Convolution Neural Network in MNIST Data

Importing packages

In [1]:

```
import tensorflow as tf
import keras
from keras.utils import np_utils
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Input, InputLayer, Dense, Dropout, ZeroPadding2D, Flatten, Activat
from keras.layers import Conv2D, MaxPooling2D
from keras.initializers import he_normal
from keras.layers.normalization import BatchNormalization
from keras.optimizers import Adam, SGD
from keras import backend as K
from keras.preprocessing.image import ImageDataGenerator
import numpy as np
import seaborn as sns
```

```
Using TensorFlow backend.
```

Function for plotting dynamic graph

In [2]:

```
%matplotlib notebook
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()
```

Loading mnist data

In [3]:

```
#https://keras.io/examples/mnist_cnn/

img_rows, img_cols = 28, 28

(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)
```

In [4]:

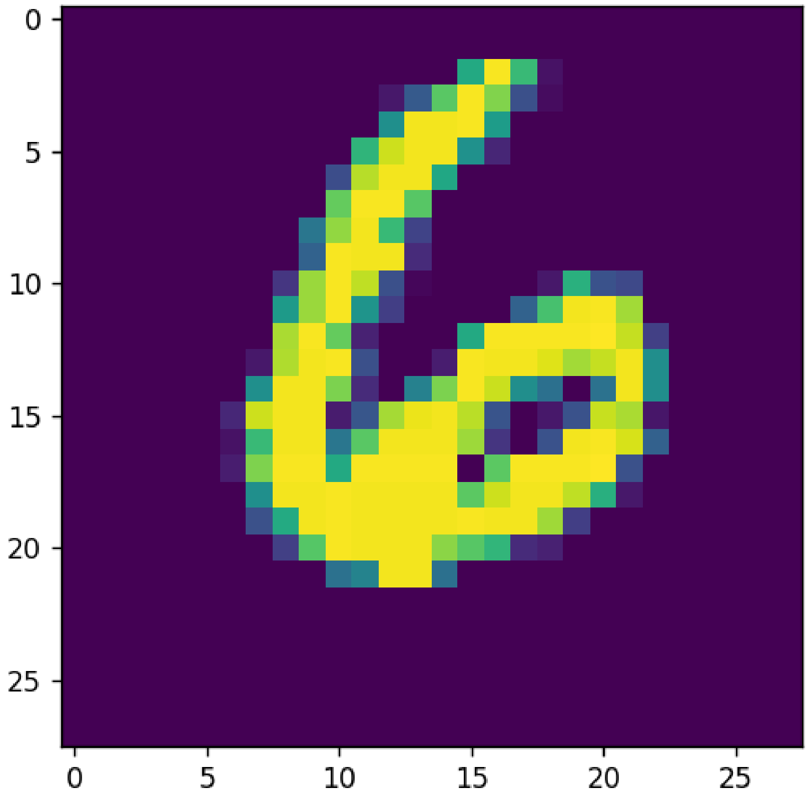
```
print("Number of training examples :", X_train.shape[0], "and each image is of shape (%d, %
print("Number of training examples :", X_test.shape[0], "and each image is of shape (%d, %c
```

```
Number of training examples : 60000 and each image is of shape (28, 28)
Number of training examples : 10000 and each image is of shape (28, 28)
```

In [5]:

```
plt.imshow(X_train[299].reshape(28,28))
```

Figure 1



Out[5]:

```
<matplotlib.image.AxesImage at 0x1fef6b0e1d0>
```

Normalizing pixel values

In [6]:

```
# Data Normalization
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')

X_train /= 255
X_test /= 255

old_v = tf.logging.get_verbosity
tf.logging.set_verbosity(tf.logging.ERROR)
```

After normalization (pixel value ranges from 0-1)

In [7]:

```
#Softmax output dim
num_classes = 10
batch_size = 140
epochs = 20
```

Model 1

In [8]:

```
# Converting y_train 10-D vector
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

# Model

model = Sequential()

model.add(Conv2D(filters=16, kernel_size=(3,3), strides =(1,1), padding = 'same',
                 activation = 'relu', input_shape = input_shape))

model.add(Conv2D(filters=32, kernel_size=(3,3), strides =(1,1), padding = 'same',
                 activation = 'relu'))

model.add(MaxPooling2D(pool_size=(2,2), strides=2))

model.add(BatchNormalization())

model.add(Dropout(0.25))

model.add(Conv2D(filters=64, kernel_size=(3,3), strides =(1,1), padding = 'same',
                 activation = 'relu'))

model.add(Conv2D(filters=128, kernel_size=(3,3), strides =(1,1), padding = 'same',
                 activation = 'relu'))

model.add(MaxPooling2D(pool_size=(2,2), strides=2))

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(units = 512, activation='relu'))


model.add(Dense(units = num_classes, activation='softmax'))

model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy']

print(model.summary(), '\n')

history = model.fit(X_train, y_train, batch_size = batch_size,
                    epochs = epochs, verbose = 1, validation_data = (X_test, y_test))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 16)        160       
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 28, 28, 32)        4640      
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 32)        0         
_________________________________________________________________
batch_normalization_1 (Batch (None, 14, 14, 32)        128       
_________________________________________________________________
dropout_1 (Dropout)          (None, 14, 14, 32)        0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 14, 14, 64)        18496     
```

```
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 14, 14, 128)       73856     
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 128)         0         
_________________________________________________________________
dropout_2 (Dropout)          (None, 7, 7, 128)         0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 6272)              0         
_________________________________________________________________
dense_1 (Dense)              (None, 512)               3211776   
_________________________________________________________________
dense_2 (Dense)              (None, 10)                5130      
=================================================================
Total params: 3,314,186
Trainable params: 3,314,122
Non-trainable params: 64
_________________________________________________________________
None

Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 14s 228us/step - loss: 0.2234
- acc: 0.9373 - val_loss: 0.0530 - val_acc: 0.9832
Epoch 2/20
60000/60000 [==============================] - 12s 192us/step - loss: 0.0489
- acc: 0.9842 - val_loss: 0.0306 - val_acc: 0.9901
Epoch 3/20
60000/60000 [==============================] - 12s 193us/step - loss: 0.0341
- acc: 0.9894 - val_loss: 0.0269 - val_acc: 0.9915
Epoch 4/20
60000/60000 [==============================] - 12s 193us/step - loss: 0.0296
- acc: 0.9902 - val_loss: 0.0263 - val_acc: 0.9906
Epoch 5/20
60000/60000 [==============================] - 12s 193us/step - loss: 0.0228
- acc: 0.9927 - val_loss: 0.0254 - val_acc: 0.9923
Epoch 6/20
60000/60000 [==============================] - 12s 193us/step - loss: 0.0212
- acc: 0.9929 - val_loss: 0.0219 - val_acc: 0.9932
Epoch 7/20
60000/60000 [==============================] - 12s 193us/step - loss: 0.0180
- acc: 0.9945 - val_loss: 0.0205 - val_acc: 0.9930
Epoch 8/20
60000/60000 [==============================] - 12s 193us/step - loss: 0.0165
- acc: 0.9946 - val_loss: 0.0207 - val_acc: 0.9934
Epoch 9/20
60000/60000 [==============================] - 12s 193us/step - loss: 0.0149
- acc: 0.9953 - val_loss: 0.0186 - val_acc: 0.9947
Epoch 10/20
60000/60000 [==============================] - 12s 193us/step - loss: 0.0141
- acc: 0.9955 - val_loss: 0.0226 - val_acc: 0.9925
Epoch 11/20
60000/60000 [==============================] - 12s 193us/step - loss: 0.0135
- acc: 0.9957 - val_loss: 0.0236 - val_acc: 0.9940
Epoch 12/20
60000/60000 [==============================] - 12s 194us/step - loss: 0.0130
- acc: 0.9960 - val_loss: 0.0295 - val_acc: 0.9926
Epoch 13/20
60000/60000 [==============================] - 12s 195us/step - loss: 0.0108
- acc: 0.9967 - val_loss: 0.0219 - val_acc: 0.9937
Epoch 14/20
60000/60000 [==============================] - 12s 194us/step - loss: 0.0127
```

```
- acc: 0.9960 - val_loss: 0.0209 - val_acc: 0.9949
Epoch 15/20
60000/60000 [==============================] - 12s 194us/step - loss: 0.0102
- acc: 0.9966 - val_loss: 0.0249 - val_acc: 0.9939
Epoch 16/20
60000/60000 [==============================] - 12s 196us/step - loss: 0.0104
- acc: 0.9970 - val_loss: 0.0208 - val_acc: 0.9951
Epoch 17/20
60000/60000 [==============================] - 12s 195us/step - loss: 0.0085
- acc: 0.9973 - val_loss: 0.0212 - val_acc: 0.9942
Epoch 18/20
60000/60000 [==============================] - 12s 194us/step - loss: 0.0099
- acc: 0.9970 - val_loss: 0.0230 - val_acc: 0.9942
Epoch 19/20
60000/60000 [==============================] - 12s 193us/step - loss: 0.0085
- acc: 0.9972 - val_loss: 0.0270 - val_acc: 0.9933
Epoch 20/20
60000/60000 [==============================] - 12s 193us/step - loss: 0.0086
- acc: 0.9972 - val_loss: 0.0236 - val_acc: 0.9938
```

