Assignment 13 - CNN - MNIST

May 29, 2019

1 Introduction

MNIST ("Modified National Institute of Standards and Technology") is the de facto “Hello World” dataset of computer vision. Since its release in 1999, this classic dataset of handwritten images has served as the basis for benchmarking classification algorithms. As new machine learning techniques emerge, MNIST remains a reliable resource for researchers and learners alike.

Objective is to correctly identify digits from a dataset of tens of thousands of handwritten images

Importing the libraries

```
In [0]: import numpy as np
        import pandas as pd
        from matplotlib import pyplot as plt
        import tensorflow as tf
        import seaborn as sns
```

Now that we have successfully imported the required libraries, let us print out the version of Tensorflow which we will working upon. Also, checking the image format from tensorflow backend.

```
In [2]: print("Version of Tensorflow:", tf.__version__)
        print("Image data format: ", tf.keras.backend.image_data_format())
```

```
Version of Tensorflow: 1.13.1
Image data format:  channels_last
```

As we can see, we are using 1.13 tensorflow version. Also, the image data format config is set to channel_last, which means that the 2D shape would be (row, cols, channels)

Loading the dataset

```
In [3]: mnist_data = tf.keras.datasets.mnist
        (x_train, y_train), (x_test, y_test) = mnist_data.load_data()

        print("INPUT : train shape: ", x_train.shape)
        print("OUTPUT: train shape: ", y_train.shape)
        print("INPUT : test shape: ", x_test.shape)
        print("OUTPUT: test shape: ", y_test.shape)
```

```
INPUT : train shape:  (60000, 28, 28)
OUTPUT: train shape:  (60000,)
INPUT : test shape:  (10000, 28, 28)
OUTPUT: test shape:  (10000,)
```

As we can see, the shape of the train and test feature are 3D i.e for training set, there are 60k rows of 28 x 28 data.

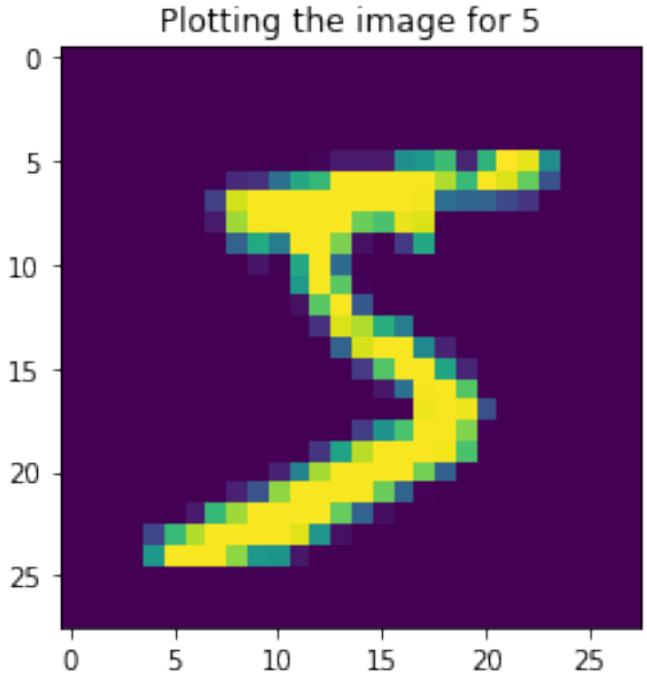
However, as we are using keras with tensorflow backend, we will have to convert the above shape to be of the form (60k * 28 x 28 X 1) i.e 4D tensor

```
In [4]: x_train_new = x_train.reshape(x_train.shape[0], x_train.shape[1], x_train.shape[2], 1)
        x_test_new = x_test.reshape(x_test.shape[0], x_test.shape[1], x_test.shape[2], 1)

        print("After reshaping -> INPUT : train shape: ", x_train_new.shape)
        print("After reshaping -> INPUT : test shape: ", x_train_new.shape)
```

```
After reshaping -> INPUT : train shape:  (60000, 28, 28, 1)
After reshaping -> INPUT : test shape:  (60000, 28, 28, 1)
```

Visualizing the Output

```
In [5]: plt.close()
        plt.title("Plotting the image for "+ str(y_train[0]))
        plt.imshow(x_train[0])
        plt.show()
```



Preprocessing the data
Now, we would be normalizing the data before creating our model.
Typically, we employ 2 methods to normalize our data for MNIST

1. Divide each element by 255
2. Just use keras.utils.normalize

We would be using the built in function for normalizing

```
In [6]: x_train_new = tf.keras.utils.normalize(x_train_new)
        x_test_new = tf.keras.utils.normalize(x_test_new)

        print("After Normalizing -> INPUT : train shape: ", x_train_new.shape)
        print("After Normalizing -> INPUT : test shape: ", x_train_new.shape)
```

```
After Normalizing -> INPUT : train shape:  (60000, 28, 28, 1)
After Normalizing -> INPUT : test shape:  (60000, 28, 28, 1)
```

So we are done with normalizing. Now we would be one hot encoding the output variable so that we are able to feed it into the output layer for softmax classifier activation function.

```
In [7]: y_train = tf.keras.utils.to_categorical(y_train, num_classes=10)
        y_test = tf.keras.utils.to_categorical(y_test, num_classes=10)

        print("After Encoding -> OUTPUT : train shape: ", y_train.shape)
        print("After Encoding -> OUTPUT : test shape: ", y_test.shape)
```

```
After Encoding -> OUTPUT : train shape:  (60000, 10)
After Encoding -> OUTPUT : test shape:  (10000, 10)
```

2 CNN Model on MNIST dataset

```
In [0]: from tensorflow.keras.models import Sequential
        from tensorflow.keras.layers import Dense, Flatten, Dropout, BatchNormalization, Conv2D,

        n_epochs = 25
        batchsize = 128

        final_output = pd.DataFrame(columns=["#ConvNets", "#Kernels/Filters", "Padding", "Stride
```

```
In [0]: # 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()
```

2.1 2 ConvNet Architecture

In this type of architecture, we will be trying out 2 convolution layer along with 2 maxpooling layer for each.

