```
# if you keras is not using tensorflow as backend set "KERAS_BACKEND=tensorflow" use this com
from keras.utils import np_utils
from keras.datasets import mnist
import seaborn as sns
from keras.initializers import RandomNormal
from keras.layers.normalization import BatchNormalization


import matplotlib.pyplot as plt
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()


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

import matplotlib.pyplot as plt
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 = 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])
```

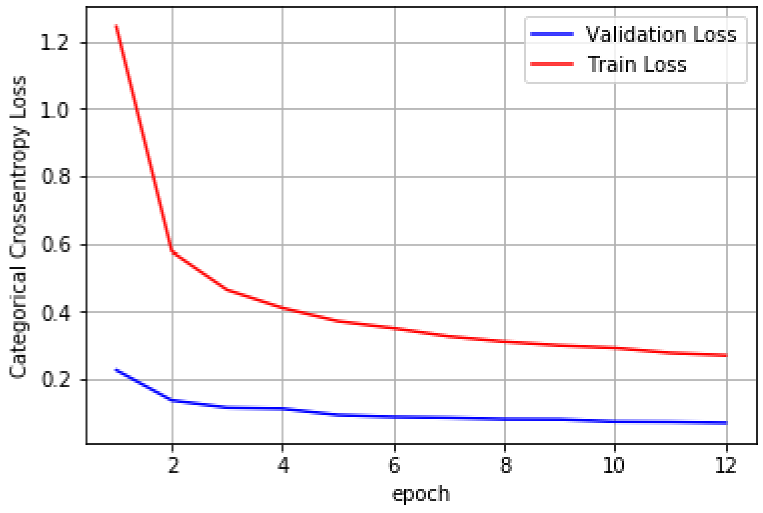
Assignment:

Model-1: 3 Conv-Layers, dropout, Max-pooling with 3*3 kernel:

```
model = Sequential()

model.add(Conv2D(32, kernel_size=(3, 3),activation='relu',input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(84, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.75))

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 = 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])
```

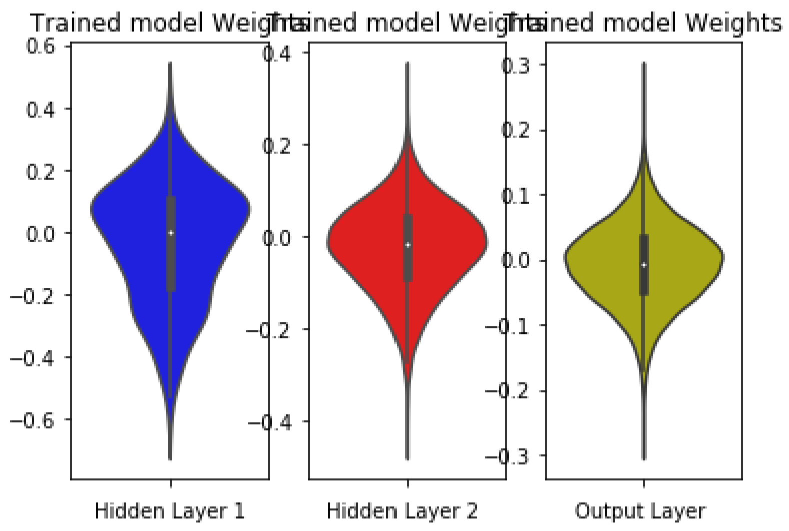
```
    WARNING:tensorflow:Large dropout rate: 0.75 (>0.5). In TensorFlow 2.x, dropout() uses dr
score = model.evaluate(x_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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))

# 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=

# 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, ax)
```

```
Test score: 0.07088609065115452
Test accuracy: 0.9791
```



```
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-2: 3 Conv-Layers, dropout, Max-pooling with 5*5 kernel:

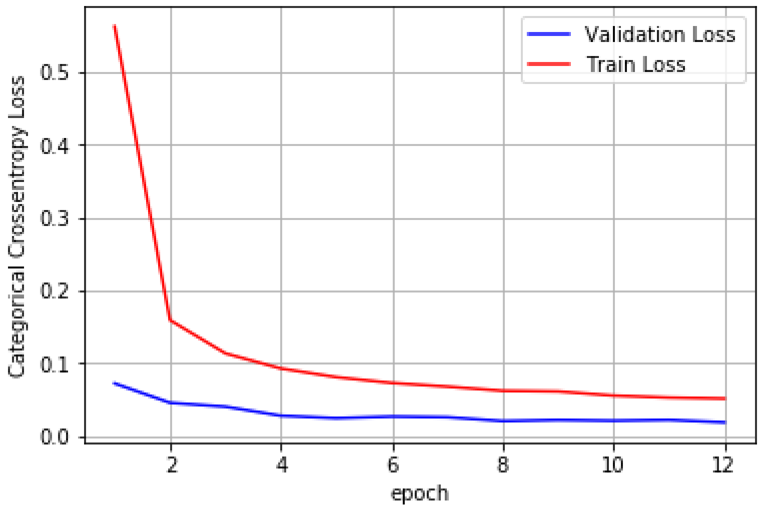
```
model = Sequential()

model.add(Conv2D(32, kernel_size=(5, 5),activation='relu',input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (5, 5), activation='relu'))
# model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(84, (5, 5), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.75))

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 = 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:Large dropout rate: 0.75 (>0.5). In TensorFlow 2.x, dropout() uses dr
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 11s 184us/step - loss: 0.5625 - acc: 0.81
Epoch 2/12
60000/60000 [==============================] - 9s 150us/step - loss: 0.1591 - acc: 0.954
Epoch 3/12
60000/60000 [==============================] - 9s 149us/step - loss: 0.1134 - acc: 0.967
Epoch 4/12
60000/60000 [==============================] - 9s 149us/step - loss: 0.0926 - acc: 0.974
Epoch 5/12
60000/60000 [==============================] - 9s 149us/step - loss: 0.0807 - acc: 0.977
Epoch 6/12
60000/60000 [==============================] - 9s 149us/step - loss: 0.0726 - acc: 0.980
Epoch 7/12
60000/60000 [==============================] - 9s 149us/step - loss: 0.0676 - acc: 0.981
Epoch 8/12
60000/60000 [==============================] - 9s 149us/step - loss: 0.0619 - acc: 0.983
Epoch 9/12
60000/60000 [==============================] - 9s 150us/step - loss: 0.0610 - acc: 0.983
Epoch 10/12
60000/60000 [==============================] - 9s 152us/step - loss: 0.0553 - acc: 0.985
Epoch 11/12
60000/60000 [==============================] - 9s 150us/step - loss: 0.0525 - acc: 0.986
Epoch 12/12
60000/60000 [==============================] - 9s 150us/step - loss: 0.0512 - acc: 0.986
Test loss: 0.018479915870346305
Test accuracy: 0.9949
```

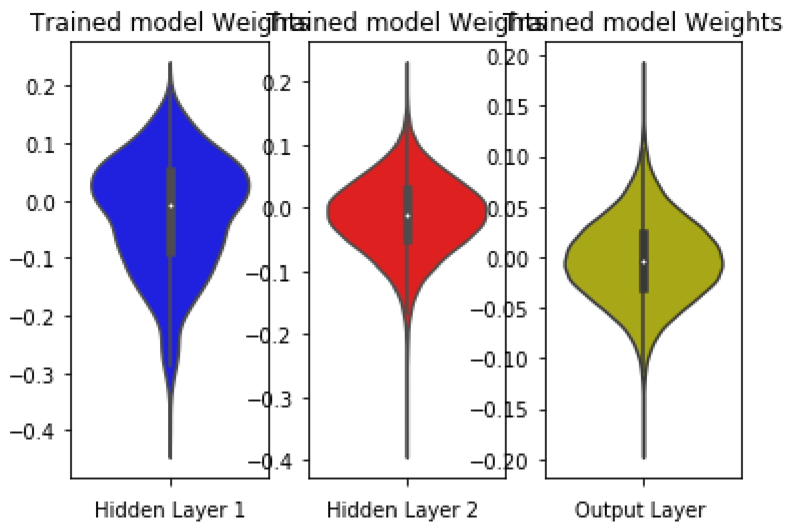
```
score = model.evaluate(x_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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))