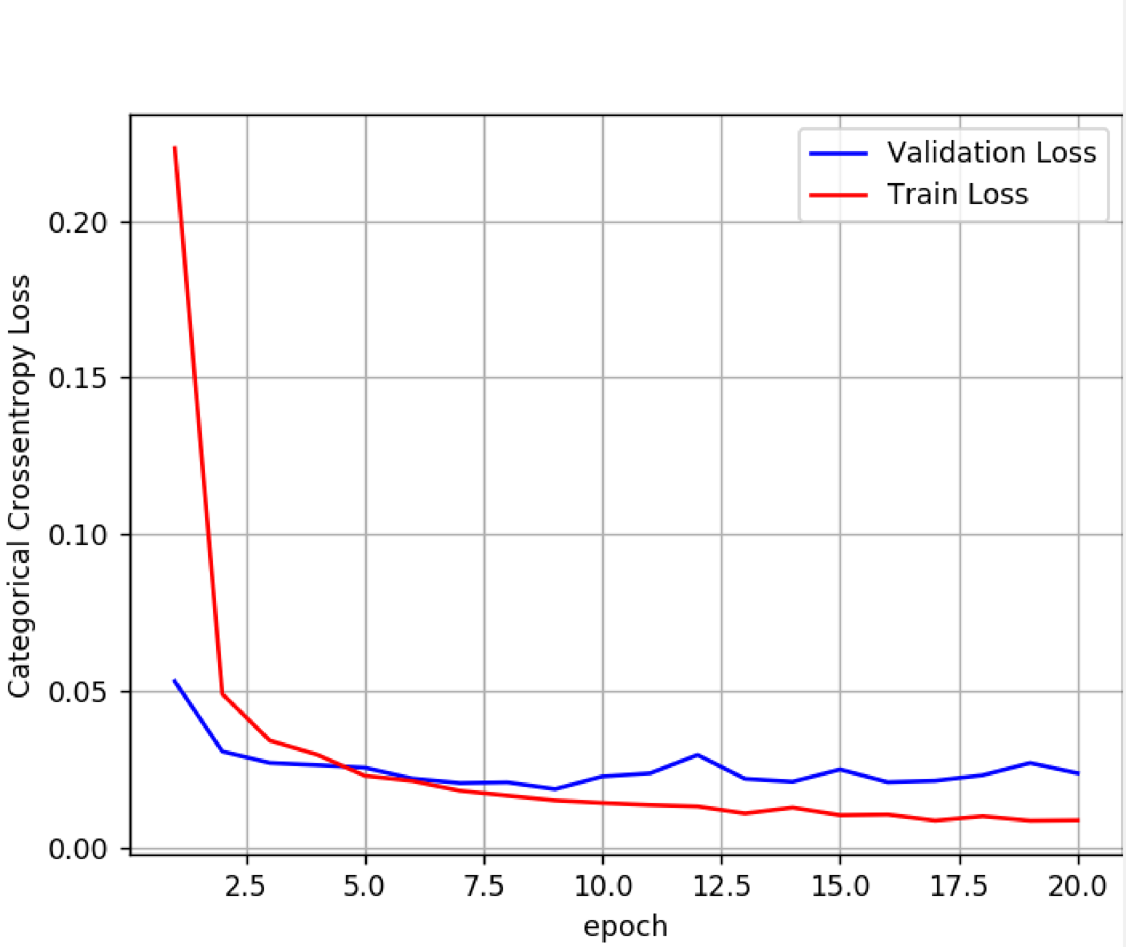
train and test loss vs Epochs

In [9]:

```
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,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, verbos

# 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: 0.023642054841191567
Test accuracy: 0.9938
```