2.1.1 3x3 Filter/Kernel

```
In [10]: m21_model = Sequential()

         # First Convolutional layer with MaxPooling
         m21_model.add(Conv2D(32, kernel_size=(3,3), activation=tf.nn.relu, input_shape=(28,28,1
         m21_model.add(MaxPooling2D(pool_size=(2,2)))

         # Second Convolutional layer with MaxPooling
         m21_model.add(Conv2D(64, kernel_size=(3,3), activation=tf.nn.relu))
         m21_model.add(MaxPooling2D(pool_size=(2,2)))

         # Three dense layers in MLP
         m21_model.add(Flatten())
         m21_model.add(Dense(128, activation=tf.nn.relu))
         m21_model.add(Dense(64, activation=tf.nn.relu))
         m21_model.add(Dense(10, activation=tf.nn.softmax))


         m21_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy
         model = m21_model.fit(x_train_new, y_train, epochs=n_epochs, batch_size=batchsize, verb

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/ops/resource_va
Instructions for updating:
Colocations handled automatically by placer.
Train on 60000 samples, validate on 10000 samples
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/ops/math_ops.py
Instructions for updating:
Use tf.cast instead.
Epoch 1/25
60000/60000 [==============================] - 7s 117us/sample - loss: 0.2405 - acc: 0.9281 - va
Epoch 2/25
60000/60000 [==============================] - 6s 97us/sample - loss: 0.0697 - acc: 0.9782 - val
Epoch 3/25
60000/60000 [==============================] - 6s 98us/sample - loss: 0.0530 - acc: 0.9833 - val
Epoch 4/25
60000/60000 [==============================] - 6s 97us/sample - loss: 0.0394 - acc: 0.9882 - val
Epoch 5/25
```