# 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=

# 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, ax)
```

```
Test score: 0.018479915870346305
Test accuracy: 0.9949
```



```
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()
```



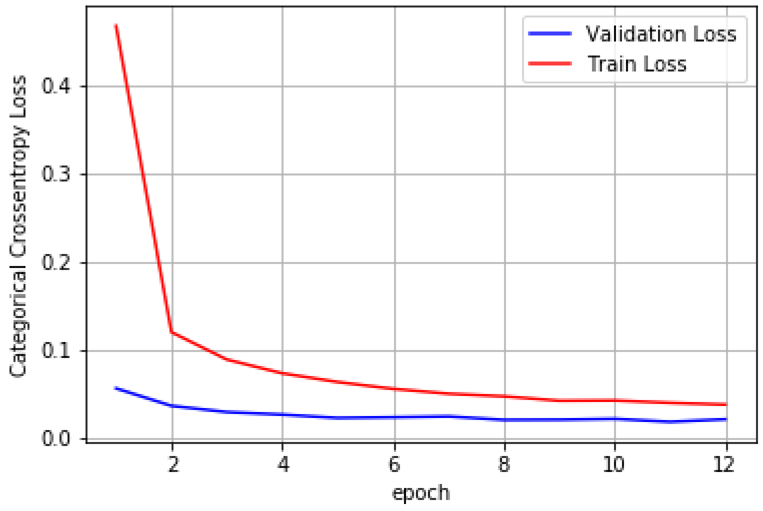
Model-3: 3 Conv-Layers, dropout, Max-pooling with 7*7 kernel:

```
model = Sequential()

model.add(Conv2D(32, kernel_size=(7, 7),activation='relu',input_shape=input_shape,padding='sa
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (7, 7), activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(84, (7, 7), activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.75))

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(),
              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])
```

```
WARNING:tensorflow:Large dropout rate: 0.75 (>0.5). In TensorFlow 2.x, dropout() uses dr
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 17s 291us/step - loss: 0.4669 - acc: 0.84
Epoch 2/12
60000/60000 [==============================] - 15s 254us/step - loss: 0.1194 - acc: 0.96
Epoch 3/12
60000/60000 [==============================] - 15s 254us/step - loss: 0.0883 - acc: 0.97
Epoch 4/12
60000/60000 [==============================] - 15s 255us/step - loss: 0.0727 - acc: 0.98
Epoch 5/12
60000/60000 [==============================] - 15s 257us/step - loss: 0.0628 - acc: 0.98
Epoch 6/12
60000/60000 [==============================] - 15s 258us/step - loss: 0.0550 - acc: 0.98
Epoch 7/12
60000/60000 [==============================] - 16s 259us/step - loss: 0.0494 - acc: 0.98
Epoch 8/12
60000/60000 [==============================] - 16s 260us/step - loss: 0.0464 - acc: 0.98
Epoch 9/12
60000/60000 [==============================] - 16s 264us/step - loss: 0.0415 - acc: 0.98
Epoch 10/12
60000/60000 [==============================] - 16s 261us/step - loss: 0.0418 - acc: 0.98
Epoch 11/12
60000/60000 [==============================] - 16s 263us/step - loss: 0.0391 - acc: 0.98
Epoch 12/12
60000/60000 [==============================] - 16s 263us/step - loss: 0.0371 - acc: 0.98
Test loss: 0.020393834800141484
Test accuracy: 0.9939
```

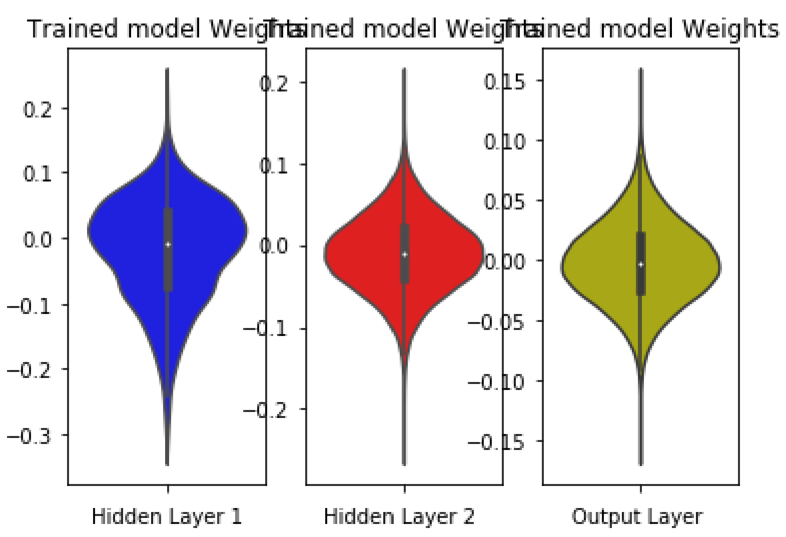
```
score = model.evaluate(x_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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))

# 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=
# 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, ax)
```

```
Test score: 0.020393834800141484
Test accuracy: 0.9939
```



```
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()
```



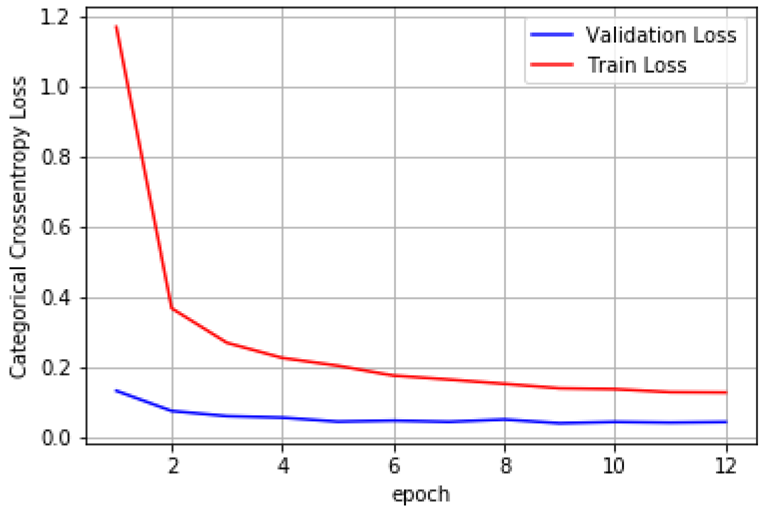
Model-4: 5 Conv-Layers, dropout, Max-pooling with 3*3 kernel:

```
model = Sequential()

model.add(Conv2D(32, kernel_size=(3, 3),activation='relu',input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(42, (3, 3), activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(52, (3, 3), activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
# model.add(Dropout(0.75))

model.add(Conv2D(62, (3, 3), activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
# model.add(Dropout(0.75))

model.add(Conv2D(72, (3, 3), activation='relu',padding='same'))
# model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.75))