Figure 2



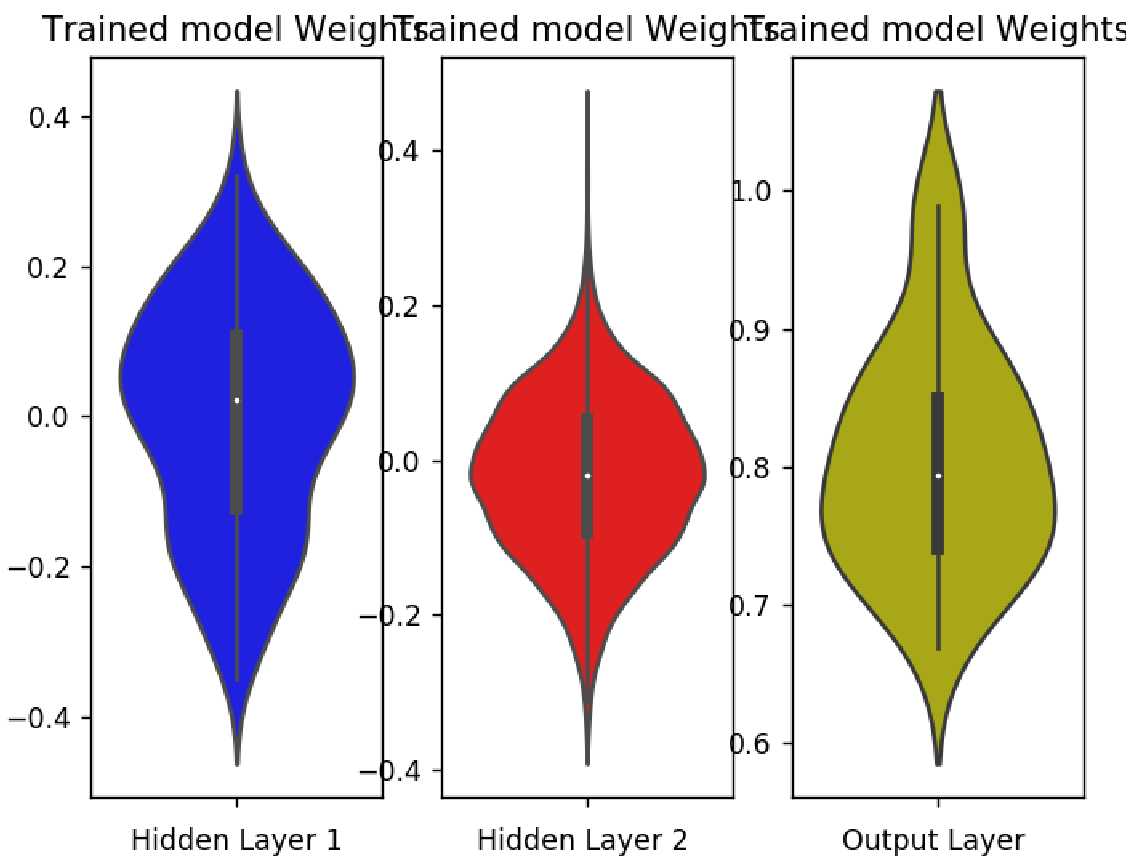
1. Validation loss kept on fluctuating in the range(0.1 to 0.3), train loss was decreasing slowly.

Violin plots of weights

In [10]:

```
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()
```

Figure 3



1. Weights at hidden layer ranges (-0.45 to 0.3).
2. Weights at hidden layer 2 ranges (-0.4 to 0.3).
3. Weights at output layer ranges (0.6 to 1.75).

Model 2

In [11]:

```
model2 = Sequential()
model2.add(Conv2D(32, kernel_size=(3, 3),
                 activation='sigmoid',
                 input_shape=input_shape))
model2.add(Conv2D(64, (3, 3), activation='sigmoid'))
model2.add(MaxPooling2D(pool_size=(2, 2)))
model2.add(Dropout(0.25))
model2.add(Flatten())
model2.add(Dense(128, activation='sigmoid'))
model2.add(Dropout(0.5))
model2.add(Dense(num_classes, activation='softmax'))

model2.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])


print(model2.summary(), '\n')

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_5 (Conv2D)            (None, 26, 26, 32)        320       
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 24, 24, 64)        18496     
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 12, 12, 64)        0         
_________________________________________________________________
dropout_3 (Dropout)          (None, 12, 12, 64)        0         
_________________________________________________________________
flatten_2 (Flatten)          (None, 9216)              0         
_________________________________________________________________
dense_3 (Dense)              (None, 128)               1179776   
_________________________________________________________________
dropout_4 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_4 (Dense)              (None, 10)                1290      
=================================================================
Total params: 1,199,882
Trainable params: 1,199,882
Non-trainable params: 0
_________________________________________________________________
None

Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 8s 129us/step - loss: 2.3103
- acc: 0.1093 - val_loss: 2.3012 - val_acc: 0.1135
Epoch 2/20
60000/60000 [==============================] - 11s 175us/step - loss: 2.3015
- acc: 0.1121 - val_loss: 2.3010 - val_acc: 0.1135
Epoch 3/20
60000/60000 [==============================] - 9s 151us/step - loss: 2.3013
```

```
- acc: 0.1124 - val_loss: 2.3011 - val_acc: 0.1135
Epoch 4/20
60000/60000 [==============================] - 9s 151us/step - loss: 2.3013
- acc: 0.1123 - val_loss: 2.3010 - val_acc: 0.1135
Epoch 5/20
60000/60000 [==============================] - 9s 151us/step - loss: 2.3013
- acc: 0.1120 - val_loss: 2.2964 - val_acc: 0.1135
Epoch 6/20
60000/60000 [==============================] - 9s 153us/step - loss: 0.8546
- acc: 0.7113 - val_loss: 0.3016 - val_acc: 0.9096
Epoch 7/20
60000/60000 [==============================] - 9s 154us/step - loss: 0.3324
- acc: 0.9010 - val_loss: 0.2146 - val_acc: 0.9371
Epoch 8/20
60000/60000 [==============================] - 9s 152us/step - loss: 0.2746
- acc: 0.9197 - val_loss: 0.1722 - val_acc: 0.9490
Epoch 9/20
60000/60000 [==============================] - 9s 153us/step - loss: 0.2348
- acc: 0.9307 - val_loss: 0.1559 - val_acc: 0.9514
Epoch 10/20
60000/60000 [==============================] - 9s 151us/step - loss: 0.2104
- acc: 0.9386 - val_loss: 0.1365 - val_acc: 0.9576
Epoch 11/20
60000/60000 [==============================] - 9s 153us/step - loss: 0.1942
- acc: 0.9429 - val_loss: 0.1224 - val_acc: 0.9615
Epoch 12/20
60000/60000 [==============================] - 9s 151us/step - loss: 0.1781
- acc: 0.9479 - val_loss: 0.1173 - val_acc: 0.9678
Epoch 13/20
60000/60000 [==============================] - 9s 151us/step - loss: 0.1674
- acc: 0.9515 - val_loss: 0.0977 - val_acc: 0.9712
Epoch 14/20
60000/60000 [==============================] - 9s 151us/step - loss: 0.1545
- acc: 0.9547 - val_loss: 0.0980 - val_acc: 0.9707
Epoch 15/20
60000/60000 [==============================] - 9s 154us/step - loss: 0.1484
- acc: 0.9571 - val_loss: 0.0931 - val_acc: 0.9715
Epoch 16/20
60000/60000 [==============================] - 9s 152us/step - loss: 0.1417
- acc: 0.9584 - val_loss: 0.0824 - val_acc: 0.9741
Epoch 17/20
60000/60000 [==============================] - 9s 151us/step - loss: 0.1325
- acc: 0.9609 - val_loss: 0.0835 - val_acc: 0.9748
Epoch 18/20
60000/60000 [==============================] - 9s 151us/step - loss: 0.1231
- acc: 0.9633 - val_loss: 0.0793 - val_acc: 0.9753
Epoch 19/20
60000/60000 [==============================] - 9s 151us/step - loss: 0.1180
- acc: 0.9650 - val_loss: 0.0779 - val_acc: 0.9760
Epoch 20/20
60000/60000 [==============================] - 9s 152us/step - loss: 0.1125
- acc: 0.9662 - val_loss: 0.0704 - val_acc: 0.9791
```

