Assignment-12 : Apply Keras on MNIST DataSet

In [0]:
```
# if you keras is not using tensorflow as backend set "KERAS_BACKEND=tensorflo
w" use this command
from keras.utils import np_utils
from keras.datasets import mnist
import seaborn as sns
from keras.initializers import RandomNormal

%matplotlib notebook
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import time
```

In [0]:
```
%matplotlib notebook
import matplotlib.pyplot as plt
import numpy as np
import time
def plot_loss(x, vy, ty, xlabel='Epoch', ylabel='Categorical Crossentropy Los
s'):
         _, = plt.plot(x, vy, 'b', label="Validation Loss")
         _, = plt.plot(x, ty, 'r', label="Train Loss")
         plt.xlabel(xlabel)
         plt.ylabel(ylabel)

         plt.grid()
         plt.legend()
         plt.grid()
         plt.show()
```

In [3]:
```
# the data, shuffled and split between train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

```
Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz
11493376/11490434 [==============================] - 2s 0us/step
```

In [4]:
```
print("Number of training examples :", X_train.shape[0], "and each image is of
shape (%d, %d)"%(X_train.shape[1], X_train.shape[2]))
print("Number of training examples :", X_test.shape[0], "and each image is of
 shape (%d, %d)"%(X_test.shape[1], X_test.shape[2]))
```

```
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 [0]:
```
# if you observe the input shape its 2 dimensional vector
# for each image we have a (28*28) vector
# we will convert the (28*28) vector into single dimensional vector of 1 * 784

X_train = X_train.reshape(X_train.shape[0], X_train.shape[1]*X_train.shape[2])
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1]*X_test.shape[2])
```

In [6]:
```
# after converting the input images from 3d to 2d vectors

print("Number of training examples :", X_train.shape[0], "and each image is of
shape (%d)"%(X_train.shape[1]))
print("Number of training examples :", X_test.shape[0], "and each image is of
 shape (%d)"%(X_test.shape[1]))
```

```
Number of training examples : 60000 and each image is of shape (784)
Number of training examples : 10000 and each image is of shape (784)
```

In [0]:
```
# if we observe the above matrix each cell is having a value between 0-255
# before we move to apply machine learning algorithms lets try to normalize th
e data
# X => (X - Xmin)/(Xmax-Xmin) = X/255

X_train = X_train/255
X_test = X_test/255
```

In [8]:
```
# here we are having a class number for each image
print("Class label of first image :", y_train[0])

# lets convert this into a 10 dimensional vector
# ex: consider an image is 5 convert it into 5 => [0, 0, 0, 0, 0, 1, 0, 0, 0,
 0]
# this conversion needed for MLPs

Y_train = np_utils.to_categorical(y_train, 10)
Y_test = np_utils.to_categorical(y_test, 10)

print("After converting the output into a vector : ",Y_train[0])
```

```
Class label of first image : 5
After converting the output into a vector :  [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

In [0]:
```
# some model parameters

output_dim = 10
input_dim = X_train.shape[1]

batch_size = 128
nb_epoch = 20
```

Multi-Layer Perceptron With 2 hidden layer

1. MLP + Relu Activation + Adam Optimizer (H1: 720 , H2 : 200)

In [0]:
```
from keras.models import Sequential
from keras.layers import Dense, Activation,Dropout, BatchNormalization
```

In [13]:
```
%%time
def Build_NN_2(input_dim, output_dim=10):
  model = Sequential()
  model.add(Dense(720, activation='relu', kernel_initializer='he_normal', inpu
t_shape=(input_dim,)))

  model.add(Dense(200, activation='relu', kernel_initializer='he_normal'))

  model.add(Dense(output_dim, activation='softmax'))

  model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['a
ccuracy'])
  print(model.summary())

  return model
```

```
CPU times: user 4 µs, sys: 0 ns, total: 4 µs
Wall time: 23.8 µs
```

In [17]:
```
model=Build_NN_2(input_dim)
print()
history = model.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch,
validation_split=0.3, verbose=1)
```

```
W0713 10:20:14.764899 140220561528704 deprecation.py:323] From /usr/local/li
b/python3.6/dist-packages/tensorflow/python/ops/math_grad.py:1250: add_dispat
ch_support.<locals>.wrapper (from tensorflow.python.ops.array_ops) is depreca
ted and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_10 (Dense)             (None, 720)               565200    
_________________________________________________________________
dense_11 (Dense)             (None, 200)               144200    
_________________________________________________________________
dense_12 (Dense)             (None, 10)                2010      
=================================================================
Total params: 711,410
Trainable params: 711,410
Non-trainable params: 0
_________________________________________________________________
None

Train on 42000 samples, validate on 18000 samples
Epoch 1/20
42000/42000 [==============================] - 5s 127us/step - loss: 0.2554 - acc: 0.9246 - val_loss: 0.1375 - val_acc: 0.9583
Epoch 2/20
42000/42000 [==============================] - 1s 34us/step - loss: 0.0917 - acc: 0.9724 - val_loss: 0.1157 - val_acc: 0.9638
Epoch 3/20
42000/42000 [==============================] - 1s 34us/step - loss: 0.0564 - acc: 0.9823 - val_loss: 0.1064 - val_acc: 0.9690
Epoch 4/20
42000/42000 [==============================] - 1s 35us/step - loss: 0.0345 - acc: 0.9894 - val_loss: 0.1036 - val_acc: 0.9711
Epoch 5/20
42000/42000 [==============================] - 2s 36us/step - loss: 0.0250 - acc: 0.9921 - val_loss: 0.0896 - val_acc: 0.9744
Epoch 6/20
42000/42000 [==============================] - 2s 37us/step - loss: 0.0181 - acc: 0.9939 - val_loss: 0.0993 - val_acc: 0.9742
Epoch 7/20
42000/42000 [==============================] - 2s 37us/step - loss: 0.0199 - acc: 0.9936 - val_loss: 0.0978 - val_acc: 0.9754
Epoch 8/20
42000/42000 [==============================] - 2s 36us/step - loss: 0.0182 - acc: 0.9944 - val_loss: 0.1027 - val_acc: 0.9735
Epoch 9/20
42000/42000 [==============================] - 2s 37us/step - loss: 0.0075 - acc: 0.9978 - val_loss: 0.1142 - val_acc: 0.9750
Epoch 10/20
42000/42000 [==============================] - 1s 35us/step - loss: 0.0122 - acc: 0.9955 - val_loss: 0.1199 - val_acc: 0.9717
Epoch 11/20
42000/42000 [==============================] - 2s 36us/step - loss: 0.0101 - acc: 0.9967 - val_loss: 0.1259 - val_acc: 0.9741
Epoch 12/20
42000/42000 [==============================] - 1s 34us/step - loss: 0.0095 - acc: 0.9972 - val_loss: 0.1256 - val_acc: 0.9704
Epoch 13/20
42000/42000 [==============================] - 1s 34us/step - loss: 0.0118 - acc: 0.9961 - val_loss: 0.1064 - val_acc: 0.9773
Epoch 14/20
42000/42000 [==============================] - 1s 34us/step - loss: 0.0103 -
```

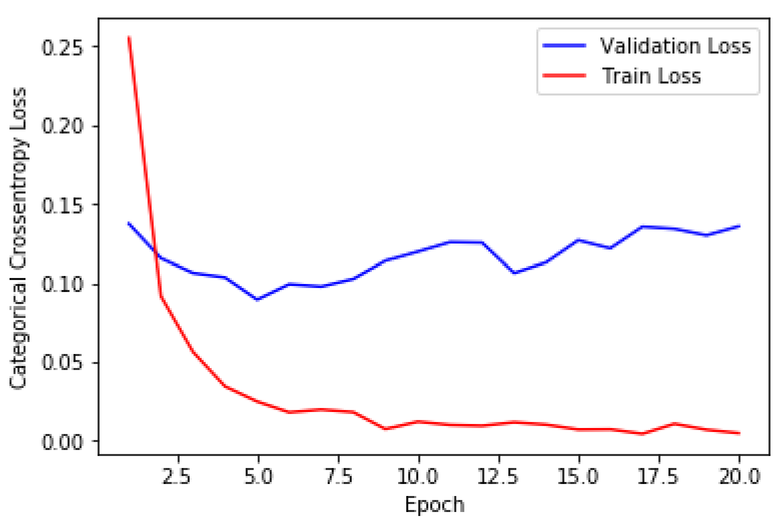
```
acc: 0.9965 - val_loss: 0.1131 - val_acc: 0.9760
Epoch 15/20
42000/42000 [==============================] - 1s 35us/step - loss: 0.0072 -
acc: 0.9978 - val_loss: 0.1271 - val_acc: 0.9744
Epoch 16/20
42000/42000 [==============================] - 1s 34us/step - loss: 0.0073 -
acc: 0.9974 - val_loss: 0.1220 - val_acc: 0.9767
Epoch 17/20
42000/42000 [==============================] - 1s 34us/step - loss: 0.0044 -
acc: 0.9986 - val_loss: 0.1356 - val_acc: 0.9746
Epoch 18/20
42000/42000 [==============================] - 1s 35us/step - loss: 0.0108 -
acc: 0.9965 - val_loss: 0.1343 - val_acc: 0.9759
Epoch 19/20
42000/42000 [==============================] - 1s 34us/step - loss: 0.0070 -
acc: 0.9977 - val_loss: 0.1302 - val_acc: 0.9754
Epoch 20/20
42000/42000 [==============================] - 1s 34us/step - loss: 0.0049 -
acc: 0.9986 - val_loss: 0.1358 - val_acc: 0.9744
```

Plot

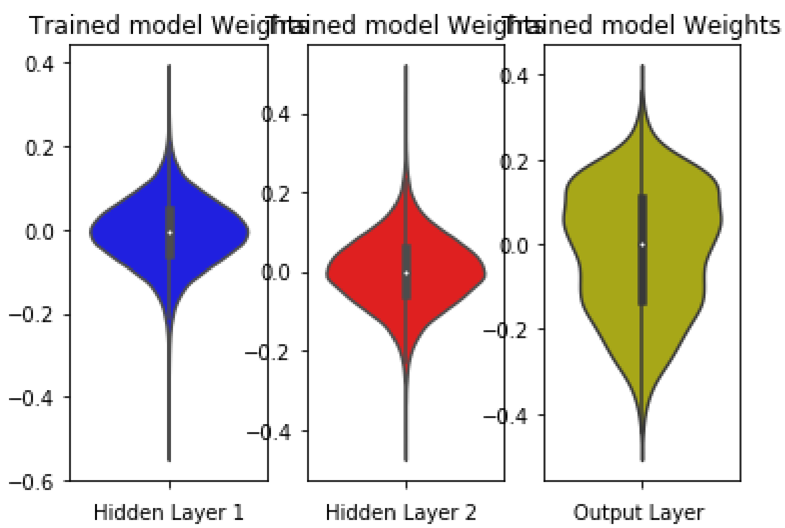
In [32]:

```
score = model.evaluate(X_test, Y_test, verbose=0)
print(f'Test Score: {score[0]}')
print(f'Test Accuracy: {score[1]}\n')


x = list(range(1, nb_epoch+1))
vy = history.history['val_loss']
ty = history.history['loss']
plot_loss(x, vy, ty)-
```

```
Test Score: 0.11015889572340877
Test Accuracy: 0.9787
```



In [33]:
```
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()
```



2. MLP + Relu Activation + Adam Optimizer + Batch Normalization (H1: 720 , H2: 200)

In [34]:
```
%%time
def Build_NN_2(input_dim, output_dim=10):
  model = Sequential()
  model.add(Dense(720, activation='relu', kernel_initializer='he_normal', inpu
t_shape=(input_dim,)))
  model.add(BatchNormalization())

  model.add(Dense(200, activation='relu', kernel_initializer='he_normal'))
  model.add(BatchNormalization())

  model.add(Dense(output_dim, activation='softmax'))

  model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['a
ccuracy'])
  print(model.summary())