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 = 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])
```

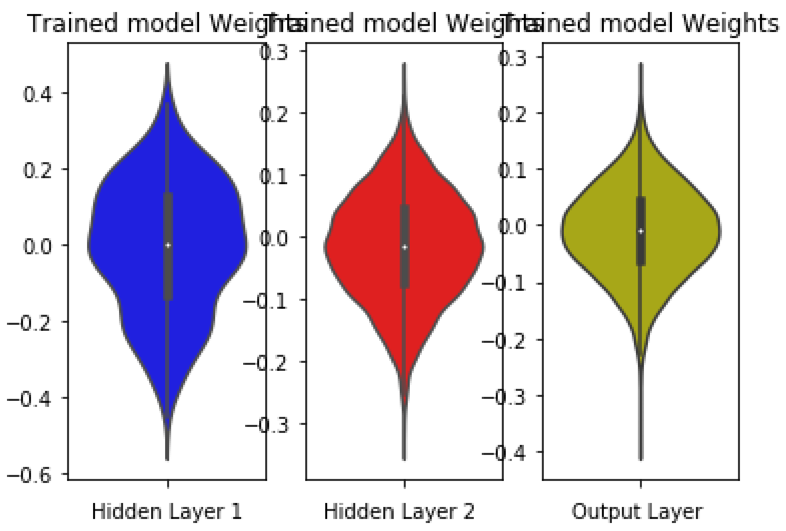
```
WARNING:tensorflow:Large dropout rate: 0.75 (>0.5). In TensorFlow 2.x, dropout() uses dr
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 12s 196us/step - loss: 1.1704 - acc: 0.58
Epoch 2/12
60000/60000 [==============================] - 9s 156us/step - loss: 0.3683 - acc: 0.883
Epoch 3/12
60000/60000 [==============================] - 9s 155us/step - loss: 0.2697 - acc: 0.917
Epoch 4/12
60000/60000 [==============================] - 9s 156us/step - loss: 0.2265 - acc: 0.930
Epoch 5/12
60000/60000 [==============================] - 9s 156us/step - loss: 0.2043 - acc: 0.939
Epoch 6/12
60000/60000 [==============================] - 9s 156us/step - loss: 0.1761 - acc: 0.947
Epoch 7/12
60000/60000 [==============================] - 9s 156us/step - loss: 0.1648 - acc: 0.951
Epoch 8/12
60000/60000 [==============================] - 9s 156us/step - loss: 0.1526 - acc: 0.954
Epoch 9/12
60000/60000 [==============================] - 9s 156us/step - loss: 0.1397 - acc: 0.958
Epoch 10/12
60000/60000 [==============================] - 9s 156us/step - loss: 0.1373 - acc: 0.959
Epoch 11/12
60000/60000 [==============================] - 9s 156us/step - loss: 0.1287 - acc: 0.961
Epoch 12/12
60000/60000 [==============================] - 9s 155us/step - loss: 0.1278 - acc: 0.962
Test loss: 0.04341736109344106
Test accuracy: 0.9919
```

```
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))

# 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=

# 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, ax)
```

```
Test score: 0.04341736109344106
Test accuracy: 0.9919
```



```
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-5: 5 Conv-Layers, dropout, Max-pooling with 5*5 kernel:

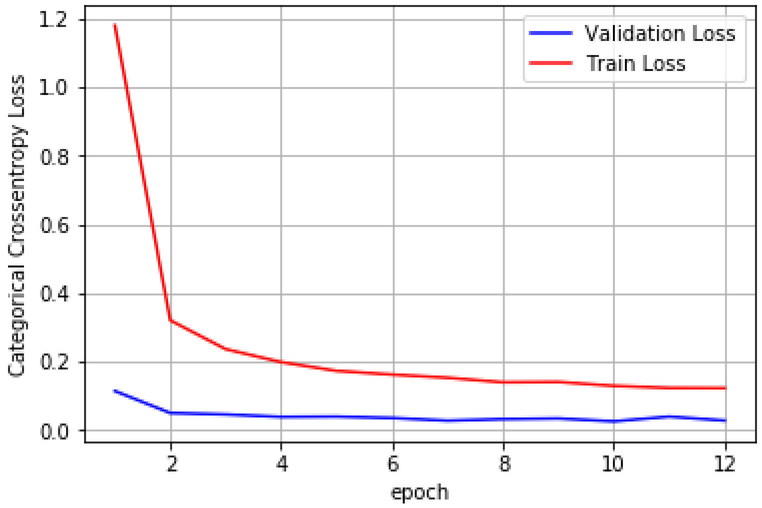
```
model = Sequential()

model.add(Conv2D(32, kernel_size=(5, 5),activation='relu',input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(42, kernel_size=(5, 5),activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(52, kernel_size=(5, 5),activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(62, (5, 5), activation='relu',padding='same'))
# model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(72, (5, 5), activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.75))

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 = 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:Large dropout rate: 0.75 (>0.5). In TensorFlow 2.x, dropout() uses dr
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 14s 241us/step - loss: 1.1804 - acc: 0.57
Epoch 2/12
60000/60000 [==============================] - 11s 190us/step - loss: 0.3211 - acc: 0.90
Epoch 3/12
60000/60000 [==============================] - 11s 187us/step - loss: 0.2371 - acc: 0.92
Epoch 4/12
60000/60000 [==============================] - 11s 190us/step - loss: 0.1991 - acc: 0.94
Epoch 5/12
60000/60000 [==============================] - 11s 188us/step - loss: 0.1736 - acc: 0.94
Epoch 6/12
60000/60000 [==============================] - 12s 193us/step - loss: 0.1627 - acc: 0.95
Epoch 7/12
60000/60000 [==============================] - 11s 190us/step - loss: 0.1534 - acc: 0.95
Epoch 8/12
60000/60000 [==============================] - 11s 191us/step - loss: 0.1401 - acc: 0.95
Epoch 9/12
60000/60000 [==============================] - 12s 193us/step - loss: 0.1411 - acc: 0.96
Epoch 10/12
60000/60000 [==============================] - 12s 195us/step - loss: 0.1298 - acc: 0.96
Epoch 11/12
60000/60000 [==============================] - 11s 189us/step - loss: 0.1237 - acc: 0.96
Epoch 12/12
60000/60000 [==============================] - 11s 190us/step - loss: 0.1231 - acc: 0.96
Test loss: 0.028561681090549428
Test accuracy: 0.9936
```

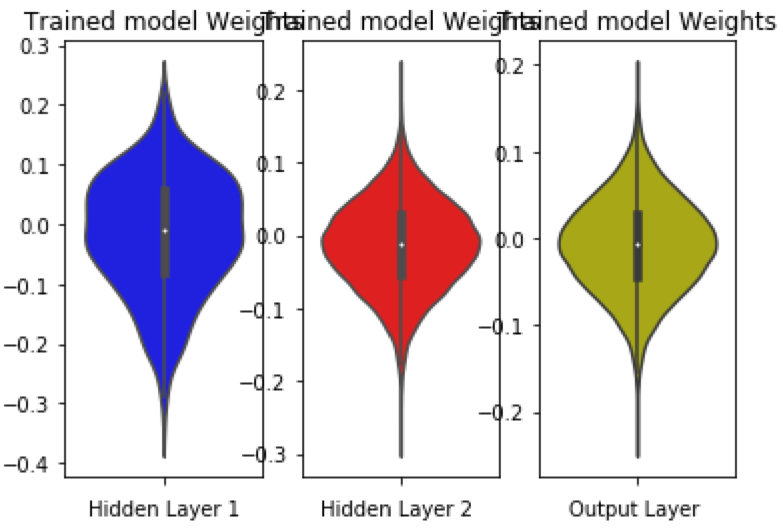
```
score = model.evaluate(x_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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))
```

```
# 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=

# 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, ax)
```

```
Test score: 0.028561681090549428
Test accuracy: 0.9936
```



```
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-6: 5 Conv-Layers, dropout, Max-pooling with 7*7 kernel:

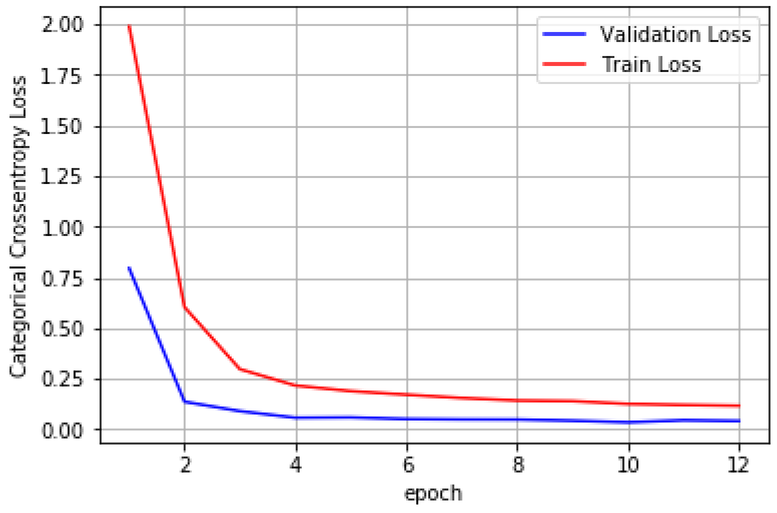
```
model = Sequential()