Out[11]:

```
<keras.callbacks.History at 0x1ff87abb940>
```

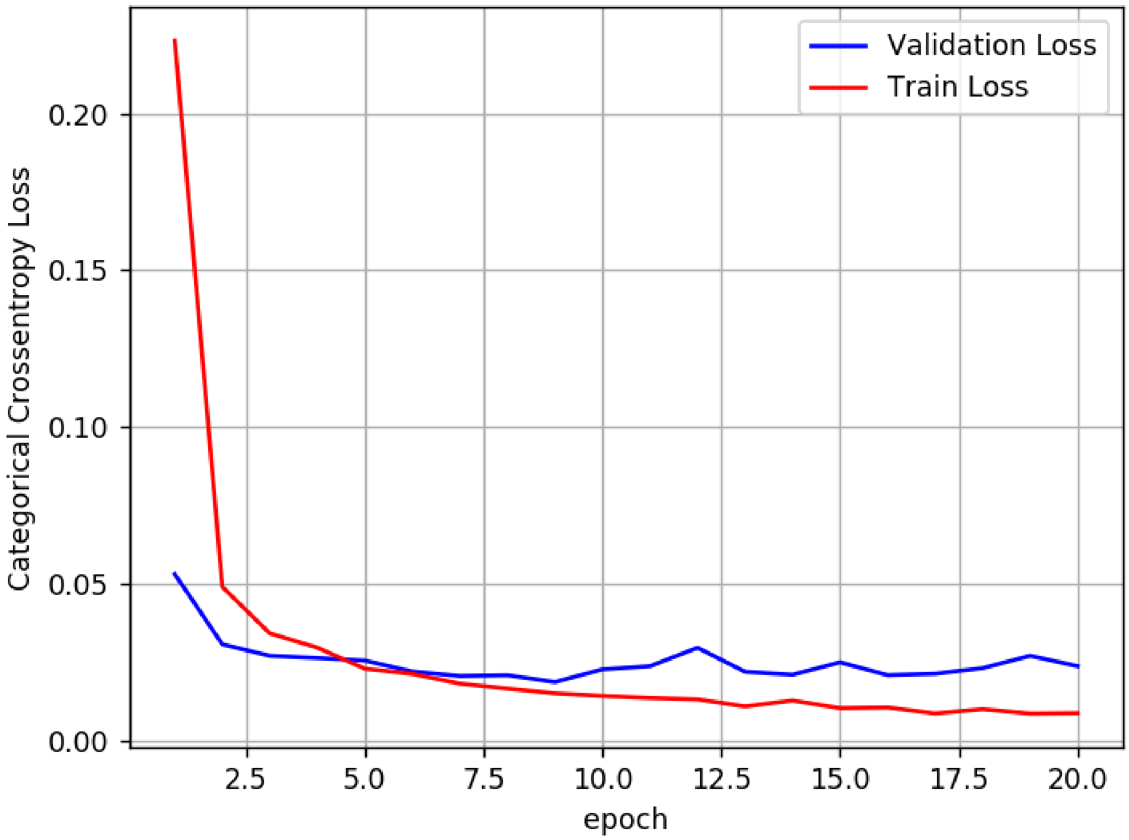
train and test loss vs Epochs

In [12]:

```
score = model2.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,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, verbos

# 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: 0.07039268123237416
Test accuracy: 0.9791
```

Figure 4



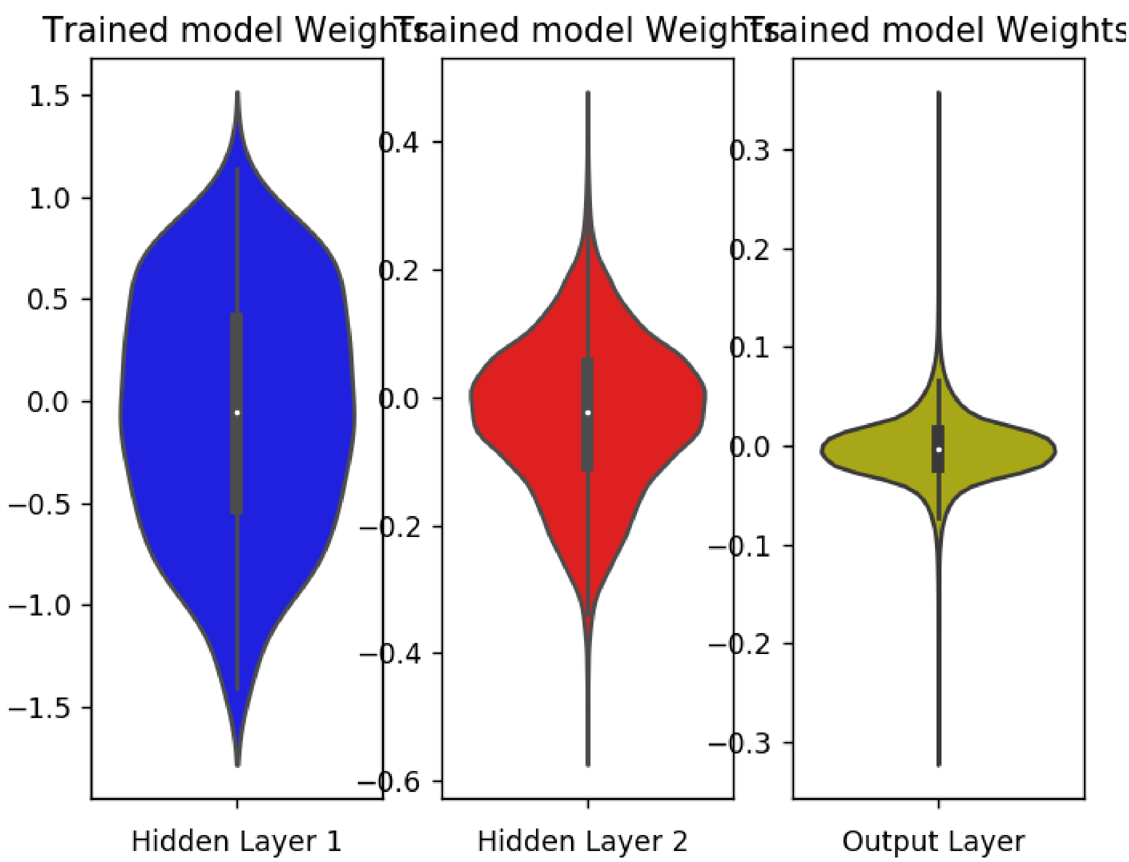
1. validation loss was fluctuating in range (0.02 to 0.03) while train loss was decreasing slowly.