  return model
```

```
CPU times: user 8 µs, sys: 0 ns, total: 8 µs
Wall time: 12.6 µs
```

In [37]:
```
model=Build_NN_2(input_dim)
print()
history = model.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch,
validation_split=0.3, verbose=1)
```

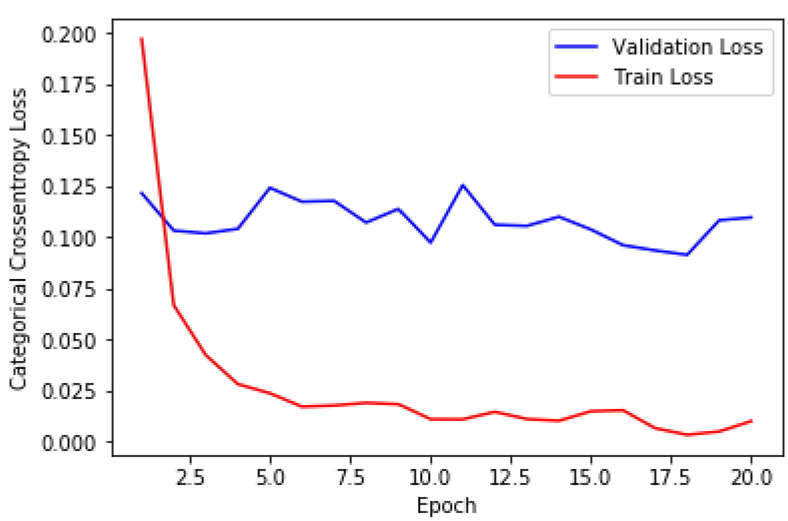
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_14 (Dense)             (None, 720)               565200    
_________________________________________________________________
batch_normalization_1 (Batch (None, 720)               2880      
_________________________________________________________________
dense_15 (Dense)             (None, 200)               144200    
_________________________________________________________________
batch_normalization_2 (Batch (None, 200)               800       
_________________________________________________________________
dense_16 (Dense)             (None, 10)                2010      
=================================================================
Total params: 715,090
Trainable params: 713,250
Non-trainable params: 1,840
_________________________________________________________________
None

Train on 42000 samples, validate on 18000 samples
Epoch 1/20
42000/42000 [==============================] - 3s 74us/step - loss: 0.1973 -
acc: 0.9413 - val_loss: 0.1216 - val_acc: 0.9645
Epoch 2/20
42000/42000 [==============================] - 2s 54us/step - loss: 0.0669 -
acc: 0.9805 - val_loss: 0.1032 - val_acc: 0.9691
Epoch 3/20
42000/42000 [==============================] - 2s 54us/step - loss: 0.0423 -
acc: 0.9869 - val_loss: 0.1019 - val_acc: 0.9706
Epoch 4/20
42000/42000 [==============================] - 2s 54us/step - loss: 0.0281 -
acc: 0.9911 - val_loss: 0.1041 - val_acc: 0.9691
Epoch 5/20
42000/42000 [==============================] - 2s 54us/step - loss: 0.0236 -
acc: 0.9928 - val_loss: 0.1242 - val_acc: 0.9671
Epoch 6/20
42000/42000 [==============================] - 2s 54us/step - loss: 0.0170 -
acc: 0.9949 - val_loss: 0.1174 - val_acc: 0.9693
Epoch 7/20
42000/42000 [==============================] - 2s 54us/step - loss: 0.0176 -
acc: 0.9939 - val_loss: 0.1178 - val_acc: 0.9687
Epoch 8/20
42000/42000 [==============================] - 2s 53us/step - loss: 0.0189 -
acc: 0.9940 - val_loss: 0.1072 - val_acc: 0.9727
Epoch 9/20
42000/42000 [==============================] - 2s 55us/step - loss: 0.0182 -
acc: 0.9942 - val_loss: 0.1138 - val_acc: 0.9711
Epoch 10/20
42000/42000 [==============================] - 2s 54us/step - loss: 0.0109 -
acc: 0.9966 - val_loss: 0.0973 - val_acc: 0.9764
Epoch 11/20
42000/42000 [==============================] - 2s 54us/step - loss: 0.0108 -
acc: 0.9965 - val_loss: 0.1255 - val_acc: 0.9703
Epoch 12/20
42000/42000 [==============================] - 2s 53us/step - loss: 0.0145 -
acc: 0.9954 - val_loss: 0.1061 - val_acc: 0.9752
Epoch 13/20
```

```
42000/42000 [==============================] - 2s 54us/step - loss: 0.0110 -
acc: 0.9963 - val_loss: 0.1055 - val_acc: 0.9760
Epoch 14/20
42000/42000 [==============================] - 2s 54us/step - loss: 0.0101 -
acc: 0.9967 - val_loss: 0.1100 - val_acc: 0.9742
Epoch 15/20
42000/42000 [==============================] - 2s 54us/step - loss: 0.0148 -
acc: 0.9954 - val_loss: 0.1037 - val_acc: 0.9747
Epoch 16/20
42000/42000 [==============================] - 2s 54us/step - loss: 0.0153 -
acc: 0.9948 - val_loss: 0.0960 - val_acc: 0.9768
Epoch 17/20
42000/42000 [==============================] - 2s 54us/step - loss: 0.0065 -
acc: 0.9979 - val_loss: 0.0935 - val_acc: 0.9789
Epoch 18/20
42000/42000 [==============================] - 2s 54us/step - loss: 0.0032 -
acc: 0.9993 - val_loss: 0.0913 - val_acc: 0.9788
Epoch 19/20
42000/42000 [==============================] - 2s 54us/step - loss: 0.0049 -
acc: 0.9987 - val_loss: 0.1083 - val_acc: 0.9769
Epoch 20/20
42000/42000 [==============================] - 2s 53us/step - loss: 0.0099 -
acc: 0.9970 - val_loss: 0.1097 - val_acc: 0.9766
```

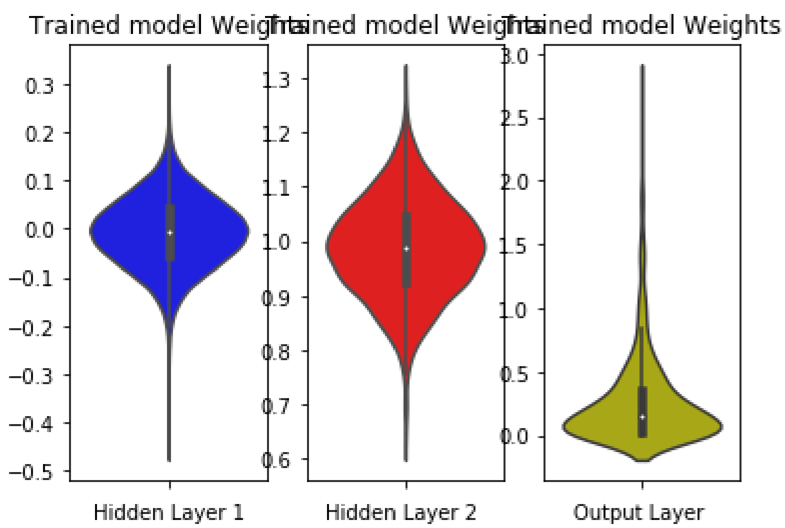
In [38]:

```
score = model.evaluate(X_test, Y_test, verbose=0)
print(f'Test Score: {score[0]}')
print(f'Test Accuracy: {score[1]}\n')


x = list(range(1, nb_epoch+1))
vy = history.history['val_loss']
ty = history.history['loss']
plot_loss(x, vy, ty)
```

```
Test Score: 0.08973172496785191
Test Accuracy: 0.9797
```



In [39]:
```
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()
```



3. MLP + Relu Activation + Adam Optimizer + Dropout (H1: 720 , H2: 200)

In [40]:
```
%%time
def Build_NN_2(input_dim, output_dim=10):
  model = Sequential()

  model.add(Dense(720, activation='relu', kernel_initializer='he_normal', inpu
t_shape=(input_dim,)))
  model.add(Dropout(0.5))


  model.add(Dense(200, activation='relu', kernel_initializer='he_normal'))
  model.add(Dropout(0.5))


  model.add(Dense(output_dim, activation='softmax'))

  model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['a
ccuracy'])
  print(model.summary())

  return model
```

```
CPU times: user 4 µs, sys: 1e+03 ns, total: 5 µs
Wall time: 7.63 µs
```

In [41]:
```
model=Build_NN_2(input_dim)
print()
history = model.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch,
validation_split=0.3, verbose=1)
```

```
W0713 10:58:05.901090 140220561528704 deprecation.py:506] From /usr/local/li
b/python3.6/dist-packages/keras/backend/tensorflow_backend.py:3445: calling d
ropout (from tensorflow.python.ops.nn_ops) with keep_prob is deprecated and w
ill be removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - k
eep_prob`.
```

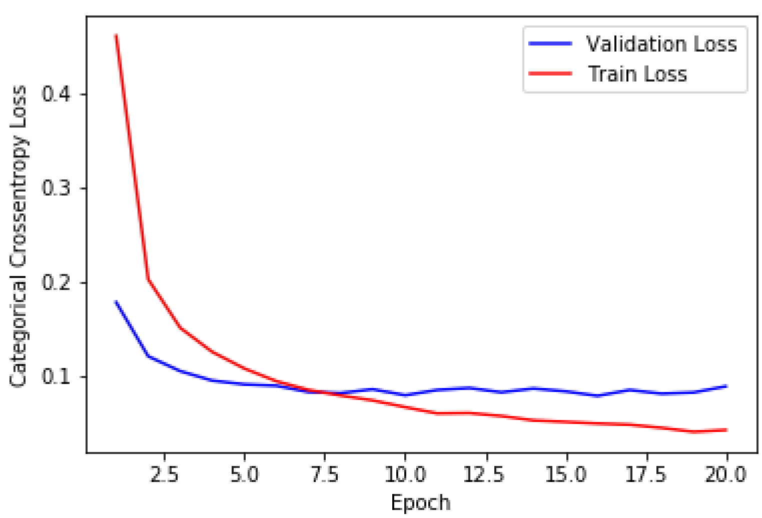
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_17 (Dense)             (None, 720)               565200
_________________________________________________________________
dropout_1 (Dropout)          (None, 720)               0
_________________________________________________________________
dense_18 (Dense)             (None, 200)               144200
_________________________________________________________________
dropout_2 (Dropout)          (None, 200)               0
_________________________________________________________________
dense_19 (Dense)             (None, 10)                2010
=================================================================
Total params: 711,410
Trainable params: 711,410
Non-trainable params: 0
_________________________________________________________________
None

Train on 42000 samples, validate on 18000 samples
Epoch 1/20
42000/42000 [==============================] - 2s 53us/step - loss: 0.4603 -
acc: 0.8564 - val_loss: 0.1784 - val_acc: 0.9467
Epoch 2/20
42000/42000 [==============================] - 2s 38us/step - loss: 0.2027 -
acc: 0.9388 - val_loss: 0.1210 - val_acc: 0.9636
Epoch 3/20
42000/42000 [==============================] - 2s 38us/step - loss: 0.1512 -
acc: 0.9545 - val_loss: 0.1053 - val_acc: 0.9689
Epoch 4/20
42000/42000 [==============================] - 2s 38us/step - loss: 0.1256 -
acc: 0.9629 - val_loss: 0.0953 - val_acc: 0.9715
Epoch 5/20
42000/42000 [==============================] - 2s 38us/step - loss: 0.1081 -
acc: 0.9675 - val_loss: 0.0913 - val_acc: 0.9722
Epoch 6/20
42000/42000 [==============================] - 2s 38us/step - loss: 0.0946 -
acc: 0.9708 - val_loss: 0.0898 - val_acc: 0.9736
Epoch 7/20
42000/42000 [==============================] - 2s 38us/step - loss: 0.0853 -
acc: 0.9739 - val_loss: 0.0831 - val_acc: 0.9765
Epoch 8/20
42000/42000 [==============================] - 2s 38us/step - loss: 0.0795 -
acc: 0.9755 - val_loss: 0.0820 - val_acc: 0.9758
Epoch 9/20
42000/42000 [==============================] - 2s 38us/step - loss: 0.0743 -
acc: 0.9770 - val_loss: 0.0861 - val_acc: 0.9761
Epoch 10/20
42000/42000 [==============================] - 2s 37us/step - loss: 0.0671 -
acc: 0.9791 - val_loss: 0.0797 - val_acc: 0.9770
Epoch 11/20
42000/42000 [==============================] - 2s 38us/step - loss: 0.0605 -
acc: 0.9804 - val_loss: 0.0853 - val_acc: 0.9775
Epoch 12/20
42000/42000 [==============================] - 2s 38us/step - loss: 0.0608 -
acc: 0.9811 - val_loss: 0.0874 - val_acc: 0.9771
Epoch 13/20
```

```
42000/42000 [==============================] - 2s 38us/step - loss: 0.0577 -
acc: 0.9816 - val_loss: 0.0828 - val_acc: 0.9778
Epoch 14/20
42000/42000 [==============================] - 2s 38us/step - loss: 0.0530 -
acc: 0.9832 - val_loss: 0.0869 - val_acc: 0.9776
Epoch 15/20
42000/42000 [==============================] - 2s 38us/step - loss: 0.0515 -
acc: 0.9836 - val_loss: 0.0838 - val_acc: 0.9784
Epoch 16/20
42000/42000 [==============================] - 2s 38us/step - loss: 0.0497 -
acc: 0.9837 - val_loss: 0.0791 - val_acc: 0.9793
Epoch 17/20
42000/42000 [==============================] - 2s 37us/step - loss: 0.0485 -
acc: 0.9844 - val_loss: 0.0853 - val_acc: 0.9783
Epoch 18/20
42000/42000 [==============================] - 2s 39us/step - loss: 0.0451 -
acc: 0.9857 - val_loss: 0.0814 - val_acc: 0.9798
Epoch 19/20
42000/42000 [==============================] - 2s 38us/step - loss: 0.0409 -
acc: 0.9865 - val_loss: 0.0827 - val_acc: 0.9794
Epoch 20/20
42000/42000 [==============================] - 2s 38us/step - loss: 0.0428 -
acc: 0.9864 - val_loss: 0.0891 - val_acc: 0.9784
```

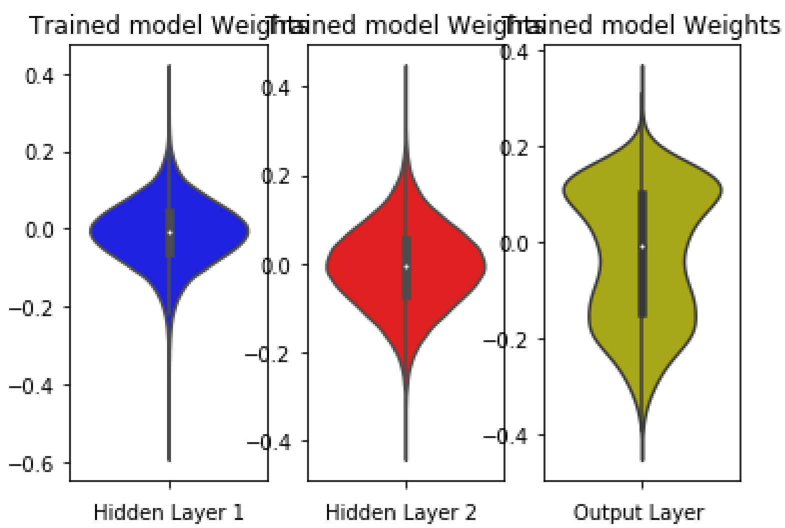
In [43]:

