________
dropout_72 (Dropout)         (None, 5, 5, 40)          0         
_________________________________________________________________
conv2d_74 (Conv2D)           (None, 5, 5, 30)          30030     
_________________________________________________________________
dropout_73 (Dropout)         (None, 5, 5, 30)          0         
_________________________________________________________________
flatten_12 (Flatten)         (None, 750)               0         
_________________________________________________________________
dense_23 (Dense)             (None, 128)               96128     
_________________________________________________________________
dropout_74 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_24 (Dense)             (None, 10)                1290      
=================================================================
Total params: 326,398
Trainable params: 326,398
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 11s 189us/step - loss: 0.6108 - acc: 0.7979 - val_loss: 0.0709 - val_acc: 0.97
Epoch 2/10
60000/60000 [==============================] - 7s 112us/step - loss: 0.1488 - acc: 0.9578 - val_loss: 0.0718 - val_acc: 0.977
Epoch 3/10
60000/60000 [==============================] - 7s 112us/step - loss: 0.1041 - acc: 0.9701 - val_loss: 0.0332 - val_acc: 0.989
Epoch 4/10
60000/60000 [==============================] - 7s 112us/step - loss: 0.0875 - acc: 0.9750 - val_loss: 0.0368 - val_acc: 0.988
Epoch 5/10
60000/60000 [==============================] - 7s 114us/step - loss: 0.0779 - acc: 0.9792 - val_loss: 0.0254 - val_acc: 0.991
Epoch 6/10
60000/60000 [==============================] - 7s 113us/step - loss: 0.0669 - acc: 0.9810 - val_loss: 0.0226 - val_acc: 0.993
Epoch 7/10
60000/60000 [==============================] - 7s 113us/step - loss: 0.0606 - acc: 0.9826 - val_loss: 0.0211 - val_acc: 0.992
Epoch 8/10
60000/60000 [==============================] - 7s 114us/step - loss: 0.0603 - acc: 0.9838 - val_loss: 0.0199 - val_acc: 0.994
Epoch 9/10
60000/60000 [==============================] - 7s 116us/step - loss: 0.0566 - acc: 0.9851 - val_loss: 0.0303 - val_acc: 0.990
Epoch 10/10
60000/60000 [==============================] - 7s 112us/step - loss: 0.0511 - acc: 0.9860 - val_loss: 0.0194 - val_acc: 0.994
```

```
loss_plot(model2)
```

```
Test score: 0.019397488478862943
Test accuracy: 0.994
```



Architecture 3:

no. of Layers = 7; Kernel_size = 6x6

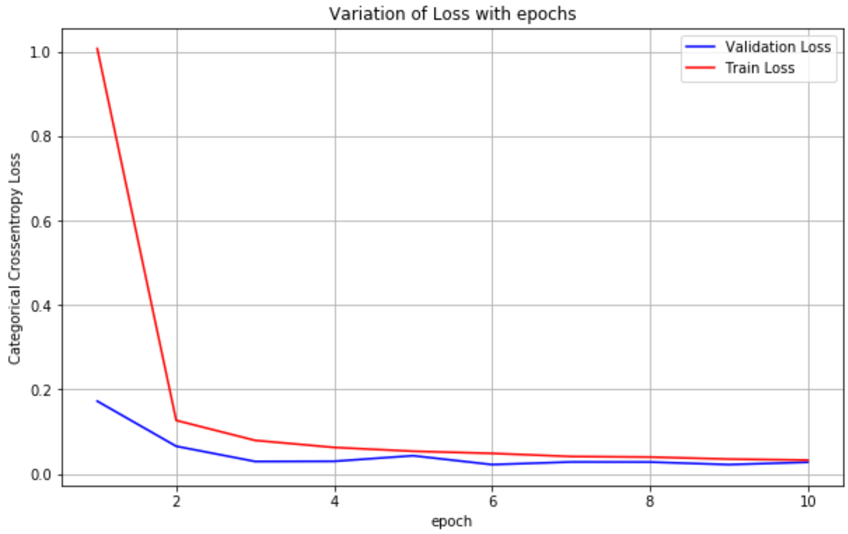
along with Batch Normalization and Dropout layers