Visualizing weights with violin plot

In [13]:

```
w_after = model2.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()
```

Figure 5



x= y=-0.

1. Weights at hidden layer 1 ranges (-1.75 to 1.5).
2. Weights at hidden layer 2 ranges (-0.4 to 0.4).
3. Weights at output layer ranges (-0.1 to 0.1).

Model 3 -- Failed Model

In [14]:

```
model3 = Sequential()
model3.add(Conv2D(32, kernel_size=(3, 3),
                 activation='sigmoid',
                 input_shape=input_shape))
model3.add(Conv2D(64, (3, 3), activation='sigmoid'))
model3.add(MaxPooling2D(pool_size=(2, 2)))
model3.add(Dropout(0.25))
model3.add(Flatten())
model3.add(Dense(128, activation='sigmoid'))
model3.add(Dropout(0.5))
model3.add(Dense(num_classes, activation='softmax'))

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

print(model3.summary(), '\n')

model3.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_7 (Conv2D)            (None, 26, 26, 32)        320       
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 24, 24, 64)        18496     
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 12, 12, 64)        0         
_________________________________________________________________
dropout_5 (Dropout)          (None, 12, 12, 64)        0         
_________________________________________________________________
flatten_3 (Flatten)          (None, 9216)              0         
_________________________________________________________________
dense_5 (Dense)              (None, 128)               1179776   
_________________________________________________________________
dropout_6 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_6 (Dense)              (None, 10)                1290      
=================================================================
Total params: 1,199,882
Trainable params: 1,199,882
Non-trainable params: 0
_________________________________________________________________
None

Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 9s 148us/step - loss: 2.3322
- acc: 0.1038 - val_loss: 2.3011 - val_acc: 0.1135
Epoch 2/20
60000/60000 [==============================] - 8s 136us/step - loss: 2.3042
- acc: 0.1047 - val_loss: 2.3011 - val_acc: 0.1135
Epoch 3/20
60000/60000 [==============================] - 8s 135us/step - loss: 2.3030
- acc: 0.1086 - val_loss: 2.3010 - val_acc: 0.1135
```

```
Epoch 4/20
60000/60000 [==============================] - 8s 135us/step - loss: 2.3022
- acc: 0.1102 - val_loss: 2.3010 - val_acc: 0.1135
Epoch 5/20
60000/60000 [==============================] - 8s 135us/step - loss: 2.3021
- acc: 0.1104 - val_loss: 2.3010 - val_acc: 0.1135
Epoch 6/20
60000/60000 [==============================] - 8s 135us/step - loss: 2.3017
- acc: 0.1102 - val_loss: 2.3010 - val_acc: 0.1135
Epoch 7/20
60000/60000 [==============================] - 8s 135us/step - loss: 2.3017
- acc: 0.1122 - val_loss: 2.3010 - val_acc: 0.1135
Epoch 8/20
60000/60000 [==============================] - 8s 136us/step - loss: 2.3017
- acc: 0.1120 - val_loss: 2.3010 - val_acc: 0.1135
Epoch 9/20
60000/60000 [==============================] - 8s 135us/step - loss: 2.3016
- acc: 0.1123 - val_loss: 2.3010 - val_acc: 0.1135
Epoch 10/20
60000/60000 [==============================] - 8s 135us/step - loss: 2.3014
- acc: 0.1121 - val_loss: 2.3010 - val_acc: 0.1135
Epoch 11/20
60000/60000 [==============================] - 8s 137us/step - loss: 2.3014
- acc: 0.1124 - val_loss: 2.3010 - val_acc: 0.1135
Epoch 12/20
60000/60000 [==============================] - 8s 135us/step - loss: 2.3013
- acc: 0.1124 - val_loss: 2.3010 - val_acc: 0.1135
Epoch 13/20
60000/60000 [==============================] - 8s 135us/step - loss: 2.3014
- acc: 0.1121 - val_loss: 2.3009 - val_acc: 0.1135
Epoch 14/20
60000/60000 [==============================] - 8s 135us/step - loss: 2.3013
- acc: 0.1120 - val_loss: 2.3009 - val_acc: 0.1135
Epoch 15/20
60000/60000 [==============================] - 8s 135us/step - loss: 2.3013
- acc: 0.1121 - val_loss: 2.3009 - val_acc: 0.1135
Epoch 16/20
60000/60000 [==============================] - 8s 136us/step - loss: 2.3014
- acc: 0.1126 - val_loss: 2.3009 - val_acc: 0.1135
Epoch 17/20
60000/60000 [==============================] - 8s 137us/step - loss: 2.3013
- acc: 0.1122 - val_loss: 2.3009 - val_acc: 0.1135
Epoch 18/20
60000/60000 [==============================] - 8s 136us/step - loss: 2.3014
- acc: 0.1121 - val_loss: 2.3009 - val_acc: 0.1135
Epoch 19/20
60000/60000 [==============================] - 8s 136us/step - loss: 2.3013
- acc: 0.1122 - val_loss: 2.3009 - val_acc: 0.1135
Epoch 20/20
60000/60000 [==============================] - 8s 136us/step - loss: 2.3014
- acc: 0.1123 - val_loss: 2.3009 - val_acc: 0.1135
```