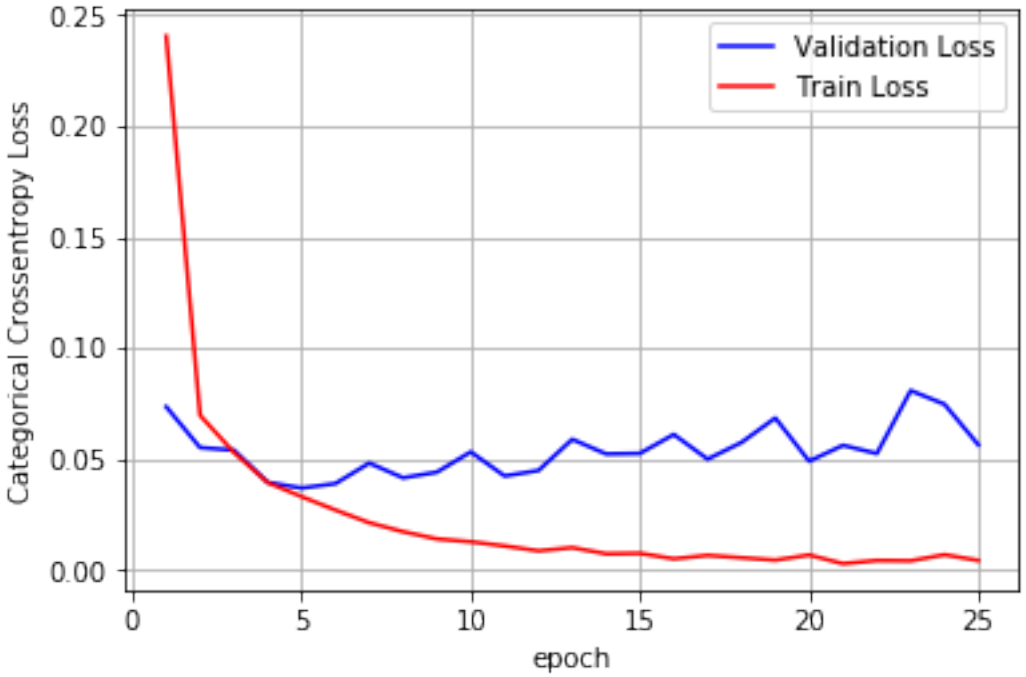
```
60000/60000 [==============================] - 6s 96us/sample - loss: 0.0331 - acc: 0.9894 - val
Epoch 6/25
60000/60000 [==============================] - 6s 96us/sample - loss: 0.0270 - acc: 0.9916 - val
Epoch 7/25
60000/60000 [==============================] - 6s 96us/sample - loss: 0.0213 - acc: 0.9931 - val
Epoch 8/25
60000/60000 [==============================] - 6s 96us/sample - loss: 0.0173 - acc: 0.9944 - val
Epoch 9/25
60000/60000 [==============================] - 6s 97us/sample - loss: 0.0140 - acc: 0.9955 - val
Epoch 10/25
60000/60000 [==============================] - 6s 96us/sample - loss: 0.0127 - acc: 0.9956 - val
Epoch 11/25
60000/60000 [==============================] - 6s 97us/sample - loss: 0.0109 - acc: 0.9964 - val
Epoch 12/25
60000/60000 [==============================] - 6s 96us/sample - loss: 0.0087 - acc: 0.9973 - val
Epoch 13/25
60000/60000 [==============================] - 6s 98us/sample - loss: 0.0101 - acc: 0.9966 - val
Epoch 14/25
60000/60000 [==============================] - 6s 97us/sample - loss: 0.0073 - acc: 0.9977 - val
Epoch 15/25
60000/60000 [==============================] - 6s 97us/sample - loss: 0.0075 - acc: 0.9974 - val
Epoch 16/25
60000/60000 [==============================] - 6s 97us/sample - loss: 0.0051 - acc: 0.9982 - val
Epoch 17/25
60000/60000 [==============================] - 6s 97us/sample - loss: 0.0066 - acc: 0.9978 - val
Epoch 18/25
60000/60000 [==============================] - 6s 96us/sample - loss: 0.0055 - acc: 0.9982 - val
Epoch 19/25
60000/60000 [==============================] - 6s 96us/sample - loss: 0.0045 - acc: 0.9986 - val
Epoch 20/25
60000/60000 [==============================] - 6s 96us/sample - loss: 0.0068 - acc: 0.9980 - val
Epoch 21/25
60000/60000 [==============================] - 6s 97us/sample - loss: 0.0029 - acc: 0.9991 - val
Epoch 22/25
60000/60000 [==============================] - 6s 96us/sample - loss: 0.0043 - acc: 0.9986 - val
Epoch 23/25
60000/60000 [==============================] - 6s 96us/sample - loss: 0.0042 - acc: 0.9987 - val
Epoch 24/25
60000/60000 [==============================] - 6s 96us/sample - loss: 0.0069 - acc: 0.9977 - val
Epoch 25/25
60000/60000 [==============================] - 6s 95us/sample - loss: 0.0043 - acc: 0.9987 - val
```

```
In [11]: print("************************************************")
         print("Printing the Model Summary")
         print(m21_model.summary())
         print("************************************************")
```

```
        score = m21_model.evaluate(x_test_new, y_test)

        print('Test score:', score[0])
        print('Test accuracy:', score[1])

        final_output = final_output.append({"#ConvNets": 2,
                                            "#Kernels/Filters": '3x3',
                                            "Padding": '-',
                                            "Stride": '2x2',
                                            "Dropout": '-',
                                            "BatchNormalization": False,
                                            "Regularization": '-',
                                            "TRAIN_LOSS": '{:.5f}'.format(model.history["loss"]
                                            "TEST_LOSS": '{:.5f}'.format(model.history["val_los
                                            "TRAIN_ACC": '{:.5f}'.format(model.history["acc"][n
                                            "TEST_ACC": '{:.5f}'.format(model.history["val_acc"


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

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

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

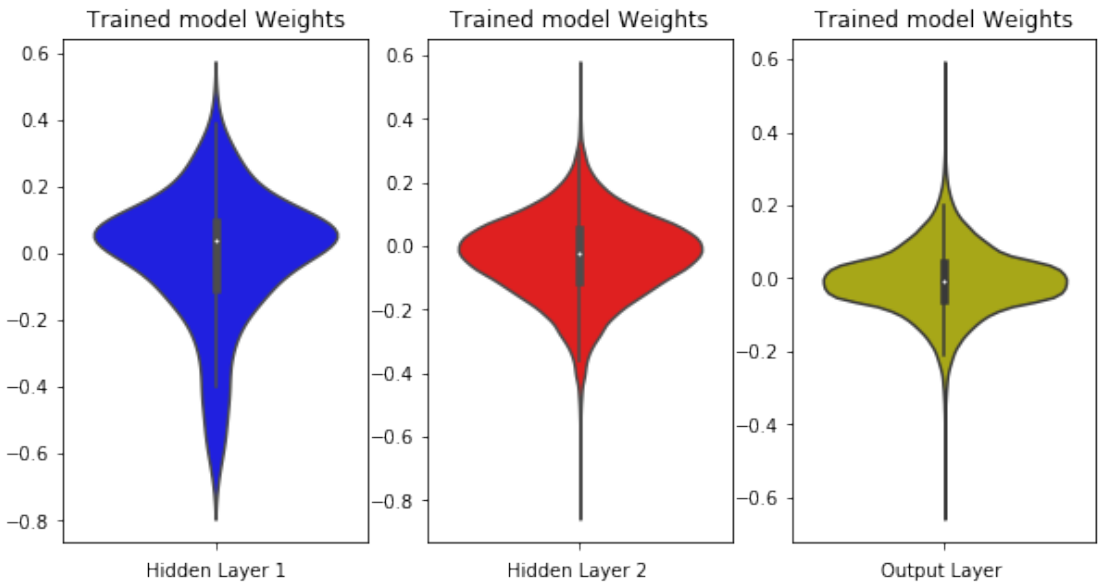
```
**********************************************
Printing the Model Summary
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 1600)              0
_________________________________________________________________
dense (Dense)                (None, 128)               204928
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256
_________________________________________________________________
```

```
dense_2 (Dense)              (None, 10)                650
=================================================================
Total params: 232,650
Trainable params: 232,650
Non-trainable params: 0
_________________________________________________________________
None
**********************************************
10000/10000 [==============================] - 1s 82us/sample - loss: 0.0564 - acc: 0.9883
Test score: 0.056394415135847066
Test accuracy: 0.9883
```



In [12]:
```
w_after = m21_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(figsize=(10, 5))
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()
```



2.1.2 5x5 Filter/Kernel

In [13]:
```
m22_model = Sequential()