```
score = model.evaluate(X_test, Y_test, verbose=0)
print(f'Test Score: {score[0]}')
print(f'Test Accuracy: {score[1]}\n')


x = list(range(1, nb_epoch+1))
vy = history.history['val_loss']
ty = history.history['loss']
plot_loss(x, vy, ty)
```

```
Test Score: 0.07207091313775582
Test Accuracy: 0.9807
```



In [44]:
```
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()
```



4. MLP + Relu Activation + Adam Optimizer + Batch Normalization + Dropout (H1: 720 , H2: 200)

In [45]:
```
%%time
def Build_NN_2(input_dim, output_dim=10):
  model = Sequential()
  model.add(Dense(720, activation='relu', kernel_initializer='he_normal', input_shape=(input_dim,)))
  model.add(BatchNormalization())
  model.add(Dropout(0.5))

  model.add(Dense(200, activation='relu', kernel_initializer='he_normal'))
  model.add(BatchNormalization())
  model.add(Dropout(0.5))

  model.add(Dense(output_dim, activation='softmax'))

  model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['accuracy'])
  print(model.summary())

  return model
```

```
CPU times: user 10 µs, sys: 2 µs, total: 12 µs
Wall time: 16.5 µs
```

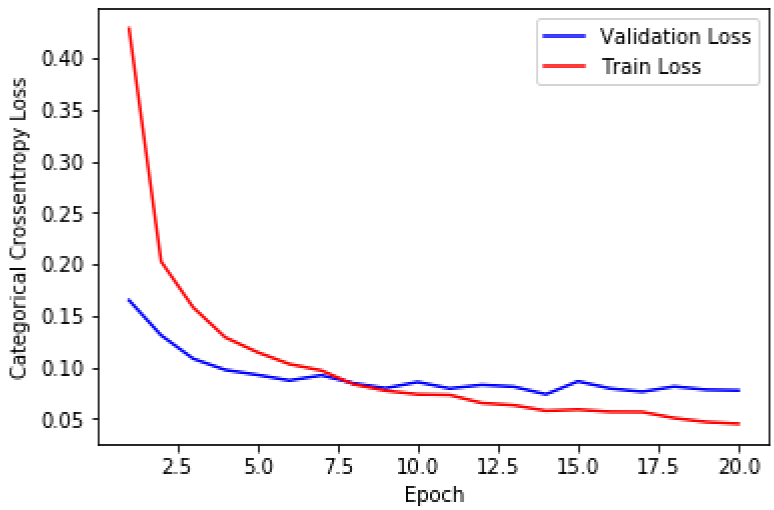
In [46]:
```
model=Build_NN_2(input_dim)
print()
history = model.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch,
validation_split=0.3, verbose=1)
print()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_20 (Dense)             (None, 720)               565200    
_________________________________________________________________
batch_normalization_3 (Batch (None, 720)               2880      
_________________________________________________________________
dropout_3 (Dropout)          (None, 720)               0         
_________________________________________________________________
dense_21 (Dense)             (None, 200)               144200    
_________________________________________________________________
batch_normalization_4 (Batch (None, 200)               800       
_________________________________________________________________
dropout_4 (Dropout)          (None, 200)               0         
_________________________________________________________________
dense_22 (Dense)             (None, 10)                2010      
=================================================================
Total params: 715,090
Trainable params: 713,250
Non-trainable params: 1,840
_________________________________________________________________
None

Train on 42000 samples, validate on 18000 samples
Epoch 1/20
42000/42000 [==============================] - 3s 79us/step - loss: 0.4287 -
acc: 0.8700 - val_loss: 0.1650 - val_acc: 0.9502
Epoch 2/20
42000/42000 [==============================] - 2s 56us/step - loss: 0.2026 -
acc: 0.9368 - val_loss: 0.1310 - val_acc: 0.9621
Epoch 3/20
42000/42000 [==============================] - 2s 57us/step - loss: 0.1581 -
acc: 0.9517 - val_loss: 0.1083 - val_acc: 0.9676
Epoch 4/20
42000/42000 [==============================] - 2s 56us/step - loss: 0.1290 -
acc: 0.9601 - val_loss: 0.0974 - val_acc: 0.9701
Epoch 5/20
42000/42000 [==============================] - 2s 56us/step - loss: 0.1146 -
acc: 0.9644 - val_loss: 0.0926 - val_acc: 0.9729
Epoch 6/20
42000/42000 [==============================] - 2s 57us/step - loss: 0.1031 -
acc: 0.9670 - val_loss: 0.0872 - val_acc: 0.9742
Epoch 7/20
42000/42000 [==============================] - 2s 56us/step - loss: 0.0968 -
acc: 0.9692 - val_loss: 0.0922 - val_acc: 0.9726
Epoch 8/20
42000/42000 [==============================] - 2s 57us/step - loss: 0.0834 -
acc: 0.9737 - val_loss: 0.0845 - val_acc: 0.9750
Epoch 9/20
42000/42000 [==============================] - 2s 57us/step - loss: 0.0772 -
acc: 0.9759 - val_loss: 0.0797 - val_acc: 0.9771
Epoch 10/20
42000/42000 [==============================] - 2s 58us/step - loss: 0.0737 -
acc: 0.9767 - val_loss: 0.0857 - val_acc: 0.9752
Epoch 11/20
42000/42000 [==============================] - 2s 58us/step - loss: 0.0731 -
acc: 0.9771 - val_loss: 0.0794 - val_acc: 0.9771
```

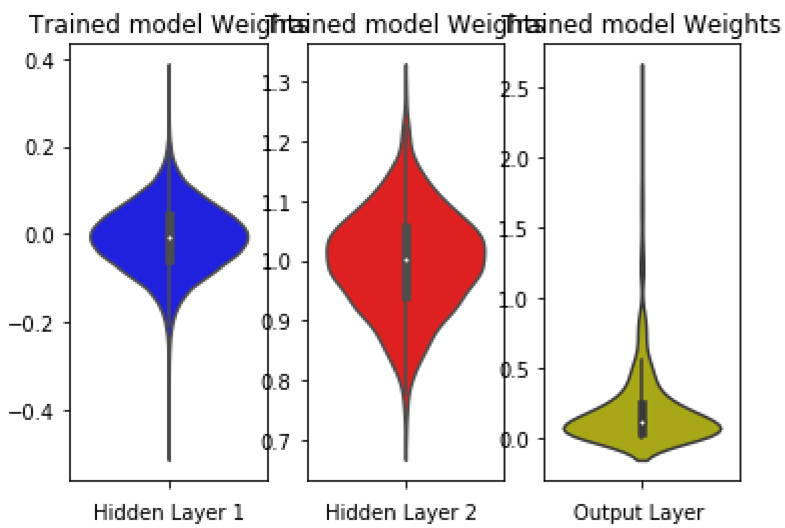
```
Epoch 12/20
42000/42000 [==============================] - 2s 57us/step - loss: 0.0653 -
acc: 0.9791 - val_loss: 0.0829 - val_acc: 0.9775
Epoch 13/20
42000/42000 [==============================] - 2s 56us/step - loss: 0.0631 -
acc: 0.9795 - val_loss: 0.0811 - val_acc: 0.9769
Epoch 14/20
42000/42000 [==============================] - 2s 56us/step - loss: 0.0577 -
acc: 0.9811 - val_loss: 0.0737 - val_acc: 0.9796
Epoch 15/20
42000/42000 [==============================] - 2s 56us/step - loss: 0.0589 -
acc: 0.9807 - val_loss: 0.0864 - val_acc: 0.9760
Epoch 16/20
42000/42000 [==============================] - 2s 57us/step - loss: 0.0568 -
acc: 0.9815 - val_loss: 0.0794 - val_acc: 0.9781
Epoch 17/20
42000/42000 [==============================] - 2s 56us/step - loss: 0.0566 -
acc: 0.9820 - val_loss: 0.0762 - val_acc: 0.9794
Epoch 18/20
42000/42000 [==============================] - 2s 56us/step - loss: 0.0507 -
acc: 0.9829 - val_loss: 0.0813 - val_acc: 0.9772
Epoch 19/20
42000/42000 [==============================] - 2s 56us/step - loss: 0.0469 -
acc: 0.9842 - val_loss: 0.0782 - val_acc: 0.9790
Epoch 20/20
42000/42000 [==============================] - 2s 57us/step - loss: 0.0451 -
acc: 0.9853 - val_loss: 0.0776 - val_acc: 0.9796
```

In [47]:
```
score = model.evaluate(X_test, Y_test, verbose=0)
print(f'Test Score: {score[0]}')
print(f'Test Accuracy: {score[1]}\n')


x = list(range(1, nb_epoch+1))
vy = history.history['val_loss']
ty = history.history['loss']
plot_loss(x, vy, ty)
```

```
Test Score: 0.06645472465842323
Test Accuracy: 0.9822
```



In [48]:
```
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()
```



Multi-Layer Perceptron With 3 hidden layer

1. MLP + Relu Activation + Adam Optimizer (H1: 700 , H2 : 360 , H3: 180)

In [49]:
```
%%time
def Build_NN_3(input_dim, output_dim=10):
  model = Sequential()

  model.add(Dense(700, activation='relu', kernel_initializer='he_normal', inpu
t_shape=(input_dim,)))

  model.add(Dense(360, activation='relu', kernel_initializer='he_normal'))

  model.add(Dense(180, activation='relu', kernel_initializer='he_normal'))


  model.add(Dense(output_dim, activation='softmax'))

  model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['a
ccuracy'])
  print(model.summary())

  return model
```

```
CPU times: user 5 µs, sys: 0 ns, total: 5 µs
Wall time: 9.54 µs
```

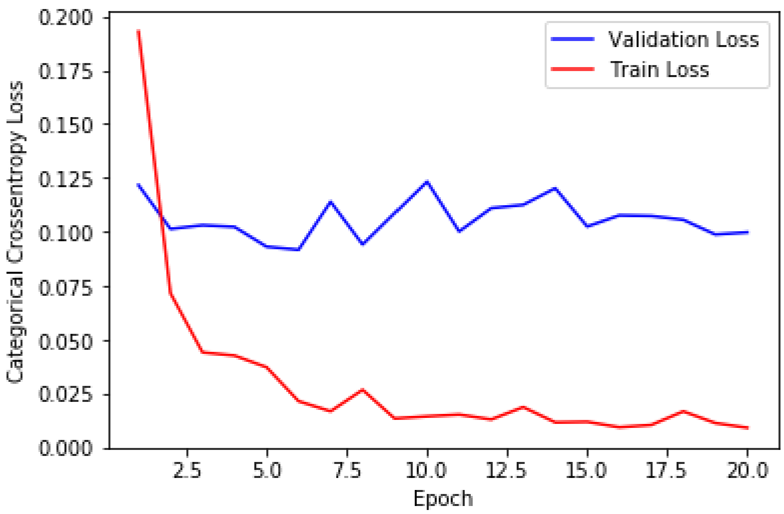
In [57]:
```
model=Build_NN_3(input_dim)
print()
history = model.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch,
validation_split=0.3, verbose=1)
print()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_29 (Dense)             (None, 700)               549500    
_________________________________________________________________
batch_normalization_9 (Batch (None, 700)               2800      
_________________________________________________________________
dense_30 (Dense)             (None, 360)               252360    
_________________________________________________________________
batch_normalization_10 (Batc (None, 360)               1440      
_________________________________________________________________
dense_31 (Dense)             (None, 180)               64980     
_________________________________________________________________
batch_normalization_11 (Batc (None, 180)               720       
_________________________________________________________________
dense_32 (Dense)             (None, 10)                1810      
=================================================================
Total params: 873,610
Trainable params: 871,130
Non-trainable params: 2,480
_________________________________________________________________
None