Out[14]:

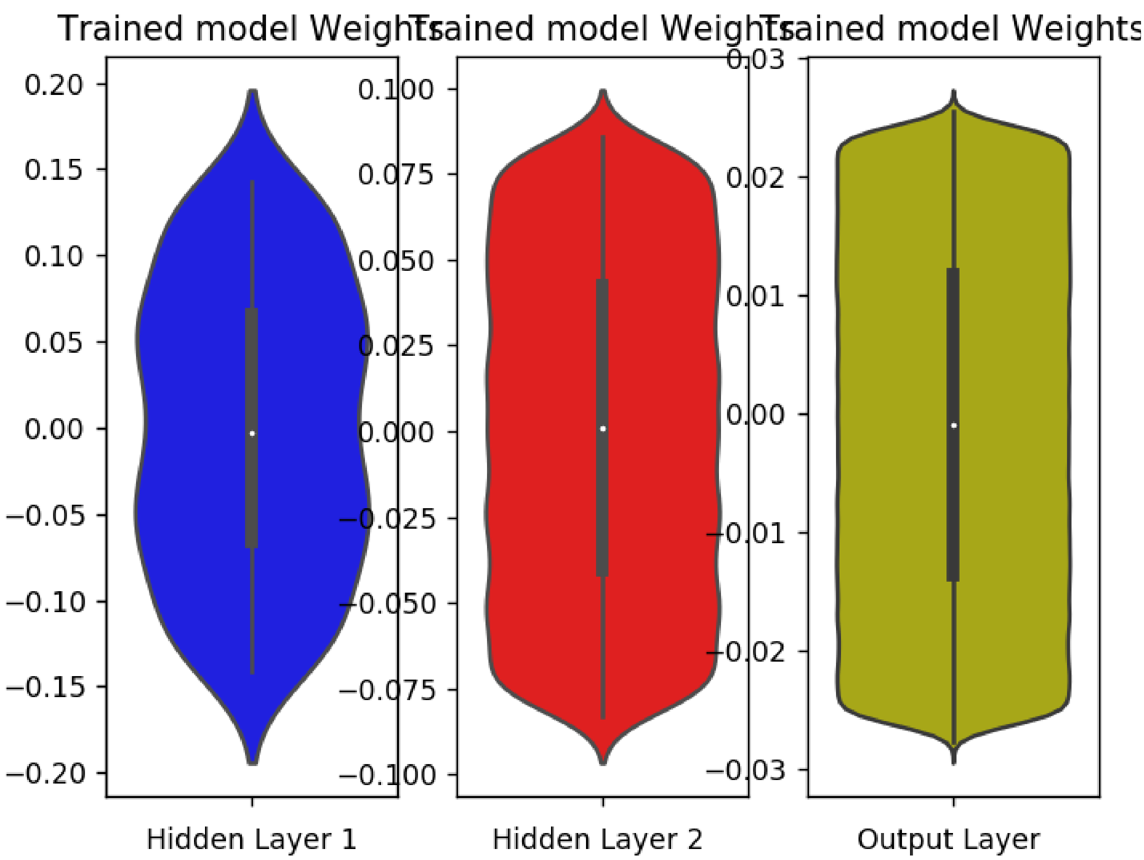
```
<keras.callbacks.History at 0x1ffb55c4128>
```

SGD Optimizer seems to have stuck at local minima

In [15]:

```
w_after = model3.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()
```

Figure 6



Downlo

1. Weights at hidden layer 1 ranges (-0.2 to 0.2).
2. Weights at hidden layer 2 ranges (-0.1 to 0.1).
3. Weights at output layer ranges (-0.03 to 0.03).

This model was not able to converge, SGD optimizer seems to have stuck at local minima