model.add(Conv2D(32, kernel_size=(7, 7),activation='relu',input_shape=input_shape,padding='sa
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(42, (7, 7), activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(52, (7, 7), activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(62, (7, 7), activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(72, (7, 7), activation='relu',padding='same'))
# model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.75))

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 = 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 [==============================] - 20s 333us/step - loss: 1.9886 - acc: 0.25
Epoch 2/12
60000/60000 [==============================] - 17s 286us/step - loss: 0.6041 - acc: 0.80
Epoch 3/12
60000/60000 [==============================] - 18s 293us/step - loss: 0.2974 - acc: 0.91
Epoch 4/12
60000/60000 [==============================] - 17s 291us/step - loss: 0.2165 - acc: 0.94
Epoch 5/12
60000/60000 [==============================] - 18s 293us/step - loss: 0.1889 - acc: 0.95
Epoch 6/12
60000/60000 [==============================] - 18s 296us/step - loss: 0.1712 - acc: 0.95
Epoch 7/12
60000/60000 [==============================] - 19s 312us/step - loss: 0.1548 - acc: 0.96
Epoch 8/12
60000/60000 [==============================] - 18s 299us/step - loss: 0.1423 - acc: 0.96
Epoch 9/12
60000/60000 [==============================] - 18s 299us/step - loss: 0.1393 - acc: 0.96
Epoch 10/12
60000/60000 [==============================] - 18s 299us/step - loss: 0.1256 - acc: 0.96
Epoch 11/12
60000/60000 [==============================] - 18s 299us/step - loss: 0.1205 - acc: 0.97
Epoch 12/12
60000/60000 [==============================] - 18s 298us/step - loss: 0.1166 - acc: 0.97
Test loss: 0.041424628414865584
Test accuracy: 0.9904
```

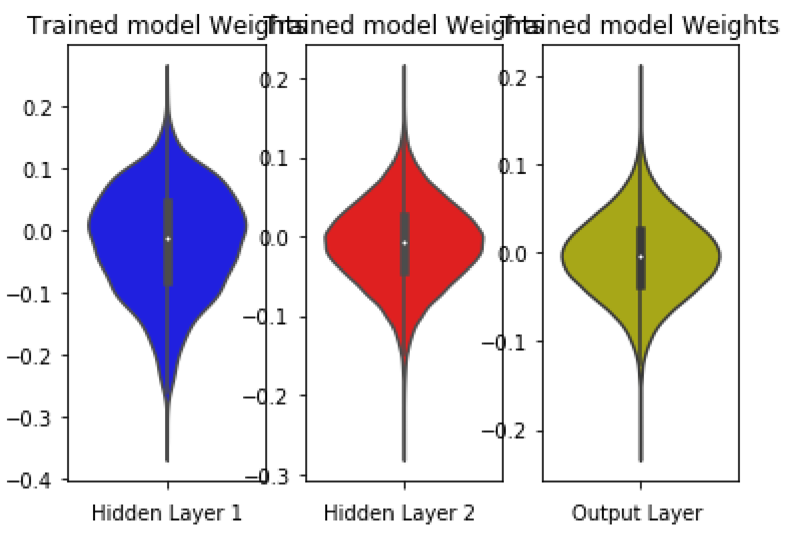
```
score = model.evaluate(x_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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))
```

```
# 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=

# 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, ax)
```

```
Test score: 0.041424628414865584
Test accuracy: 0.9904
```



```
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()
```



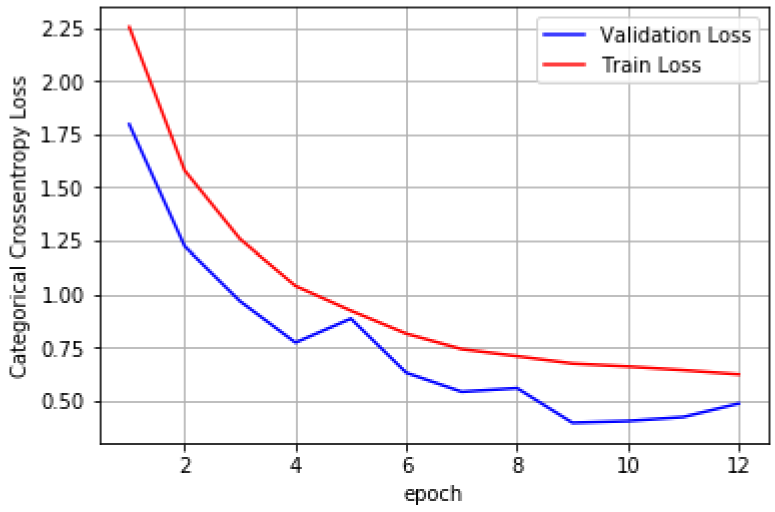
Model-7: 7 Conv-Layers, dropout, Max-pooling with 3*3 kernel:

```
model = Sequential()

model.add(Conv2D(32, kernel_size=(3, 3),activation='relu',input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(42, (3, 3), activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(52, (3, 3), activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
# model.add(Dropout(0.75))

model.add(Conv2D(62, (3, 3), activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
# model.add(Dropout(0.75))

model.add(Conv2D(72, (3, 3), activation='relu',padding='same'))
# model.add(MaxPooling2D(pool_size=(2, 2)))
# model.add(Dropout(0.75))

model.add(Conv2D(50, (3, 3), activation='relu',padding='same'))
model.add(Conv2D(50, (5, 5), activation='relu',padding='same'))
# model.add(MaxPooling2D(pool_size=(2, 2)))
# model.add(Dropout(0.75))

model.add(Conv2D(20, (3, 3), activation='relu',padding='same'))
# model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.75))