# First Convolutional layer with MaxPooling
m22_model.add(Conv2D(32, kernel_size=(5,5), activation=tf.nn.relu, input_shape=(28,28,1
m22_model.add(MaxPooling2D(pool_size=(2,2)))

# Second Convolutional layer with MaxPooling
m22_model.add(Conv2D(64, kernel_size=(5,5), activation=tf.nn.relu))
m22_model.add(MaxPooling2D(pool_size=(2,2)))

# Three dense layers in MLP
m22_model.add(Flatten())
m22_model.add(Dense(128, activation=tf.nn.relu))
```

```
m22_model.add(Dense(64, activation=tf.nn.relu))
m22_model.add(Dense(10, activation=tf.nn.softmax))


m22_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy
model = m22_model.fit(x_train_new, y_train, epochs=n_epochs, batch_size=batchsize, verb
```

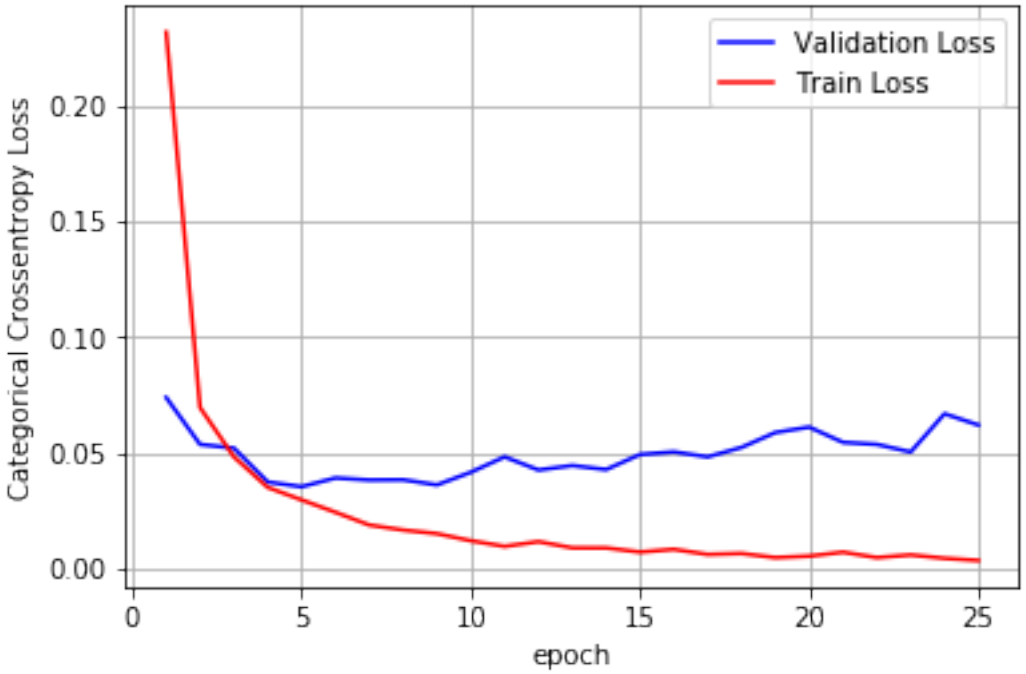
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/25
60000/60000 [==============================] - 6s 101us/sample - loss: 0.2320 - acc: 0.9293 - va
Epoch 2/25
60000/60000 [==============================] - 5s 87us/sample - loss: 0.0696 - acc: 0.9778 - val
Epoch 3/25
60000/60000 [==============================] - 5s 89us/sample - loss: 0.0482 - acc: 0.9848 - val
Epoch 4/25
60000/60000 [==============================] - 5s 89us/sample - loss: 0.0350 - acc: 0.9888 - val
Epoch 5/25
60000/60000 [==============================] - 5s 89us/sample - loss: 0.0296 - acc: 0.9901 - val
Epoch 6/25
60000/60000 [==============================] - 5s 89us/sample - loss: 0.0243 - acc: 0.9923 - val
Epoch 7/25
60000/60000 [==============================] - 5s 89us/sample - loss: 0.0187 - acc: 0.9940 - val
Epoch 8/25
60000/60000 [==============================] - 5s 89us/sample - loss: 0.0166 - acc: 0.9947 - val
Epoch 9/25
60000/60000 [==============================] - 5s 88us/sample - loss: 0.0150 - acc: 0.9952 - val
Epoch 10/25
60000/60000 [==============================] - 5s 89us/sample - loss: 0.0119 - acc: 0.9961 - val
Epoch 11/25
60000/60000 [==============================] - 5s 89us/sample - loss: 0.0095 - acc: 0.9967 - val
Epoch 12/25
60000/60000 [==============================] - 5s 89us/sample - loss: 0.0115 - acc: 0.9962 - val
Epoch 13/25
60000/60000 [==============================] - 5s 89us/sample - loss: 0.0089 - acc: 0.9969 - val
Epoch 14/25
60000/60000 [==============================] - 5s 89us/sample - loss: 0.0088 - acc: 0.9971 - val
Epoch 15/25
60000/60000 [==============================] - 5s 89us/sample - loss: 0.0070 - acc: 0.9978 - val
Epoch 16/25
60000/60000 [==============================] - 5s 89us/sample - loss: 0.0082 - acc: 0.9975 - val
Epoch 17/25
60000/60000 [==============================] - 5s 89us/sample - loss: 0.0060 - acc: 0.9981 - val
Epoch 18/25
60000/60000 [==============================] - 5s 89us/sample - loss: 0.0064 - acc: 0.9977 - val
Epoch 19/25
60000/60000 [==============================] - 5s 89us/sample - loss: 0.0046 - acc: 0.9985 - val
Epoch 20/25
60000/60000 [==============================] - 5s 89us/sample - loss: 0.0053 - acc: 0.9982 - val
```

```
Epoch 21/25
60000/60000 [==============================] - 5s 89us/sample - loss: 0.0070 - acc: 0.9976 - val
Epoch 22/25
60000/60000 [==============================] - 5s 89us/sample - loss: 0.0046 - acc: 0.9985 - val
Epoch 23/25
60000/60000 [==============================] - 5s 89us/sample - loss: 0.0058 - acc: 0.9979 - val
Epoch 24/25
60000/60000 [==============================] - 5s 89us/sample - loss: 0.0043 - acc: 0.9986 - val
Epoch 25/25
60000/60000 [==============================] - 5s 89us/sample - loss: 0.0034 - acc: 0.9990 - val
```