```
from keras.layers.normalization import BatchNormalization

model3 = Sequential()
#Layer 1
model3.add(Conv2D(70, kernel_size=(6, 6), padding='same',
                 activation='relu',
                 input_shape=input_shape))
model3.add(BatchNormalization())

#Layer 2
model3.add(Conv2D(80, (6, 6), padding='same', activation='relu'))
model3.add(BatchNormalization())
model3.add(MaxPooling2D(pool_size=(2, 2), padding='same'))
model3.add(Dropout(0.5))

#Layer 3
model3.add(Conv2D(60, (6, 6), padding='same', activation='relu'))
model3.add(BatchNormalization())
model3.add(Dropout(0.5))

#Layer 4
model3.add(Conv2D(50, (6, 6), padding='same', activation='relu'))
model3.add(BatchNormalization())
model3.add(MaxPooling2D(pool_size=(2, 2), padding='same'))
model3.add(Dropout(0.5))

#Layer 5
model3.add(Conv2D(48, (6, 6), padding='same', activation='relu'))
model3.add(BatchNormalization())
model3.add(Dropout(0.5))

#Layer 6
model3.add(Conv2D(36, (6, 6), padding='same', activation='relu'))
model3.add(BatchNormalization())
model3.add(Dropout(0.5))

#Layer 7
model3.add(Conv2D(24, (6, 6), padding='same', activation='relu'))
model3.add(BatchNormalization())
model3.add(MaxPooling2D(pool_size=(2, 2), padding='same'))
model3.add(Dropout(0.5))

model3.add(Flatten())
model3.add(Dense(128, activation='relu'))
model3.add(Dropout(0.5))
model3.add(Dense(num_classes, activation='softmax'))

print(model3.summary())

model3.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

history = model3.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, verbose=1, validation_data=(x_test, y_test))

```