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 = 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])
```

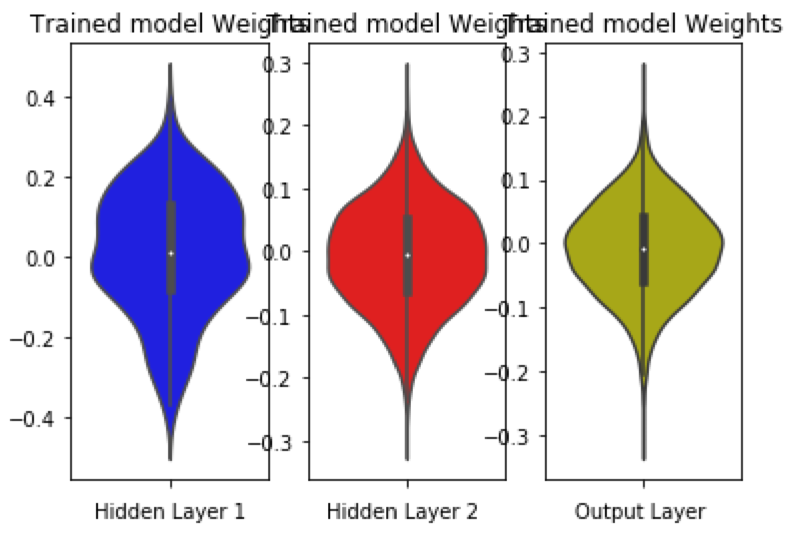
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 13s 209us/step - loss: 2.2540 - acc: 0.14
Epoch 2/12
60000/60000 [==============================] - 10s 164us/step - loss: 1.5798 - acc: 0.35
Epoch 3/12
60000/60000 [==============================] - 10s 164us/step - loss: 1.2581 - acc: 0.51
Epoch 4/12
60000/60000 [==============================] - 10s 164us/step - loss: 1.0393 - acc: 0.64
Epoch 5/12
60000/60000 [==============================] - 10s 164us/step - loss: 0.9233 - acc: 0.69
Epoch 6/12
60000/60000 [==============================] - 10s 165us/step - loss: 0.8153 - acc: 0.75
Epoch 7/12
60000/60000 [==============================] - 10s 165us/step - loss: 0.7425 - acc: 0.77
Epoch 8/12
60000/60000 [==============================] - 10s 164us/step - loss: 0.7093 - acc: 0.78
Epoch 9/12
60000/60000 [==============================] - 10s 165us/step - loss: 0.6748 - acc: 0.79
Epoch 10/12
60000/60000 [==============================] - 10s 167us/step - loss: 0.6608 - acc: 0.79
Epoch 11/12
60000/60000 [==============================] - 10s 169us/step - loss: 0.6436 - acc: 0.80
Epoch 12/12
60000/60000 [==============================] - 10s 167us/step - loss: 0.6238 - acc: 0.81
Test loss: 0.4869926846504211
Test accuracy: 0.8385
```

```
score = model.evaluate(x_test, y_test, verbose=0)
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))

# 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=

# 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, ax)
```

```
Test score: 0.4869926846504211
Test accuracy: 0.8385
```



```
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-8: 7 Conv-Layers, dropout, Max-pooling with 5*5 kernel:

```
input_shape
```

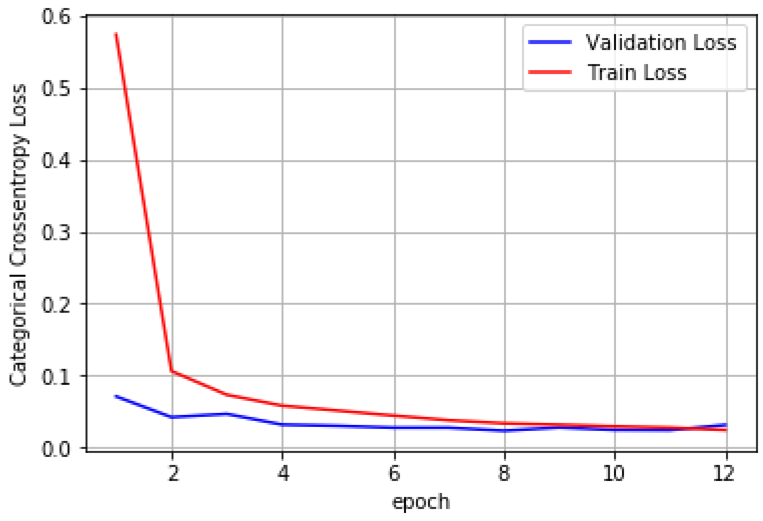
```
(28, 28, 1)
```

```
model = Sequential()

model.add(Conv2D(32, kernel_size=(5, 5),activation='relu',input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(42, kernel_size=(5, 5),activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.20))
model.add(Conv2D(54, kernel_size=(5, 5),activation='relu',padding='same'))
# model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(30, kernel_size=(5, 5),activation='relu',padding='same'))
# model.add(MaxPooling2D(pool_size=(2, 2)))
# model.add(Dropout(0.32))

model.add(Conv2D(22, kernel_size=(5, 5),activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
# model.add(Dropout(0.25))

model.add(Conv2D(10, (5, 5), activation='relu',padding='same'))
# model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.13))

model.add(Conv2D(20, (5, 5), activation='relu',padding='same'))
# model.add(MaxPooling2D(pool_size=(2, 2)))
# model.add(Dropout(0.20))

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 = 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])
```

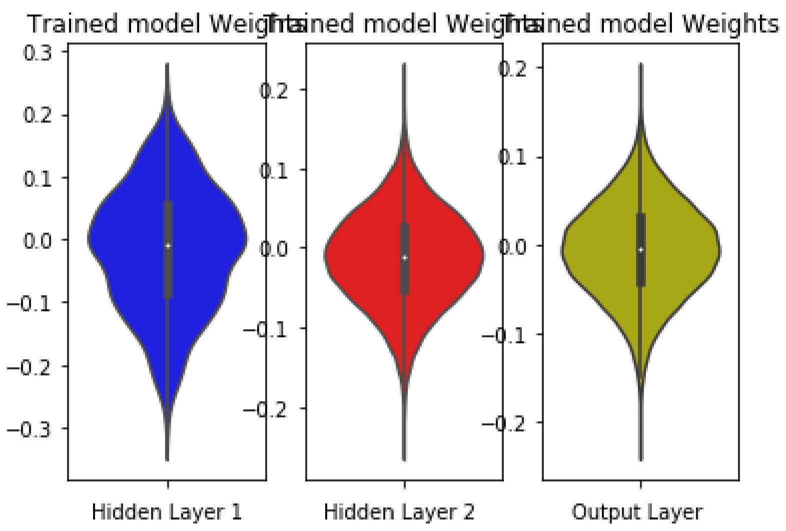
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 14s 232us/step - loss: 0.5739 - acc: 0.81
Epoch 2/12
60000/60000 [==============================] - 11s 183us/step - loss: 0.1063 - acc: 0.97
Epoch 3/12
60000/60000 [==============================] - 11s 183us/step - loss: 0.0733 - acc: 0.98
Epoch 4/12
60000/60000 [==============================] - 11s 183us/step - loss: 0.0581 - acc: 0.98
Epoch 5/12
60000/60000 [==============================] - 11s 184us/step - loss: 0.0513 - acc: 0.98
Epoch 6/12
60000/60000 [==============================] - 11s 184us/step - loss: 0.0443 - acc: 0.98
Epoch 7/12
60000/60000 [==============================] - 11s 184us/step - loss: 0.0378 - acc: 0.98
Epoch 8/12
60000/60000 [==============================] - 11s 185us/step - loss: 0.0335 - acc: 0.99
Epoch 9/12
60000/60000 [==============================] - 11s 185us/step - loss: 0.0316 - acc: 0.99
Epoch 10/12
60000/60000 [==============================] - 11s 185us/step - loss: 0.0293 - acc: 0.99
Epoch 11/12
60000/60000 [==============================] - 11s 185us/step - loss: 0.0276 - acc: 0.99
Epoch 12/12
60000/60000 [==============================] - 11s 186us/step - loss: 0.0243 - acc: 0.99
Test loss: 0.031318415241015876
Test accuracy: 0.9923
```

```
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))

# 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=

# 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, ax)
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.031318415241015876
Test accuracy: 0.9923
```



```
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-9: 7 Conv-Layers, dropout, Max-pooling with 7*7 kernel:

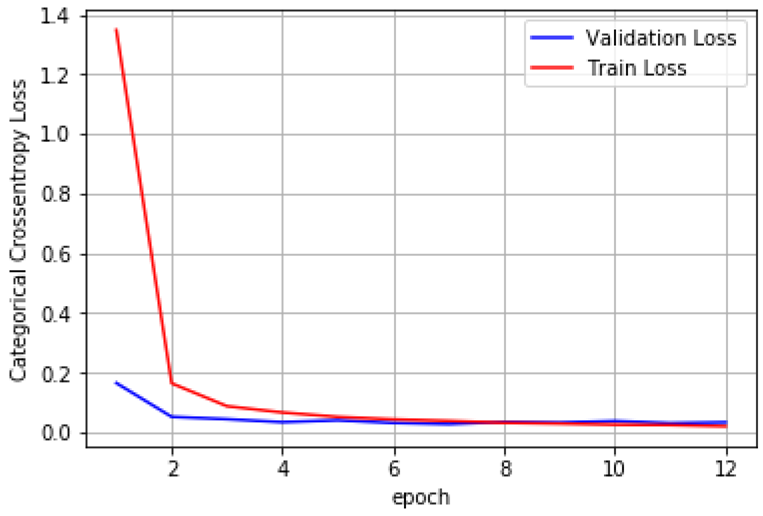
```
model = Sequential()

