```
# Credits: https://github.com/keras-team/keras/blob/master/examples/mnist_cnn.p
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
import matplotlib.pyplot as plt
import seaborn as sns
%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()

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(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
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'])

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

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/t

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/t

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/t
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - ke
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/optimizer

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/t

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/t

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/t

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/t

Train on 60000 samples, validate on 10000 samples
Epoch 1/12
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/t

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/t

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/t

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/t

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/t

60000/60000 [==============================] - 20s 338us/step - loss: 0.2541 -
Epoch 2/12
60000/60000 [==============================] - 6s 100us/step - loss: 0.0840 -
Epoch 3/12
60000/60000 [==============================] - 6s 102us/step - loss: 0.0637 -
Epoch 4/12
60000/60000 [==============================] - 6s 102us/step - loss: 0.0518 -
Epoch 5/12
60000/60000 [==============================] - 6s 102us/step - loss: 0.0443 -
Epoch 6/12
60000/60000 [==============================] - 6s 102us/step - loss: 0.0405 -
Epoch 7/12
60000/60000 [==============================] - 6s 104us/step - loss: 0.0355 -
Epoch 8/12
60000/60000 [==============================] - 6s 101us/step - loss: 0.0327 -
Epoch 9/12
60000/60000 [==============================] - 6s 102us/step - loss: 0.0296 -
Epoch 10/12
60000/60000 [==============================] - 6s 102us/step - loss: 0.0262 -
Epoch 11/12
60000/60000 [==============================] - 6s 102us/step - loss: 0.0254 -
Epoch 12/12
60000/60000 [==============================] - 6s 102us/step - loss: 0.0256 -
Test loss: 0.027925510946988835
Test accuracy: 0.9905
```

================ Assignment ======================

```
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, 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()
```

Model 1: 7 conv layers + filter size = 3x3 + relu activation + d

```
model = Sequential()

model.add(Conv2D(20, kernel_size=(3, 3),activation='relu',input_shape=input_sha
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(25, (3, 3), activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(35, (3, 3), activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(45, (3, 3), activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(55, (3, 3), activation='relu',padding='same'))


model.add(Conv2D(65, (3, 3), activation='relu',padding='same'))

```

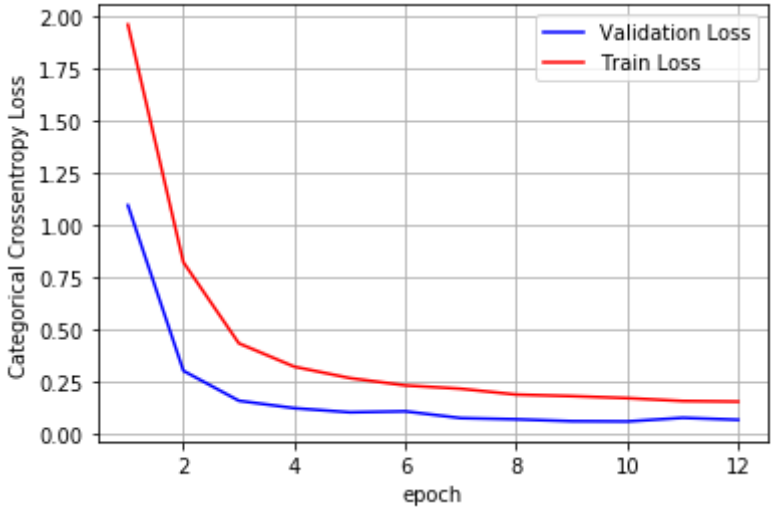
```
22
23 model.add(Conv2D(75, (3, 3), activation='relu',padding='same'))
24 model.add(Dropout(0.75))
25
26 model.add(Flatten())
27 model.add(Dense(128, activation='relu'))
28 model.add(Dropout(0.5))
29 model.add(Dense(num_classes, activation='softmax'))
30
31 model.compile(loss=keras.losses.categorical_crossentropy,
32               optimizer=keras.optimizers.Adadelta(),
33               metrics=['accuracy'])
34
35 history = model.fit(x_train, y_train,
36           batch_size=batch_size,
37           epochs=epochs,
38           verbose=1,
39           validation_data=(x_test, y_test))
40 score = model.evaluate(x_test, y_test, verbose=0)
41 print('Test loss:', score[0])
42 print('Test accuracy:', score[1])
43
```

```
WARNING:tensorflow:Large dropout rate: 0.75 (>0.5). In TensorFlow 2.x, dropout
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 6s 92us/step - loss: 1.9629 - a
Epoch 2/12
60000/60000 [==============================] - 5s 77us/step - loss: 0.8247 - a
Epoch 3/12
60000/60000 [==============================] - 5s 75us/step - loss: 0.4365 - a
Epoch 4/12
60000/60000 [==============================] - 5s 75us/step - loss: 0.3244 - a
Epoch 5/12
60000/60000 [==============================] - 5s 76us/step - loss: 0.2699 - a
Epoch 6/12
60000/60000 [==============================] - 4s 74us/step - loss: 0.2348 - a
Epoch 7/12
60000/60000 [==============================] - 4s 74us/step - loss: 0.2188 - a
Epoch 8/12
60000/60000 [==============================] - 5s 75us/step - loss: 0.1909 - a
Epoch 9/12
60000/60000 [==============================] - 5s 77us/step - loss: 0.1835 - a
Epoch 10/12
60000/60000 [==============================] - 5s 75us/step - loss: 0.1739 - a
Epoch 11/12
60000/60000 [==============================] - 5s 76us/step - loss: 0.1600 - a
Epoch 12/12
60000/60000 [==============================] - 5s 77us/step - loss: 0.1572 - a
Test loss: 0.07009304843556602
Test accuracy: 0.986
```

```
1 score = model.evaluate(x_test, y_test, verbose=0)
2 print('Test score:', score[0])
3 print('Test accuracy:', score[1])
4
5 fig,ax = plt.subplots(1,1)
```