```
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_63 (Conv2D)           (None, 28, 28, 70)        2590      
_________________________________________________________________
batch_normalization_49 (Batc (None, 28, 28, 70)        280       
_________________________________________________________________
conv2d_64 (Conv2D)           (None, 28, 28, 80)        201680    
_________________________________________________________________
batch_normalization_50 (Batc (None, 28, 28, 80)        320       
_________________________________________________________________
max_pooling2d_52 (MaxPooling (None, 14, 14, 80)        0         
_________________________________________________________________
dropout_63 (Dropout)         (None, 14, 14, 80)        0         
_________________________________________________________________
conv2d_65 (Conv2D)           (None, 14, 14, 60)        172860    
_________________________________________________________________
batch_normalization_51 (Batc (None, 14, 14, 60)        240       
_________________________________________________________________
dropout_64 (Dropout)         (None, 14, 14, 60)        0         
_________________________________________________________________
conv2d_66 (Conv2D)           (None, 14, 14, 50)        108050    
_________________________________________________________________
batch_normalization_52 (Batc (None, 14, 14, 50)        200       
_________________________________________________________________
max_pooling2d_53 (MaxPooling (None, 7, 7, 50)          0         
_________________________________________________________________
dropout_65 (Dropout)         (None, 7, 7, 50)          0         
_________________________________________________________________
conv2d_67 (Conv2D)           (None, 7, 7, 48)          86448     
_________________________________________________________________
batch_normalization_53 (Batc (None, 7, 7, 48)          192       
_________________________________________________________________
dropout_66 (Dropout)         (None, 7, 7, 48)          0         
_________________________________________________________________
conv2d_68 (Conv2D)           (None, 7, 7, 36)          62244     
_________________________________________________________________
batch_normalization_54 (Batc (None, 7, 7, 36)          144       
_________________________________________________________________
dropout_67 (Dropout)         (None, 7, 7, 36)          0         
_________________________________________________________________
conv2d_69 (Conv2D)           (None, 7, 7, 24)          31128     
_________________________________________________________________
batch_normalization_55 (Batc (None, 7, 7, 24)          96        
_________________________________________________________________
max_pooling2d_54 (MaxPooling (None, 4, 4, 24)          0         
_________________________________________________________________
dropout_68 (Dropout)         (None, 4, 4, 24)          0         
_________________________________________________________________
flatten_11 (Flatten)         (None, 384)               0         
_________________________________________________________________
dense_21 (Dense)             (None, 128)               49280     
_________________________________________________________________
dropout_69 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_22 (Dense)             (None, 10)                1290      
=================================================================
Total params: 717,042
Trainable params: 716,306
Non-trainable params: 736
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 23s 384us/step - loss: 1.0074 - acc: 0.6573 - val_loss: 0.1724 - val_acc: 0.96
Epoch 2/10
60000/60000 [==============================] - 18s 296us/step - loss: 0.1268 - acc: 0.9660 - val_loss: 0.0655 - val_acc: 0.98
Epoch 3/10
60000/60000 [==============================] - 18s 299us/step - loss: 0.0793 - acc: 0.9792 - val_loss: 0.0294 - val_acc: 0.99
Epoch 4/10
60000/60000 [==============================] - 18s 297us/step - loss: 0.0627 - acc: 0.9842 - val_loss: 0.0299 - val_acc: 0.99
Epoch 5/10
60000/60000 [==============================] - 18s 297us/step - loss: 0.0537 - acc: 0.9861 - val_loss: 0.0431 - val_acc: 0.98
Epoch 6/10
60000/60000 [==============================] - 18s 297us/step - loss: 0.0486 - acc: 0.9877 - val_loss: 0.0221 - val_acc: 0.99
Epoch 7/10
60000/60000 [==============================] - 18s 296us/step - loss: 0.0413 - acc: 0.9892 - val_loss: 0.0283 - val_acc: 0.99
Epoch 8/10
60000/60000 [==============================] - 18s 296us/step - loss: 0.0399 - acc: 0.9904 - val_loss: 0.0281 - val_acc: 0.99
Epoch 9/10
60000/60000 [==============================] - 18s 297us/step - loss: 0.0351 - acc: 0.9909 - val_loss: 0.0220 - val_acc: 0.99
Epoch 10/10
60000/60000 [==============================] - 18s 298us/step - loss: 0.0326 - acc: 0.9914 - val_loss: 0.0277 - val_acc: 0.99
```

```
loss_plot(model3)
```

```
Test score: 0.02765477129972496
Test accuracy: 0.9945
```



Summary

```python
# Please compare all your models using Prettytable library
from prettytable import PrettyTable
x = PrettyTable()
print('\nResults : CNN on MNIST data')
print(40*'=')

x.field_names = ["Hidden layers", "Kernel size", "Padding", "Max pooling", "epochs", "Train error", "Train accuracey", "Test error", "Te
x.add_row([3, '3X3', 'valid', '(2,2)', 10, 0.0623, '98.18 %', 0.0207, '99.35 %'])
x.add_row([5, '5X5', 'same', '(2,2)', 10, 0.0511, '98.60 %', 0.0194, '99.40 %'])
x.add_row([7, '6X6', 'same', '(2,2)', 10, 0.0326, '99.14 %', 0.0277, '99.45 %'])

print(x)
```

```
Results : CNN on MNIST data
========================================
+---------------+-------------+---------+-------------+--------+-------------+-----------------+------------+---------------+
| Hidden layers | Kernel size | Padding | Max pooling | epochs | Train error | Train accuracey | Test error | Test accuracy |
+---------------+-------------+---------+-------------+--------+-------------+-----------------+------------+---------------+
|       3       |     3X3     |  valid  |    (2,2)    |   10   |    0.0623   |     98.18 %     |   0.0207   |    99.35 %    |
|       5       |     5X5     |   same  |    (2,2)    |   10   |    0.0511   |     98.60 %     |   0.0194   |    99.40 %    |
|       7       |     6X6     |   same  |    (2,2)    |   10   |    0.0326   |     99.14 %     |   0.0277   |    99.45 %    |
+---------------+-------------+---------+-------------+--------+-------------+-----------------+------------+---------------+
```