model.add(Conv2D(32, kernel_size=(7, 7),activation='relu',input_shape=input_shape,padding='sa
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(42, (7, 7), activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
# model.add(Dropout(0.25))

model.add(Conv2D(32, (7, 7), activation='relu',padding='same'))
# model.add(MaxPooling2D(pool_size=(2, 2)))
# model.add(Dropout(0.25))

model.add(Conv2D(70, (7, 7), activation='relu',padding='same'))
# model.add(MaxPooling2D(pool_size=(2, 2)))
# model.add(Dropout(0.25))

model.add(Conv2D(23, (7, 7), activation='relu',padding='same'))
# model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.20))

model.add(Conv2D(11, (7, 7), activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.5))

model.add(Conv2D(11, (7, 7), activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.001))
model.add(Dropout(0.001))

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 = 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 [==============================] - 19s 315us/step - loss: 1.3484 - acc: 0.51
Epoch 2/12
60000/60000 [==============================] - 17s 287us/step - loss: 0.1644 - acc: 0.95
Epoch 3/12
60000/60000 [==============================] - 17s 289us/step - loss: 0.0876 - acc: 0.97
Epoch 4/12
60000/60000 [==============================] - 17s 291us/step - loss: 0.0665 - acc: 0.98
Epoch 5/12
60000/60000 [==============================] - 18s 293us/step - loss: 0.0517 - acc: 0.98
Epoch 6/12
60000/60000 [==============================] - 18s 293us/step - loss: 0.0433 - acc: 0.99
Epoch 7/12
60000/60000 [==============================] - 18s 295us/step - loss: 0.0381 - acc: 0.99
Epoch 8/12
60000/60000 [==============================] - 18s 295us/step - loss: 0.0318 - acc: 0.99
Epoch 9/12
60000/60000 [==============================] - 18s 295us/step - loss: 0.0292 - acc: 0.99
Epoch 10/12
60000/60000 [==============================] - 18s 295us/step - loss: 0.0257 - acc: 0.99
Epoch 11/12
60000/60000 [==============================] - 18s 299us/step - loss: 0.0243 - acc: 0.99
Epoch 12/12
60000/60000 [==============================] - 18s 299us/step - loss: 0.0203 - acc: 0.99
Test loss: 0.03348360838291346
Test accuracy: 0.9936
```

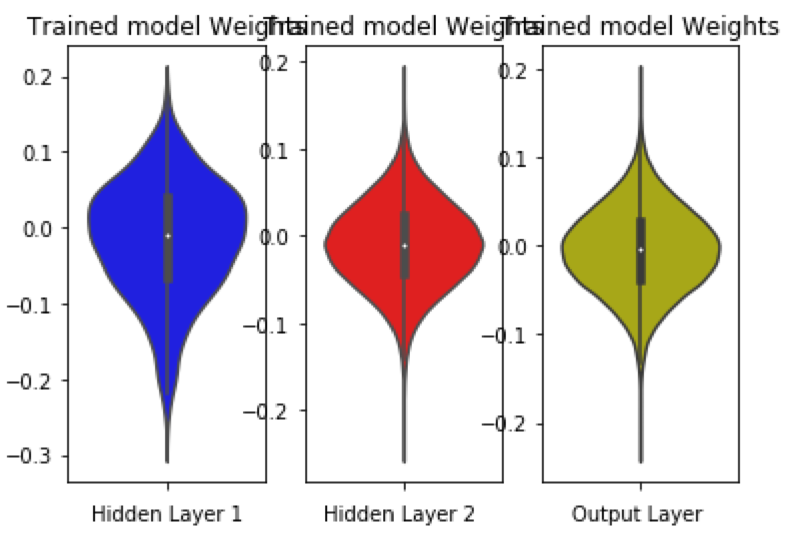
```
score = model.evaluate(x_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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))

# 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=

# 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, ax)
```

```
Test score: 0.03348360838291346
Test accuracy: 0.9936
```



```
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()
```



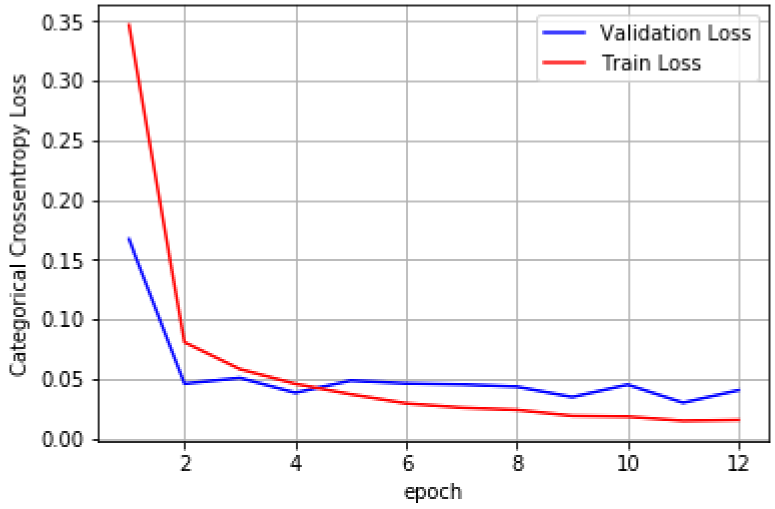
Model-10: 7 Conv-Layers, dropout, Max-pooling with 7*7 kernel with Ba

```
model = Sequential()

model.add(Conv2D(32, kernel_size=(7, 7),activation='relu',input_shape=input_shape,padding='sa
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(42, (7, 7), activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
# model.add(Dropout(0.25))

model.add(Conv2D(32, (7, 7), activation='relu',padding='same'))
model.add(BatchNormalization())
# model.add(MaxPooling2D(pool_size=(2, 2)))
# model.add(Dropout(0.25))

model.add(Conv2D(70, (7, 7), activation='relu',padding='same'))
model.add(BatchNormalization())
# model.add(MaxPooling2D(pool_size=(2, 2)))
# model.add(Dropout(0.25))
# model.add(Dropout(0.25))

model.add(Conv2D(23, (7, 7), activation='relu',padding='same'))
model.add(BatchNormalization())
# model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.20))

model.add(Conv2D(11, (7, 7), activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.5))