Train on 42000 samples, validate on 18000 samples
Epoch 1/20
42000/42000 [==============================] - 4s 101us/step - loss: 0.1929 - acc: 0.9418 - val_loss: 0.1217 - val_acc: 0.9631
Epoch 2/20
42000/42000 [==============================] - 3s 67us/step - loss: 0.0718 - acc: 0.9776 - val_loss: 0.1013 - val_acc: 0.9697
Epoch 3/20
42000/42000 [==============================] - 3s 67us/step - loss: 0.0444 - acc: 0.9862 - val_loss: 0.1031 - val_acc: 0.9703
Epoch 4/20
42000/42000 [==============================] - 3s 68us/step - loss: 0.0429 - acc: 0.9857 - val_loss: 0.1023 - val_acc: 0.9693
Epoch 5/20
42000/42000 [==============================] - 3s 68us/step - loss: 0.0375 - acc: 0.9877 - val_loss: 0.0931 - val_acc: 0.9747
Epoch 6/20
42000/42000 [==============================] - 3s 68us/step - loss: 0.0217 - acc: 0.9929 - val_loss: 0.0917 - val_acc: 0.9756
Epoch 7/20
42000/42000 [==============================] - 3s 67us/step - loss: 0.0171 - acc: 0.9947 - val_loss: 0.1140 - val_acc: 0.9703
Epoch 8/20
42000/42000 [==============================] - 3s 68us/step - loss: 0.0270 - acc: 0.9909 - val_loss: 0.0941 - val_acc: 0.9746
Epoch 9/20
42000/42000 [==============================] - 3s 67us/step - loss: 0.0137 - acc: 0.9953 - val_loss: 0.1088 - val_acc: 0.9757
Epoch 10/20
42000/42000 [==============================] - 3s 68us/step - loss: 0.0147 - acc: 0.9948 - val_loss: 0.1233 - val_acc: 0.9704
Epoch 11/20
42000/42000 [==============================] - 3s 68us/step - loss: 0.0155 - acc: 0.9948 - val_loss: 0.1002 - val_acc: 0.9758
```

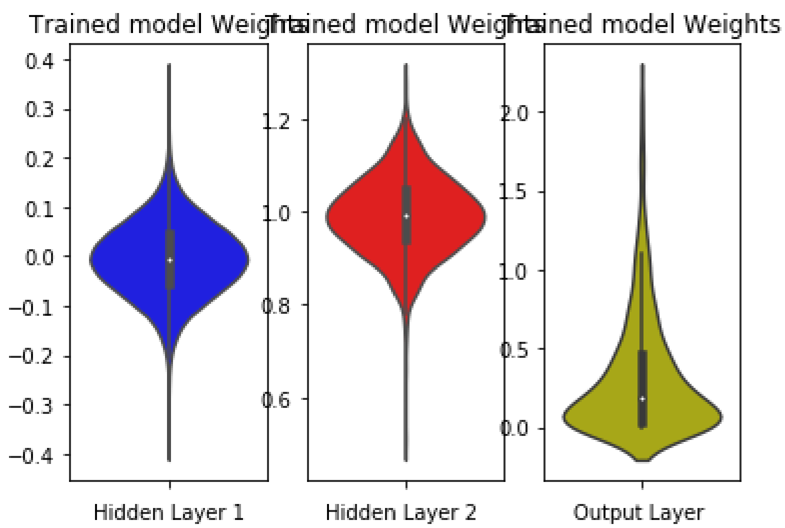
```
Epoch 12/20
42000/42000 [==============================] - 3s 68us/step - loss: 0.0133 -
acc: 0.9956 - val_loss: 0.1110 - val_acc: 0.9739
Epoch 13/20
42000/42000 [==============================] - 3s 68us/step - loss: 0.0190 -
acc: 0.9934 - val_loss: 0.1125 - val_acc: 0.9716
Epoch 14/20
42000/42000 [==============================] - 3s 67us/step - loss: 0.0119 -
acc: 0.9961 - val_loss: 0.1202 - val_acc: 0.9730
Epoch 15/20
42000/42000 [==============================] - 3s 68us/step - loss: 0.0122 -
acc: 0.9959 - val_loss: 0.1025 - val_acc: 0.9761
Epoch 16/20
42000/42000 [==============================] - 3s 68us/step - loss: 0.0097 -
acc: 0.9965 - val_loss: 0.1077 - val_acc: 0.9761
Epoch 17/20
42000/42000 [==============================] - 3s 67us/step - loss: 0.0107 -
acc: 0.9967 - val_loss: 0.1074 - val_acc: 0.9763
Epoch 18/20
42000/42000 [==============================] - 3s 67us/step - loss: 0.0170 -
acc: 0.9942 - val_loss: 0.1056 - val_acc: 0.9750
Epoch 19/20
42000/42000 [==============================] - 3s 68us/step - loss: 0.0116 -
acc: 0.9968 - val_loss: 0.0988 - val_acc: 0.9777
Epoch 20/20
42000/42000 [==============================] - 3s 67us/step - loss: 0.0094 -
acc: 0.9970 - val_loss: 0.0998 - val_acc: 0.9785
```

In [58]:
```
score = model.evaluate(X_test, Y_test, verbose=0)
print(f'Test Score: {score[0]}')
print(f'Test Accuracy: {score[1]}\n')


x = list(range(1, nb_epoch+1))
vy = history.history['val_loss']
ty = history.history['loss']
plot_loss(x, vy, ty)
```

```
Test Score: 0.09005934175993825
Test Accuracy: 0.981
```



In [59]:
```
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()
```



2. MLP + Relu Activation + Adam Optimizer + Batch Normalization (H1: 700 , H2: 360 , H3: 180)

In [60]:
```
%%time
def Build_NN_3(input_dim, output_dim=10):
  model = Sequential()

  model.add(Dense(700, activation='relu', kernel_initializer='he_normal', inpu
t_shape=(input_dim,)))
  model.add(BatchNormalization())

  model.add(Dense(360, activation='relu', kernel_initializer='he_normal'))
  model.add(BatchNormalization())

  model.add(Dense(180, activation='relu', kernel_initializer='he_normal'))
  model.add(BatchNormalization())


  model.add(Dense(output_dim, activation='softmax'))

  model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['a
ccuracy'])
  print(model.summary())

  return model
```

```
CPU times: user 10 µs, sys: 0 ns, total: 10 µs
Wall time: 14.3 µs
```

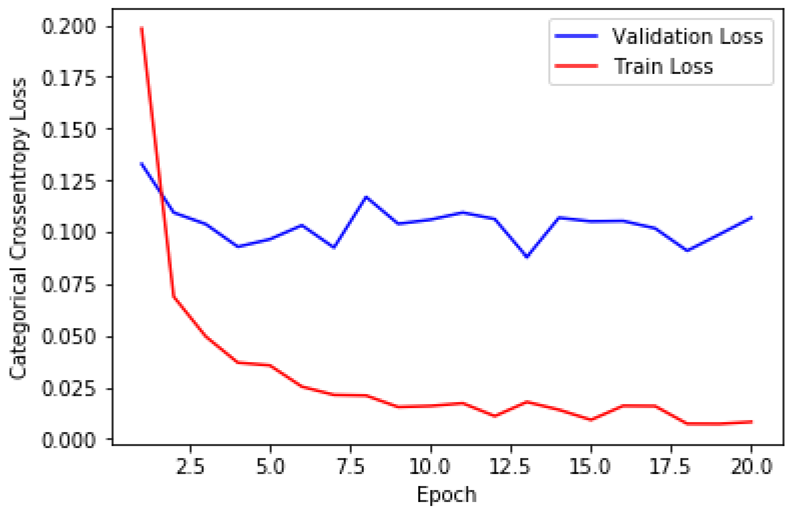
In [61]:
```
model=Build_NN_3(input_dim)
print()
history = model.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch,
validation_split=0.3, verbose=1)
print()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_33 (Dense)             (None, 700)               549500    
_________________________________________________________________
batch_normalization_12 (Batc (None, 700)               2800      
_________________________________________________________________
dense_34 (Dense)             (None, 360)               252360    
_________________________________________________________________
batch_normalization_13 (Batc (None, 360)               1440      
_________________________________________________________________
dense_35 (Dense)             (None, 180)               64980     
_________________________________________________________________
batch_normalization_14 (Batc (None, 180)               720       
_________________________________________________________________
dense_36 (Dense)             (None, 10)                1810      
=================================================================
Total params: 873,610
Trainable params: 871,130
Non-trainable params: 2,480
_________________________________________________________________
None

Train on 42000 samples, validate on 18000 samples
Epoch 1/20
42000/42000 [==============================] - 4s 105us/step - loss: 0.1984 - 
acc: 0.9399 - val_loss: 0.1328 - val_acc: 0.9596
Epoch 2/20
42000/42000 [==============================] - 3s 68us/step - loss: 0.0689 - 
acc: 0.9785 - val_loss: 0.1093 - val_acc: 0.9658
Epoch 3/20
42000/42000 [==============================] - 3s 67us/step - loss: 0.0495 - 
acc: 0.9847 - val_loss: 0.1037 - val_acc: 0.9711
Epoch 4/20
42000/42000 [==============================] - 3s 67us/step - loss: 0.0369 - 
acc: 0.9879 - val_loss: 0.0927 - val_acc: 0.9742
Epoch 5/20
42000/42000 [==============================] - 3s 67us/step - loss: 0.0356 - 
acc: 0.9882 - val_loss: 0.0964 - val_acc: 0.9733
Epoch 6/20
42000/42000 [==============================] - 3s 67us/step - loss: 0.0253 - 
acc: 0.9914 - val_loss: 0.1031 - val_acc: 0.9722
Epoch 7/20
42000/42000 [==============================] - 3s 68us/step - loss: 0.0213 - 
acc: 0.9931 - val_loss: 0.0923 - val_acc: 0.9758
Epoch 8/20
42000/42000 [==============================] - 3s 67us/step - loss: 0.0210 - 
acc: 0.9932 - val_loss: 0.1169 - val_acc: 0.9687
Epoch 9/20
42000/42000 [==============================] - 3s 67us/step - loss: 0.0154 - 
acc: 0.9952 - val_loss: 0.1039 - val_acc: 0.9731
Epoch 10/20
42000/42000 [==============================] - 3s 67us/step - loss: 0.0159 - 
acc: 0.9948 - val_loss: 0.1059 - val_acc: 0.9739
Epoch 11/20
42000/42000 [==============================] - 3s 67us/step - loss: 0.0172 - 
acc: 0.9942 - val_loss: 0.1093 - val_acc: 0.9746
```

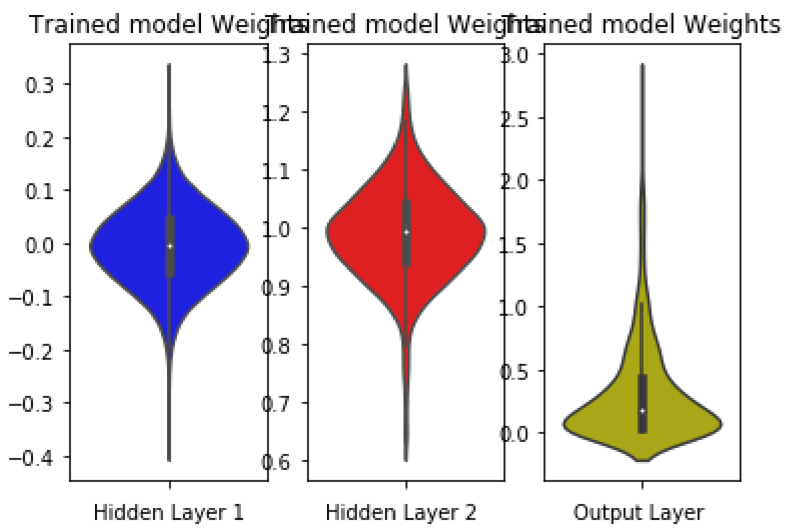
```
Epoch 12/20
42000/42000 [==============================] - 3s 67us/step - loss: 0.0110 -
acc: 0.9964 - val_loss: 0.1062 - val_acc: 0.9739
Epoch 13/20
42000/42000 [==============================] - 3s 68us/step - loss: 0.0179 -
acc: 0.9942 - val_loss: 0.0878 - val_acc: 0.9795
Epoch 14/20
42000/42000 [==============================] - 3s 67us/step - loss: 0.0141 -
acc: 0.9957 - val_loss: 0.1068 - val_acc: 0.9751
Epoch 15/20
42000/42000 [==============================] - 3s 67us/step - loss: 0.0092 -
acc: 0.9971 - val_loss: 0.1050 - val_acc: 0.9762
Epoch 16/20
42000/42000 [==============================] - 3s 67us/step - loss: 0.0160 -
acc: 0.9947 - val_loss: 0.1053 - val_acc: 0.9758
Epoch 17/20
42000/42000 [==============================] - 3s 68us/step - loss: 0.0159 -
acc: 0.9948 - val_loss: 0.1017 - val_acc: 0.9761
Epoch 18/20
42000/42000 [==============================] - 3s 67us/step - loss: 0.0073 -
acc: 0.9978 - val_loss: 0.0908 - val_acc: 0.9790
Epoch 19/20
42000/42000 [==============================] - 3s 67us/step - loss: 0.0072 -
acc: 0.9979 - val_loss: 0.0987 - val_acc: 0.9777
Epoch 20/20
42000/42000 [==============================] - 3s 68us/step - loss: 0.0082 -
acc: 0.9970 - val_loss: 0.1068 - val_acc: 0.9775
```

In [62]:
```
score = model.evaluate(X_test, Y_test, verbose=0)
print(f'Test Score: {score[0]}')
print(f'Test Accuracy: {score[1]}\n')


x = list(range(1, nb_epoch+1))
vy = history.history['val_loss']
ty = history.history['loss']
plot_loss(x, vy, ty)
```