Model 4 -- Best Model

In [17]:

```
model4 = Sequential()

model4.add(Conv2D(filters=16, kernel_size=(5,5), strides =(1,1), padding = 'same',
                 activation = 'relu', input_shape = input_shape,
                 kernel_initializer = he_normal(seed=None)))

model4.add(Conv2D(filters=32, kernel_size=(5,5), strides =(1,1), padding = 'same',
                 activation = 'relu', kernel_initializer = he_normal(seed=None)))

model4.add(MaxPooling2D(pool_size=(2,2), strides=2))

model4.add(BatchNormalization())

model4.add(Dropout(0.25))

model4.add(Conv2D(filters=64, kernel_size=(5,5), strides =(1,1), padding = 'same',
                 activation = 'relu', kernel_initializer = he_normal(seed=None)))

model4.add(Conv2D(filters=64, kernel_size=(5,5), strides =(1,1), padding = 'same',
                 activation = 'relu', kernel_initializer = he_normal(seed=None)))

model4.add(MaxPooling2D(pool_size=(2,2), strides=2))

model4.add(Dropout(0.25))

model4.add(Conv2D(filters=128, kernel_size=(5,5), strides =(1,1), padding = 'same',
                 activation = 'relu', kernel_initializer = he_normal(seed=None)))

model4.add(MaxPooling2D(pool_size=(2,2), strides=2))

model4.add(Dropout(0.25))

model4.add(Conv2D(filters=256, kernel_size=(5,5), strides =(1,1), padding = 'same',
                 activation = 'relu', kernel_initializer = he_normal(seed=None)))

model4.add(MaxPooling2D(pool_size=(2,2), strides=2))


model4.add(Flatten())

model4.add(Dense(units = 512, activation='relu', kernel_initializer= he_normal(seed=None)))

model4.add(BatchNormalization())

model4.add(Dropout(0.25))

model4.add(Dense(units = 64, activation='relu', kernel_initializer= he_normal(seed=None)))

model4.add(BatchNormalization())

model4.add(Dropout(0.25))

model4.add(Dense(units = num_classes, activation='softmax'))

model4.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'

print(model4.summary(), '\n')

history = model4.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_9 (Conv2D)            (None, 28, 28, 16)        416       
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 28, 28, 32)        12832     
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 14, 14, 32)        0         
_________________________________________________________________
batch_normalization_2 (Batch (None, 14, 14, 32)        128       
_________________________________________________________________
dropout_7 (Dropout)          (None, 14, 14, 32)        0         
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 14, 14, 64)        51264     
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 14, 14, 64)        102464    
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 7, 7, 64)          0         
_________________________________________________________________
dropout_8 (Dropout)          (None, 7, 7, 64)          0         
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 7, 7, 128)         204928    
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 3, 3, 128)         0         
_________________________________________________________________
dropout_9 (Dropout)          (None, 3, 3, 128)         0         
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 3, 3, 256)         819456    
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 1, 1, 256)         0         
_________________________________________________________________
flatten_4 (Flatten)          (None, 256)               0         
_________________________________________________________________
dense_7 (Dense)              (None, 512)               131584    
_________________________________________________________________
batch_normalization_3 (Batch (None, 512)               2048      
_________________________________________________________________
dropout_10 (Dropout)         (None, 512)               0         
_________________________________________________________________
dense_8 (Dense)              (None, 64)                32832     
_________________________________________________________________
batch_normalization_4 (Batch (None, 64)                256       
_________________________________________________________________
dropout_11 (Dropout)         (None, 64)                0         
_________________________________________________________________
dense_9 (Dense)              (None, 10)                650       
=================================================================
Total params: 1,358,858
Trainable params: 1,357,642
Non-trainable params: 1,216
_________________________________________________________________
None

Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 23s 388us/step - loss: 0.3097
- acc: 0.9071 - val_loss: 0.0473 - val_acc: 0.9841
Epoch 2/20
```

```
60000/60000 [==============================] - 20s 332us/step - loss: 0.0625
- acc: 0.9818 - val_loss: 0.0478 - val_acc: 0.9827
Epoch 3/20
60000/60000 [==============================] - 20s 332us/step - loss: 0.0472
- acc: 0.9862 - val_loss: 0.0302 - val_acc: 0.9906
Epoch 4/20
60000/60000 [==============================] - 20s 334us/step - loss: 0.0411
- acc: 0.9881 - val_loss: 0.0282 - val_acc: 0.9905
Epoch 5/20
60000/60000 [==============================] - 20s 334us/step - loss: 0.0335
- acc: 0.9898 - val_loss: 0.0296 - val_acc: 0.9905
Epoch 6/20
60000/60000 [==============================] - 20s 334us/step - loss: 0.0281
- acc: 0.9919 - val_loss: 0.1630 - val_acc: 0.9523
Epoch 7/20
60000/60000 [==============================] - 20s 335us/step - loss: 0.0267
- acc: 0.9925 - val_loss: 0.0218 - val_acc: 0.9932
Epoch 8/20
60000/60000 [==============================] - 20s 334us/step - loss: 0.0242
- acc: 0.9927 - val_loss: 0.0245 - val_acc: 0.9930
Epoch 9/20
60000/60000 [==============================] - 20s 336us/step - loss: 0.0231
- acc: 0.9927 - val_loss: 0.0208 - val_acc: 0.9937
Epoch 10/20
60000/60000 [==============================] - 20s 335us/step - loss: 0.0202
- acc: 0.9938 - val_loss: 0.0369 - val_acc: 0.9891
Epoch 11/20
60000/60000 [==============================] - 20s 336us/step - loss: 0.0180
- acc: 0.9946 - val_loss: 0.0337 - val_acc: 0.9914
Epoch 12/20
60000/60000 [==============================] - 20s 336us/step - loss: 0.0198
- acc: 0.9938 - val_loss: 0.0292 - val_acc: 0.9915
Epoch 13/20
60000/60000 [==============================] - 20s 335us/step - loss: 0.0172
- acc: 0.9945 - val_loss: 0.0313 - val_acc: 0.9920
Epoch 14/20
60000/60000 [==============================] - 20s 336us/step - loss: 0.0155
- acc: 0.9954 - val_loss: 0.0274 - val_acc: 0.9928
Epoch 15/20
60000/60000 [==============================] - 20s 336us/step - loss: 0.0128
- acc: 0.9958 - val_loss: 0.0263 - val_acc: 0.9924
Epoch 16/20
60000/60000 [==============================] - 20s 334us/step - loss: 0.0132
- acc: 0.9963 - val_loss: 0.0211 - val_acc: 0.9937
Epoch 17/20
60000/60000 [==============================] - 20s 336us/step - loss: 0.0134
- acc: 0.9960 - val_loss: 0.0243 - val_acc: 0.9933
Epoch 18/20
60000/60000 [==============================] - 20s 335us/step - loss: 0.0131
- acc: 0.9962 - val_loss: 0.0243 - val_acc: 0.9939
Epoch 19/20
60000/60000 [==============================] - 20s 338us/step - loss: 0.0132
- acc: 0.9958 - val_loss: 0.0213 - val_acc: 0.9938
Epoch 20/20
60000/60000 [==============================] - 20s 338us/step - loss: 0.0119
- acc: 0.9962 - val_loss: 0.0189 - val_acc: 0.9947
```