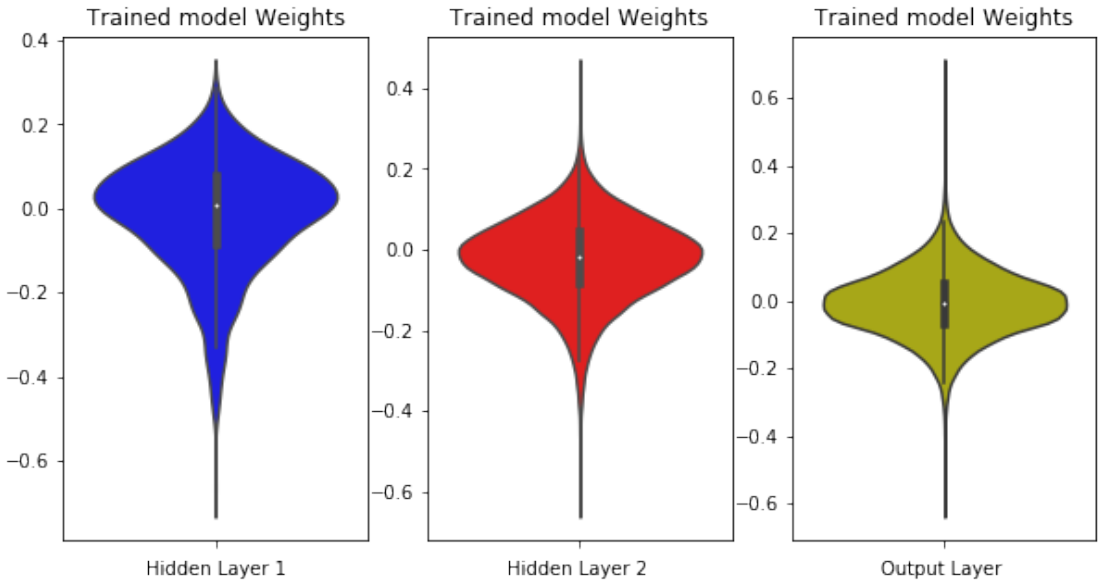
```
In [14]: print("************************************************")
         print("Printing the Model Summary")
         print(m22_model.summary())
         print("************************************************")

         score = m22_model.evaluate(x_test_new, y_test)

         print('Test score:', score[0])
         print('Test accuracy:', score[1])

         final_output = final_output.append({"#ConvNets": 2,
                                             "#Kernels/Filters": '5x5',
                                             "Padding": '-',
                                             "Stride": '2x2',
                                             "Dropout": '-',
                                             "BatchNormalization": False,
                                             "Regularization": '-',
                                             "TRAIN_LOSS": '{:.5f}'.format(model.history["loss"]
                                             "TEST_LOSS": '{:.5f}'.format(model.history["val_los
                                             "TRAIN_ACC": '{:.5f}'.format(model.history["acc"][n
                                             "TEST_ACC": '{:.5f}'.format(model.history["val_acc"


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

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

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

```
************************************************
Printing the Model Summary
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 24, 24, 32)        832
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 12, 12, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 8, 8, 64)          51264
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 4, 4, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_3 (Dense)              (None, 128)               131200
_________________________________________________________________
dense_4 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_5 (Dense)              (None, 10)                650
=================================================================
Total params: 192,202
Trainable params: 192,202
Non-trainable params: 0
_________________________________________________________________
None
**********************************************
10000/10000 [==============================] - 1s 84us/sample - loss: 0.0620 - acc: 0.9887
Test score: 0.061998865842755914
Test accuracy: 0.9887
```



In [15]:
```
w_after = m22_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(figsize=(10, 5))
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()
```



2.1.3 3x3 Filter/Kernel with Dropout and Weight Regularization & Initialization and Batch Normalization

In [16]:
```
m23_model = Sequential()

# First Convolutional layer with MaxPooling
m23_model.add(Conv2D(32, kernel_size=(3,3), activation=tf.nn.relu, input_shape=(28,28,1
                     kernel_initializer='he_normal', kernel_regularizer=tf.keras.regula
m23_model.add(MaxPooling2D(pool_size=(2,2)))

m23_model.add(Dropout(rate=0.5))

# Second Convolutional layer with MaxPooling
m23_model.add(Conv2D(64, kernel_size=(3,3), activation=tf.nn.relu,
                     kernel_initializer='he_normal', kernel_regularizer=tf.keras.regular
m23_model.add(MaxPooling2D(pool_size=(2,2)))

m23_model.add(Dropout(rate=0.5))