```
Test Score: 0.08819649735184722
Test Accuracy: 0.9787
```



In [63]:
```
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()
```



3. MLP + Relu Activation + Adam Optimizer + Dropout (H1: 700 , H2: 360 , H3: 180)

In [64]:
```
%time
def Build_NN_3(input_dim, output_dim=10):
  model = Sequential()

  model.add(Dense(700, activation='relu', kernel_initializer='he_normal', input_shape=(input_dim,)))

  model.add(Dense(360, activation='relu', kernel_initializer='he_normal'))

  model.add(Dense(180, activation='relu', kernel_initializer='he_normal'))


  model.add(Dense(output_dim, activation='softmax'))

  model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['accuracy'])
  print(model.summary())

  return model
```

```
CPU times: user 3 µs, sys: 1e+03 ns, total: 4 µs
Wall time: 7.15 µs
```

In [65]:
```
model=Build_NN_3(input_dim)
print()
history = model.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch,
validation_split=0.3, verbose=1)
print()
```

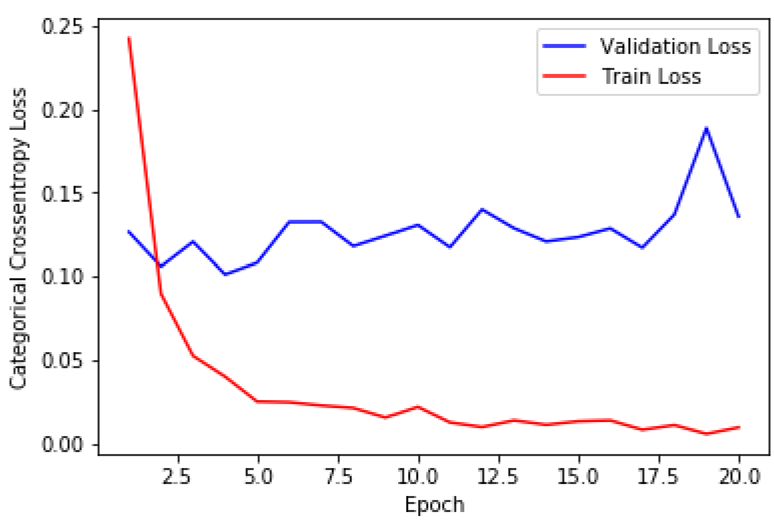
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_37 (Dense)             (None, 700)               549500    
_________________________________________________________________
dense_38 (Dense)             (None, 360)               252360    
_________________________________________________________________
dense_39 (Dense)             (None, 180)               64980     
_________________________________________________________________
dense_40 (Dense)             (None, 10)                1810      
=================================================================
Total params: 868,650
Trainable params: 868,650
Non-trainable params: 0
_________________________________________________________________
None

Train on 42000 samples, validate on 18000 samples
Epoch 1/20
42000/42000 [==============================] - 3s 72us/step - loss: 0.2423 - acc: 0.9256 - val_loss: 0.1264 - val_acc: 0.9612
Epoch 2/20
42000/42000 [==============================] - 2s 39us/step - loss: 0.0896 - acc: 0.9723 - val_loss: 0.1058 - val_acc: 0.9699
Epoch 3/20
42000/42000 [==============================] - 2s 39us/step - loss: 0.0524 - acc: 0.9838 - val_loss: 0.1206 - val_acc: 0.9635
Epoch 4/20
42000/42000 [==============================] - 2s 39us/step - loss: 0.0399 - acc: 0.9864 - val_loss: 0.1011 - val_acc: 0.9712
Epoch 5/20
42000/42000 [==============================] - 2s 39us/step - loss: 0.0250 - acc: 0.9921 - val_loss: 0.1080 - val_acc: 0.9724
Epoch 6/20
42000/42000 [==============================] - 2s 39us/step - loss: 0.0246 - acc: 0.9912 - val_loss: 0.1324 - val_acc: 0.9676
Epoch 7/20
42000/42000 [==============================] - 2s 39us/step - loss: 0.0226 - acc: 0.9927 - val_loss: 0.1324 - val_acc: 0.9698
Epoch 8/20
42000/42000 [==============================] - 2s 39us/step - loss: 0.0211 - acc: 0.9930 - val_loss: 0.1179 - val_acc: 0.9712
Epoch 9/20
42000/42000 [==============================] - 2s 39us/step - loss: 0.0154 - acc: 0.9949 - val_loss: 0.1241 - val_acc: 0.9739
Epoch 10/20
42000/42000 [==============================] - 2s 40us/step - loss: 0.0218 - acc: 0.9925 - val_loss: 0.1305 - val_acc: 0.9701
Epoch 11/20
42000/42000 [==============================] - 2s 39us/step - loss: 0.0125 - acc: 0.9958 - val_loss: 0.1172 - val_acc: 0.9747
Epoch 12/20
42000/42000 [==============================] - 2s 39us/step - loss: 0.0097 - acc: 0.9965 - val_loss: 0.1398 - val_acc: 0.9723
Epoch 13/20
42000/42000 [==============================] - 2s 39us/step - loss: 0.0137 - acc: 0.9957 - val_loss: 0.1286 - val_acc: 0.9738
```

```
Epoch 14/20
42000/42000 [==============================] - 2s 39us/step - loss: 0.0111 -
acc: 0.9963 - val_loss: 0.1207 - val_acc: 0.9779
Epoch 15/20
42000/42000 [==============================] - 2s 40us/step - loss: 0.0132 -
acc: 0.9959 - val_loss: 0.1233 - val_acc: 0.9749
Epoch 16/20
42000/42000 [==============================] - 2s 39us/step - loss: 0.0137 -
acc: 0.9964 - val_loss: 0.1285 - val_acc: 0.9734
Epoch 17/20
42000/42000 [==============================] - 2s 40us/step - loss: 0.0081 -
acc: 0.9973 - val_loss: 0.1169 - val_acc: 0.9784
Epoch 18/20
42000/42000 [==============================] - 2s 39us/step - loss: 0.0109 -
acc: 0.9965 - val_loss: 0.1368 - val_acc: 0.9756
Epoch 19/20
42000/42000 [==============================] - 2s 40us/step - loss: 0.0056 -
acc: 0.9985 - val_loss: 0.1886 - val_acc: 0.9686
Epoch 20/20
42000/42000 [==============================] - 2s 39us/step - loss: 0.0095 -
acc: 0.9972 - val_loss: 0.1356 - val_acc: 0.9761
```

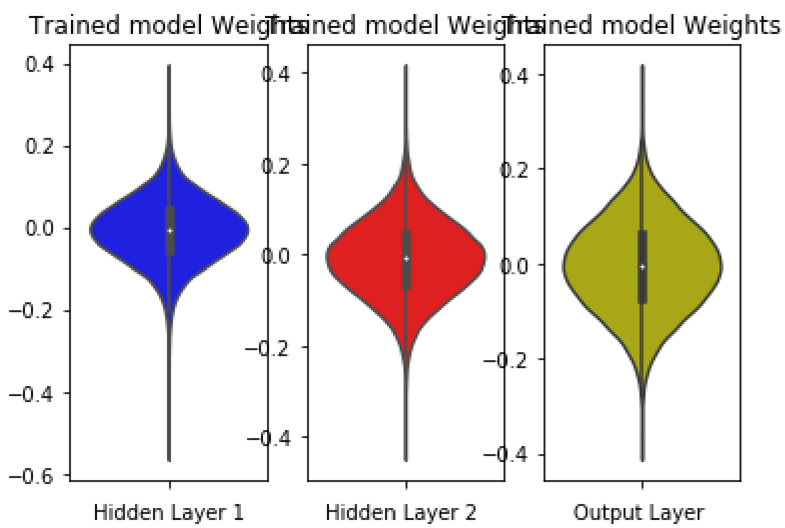
In [66]:

```
score = model.evaluate(X_test, Y_test, verbose=0)
print(f'Test Score: {score[0]}')
print(f'Test Accuracy: {score[1]}\n')


x = list(range(1, nb_epoch+1))
vy = history.history['val_loss']
ty = history.history['loss']
plot_loss(x, vy, ty)
```

```
Test Score: 0.11028124448086878
Test Accuracy: 0.9769
```



In [67]:
```
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()
```



4. MLP + Relu Activation + Adam Optimizer + Batch Normalization + Dropout (H1: 700 , H2: 360 , H3: 180)

In [0]:
```
def Build_NN_3(input_dim, output_dim=10):
  model = Sequential()
  model.add(Dense(720, activation='relu', kernel_initializer='he_normal', inpu
t_shape=(input_dim,)))
  model.add(BatchNormalization())
  model.add(Dropout(0.5))

  model.add(Dense(200, activation='relu', kernel_initializer='he_normal'))
  model.add(BatchNormalization())
  model.add(Dropout(0.5))

  model.add(Dense(output_dim, activation='softmax'))

  model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['a
ccuracy'])
  print(model.summary())

  return model
```

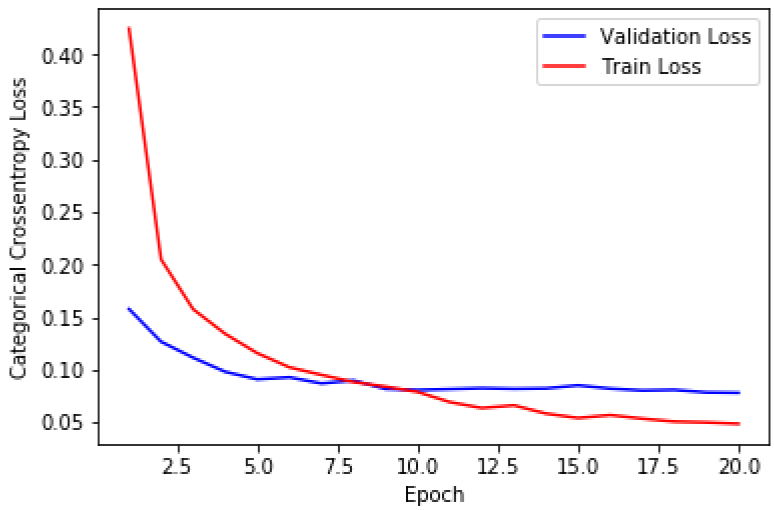
In [69]:
```
model=Build_NN_3(input_dim)
print()
history = model.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch,
validation_split=0.3, verbose=1)
print()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_41 (Dense)             (None, 720)               565200    
_________________________________________________________________
batch_normalization_15 (Batc (None, 720)               2880      
_________________________________________________________________
dropout_9 (Dropout)          (None, 720)               0         
_________________________________________________________________
dense_42 (Dense)             (None, 200)               144200    
_________________________________________________________________
batch_normalization_16 (Batc (None, 200)               800       
_________________________________________________________________
dropout_10 (Dropout)         (None, 200)               0         
_________________________________________________________________
dense_43 (Dense)             (None, 10)                2010      
=================================================================
Total params: 715,090
Trainable params: 713,250
Non-trainable params: 1,840
_________________________________________________________________
None

Train on 42000 samples, validate on 18000 samples
Epoch 1/20
42000/42000 [==============================] - 4s 96us/step - loss: 0.4246 - 
acc: 0.8723 - val_loss: 0.1577 - val_acc: 0.9528
Epoch 2/20
42000/42000 [==============================] - 3s 60us/step - loss: 0.2050 - 
acc: 0.9397 - val_loss: 0.1266 - val_acc: 0.9621
Epoch 3/20
42000/42000 [==============================] - 3s 60us/step - loss: 0.1575 - 
acc: 0.9512 - val_loss: 0.1114 - val_acc: 0.9671
Epoch 4/20
42000/42000 [==============================] - 3s 60us/step - loss: 0.1341 - 
acc: 0.9597 - val_loss: 0.0980 - val_acc: 0.9701
Epoch 5/20
42000/42000 [==============================] - 3s 61us/step - loss: 0.1157 - 
acc: 0.9634 - val_loss: 0.0908 - val_acc: 0.9726
Epoch 6/20
42000/42000 [==============================] - 3s 60us/step - loss: 0.1023 - 
acc: 0.9679 - val_loss: 0.0927 - val_acc: 0.9742
Epoch 7/20
42000/42000 [==============================] - 2s 58us/step - loss: 0.0950 - 
acc: 0.9703 - val_loss: 0.0869 - val_acc: 0.9761
Epoch 8/20
42000/42000 [==============================] - 2s 58us/step - loss: 0.0882 - 
acc: 0.9718 - val_loss: 0.0897 - val_acc: 0.9743
Epoch 9/20
42000/42000 [==============================] - 2s 58us/step - loss: 0.0839 - 
acc: 0.9744 - val_loss: 0.0816 - val_acc: 0.9764
Epoch 10/20
42000/42000 [==============================] - 2s 58us/step - loss: 0.0788 - 
acc: 0.9747 - val_loss: 0.0806 - val_acc: 0.9767
Epoch 11/20
42000/42000 [==============================] - 2s 59us/step - loss: 0.0692 - 
acc: 0.9781 - val_loss: 0.0815 - val_acc: 0.9769
```