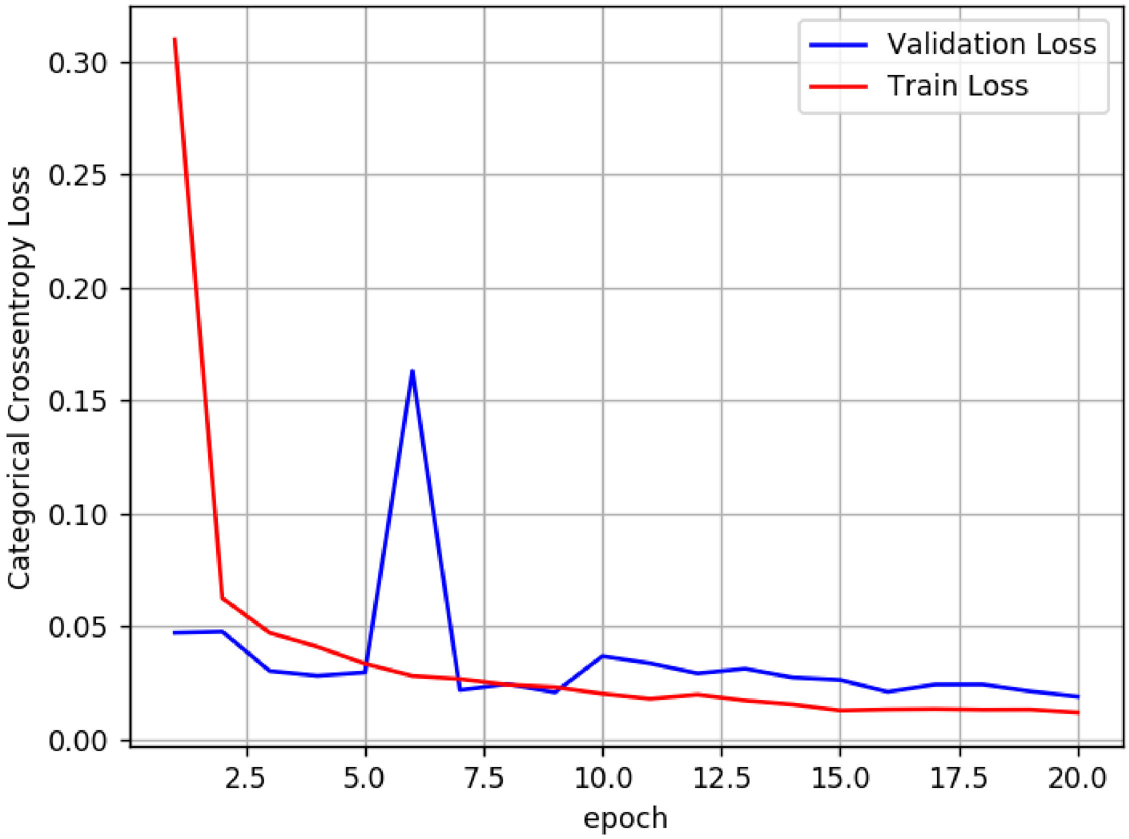
Train and test loss vs epochs

In [18]:

```
score = model4.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,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, verbos

# 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: 0.018944656266622841
Test accuracy: 0.9947
```

Figure 7



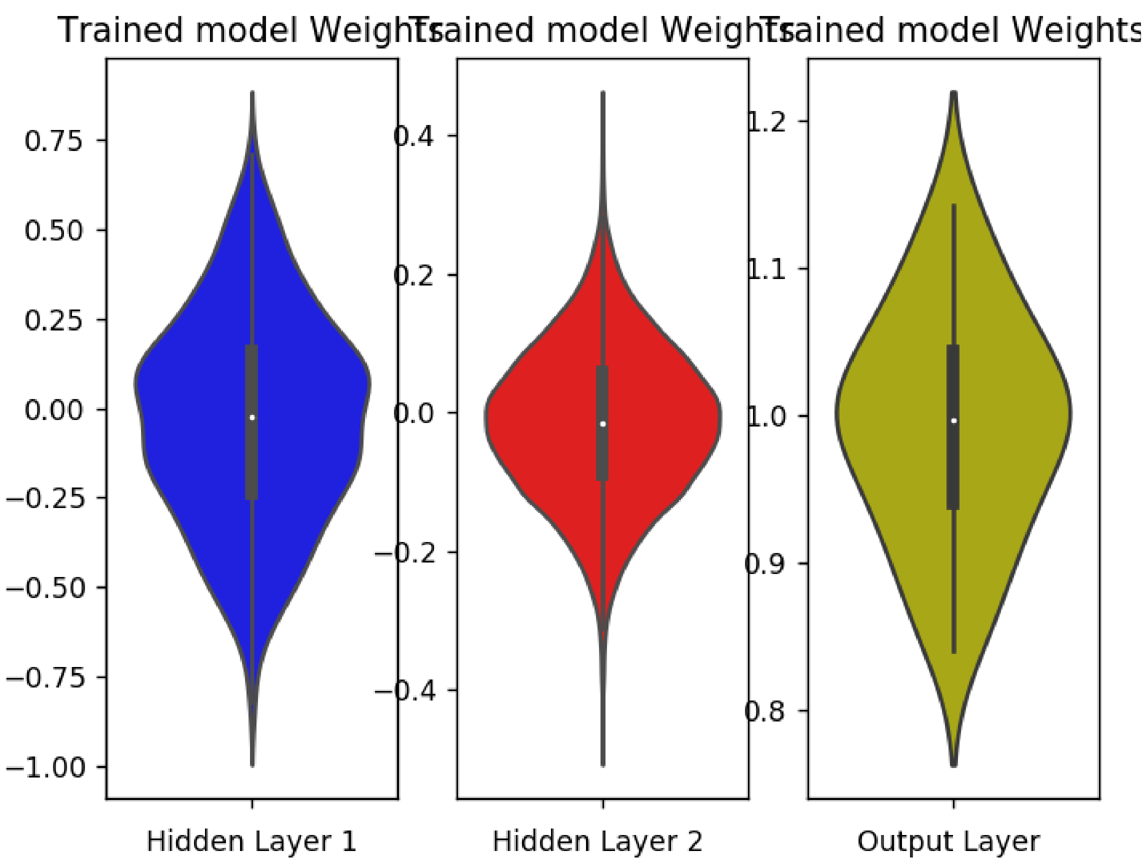
1. Both train and test loss are decreasing with number of epochs.

Visualizing weights with violin plot

In [19]:

```
w_after = model4.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()
```

Figure 8



1. Weights at hidden layer 1 ranges (-1.25 to 1).
2. Weights at hidden layer 2 ranges (-0.3 to 0.3).
3. Weights at output layer ranges (0.75 to i.25).

In [24]:

```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["model", "No. convolution layers","Activation function","Optimizer","trair
x.add_row(["Model 1", 4 , "ReLU" , "adam" ,0.008 , 0.026 , "99.7%" , "99.3%"])
x.add_row(["Model 2", 2 ,  "Sigmoid" , "adadelta" ,  0.11 , 0.07 , "97%" , "98%"])
x.add_row(["Model 3", 2 ,  "Sigmoid" , "SGD" ,  2.30 , 2.30 , "11%" , "11%"])
x.add_row(["Model 4", 6 ,  "ReLU" , "adam" , 0.01 , 0.01 , "99.6%" , "99.4%"])

print(x)
```

```
+---------+------------------------+---------------------+-----------+------
------+-----------+---------------+---------------+
|  model  | No. convolution layers | Activation function | Optimizer | train
loss | test loss | Train accuracy | Test accuracy |
+---------+------------------------+---------------------+-----------+------
------+-----------+---------------+---------------+
| Model 1 |           4            |         ReLU        |    adam   |  0.0
08    |   0.026   |     99.7%     |     99.3%     |
| Model 2 |           2            |       Sigmoid       |  adadelta |   0.
11    |    0.07   |      97%      |      98%      |
| Model 3 |           2            |       Sigmoid       |    SGD    |   2.
3     |    2.3    |      11%      |      11%      |
| Model 4 |           6            |         ReLU        |    adam   |   0.
01    |    0.01   |     99.6%     |     99.4%     |
+---------+------------------------+---------------------+-----------+------
------+-----------+---------------+---------------+
```

Obeservations:

1. Model 1 was overfitting as train loss:0.008 and test loss:0.026.
2. Model 2 has performed better.
3. Model 3 that has SGD optimizer seems to have stuck at local minima, hence model 3 is the worst model.
4. He_normal weight initialization has slightly fasten the convergence of the model.
5. Model 4 is the best model, highest accuracy, fast convergence with no overfitting.
6. Adam optimizer has performed better than other optimizers.