# Three dense layers in MLP
m23_model.add(Flatten())
m23_model.add(Dense(128, activation=tf.nn.relu, kernel_initializer='he_normal', kernel_
m23_model.add(BatchNormalization())
m23_model.add(Dropout(rate=0.5))
m23_model.add(Dense(64, activation=tf.nn.relu, kernel_initializer='he_normal', kernel_r
m23_model.add(BatchNormalization())
m23_model.add(Dropout(rate=0.5))
```

```
        m23_model.add(Dense(10, activation=tf.nn.softmax))


        m23_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy
        model = m23_model.fit(x_train_new, y_train, epochs=n_epochs, batch_size=batchsize, verb
```

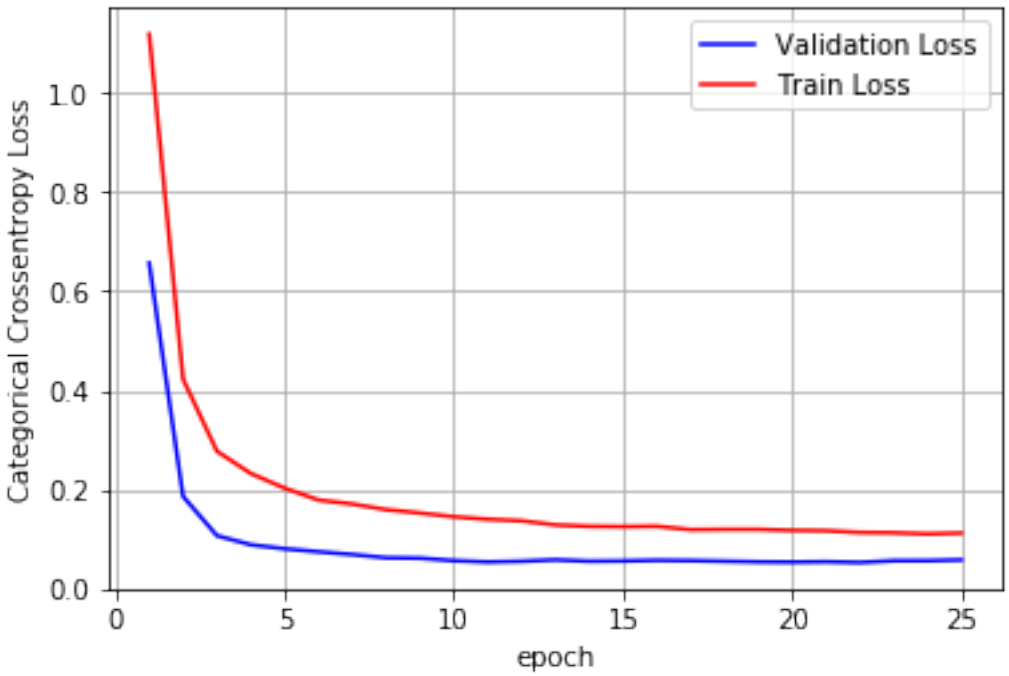
```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/keras/layers/co
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
Train on 60000 samples, validate on 10000 samples
Epoch 1/25
60000/60000 [==============================] - 9s 143us/sample - loss: 1.1191 - acc: 0.6388 - va
Epoch 2/25
60000/60000 [==============================] - 8s 130us/sample - loss: 0.4228 - acc: 0.8740 - va
Epoch 3/25
60000/60000 [==============================] - 8s 129us/sample - loss: 0.2772 - acc: 0.9200 - va
Epoch 4/25
60000/60000 [==============================] - 8s 130us/sample - loss: 0.2323 - acc: 0.9350 - va
Epoch 5/25
60000/60000 [==============================] - 8s 129us/sample - loss: 0.2026 - acc: 0.9440 - va
Epoch 6/25
60000/60000 [==============================] - 8s 129us/sample - loss: 0.1786 - acc: 0.9511 - va
Epoch 7/25
60000/60000 [==============================] - 8s 129us/sample - loss: 0.1705 - acc: 0.9537 - va
Epoch 8/25
60000/60000 [==============================] - 8s 129us/sample - loss: 0.1592 - acc: 0.9571 - va
Epoch 9/25
60000/60000 [==============================] - 8s 129us/sample - loss: 0.1523 - acc: 0.9592 - va
Epoch 10/25
60000/60000 [==============================] - 8s 129us/sample - loss: 0.1448 - acc: 0.9619 - va
Epoch 11/25
60000/60000 [==============================] - 8s 128us/sample - loss: 0.1397 - acc: 0.9628 - va
Epoch 12/25
60000/60000 [==============================] - 8s 128us/sample - loss: 0.1367 - acc: 0.9636 - va
Epoch 13/25
60000/60000 [==============================] - 8s 129us/sample - loss: 0.1283 - acc: 0.9661 - va
Epoch 14/25
60000/60000 [==============================] - 8s 128us/sample - loss: 0.1255 - acc: 0.9678 - va
Epoch 15/25
60000/60000 [==============================] - 8s 128us/sample - loss: 0.1248 - acc: 0.9686 - va
Epoch 16/25
60000/60000 [==============================] - 8s 129us/sample - loss: 0.1256 - acc: 0.9675 - va
Epoch 17/25
60000/60000 [==============================] - 8s 128us/sample - loss: 0.1187 - acc: 0.9704 - va
Epoch 18/25
60000/60000 [==============================] - 8s 128us/sample - loss: 0.1193 - acc: 0.9706 - va
Epoch 19/25
60000/60000 [==============================] - 8s 128us/sample - loss: 0.1193 - acc: 0.9701 - va
```