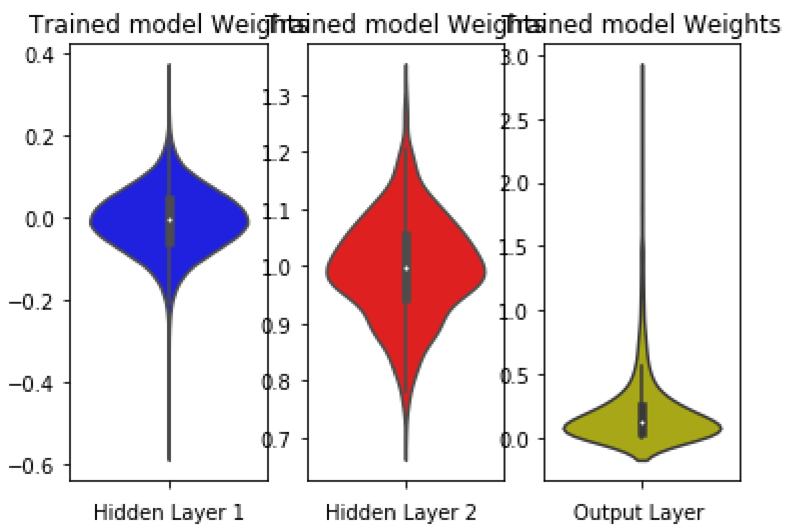
```
Epoch 12/20
42000/42000 [==============================] - 3s 60us/step - loss: 0.0635 - 
acc: 0.9792 - val_loss: 0.0824 - val_acc: 0.9772
Epoch 13/20
42000/42000 [==============================] - 3s 60us/step - loss: 0.0661 - 
acc: 0.9787 - val_loss: 0.0818 - val_acc: 0.9777
Epoch 14/20
42000/42000 [==============================] - 2s 58us/step - loss: 0.0582 - 
acc: 0.9810 - val_loss: 0.0822 - val_acc: 0.9775
Epoch 15/20
42000/42000 [==============================] - 2s 58us/step - loss: 0.0540 - 
acc: 0.9820 - val_loss: 0.0851 - val_acc: 0.9777
Epoch 16/20
42000/42000 [==============================] - 2s 58us/step - loss: 0.0568 - 
acc: 0.9820 - val_loss: 0.0820 - val_acc: 0.9786
Epoch 17/20
42000/42000 [==============================] - 2s 58us/step - loss: 0.0534 - 
acc: 0.9832 - val_loss: 0.0803 - val_acc: 0.9790
Epoch 18/20
42000/42000 [==============================] - 2s 58us/step - loss: 0.0506 - 
acc: 0.9837 - val_loss: 0.0808 - val_acc: 0.9790
Epoch 19/20
42000/42000 [==============================] - 2s 58us/step - loss: 0.0498 - 
acc: 0.9838 - val_loss: 0.0784 - val_acc: 0.9802
Epoch 20/20
42000/42000 [==============================] - 2s 58us/step - loss: 0.0485 - 
acc: 0.9830 - val_loss: 0.0780 - val_acc: 0.9803
```

In [70]:
```
score = model.evaluate(X_test, Y_test, verbose=0)
print(f'Test Score: {score[0]}')
print(f'Test Accuracy: {score[1]}\n')


x = list(range(1, nb_epoch+1))
vy = history.history['val_loss']
ty = history.history['loss']
plot_loss(x, vy, ty)
```

```
Test Score: 0.06508726586559788
Test Accuracy: 0.9819
```



In [71]:
```
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()
```



Multi-Layer Perceptron With 5 hidden layer

1. MLP + Relu Activation + Adam Optimizer (H1: 720 , H2 : 360 , H3: 180 , H4: 90 , H5: 45)

In [72]:
```
%%time
def Build_NN_5(input_dim, output_dim=10):
  model = Sequential()

  model.add(Dense(720, activation='relu', kernel_initializer='he_normal', inpu
t_shape=(input_dim,)))

  model.add(Dense(360, activation='relu', kernel_initializer='he_normal'))

  model.add(Dense(180, activation='relu', kernel_initializer='he_normal'))

  model.add(Dense(90, activation='relu', kernel_initializer='he_normal'))

  model.add(Dense(45, activation='relu', kernel_initializer='he_normal'))

  model.add(Dense(output_dim, activation='softmax'))

  model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['a
ccuracy'])
  print(model.summary())

  return model
```

```
CPU times: user 5 µs, sys: 0 ns, total: 5 µs
Wall time: 10.5 µs
```

In [73]:
```
model=Build_NN_5(input_dim)
print()
history = model.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch,
validation_split=0.3, verbose=1)
print()
```

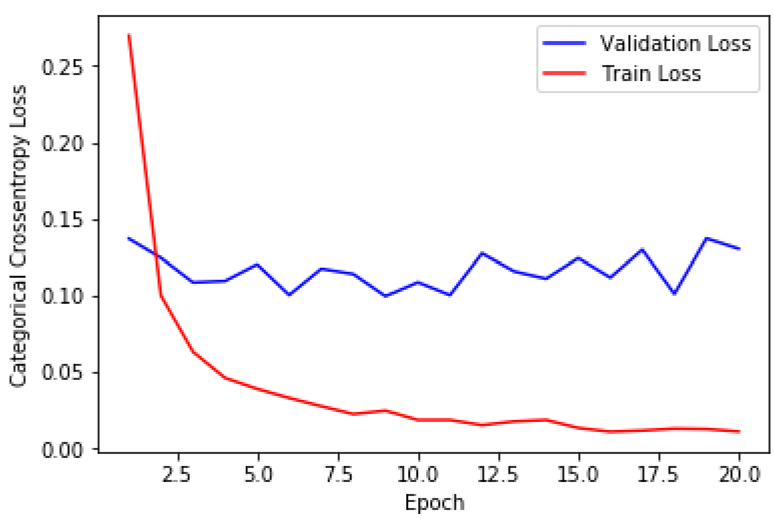
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_44 (Dense)             (None, 720)               565200    
_________________________________________________________________
dense_45 (Dense)             (None, 360)               259560    
_________________________________________________________________
dense_46 (Dense)             (None, 180)               64980     
_________________________________________________________________
dense_47 (Dense)             (None, 90)                16290     
_________________________________________________________________
dense_48 (Dense)             (None, 45)                4095      
_________________________________________________________________
dense_49 (Dense)             (None, 10)                460       
=================================================================
Total params: 910,585
Trainable params: 910,585
Non-trainable params: 0
_________________________________________________________________
None

Train on 42000 samples, validate on 18000 samples
Epoch 1/20
42000/42000 [==============================] - 4s 87us/step - loss: 0.2700 - acc: 0.9175 - val_loss: 0.1370 - val_acc: 0.9585
Epoch 2/20
42000/42000 [==============================] - 2s 47us/step - loss: 0.1000 - acc: 0.9695 - val_loss: 0.1245 - val_acc: 0.9629
Epoch 3/20
42000/42000 [==============================] - 2s 46us/step - loss: 0.0632 - acc: 0.9804 - val_loss: 0.1086 - val_acc: 0.9690
Epoch 4/20
42000/42000 [==============================] - 2s 46us/step - loss: 0.0461 - acc: 0.9850 - val_loss: 0.1095 - val_acc: 0.9692
Epoch 5/20
42000/42000 [==============================] - 2s 45us/step - loss: 0.0389 - acc: 0.9879 - val_loss: 0.1203 - val_acc: 0.9688
Epoch 6/20
42000/42000 [==============================] - 2s 45us/step - loss: 0.0330 - acc: 0.9895 - val_loss: 0.1003 - val_acc: 0.9739
Epoch 7/20
42000/42000 [==============================] - 2s 46us/step - loss: 0.0276 - acc: 0.9915 - val_loss: 0.1175 - val_acc: 0.9709
Epoch 8/20
42000/42000 [==============================] - 2s 45us/step - loss: 0.0224 - acc: 0.9923 - val_loss: 0.1141 - val_acc: 0.9722
Epoch 9/20
42000/42000 [==============================] - 2s 45us/step - loss: 0.0247 - acc: 0.9922 - val_loss: 0.0996 - val_acc: 0.9757
Epoch 10/20
42000/42000 [==============================] - 2s 46us/step - loss: 0.0186 - acc: 0.9938 - val_loss: 0.1086 - val_acc: 0.9741
Epoch 11/20
42000/42000 [==============================] - 2s 47us/step - loss: 0.0186 - acc: 0.9942 - val_loss: 0.1003 - val_acc: 0.9785
Epoch 12/20
42000/42000 [==============================] - 2s 45us/step - loss: 0.0152 -
```

```
acc: 0.9951 - val_loss: 0.1275 - val_acc: 0.9745
Epoch 13/20
42000/42000 [==============================] - 2s 45us/step - loss: 0.0176 -
acc: 0.9945 - val_loss: 0.1158 - val_acc: 0.9747
Epoch 14/20
42000/42000 [==============================] - 2s 45us/step - loss: 0.0186 -
acc: 0.9940 - val_loss: 0.1110 - val_acc: 0.9771
Epoch 15/20
42000/42000 [==============================] - 2s 45us/step - loss: 0.0134 -
acc: 0.9959 - val_loss: 0.1244 - val_acc: 0.9718
Epoch 16/20
42000/42000 [==============================] - 2s 46us/step - loss: 0.0109 -
acc: 0.9965 - val_loss: 0.1117 - val_acc: 0.9760
Epoch 17/20
42000/42000 [==============================] - 2s 45us/step - loss: 0.0117 -
acc: 0.9965 - val_loss: 0.1298 - val_acc: 0.9733
Epoch 18/20
42000/42000 [==============================] - 2s 46us/step - loss: 0.0129 -
acc: 0.9960 - val_loss: 0.1011 - val_acc: 0.9780
Epoch 19/20
42000/42000 [==============================] - 2s 46us/step - loss: 0.0126 -
acc: 0.9963 - val_loss: 0.1372 - val_acc: 0.9706
Epoch 20/20
42000/42000 [==============================] - 2s 45us/step - loss: 0.0110 -
acc: 0.9967 - val_loss: 0.1305 - val_acc: 0.9772
```

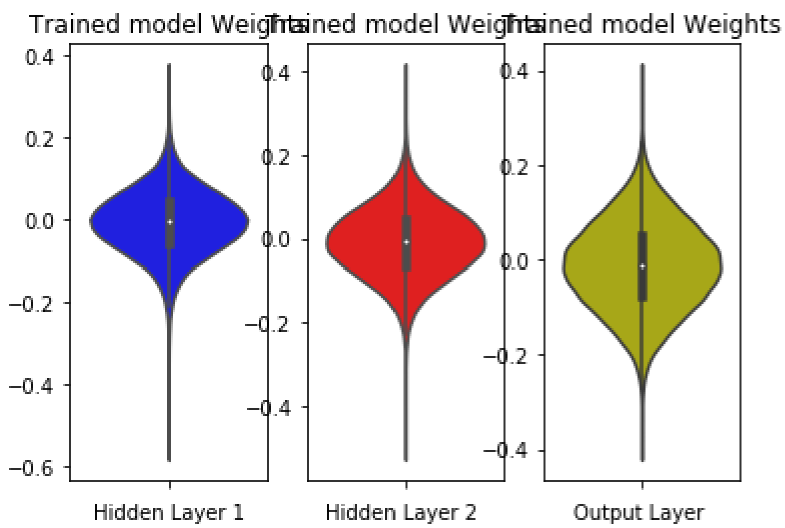
In [74]:

```
score = model.evaluate(X_test, Y_test, verbose=0)
print(f'Test Score: {score[0]}')
print(f'Test Accuracy: {score[1]}\n')


x = list(range(1, nb_epoch+1))
vy = history.history['val_loss']
ty = history.history['loss']
plot_loss(x, vy, ty)
```