model.add(Conv2D(11, (7, 7), 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.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])
```

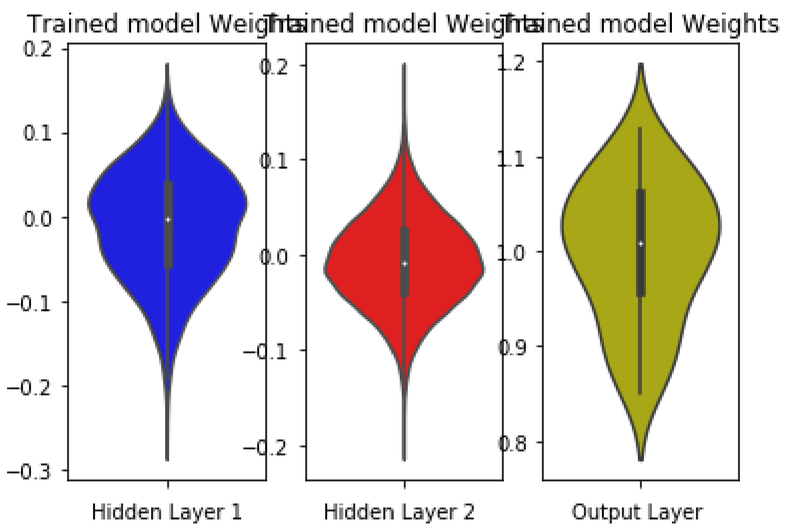
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 23s 388us/step - loss: 0.3466 - acc: 0.89
Epoch 2/12
60000/60000 [==============================] - 22s 360us/step - loss: 0.0805 - acc: 0.97
Epoch 3/12
60000/60000 [==============================] - 22s 363us/step - loss: 0.0579 - acc: 0.98
Epoch 4/12
60000/60000 [==============================] - 22s 365us/step - loss: 0.0456 - acc: 0.98
Epoch 5/12
60000/60000 [==============================] - 22s 361us/step - loss: 0.0368 - acc: 0.99
Epoch 6/12
60000/60000 [==============================] - 22s 364us/step - loss: 0.0291 - acc: 0.99
Epoch 7/12
60000/60000 [==============================] - 24s 395us/step - loss: 0.0255 - acc: 0.99
Epoch 8/12
60000/60000 [==============================] - 23s 377us/step - loss: 0.0236 - acc: 0.99
Epoch 9/12
60000/60000 [==============================] - 23s 378us/step - loss: 0.0186 - acc: 0.99
Epoch 10/12
60000/60000 [==============================] - 23s 389us/step - loss: 0.0180 - acc: 0.99
Epoch 11/12
60000/60000 [==============================] - 23s 388us/step - loss: 0.0144 - acc: 0.99
Epoch 12/12
60000/60000 [==============================] - 23s 391us/step - loss: 0.0152 - acc: 0.99
Test loss: 0.04018195565084179
Test accuracy: 0.9914
```

```
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))

# 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=

# 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, ax)
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.04018195565084179
Test accuracy: 0.9914
```



```
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-11: 7 Conv-Layers, dropout, Max-pooling with 7*7 kernel with Ba

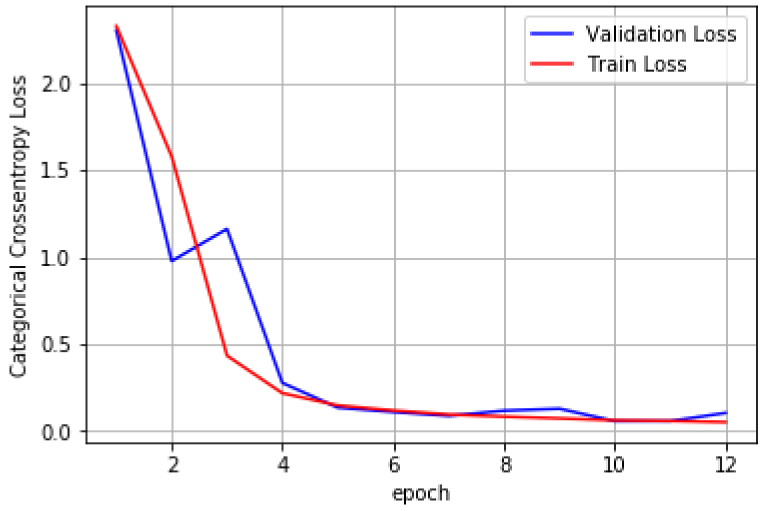
```
model = Sequential()

model.add(Conv2D(32, kernel_size=(7, 7),activation='sigmoid',input_shape=input_shape,padding=
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(42, (7, 7), activation='sigmoid',padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
# model.add(Dropout(0.25))

model.add(Conv2D(32, (7, 7), activation='sigmoid',padding='same'))
model.add(BatchNormalization())
# model.add(MaxPooling2D(pool_size=(2, 2)))
# model.add(Dropout(0.25))

model.add(Conv2D(70, (7, 7), activation='sigmoid',padding='same'))
model.add(BatchNormalization())
# model.add(MaxPooling2D(pool_size=(2, 2)))
# model.add(Dropout(0.25))

model.add(Conv2D(23, (7, 7), activation='sigmoid',padding='same'))
model.add(BatchNormalization())
# model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.20))

model.add(Conv2D(11, (7, 7), activation='sigmoid',padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.5))

model.add(Conv2D(11, (7, 7), activation='sigmoid',padding='same'))

model.add(Flatten())
model.add(Dense(128, activation='sigmoid'))
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 = 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 [==============================] - 22s 369us/step - loss: 2.3255 - acc: 0.10
Epoch 2/12
60000/60000 [==============================] - 20s 328us/step - loss: 1.5777 - acc: 0.40
Epoch 3/12
60000/60000 [==============================] - 20s 331us/step - loss: 0.4337 - acc: 0.89
Epoch 4/12
60000/60000 [==============================] - 20s 335us/step - loss: 0.2184 - acc: 0.95
Epoch 5/12
60000/60000 [==============================] - 20s 335us/step - loss: 0.1483 - acc: 0.96
Epoch 6/12
60000/60000 [==============================] - 20s 340us/step - loss: 0.1189 - acc: 0.97
Epoch 7/12
60000/60000 [==============================] - 20s 340us/step - loss: 0.0961 - acc: 0.97
Epoch 8/12
60000/60000 [==============================] - 20s 341us/step - loss: 0.0837 - acc: 0.98
Epoch 9/12
60000/60000 [==============================] - 21s 346us/step - loss: 0.0737 - acc: 0.98
Epoch 10/12
60000/60000 [==============================] - 21s 344us/step - loss: 0.0627 - acc: 0.98
Epoch 11/12
60000/60000 [==============================] - 21s 345us/step - loss: 0.0591 - acc: 0.98
Epoch 12/12
60000/60000 [==============================] - 21s 344us/step - loss: 0.0523 - acc: 0.98
Test loss: 0.10474583289409056
Test accuracy: 0.9764
```

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

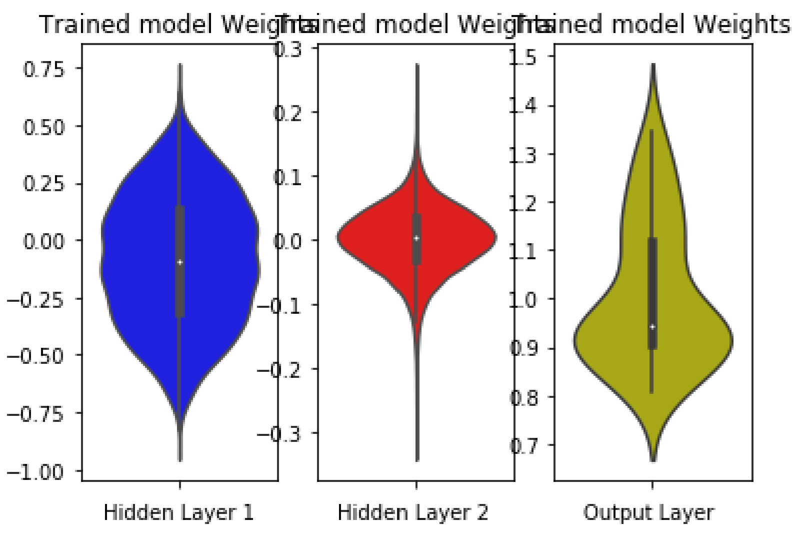
```
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,12+1))

# 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=

# 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, ax)
```

```
Test score: 0.10474583289409056
Test accuracy: 0.9764
```



```
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-12: 4 Conv-Layers, dropout, Max-pooling with 4*4 kernel with Ba