```
Epoch 20/25
60000/60000 [==============================] - 8s 127us/sample - loss: 0.1173 - acc: 0.9718 - va
Epoch 21/25
60000/60000 [==============================] - 8s 128us/sample - loss: 0.1166 - acc: 0.9709 - va
Epoch 22/25
60000/60000 [==============================] - 8s 127us/sample - loss: 0.1131 - acc: 0.9729 - va
Epoch 23/25
60000/60000 [==============================] - 8s 127us/sample - loss: 0.1125 - acc: 0.9729 - va
Epoch 24/25
60000/60000 [==============================] - 8s 128us/sample - loss: 0.1102 - acc: 0.9738 - va
Epoch 25/25
60000/60000 [==============================] - 8s 128us/sample - loss: 0.1119 - acc: 0.9736 - va
```

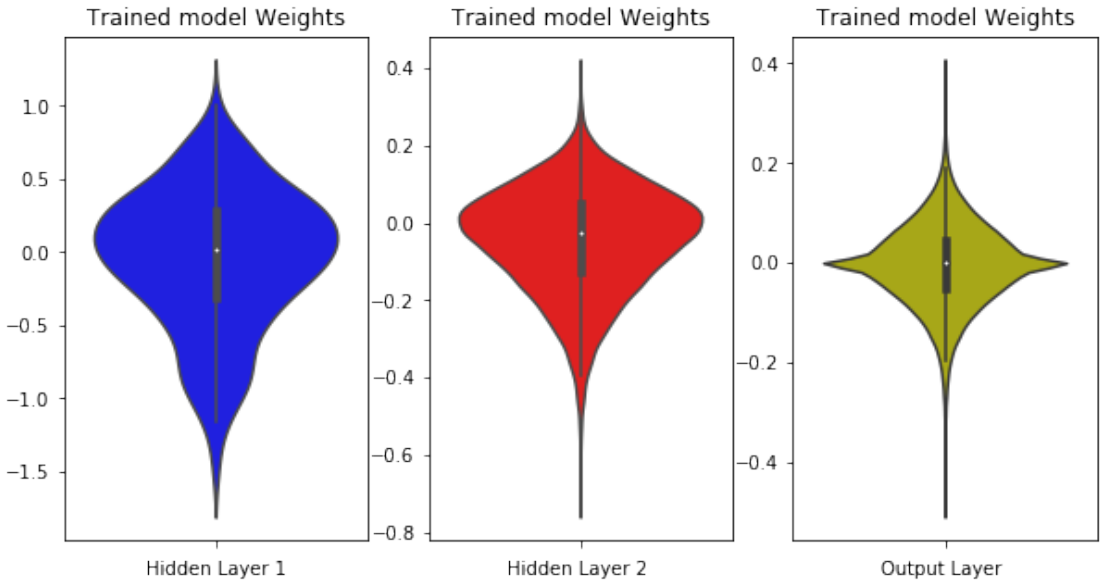
```
In [17]: print("************************************************")
         print("Printing the Model Summary")
         print(m23_model.summary())
         print("************************************************")

         score = m23_model.evaluate(x_test_new, y_test)

         print('Test score:', score[0])
         print('Test accuracy:', score[1])

         final_output = final_output.append({"#ConvNets": 2,
                                             "#Kernels/Filters": '3x3',
                                             "Padding": '-',
                                             "Stride": '2x2',
                                             "Dropout": True,
                                             "BatchNormalization": True,
                                             "Regularization": 'L2 (0.00001)',
                                             "TRAIN_LOSS": '{:.5f}'.format(model.history["loss"]
                                             "TEST_LOSS": '{:.5f}'.format(model.history["val_los
                                             "TRAIN_ACC": '{:.5f}'.format(model.history["acc"][n
                                             "TEST_ACC": '{:.5f}'.format(model.history["val_acc"

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

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

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

```
************************************************
Printing the Model Summary
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 13, 13, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 5, 5, 64)          0
_________________________________________________________________
flatten_2 (Flatten)          (None, 1600)              0
_________________________________________________________________
dense_6 (Dense)              (None, 128)               204928
_________________________________________________________________
batch_normalization_v1 (Batc (None, 128)               512
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_7 (Dense)              (None, 64)                8256
_________________________________________________________________
batch_normalization_v1_1 (Ba (None, 64)                256
_________________________________________________________________
dropout_3 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_8 (Dense)              (None, 10)                650
=================================================================
Total params: 233,418
Trainable params: 233,034
Non-trainable params: 384
_________________________________________________________________
None
**********************************************
10000/10000 [==============================] - 1s 103us/sample - loss: 0.0577 - acc: 0.9881
Test score: 0.05770807054638863
Test accuracy: 0.9881
```



In [18]:
```
w_after = m23_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(figsize=(10, 5))
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()
```



2.1.4 5x5 Kernel/Filter with Dropout, Weight Regularization/ initialization and Batch Normalization

```
In [19]: m24_model = Sequential()

         # First Convolutional layer with MaxPooling
         m24_model.add(Conv2D(32, kernel_size=(5,5), activation=tf.nn.relu, input_shape=(28,28,1
                              kernel_initializer='he_normal', kernel_regularizer=tf.keras.regula
         m24_model.add(MaxPooling2D(pool_size=(2,2)))

         m24_model.add(Dropout(rate=0.5))

         # Second Convolutional layer with MaxPooling
         m24_model.add(Conv2D(64, kernel_size=(5,5), activation=tf.nn.relu,
                              kernel_initializer='he_normal', kernel_regularizer=tf.keras.regular
         m24_model.add(MaxPooling2D(pool_size=(2,2)))

         m24_model.add(Dropout(rate=0.5))

         # Three dense layers in MLP
         m24_model.add(Flatten())
         m24_model.add(Dense(128, activation=tf.nn.relu, kernel_initializer='he_normal', kernel_
         m24_model.add(BatchNormalization())
         m24_model.add(Dropout(rate=0.5))
         m24_model.add(Dense(64, activation=tf.nn.relu, kernel_initializer='he_normal', kernel_r
         m24_model.add(BatchNormalization())
         m24_model.add(Dropout(rate=0.5))
```

```
m24_model.add(Dense(10, activation=tf.nn.softmax))


m24_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy
model = m24_model.fit(x_train_new, y_train, epochs=n_epochs, batch_size=batchsize, verb
```