```
Test Score: 0.08912124511943212
Test Accuracy: 0.9823
```



In [75]:
```
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()
```



2. MLP + Relu Activation + Adam Optimizer + Batch Normalization (H1: 720 , H2 : 360 , H3: 180 , H4: 90 , H5: 45)

In [76]:
```
%%time
def Build_NN_5(input_dim, output_dim=10):
  model = Sequential()

  model.add(Dense(720, activation='relu', kernel_initializer='he_normal', inpu
t_shape=(input_dim,)))
  model.add(BatchNormalization())

  model.add(Dense(360, activation='relu', kernel_initializer='he_normal'))
  model.add(BatchNormalization())

  model.add(Dense(180, activation='relu', kernel_initializer='he_normal'))
  model.add(BatchNormalization())

  model.add(Dense(90, activation='relu', kernel_initializer='he_normal'))
  model.add(BatchNormalization())

  model.add(Dense(45, activation='relu', kernel_initializer='he_normal'))
  model.add(BatchNormalization())

  model.add(Dense(output_dim, activation='softmax'))

  model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['a
ccuracy'])
  print(model.summary())

  return model
```

```
CPU times: user 0 ns, sys: 9 µs, total: 9 µs
Wall time: 13.1 µs
```

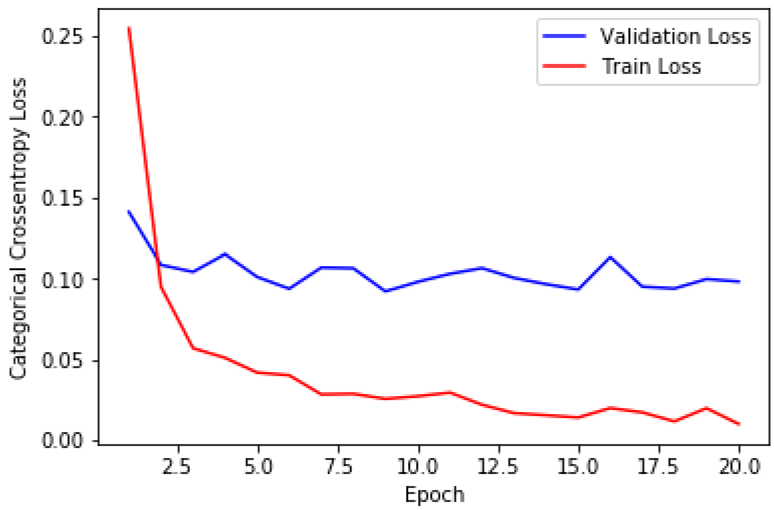
In [77]:
```
model=Build_NN_5(input_dim)
print()
history = model.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch,
validation_split=0.3, verbose=1)
print()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_50 (Dense)             (None, 720)               565200    
_________________________________________________________________
batch_normalization_17 (Batc (None, 720)               2880      
_________________________________________________________________
dense_51 (Dense)             (None, 360)               259560    
_________________________________________________________________
batch_normalization_18 (Batc (None, 360)               1440      
_________________________________________________________________
dense_52 (Dense)             (None, 180)               64980     
_________________________________________________________________
batch_normalization_19 (Batc (None, 180)               720       
_________________________________________________________________
dense_53 (Dense)             (None, 90)                16290     
_________________________________________________________________
batch_normalization_20 (Batc (None, 90)                360       
_________________________________________________________________
dense_54 (Dense)             (None, 45)                4095      
_________________________________________________________________
batch_normalization_21 (Batc (None, 45)                180       
_________________________________________________________________
dense_55 (Dense)             (None, 10)                460       
=================================================================
Total params: 916,165
Trainable params: 913,375
Non-trainable params: 2,790
_________________________________________________________________
None

Train on 42000 samples, validate on 18000 samples
Epoch 1/20
42000/42000 [==============================] - 6s 154us/step - loss: 0.2544 - 
acc: 0.9275 - val_loss: 0.1409 - val_acc: 0.9579
Epoch 2/20
42000/42000 [==============================] - 4s 93us/step - loss: 0.0949 - 
acc: 0.9723 - val_loss: 0.1084 - val_acc: 0.9677
Epoch 3/20
42000/42000 [==============================] - 4s 94us/step - loss: 0.0569 - 
acc: 0.9834 - val_loss: 0.1041 - val_acc: 0.9702
Epoch 4/20
42000/42000 [==============================] - 4s 95us/step - loss: 0.0509 - 
acc: 0.9838 - val_loss: 0.1150 - val_acc: 0.9663
Epoch 5/20
42000/42000 [==============================] - 4s 94us/step - loss: 0.0418 - 
acc: 0.9865 - val_loss: 0.1010 - val_acc: 0.9714
Epoch 6/20
42000/42000 [==============================] - 4s 94us/step - loss: 0.0401 - 
acc: 0.9879 - val_loss: 0.0937 - val_acc: 0.9753
Epoch 7/20
42000/42000 [==============================] - 4s 96us/step - loss: 0.0284 - 
acc: 0.9905 - val_loss: 0.1068 - val_acc: 0.9724
Epoch 8/20
42000/42000 [==============================] - 4s 95us/step - loss: 0.0287 - 
acc: 0.9906 - val_loss: 0.1064 - val_acc: 0.9718
Epoch 9/20
```

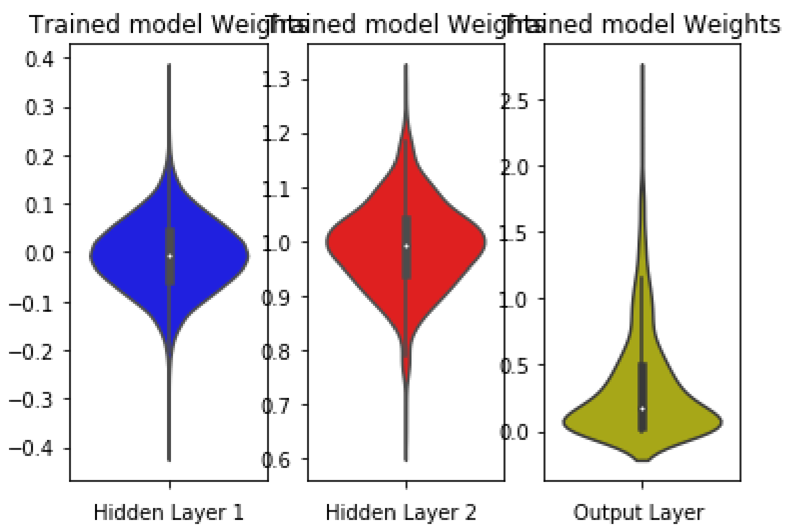
```
42000/42000 [==============================] - 4s 93us/step - loss: 0.0256 -
acc: 0.9918 - val_loss: 0.0921 - val_acc: 0.9753
Epoch 10/20
42000/42000 [==============================] - 4s 94us/step - loss: 0.0272 -
acc: 0.9915 - val_loss: 0.0980 - val_acc: 0.9744
Epoch 11/20
42000/42000 [==============================] - 4s 93us/step - loss: 0.0294 -
acc: 0.9900 - val_loss: 0.1029 - val_acc: 0.9743
Epoch 12/20
42000/42000 [==============================] - 4s 93us/step - loss: 0.0219 -
acc: 0.9927 - val_loss: 0.1064 - val_acc: 0.9718
Epoch 13/20
42000/42000 [==============================] - 4s 93us/step - loss: 0.0167 -
acc: 0.9944 - val_loss: 0.1004 - val_acc: 0.9759
Epoch 14/20
42000/42000 [==============================] - 4s 94us/step - loss: 0.0154 -
acc: 0.9951 - val_loss: 0.0965 - val_acc: 0.9766
Epoch 15/20
42000/42000 [==============================] - 4s 93us/step - loss: 0.0140 -
acc: 0.9955 - val_loss: 0.0932 - val_acc: 0.9782
Epoch 16/20
42000/42000 [==============================] - 4s 93us/step - loss: 0.0199 -
acc: 0.9934 - val_loss: 0.1133 - val_acc: 0.9726
Epoch 17/20
42000/42000 [==============================] - 4s 93us/step - loss: 0.0172 -
acc: 0.9945 - val_loss: 0.0950 - val_acc: 0.9779
Epoch 18/20
42000/42000 [==============================] - 4s 94us/step - loss: 0.0116 -
acc: 0.9964 - val_loss: 0.0938 - val_acc: 0.9765
Epoch 19/20
42000/42000 [==============================] - 4s 93us/step - loss: 0.0198 -
acc: 0.9937 - val_loss: 0.0997 - val_acc: 0.9764
Epoch 20/20
42000/42000 [==============================] - 4s 93us/step - loss: 0.0101 -
acc: 0.9970 - val_loss: 0.0982 - val_acc: 0.9769
```

In [78]:
```
score = model.evaluate(X_test, Y_test, verbose=0)
print(f'Test Score: {score[0]}')
print(f'Test Accuracy: {score[1]}\n')


x = list(range(1, nb_epoch+1))
vy = history.history['val_loss']
ty = history.history['loss']
plot_loss(x, vy, ty)
```

```
Test Score: 0.09106637627696619
Test Accuracy: 0.9772
```



In [79]:
```
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()
```



3. MLP + Relu Activation + Adam Optimizer + Dropout (H1: 720 , H2: 360 , H3: 180 , H4: 90 , H5: 45)

In [80]:
```
%%time
def Build_NN_5(input_dim, output_dim=10):
  model = Sequential()

  model.add(Dense(720, activation='relu', kernel_initializer='he_normal', inpu
t_shape=(input_dim,)))
  model.add(Dropout(0.5))

  model.add(Dense(360, activation='relu', kernel_initializer='he_normal'))
  model.add(Dropout(0.5))

  model.add(Dense(180, activation='relu', kernel_initializer='he_normal'))
  model.add(Dropout(0.5))

  model.add(Dense(90, activation='relu', kernel_initializer='he_normal'))
  model.add(Dropout(0.5))

  model.add(Dense(45, activation='relu', kernel_initializer='he_normal'))
  model.add(Dropout(0.5))

  model.add(Dense(output_dim, activation='softmax'))

  model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['a
ccuracy'])
  print(model.summary())

  return model
```

```
CPU times: user 6 µs, sys: 0 ns, total: 6 µs
Wall time: 9.78 µs
```

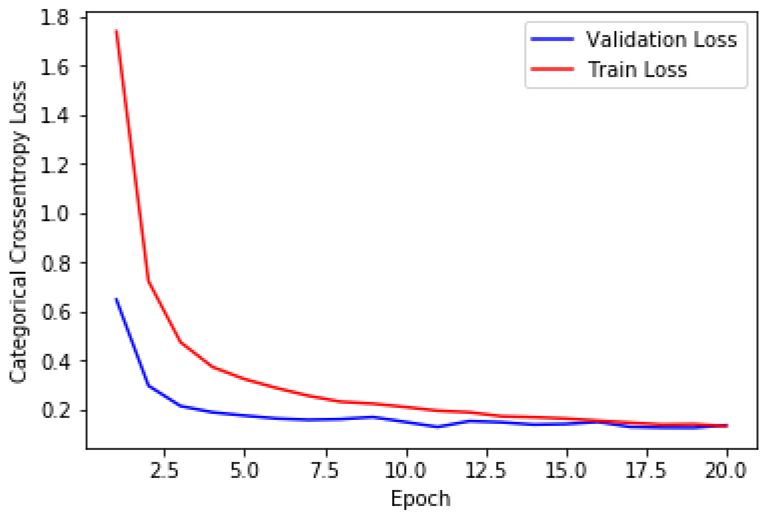
In [81]:
```
model=Build_NN_5(input_dim)
print()
history = model.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch,
validation_split=0.3, verbose=1)
print()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_56 (Dense)             (None, 720)               565200    
_________________________________________________________________
dropout_11 (Dropout)         (None, 720)               0         
_________________________________________________________________
dense_57 (Dense)             (None, 360)               259560    
_________________________________________________________________
dropout_12 (Dropout)         (None, 360)               0         
_________________________________________________________________
dense_58 (Dense)             (None, 180)               64980     
_________________________________________________________________
dropout_13 (Dropout)         (None, 180)               0         
_________________________________________________________________
dense_59 (Dense)             (None, 90)                16290     
_________________________________________________________________
dropout_14 (Dropout)         (None, 90)                0         
_________________________________________________________________
dense_60 (Dense)             (None, 45)                4095      
_________________________________________________________________
dropout_15 (Dropout)         (None, 45)                0         
_________________________________________________________________
dense_61 (Dense)             (None, 10)                460       
=================================================================
Total params: 910,585
Trainable params: 910,585
Non-trainable params: 0
_________________________________________________________________
None