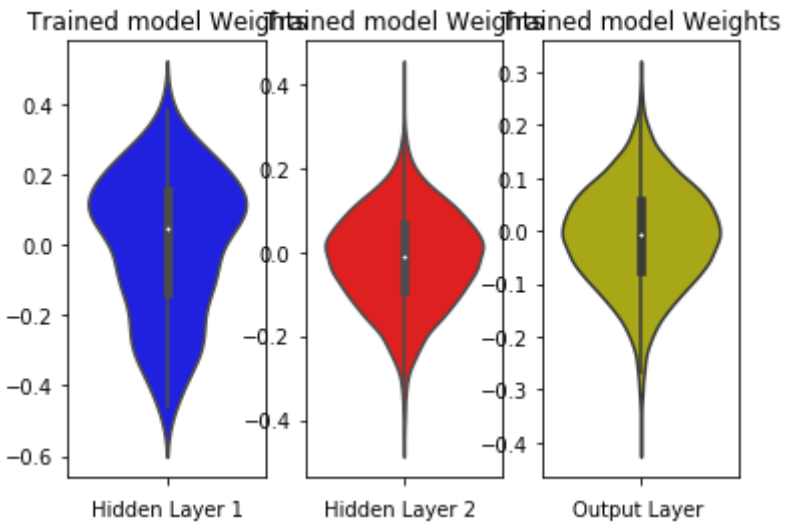
```
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,12+1))
# we will get val_loss and val_acc only when you pass the paramter validation_d
# 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 numb

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

```
Test score: 0.07009304843556602
Test accuracy: 0.986
```




```
w_after = model.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()
plt.title("Weight matrices after model trained")
plt.subplot(1, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

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

plt.subplot(1, 3, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
```

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



Model 2: 5 conv layers + filter size = 3x3 + relu activation + d

```
model = Sequential()

model.add(Conv2D(40, kernel_size=(3, 3),activation='relu',input_shape=input_sha
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(40, (3, 3), activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(50, (3, 3), activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(55, (3, 3), activation='relu',padding='same'))
#model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(55, (3, 3), activation='relu',padding='same'))
#model.add(MaxPooling2D(pool_size=(2, 2)))


model.add(Flatten())
model.add(Dense(128, activation='relu'))
```

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

model.summary()
```