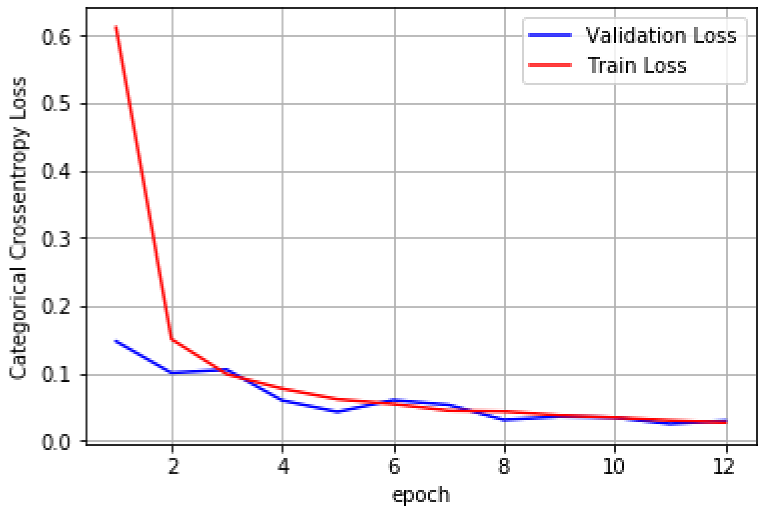
```
model = Sequential()

model.add(Conv2D(32, kernel_size=(4, 4),activation='sigmoid',input_shape=input_shape,padding=
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(42, (4, 4), activation='sigmoid',padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
# model.add(Dropout(0.25))

model.add(Conv2D(32, (4, 4), activation='sigmoid',padding='same'))
model.add(BatchNormalization())
# model.add(MaxPooling2D(pool_size=(2, 2)))
# model.add(Dropout(0.25))

model.add(Conv2D(70, (4, 4), activation='sigmoid',padding='same'))
model.add(BatchNormalization())
# model.add(MaxPooling2D(pool_size=(2, 2)))
# model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(128, activation='sigmoid'))
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 = 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 [==============================] - 17s 277us/step - loss: 0.6117 - acc: 0.80
Epoch 2/12
60000/60000 [==============================] - 14s 232us/step - loss: 0.1505 - acc: 0.95
Epoch 3/12
60000/60000 [==============================] - 14s 232us/step - loss: 0.0984 - acc: 0.97
Epoch 4/12
60000/60000 [==============================] - 14s 231us/step - loss: 0.0770 - acc: 0.97
Epoch 5/12
60000/60000 [==============================] - 14s 231us/step - loss: 0.0612 - acc: 0.98
Epoch 6/12
60000/60000 [==============================] - 14s 232us/step - loss: 0.0541 - acc: 0.98
Epoch 7/12
60000/60000 [==============================] - 14s 232us/step - loss: 0.0443 - acc: 0.98
Epoch 8/12
60000/60000 [==============================] - 14s 234us/step - loss: 0.0431 - acc: 0.98
Epoch 9/12
60000/60000 [==============================] - 14s 235us/step - loss: 0.0373 - acc: 0.98
Epoch 10/12
60000/60000 [==============================] - 14s 236us/step - loss: 0.0341 - acc: 0.98
Epoch 11/12
60000/60000 [==============================] - 14s 236us/step - loss: 0.0297 - acc: 0.99
Epoch 12/12
60000/60000 [==============================] - 14s 238us/step - loss: 0.0266 - acc: 0.99
Test loss: 0.02927502671419061
Test accuracy: 0.9898
```

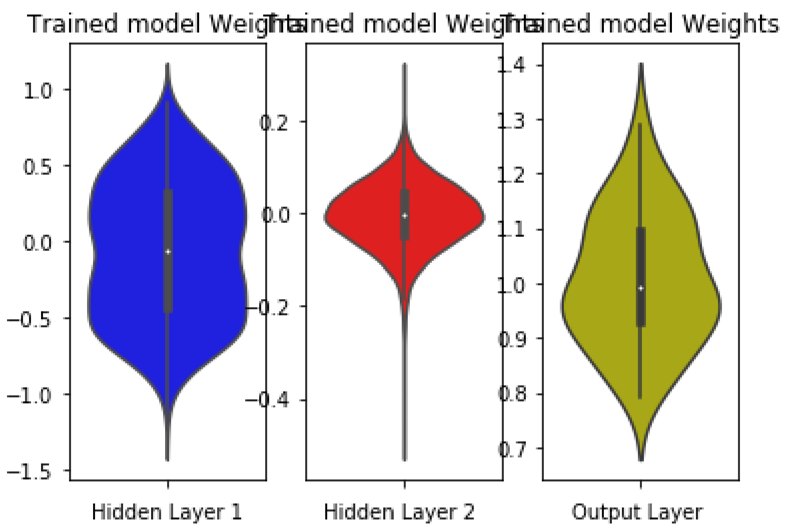
```
score = model.evaluate(x_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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))

# 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=

# 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, ax)
```

```
Test score: 0.02927502671419061
Test accuracy: 0.9898
```



```
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()
```



Results(Pretty Table):

```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Model","Layers","Kernels", "Test loss", "Test Accuracy"]
x.add_row(["1","3", "(3*3)" ,"0.070", "0.978"])
x.add_row(["2","3", "(5*5)" ,"0.021", "0.994"])
x.add_row(["3","3", "(7*7)" ,"0.020", "0.994"])
x.add_row(["4","5", "(3*3)" ,"0.038", "0.990"])
x.add_row(["5","5", "(5*5)" ,"0.028", "0.993"])
x.add_row(["6","5", "(7*7)" ,"0.041", "0.992"])
x.add_row(["7","7", "(3*3)" ,"0.304", "0.889"])
x.add_row(["8","7", "(5*5)" ,"0.024", "0.993"])
x.add_row(["9","7", "(7*7)" ,"0.043", "0.992"])
x.add_row(["10","7", "(7*7)" ,"0.040", "0.991"])
x.add_row(["11","7", "(7*7)" ,"0.104", "0.976"])
x.add_row(["12","4", "(4*4)" ,"0.029", "0.989"])
print(x)
```

```
+-------+--------+---------+-----------+---------------+
| Model | Layers | Kernels | Test loss | Test Accuracy |
+-------+--------+---------+-----------+---------------+
|   1   |   3    |  (3*3)  |   0.070   |     0.978     |
|   2   |   3    |  (5*5)  |   0.021   |     0.994     |
|   3   |   3    |  (7*7)  |   0.020   |     0.994     |
|   4   |   5    |  (3*3)  |   0.038   |     0.990     |
|   5   |   5    |  (5*5)  |   0.028   |     0.993     |
|   6   |   5    |  (7*7)  |   0.041   |     0.992     |
|   7   |   7    |  (3*3)  |   0.304   |     0.889     |
|   8   |   7    |  (5*5)  |   0.024   |     0.993     |
|   9   |   7    |  (7*7)  |   0.043   |     0.992     |
|   10  |   7    |  (7*7)  |   0.040   |     0.991     |
|   11  |   7    |  (7*7)  |   0.104   |     0.976     |
|   12  |   4    |  (4*4)  |   0.029   |     0.989     |
+-------+--------+---------+-----------+---------------+
```

Conclusion :

1. As you can see from the above table , i ran the first 3 of 3 layers with different kernels and got a max accuracy
2. For the next three models i have given 5 layers of Convolution with again various kernels and got max accura
3. For the next three models i have given 7 layers of Convolution with again various kernels and got max accura
4. Now for the 10th model i used Batch normalization , but didn't improved much
5. For 11th model i've used batch normalization along with sigmoid activations units in every hidden layers and than other models
6. And for the final model i've given 4 layers of Convolutions with Sigmoid activations and along with that i also 0.989