Train on 42000 samples, validate on 18000 samples
Epoch 1/20
42000/42000 [==============================] - 4s 100us/step - loss: 1.7393 - acc: 0.3657 - val_loss: 0.6480 - val_acc: 0.8299
Epoch 2/20
42000/42000 [==============================] - 2s 51us/step - loss: 0.7234 - acc: 0.7688 - val_loss: 0.2972 - val_acc: 0.9284
Epoch 3/20
42000/42000 [==============================] - 2s 51us/step - loss: 0.4746 - acc: 0.8765 - val_loss: 0.2142 - val_acc: 0.9476
Epoch 4/20
42000/42000 [==============================] - 2s 52us/step - loss: 0.3734 - acc: 0.9103 - val_loss: 0.1886 - val_acc: 0.9560
Epoch 5/20
42000/42000 [==============================] - 2s 53us/step - loss: 0.3239 - acc: 0.9246 - val_loss: 0.1751 - val_acc: 0.9594
Epoch 6/20
42000/42000 [==============================] - 2s 52us/step - loss: 0.2874 - acc: 0.9342 - val_loss: 0.1633 - val_acc: 0.9612
Epoch 7/20
42000/42000 [==============================] - 2s 52us/step - loss: 0.2557 - acc: 0.9405 - val_loss: 0.1579 - val_acc: 0.9654
Epoch 8/20
42000/42000 [==============================] - 2s 52us/step - loss: 0.2316 - acc: 0.9480 - val_loss: 0.1607 - val_acc: 0.9656
Epoch 9/20
```

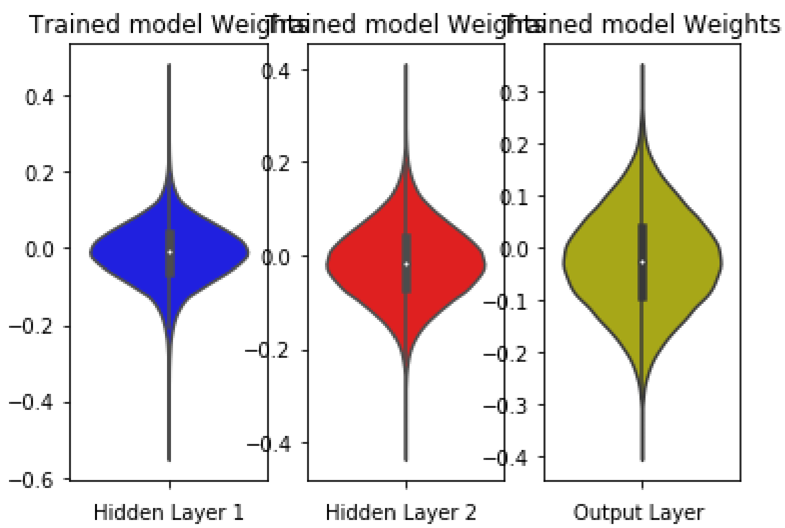
```
42000/42000 [==============================] - 2s 51us/step - loss: 0.2238 -
acc: 0.9493 - val_loss: 0.1692 - val_acc: 0.9669
Epoch 10/20
42000/42000 [==============================] - 2s 52us/step - loss: 0.2101 -
acc: 0.9530 - val_loss: 0.1490 - val_acc: 0.9666
Epoch 11/20
42000/42000 [==============================] - 2s 52us/step - loss: 0.1951 -
acc: 0.9570 - val_loss: 0.1288 - val_acc: 0.9718
Epoch 12/20
42000/42000 [==============================] - 2s 52us/step - loss: 0.1886 -
acc: 0.9581 - val_loss: 0.1527 - val_acc: 0.9694
Epoch 13/20
42000/42000 [==============================] - 2s 51us/step - loss: 0.1723 -
acc: 0.9620 - val_loss: 0.1479 - val_acc: 0.9713
Epoch 14/20
42000/42000 [==============================] - 2s 51us/step - loss: 0.1686 -
acc: 0.9625 - val_loss: 0.1383 - val_acc: 0.9724
Epoch 15/20
42000/42000 [==============================] - 2s 52us/step - loss: 0.1632 -
acc: 0.9637 - val_loss: 0.1408 - val_acc: 0.9738
Epoch 16/20
42000/42000 [==============================] - 2s 51us/step - loss: 0.1544 -
acc: 0.9666 - val_loss: 0.1496 - val_acc: 0.9709
Epoch 17/20
42000/42000 [==============================] - 2s 52us/step - loss: 0.1463 -
acc: 0.9678 - val_loss: 0.1291 - val_acc: 0.9730
Epoch 18/20
42000/42000 [==============================] - 2s 51us/step - loss: 0.1386 -
acc: 0.9688 - val_loss: 0.1261 - val_acc: 0.9745
Epoch 19/20
42000/42000 [==============================] - 2s 52us/step - loss: 0.1402 -
acc: 0.9695 - val_loss: 0.1256 - val_acc: 0.9744
Epoch 20/20
42000/42000 [==============================] - 2s 51us/step - loss: 0.1322 -
acc: 0.9706 - val_loss: 0.1360 - val_acc: 0.9751
```

In [82]:
```
score = model.evaluate(X_test, Y_test, verbose=0)
print(f'Test Score: {score[0]}')
print(f'Test Accuracy: {score[1]}\n')


x = list(range(1, nb_epoch+1))
vy = history.history['val_loss']
ty = history.history['loss']
plot_loss(x, vy, ty)
```

```
Test Score: 0.11893713512013138
Test Accuracy: 0.9776
```



In [83]:
```
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()
```



4. MLP + Relu Activation + Adam Optimizer + Batch Normalization + Dropout (H1: 720 , H2: 360 , H3: 180 , H4: 180 , H5: 45)

In [84]:
```
%%time
def Build_NN_5(input_dim, output_dim=10):
  model = Sequential()

  model.add(Dense(720, activation='relu', kernel_initializer='he_normal', input_shape=(input_dim,)))
  model.add(BatchNormalization())
  model.add(Dropout(0.5))

  model.add(Dense(360, activation='relu', kernel_initializer='he_normal'))
  model.add(BatchNormalization())
  model.add(Dropout(0.5))

  model.add(Dense(180, activation='relu', kernel_initializer='he_normal'))
  model.add(BatchNormalization())
  model.add(Dropout(0.5))

  model.add(Dense(90, activation='relu', kernel_initializer='he_normal'))
  model.add(BatchNormalization())
  model.add(Dropout(0.5))

  model.add(Dense(45, activation='relu', kernel_initializer='he_normal'))
  model.add(BatchNormalization())
  model.add(Dropout(0.5))

  model.add(Dense(output_dim, activation='softmax'))

  model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['accuracy'])
  print(model.summary())

  return model
```

```
CPU times: user 7 µs, sys: 1e+03 ns, total: 8 µs
Wall time: 11.4 µs
```

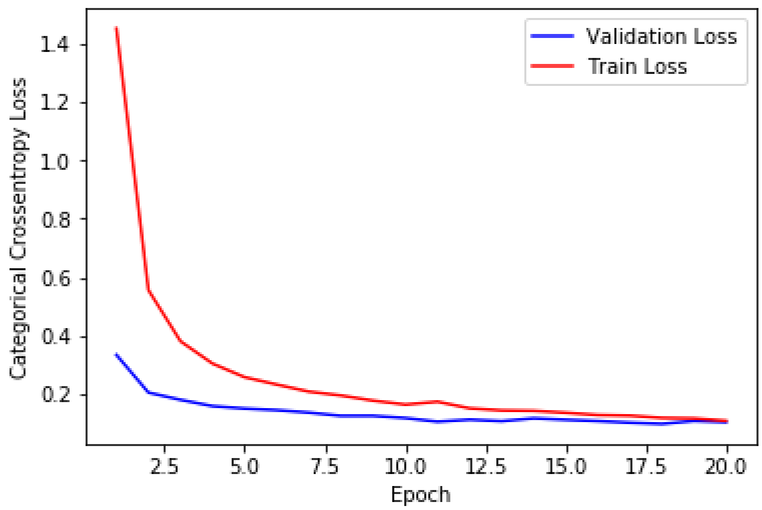
In [85]:
```
model=Build_NN_5(input_dim)
print()
history = model.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch,
validation_split=0.3, verbose=1)
print()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_62 (Dense)             (None, 720)               565200    
_________________________________________________________________
batch_normalization_22 (Batc (None, 720)               2880      
_________________________________________________________________
dropout_16 (Dropout)         (None, 720)               0         
_________________________________________________________________
dense_63 (Dense)             (None, 360)               259560    
_________________________________________________________________
batch_normalization_23 (Batc (None, 360)               1440      
_________________________________________________________________
dropout_17 (Dropout)         (None, 360)               0         
_________________________________________________________________
dense_64 (Dense)             (None, 180)               64980     
_________________________________________________________________
batch_normalization_24 (Batc (None, 180)               720       
_________________________________________________________________
dropout_18 (Dropout)         (None, 180)               0         
_________________________________________________________________
dense_65 (Dense)             (None, 90)                16290     
_________________________________________________________________
batch_normalization_25 (Batc (None, 90)                360       
_________________________________________________________________
dropout_19 (Dropout)         (None, 90)                0         
_________________________________________________________________
dense_66 (Dense)             (None, 45)                4095      
_________________________________________________________________
batch_normalization_26 (Batc (None, 45)                180       
_________________________________________________________________
dropout_20 (Dropout)         (None, 45)                0         
_________________________________________________________________
dense_67 (Dense)             (None, 10)                460       
=================================================================
Total params: 916,165
Trainable params: 913,375
Non-trainable params: 2,790
_________________________________________________________________
None

Train on 42000 samples, validate on 18000 samples
Epoch 1/20
42000/42000 [==============================] - 7s 174us/step - loss: 1.4528 - 
acc: 0.5301 - val_loss: 0.3335 - val_acc: 0.9094
Epoch 2/20
42000/42000 [==============================] - 4s 99us/step - loss: 0.5557 - 
acc: 0.8435 - val_loss: 0.2039 - val_acc: 0.9429
Epoch 3/20
42000/42000 [==============================] - 4s 100us/step - loss: 0.3801 - 
acc: 0.8986 - val_loss: 0.1790 - val_acc: 0.9519
Epoch 4/20
42000/42000 [==============================] - 4s 99us/step - loss: 0.3034 - 
acc: 0.9224 - val_loss: 0.1574 - val_acc: 0.9589
Epoch 5/20
42000/42000 [==============================] - 4s 98us/step - loss: 0.2567 - 
acc: 0.9350 - val_loss: 0.1489 - val_acc: 0.9623
```

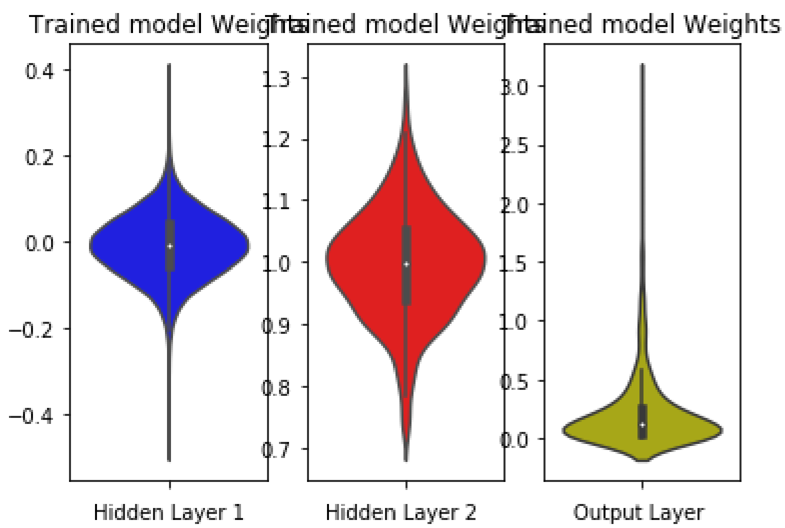
```
Epoch 6/20
42000/42000 [==============================] - 4s 100us/step - loss: 0.2315 -
acc: 0.9423 - val_loss: 0.1436 - val_acc: 0.9649
Epoch 7/20
42000/42000 [==============================] - 4s 101us/step - loss: 0.2072 -
acc: 0.9472 - val_loss: 0.1352 - val_acc: 0.9662
Epoch 8/20
42000/42000 [==============================] - 4s 101us/step - loss: 0.1939 -
acc: 0.9512 - val_loss: 0.1239 - val_acc: 0.9695
Epoch 9/20
42000/42000 [==============================] - 4s 99us/step - loss: 0.1764 -
acc: 0.9554 - val_loss: 0.1240 - val_acc: 0.9707
Epoch 10/20
42000/42000 [==============================] - 4s 99us/step - loss: 0.1626 -
acc: 0.9611 - val_loss: 0.1164 - val_acc: 0.9731
Epoch 11/20
42000/42000 [==============================] - 4s 99us/step - loss: 0.1725 -
acc: 0.9576 - val_loss: 0.1039 - val_acc: 0.9752
Epoch 12/20
42000/42000 [==============================] - 4s 100us/step - loss: 0.1498 -
acc: 0.9632 - val_loss: 0.1109 - val_acc: 0.9731
Epoch 13/20
42000/42000 [==============================] - 4s 100us/step - loss: 0.1426 -
acc: 0.9650 - val_loss: 0.1057 - val_acc: 0.9747
Epoch 14/20
42000/42000 [==============================] - 4s 99us/step - loss: 0.1412 -
acc: 0.9649 - val_loss: 0.1155 - val_acc: 0.9740
Epoch 15/20
42000/42000 [==============================] - 4s 99us/step - loss: 0.1345 -
acc: 0.9671 - val_loss: 0.1104 - val_acc: 0.9739
Epoch 16/20
42000/42000 [==============================] - 4s 100us/step - loss: 0.1268 -
acc: 0.9688 - val_loss: 0.1060 - val_acc: 0.9759
Epoch 17/20
42000/42000 [==============================] - 4s 99us/step - loss: 0.1247 -
acc: 0.9706 - val_loss: 0.1005 - val_acc: 0.9761
Epoch 18/20
42000/42000 [==============================] - 4s 99us/step - loss: 0.1169 -
acc: 0.9716 - val_loss: 0.0963 - val_acc: 0.9778
Epoch 19/20
42000/42000 [==============================] - 4s 100us/step - loss: 0.1152 -
acc: 0.9719 - val_loss: 0.1065 - val_acc: 0.9750
Epoch 20/20
42000/42000 [==============================] - 4s 99us/step - loss: 0.1069 -
acc: 0.9746 - val_loss: 0.1025 - val_acc: 0.9763
```

In [86]:
```
score = model.evaluate(X_test, Y_test, verbose=0)
print(f'Test Score: {score[0]}')
print(f'Test Accuracy: {score[1]}\n')


x = list(range(1, nb_epoch+1))
vy = history.history['val_loss']
ty = history.history['loss']
plot_loss(x, vy, ty)
```

```
Test Score: 0.08941399474844802
Test Accuracy: 0.9788
```



In [87]:
```
w_after = model.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)


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

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

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



Conclusion

1. I have tried 3 diffrent architecture having layers 2 , 3 and 5 respectively.

1. Subsequent increase in depth of the Network (3 and 5 Hidden Layers) shows not much improvement in the Test Accuracy.

1. Also, the Crossover point between Train and Validation Loss is increasing as the Networks depth increases. This is evident from the Loss vs Epoch curves.