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_37 (Conv2D)           (None, 26, 26, 40)        400       
_________________________________________________________________
max_pooling2d_27 (MaxPooling (None, 13, 13, 40)        0         
_________________________________________________________________
dropout_21 (Dropout)         (None, 13, 13, 40)        0         
_________________________________________________________________
conv2d_38 (Conv2D)           (None, 13, 13, 40)        14440     
_________________________________________________________________
max_pooling2d_28 (MaxPooling (None, 6, 6, 40)          0         
_________________________________________________________________
dropout_22 (Dropout)         (None, 6, 6, 40)          0         
_________________________________________________________________
conv2d_39 (Conv2D)           (None, 6, 6, 50)          18050     
_________________________________________________________________
max_pooling2d_29 (MaxPooling (None, 3, 3, 50)          0         
_________________________________________________________________
conv2d_40 (Conv2D)           (None, 3, 3, 55)          24805     
_________________________________________________________________
conv2d_41 (Conv2D)           (None, 3, 3, 55)          27280     
_________________________________________________________________
flatten_6 (Flatten)          (None, 495)               0         
_________________________________________________________________
dense_11 (Dense)             (None, 128)               63488     
_________________________________________________________________
dropout_23 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_12 (Dense)             (None, 10)                1290      
=================================================================
Total params: 149,753
Trainable params: 149,753
Non-trainable params: 0
_________________________________________________________________
```

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

history = model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1
```

```
          verbose=1,
          validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

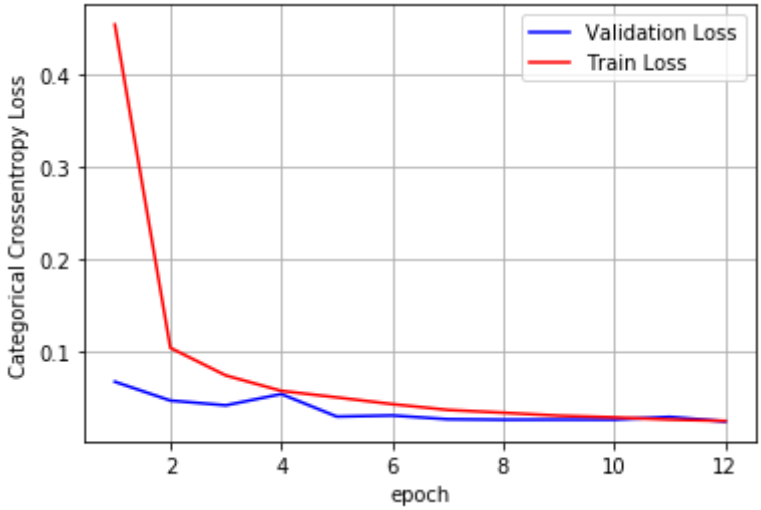
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 6s 104us/step - loss: 0.4544 -
Epoch 2/12
60000/60000 [==============================] - 5s 83us/step - loss: 0.1046 - a
Epoch 3/12
60000/60000 [==============================] - 5s 83us/step - loss: 0.0747 - a
Epoch 4/12
60000/60000 [==============================] - 5s 85us/step - loss: 0.0580 - a
Epoch 5/12
60000/60000 [==============================] - 5s 83us/step - loss: 0.0510 - a
Epoch 6/12
60000/60000 [==============================] - 5s 83us/step - loss: 0.0437 - a
Epoch 7/12
60000/60000 [==============================] - 5s 83us/step - loss: 0.0374 - a
Epoch 8/12
60000/60000 [==============================] - 5s 86us/step - loss: 0.0343 - a
Epoch 9/12
60000/60000 [==============================] - 5s 82us/step - loss: 0.0311 - a
Epoch 10/12
60000/60000 [==============================] - 5s 85us/step - loss: 0.0293 - a
Epoch 11/12
60000/60000 [==============================] - 5s 82us/step - loss: 0.0269 - a
Epoch 12/12
60000/60000 [==============================] - 5s 83us/step - loss: 0.0255 - a
Test loss: 0.024910171242001615
Test accuracy: 0.9927
```

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

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,12+1))
# we will get val_loss and val_acc only when you pass the paramter validation_d
# 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 numb
```

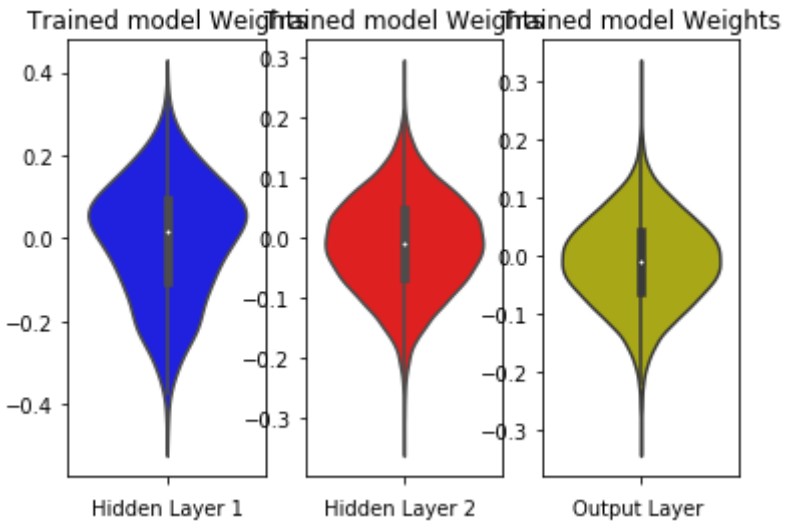
```
# for each key in history.history we will have a list of length equal to numb
```

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

```
Test score: 0.024910171242001615
Test accuracy: 0.9927
```




```
w_after = model.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()
plt.title("Weight matrices after model trained")
plt.subplot(1, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

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

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



Model 3: 3 Convolution Layers + filter size = 5x5 + dropout +

```
model = Sequential()

model.add(Conv2D(40, kernel_size=(5, 5),activation='relu',input_shape=input_sha
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(50, (5, 5), activation='relu',padding='same'))


model.add(Conv2D(55, (5, 5), activation='relu',padding='same'))

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

model.summary()
```

```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_42 (Conv2D)           (None, 24, 24, 40)        1040      
_________________________________________________________________
max_pooling2d_30 (MaxPooling (None, 12, 12, 40)        0         
_________________________________________________________________
dropout_24 (Dropout)         (None, 12, 12, 40)        0         
_________________________________________________________________
conv2d_43 (Conv2D)           (None, 12, 12, 50)        50050     
_________________________________________________________________
conv2d_44 (Conv2D)           (None, 12, 12, 55)        68805     
_________________________________________________________________
flatten_7 (Flatten)          (None, 7920)              0         
_________________________________________________________________
dense_13 (Dense)             (None, 128)               1013888   
_________________________________________________________________
dropout_25 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_14 (Dense)             (None, 10)                1290      
=================================================================
Total params: 1,135,073
Trainable params: 1,135,073
Non-trainable params: 0
_________________________________________________________________
```

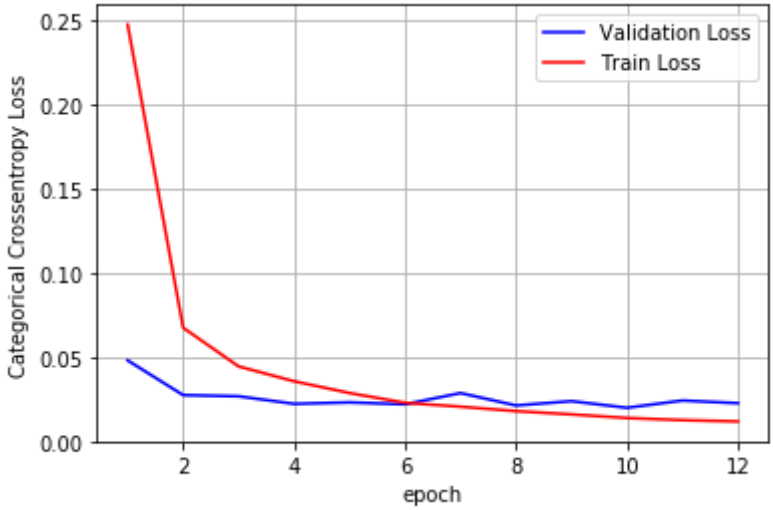
```
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

history = model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))
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 [==============================] - 7s 118us/step - loss: 0.2478 -
Epoch 2/12
60000/60000 [==============================] - 6s 99us/step - loss: 0.0675 - a
Epoch 3/12
60000/60000 [==============================] - 6s 101us/step - loss: 0.0445 -
Epoch 4/12
60000/60000 [==============================] - 6s 99us/step - loss: 0.0356 - a
Epoch 5/12
60000/60000 [==============================] - 6s 100us/step - loss: 0.0286 -
Epoch 6/12
60000/60000 [==============================] - 6s 101us/step - loss: 0.0228 -
Epoch 7/12
60000/60000 [==============================] - 6s 99us/step - loss: 0.0205 - a
Epoch 8/12
60000/60000 [==============================] - 6s 105us/step - loss: 0.0178 -
Epoch 9/12
60000/60000 [==============================] - 6s 103us/step - loss: 0.0159 -
Epoch 10/12
60000/60000 [==============================] - 6s 101us/step - loss: 0.0137 -
Epoch 11/12
60000/60000 [==============================] - 6s 99us/step - loss: 0.0125 - a
Epoch 12/12
60000/60000 [==============================] - 6s 99us/step - loss: 0.0117 - a
Test loss: 0.022570345474630403
Test accuracy: 0.9934
```

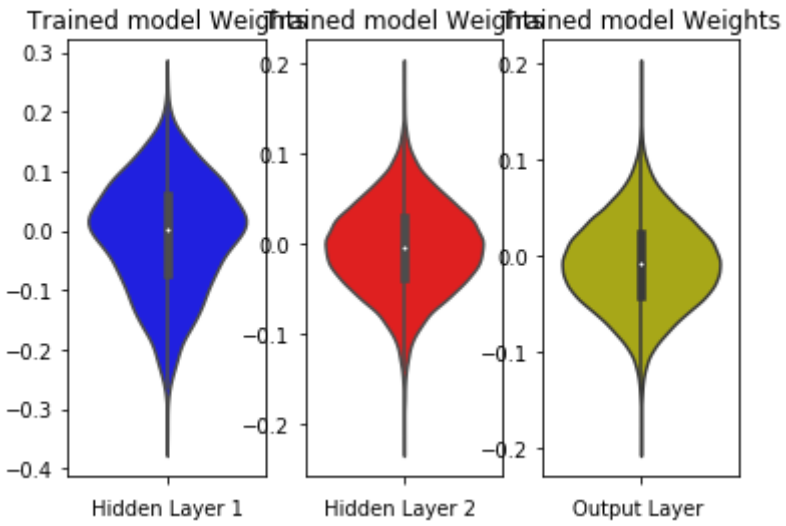
```
score = model.evaluate(x_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,12+1))
# we will get val_loss and val_acc only when you pass the paramter validation_d
# 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 numb
```

```
# for each key in history.history we will have a list of length equal to numb

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

```
Test score: 0.022570345474630403
Test accuracy: 0.9934
```




```
w_after = model.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()
plt.title("Weight matrices after model trained")
plt.subplot(1, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

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

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



Conclusion

```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Model","Conv Layers","Filter Size", "Max Pool", "Test Accurac
x.add_row(["1","7", "(3X3)" ,"(2x2)", "98.60"])
x.add_row(["2","5", "(3x3)" ,"(2x2)", "99.27"])
x.add_row(["3","3", "(5x5)" ,"(2x2)", "99.34"])
print(x)
```

```
+-------+-------------+-------------+----------+---------------+
| Model | Conv Layers | Filter Size | Max Pool | Test Accuracy |
+-------+-------------+-------------+----------+---------------+
|   1   |      7      |    (3X3)    |  (2x2)   |     98.60     |
|   2   |      5      |    (3x3)    |  (2x2)   |     99.27     |
|   3   |      3      |    (5x5)    |  (2x2)   |     99.34     |
+-------+-------------+-------------+----------+---------------+
```