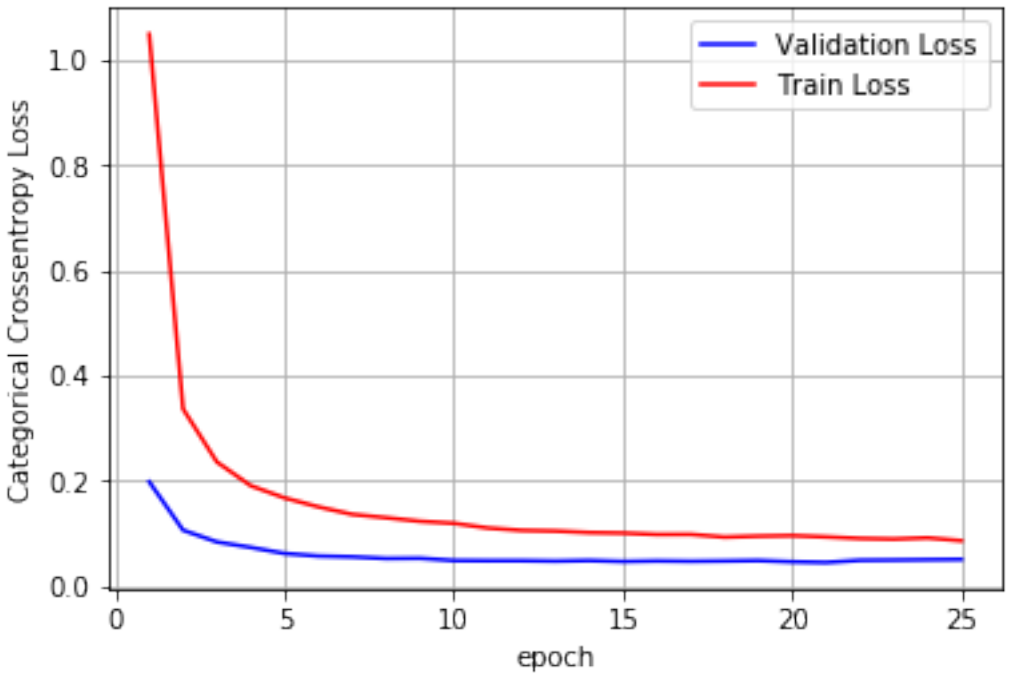
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/25
60000/60000 [==============================] - 8s 134us/sample - loss: 1.0503 - acc: 0.6756 - va
Epoch 2/25
60000/60000 [==============================] - 7s 121us/sample - loss: 0.3364 - acc: 0.9034 - va
Epoch 3/25
60000/60000 [==============================] - 7s 121us/sample - loss: 0.2353 - acc: 0.9347 - va
Epoch 4/25
60000/60000 [==============================] - 7s 120us/sample - loss: 0.1901 - acc: 0.9484 - va
Epoch 5/25
60000/60000 [==============================] - 7s 120us/sample - loss: 0.1669 - acc: 0.9546 - va
Epoch 6/25
60000/60000 [==============================] - 7s 121us/sample - loss: 0.1498 - acc: 0.9595 - va
Epoch 7/25
60000/60000 [==============================] - 7s 120us/sample - loss: 0.1355 - acc: 0.9646 - va
Epoch 8/25
60000/60000 [==============================] - 7s 120us/sample - loss: 0.1291 - acc: 0.9658 - va
Epoch 9/25
60000/60000 [==============================] - 7s 115us/sample - loss: 0.1223 - acc: 0.9679 - va
Epoch 10/25
60000/60000 [==============================] - 7s 116us/sample - loss: 0.1185 - acc: 0.9699 - va
Epoch 11/25
60000/60000 [==============================] - 7s 118us/sample - loss: 0.1096 - acc: 0.9718 - va
Epoch 12/25
60000/60000 [==============================] - 7s 118us/sample - loss: 0.1050 - acc: 0.9727 - va
Epoch 13/25
60000/60000 [==============================] - 7s 117us/sample - loss: 0.1038 - acc: 0.9742 - va
Epoch 14/25
60000/60000 [==============================] - 7s 118us/sample - loss: 0.1009 - acc: 0.9743 - va
Epoch 15/25
60000/60000 [==============================] - 7s 118us/sample - loss: 0.0999 - acc: 0.9751 - va
Epoch 16/25
60000/60000 [==============================] - 7s 117us/sample - loss: 0.0973 - acc: 0.9761 - va
Epoch 17/25
60000/60000 [==============================] - 7s 117us/sample - loss: 0.0974 - acc: 0.9764 - va
Epoch 18/25
60000/60000 [==============================] - 7s 120us/sample - loss: 0.0923 - acc: 0.9783 - va
Epoch 19/25
60000/60000 [==============================] - 7s 121us/sample - loss: 0.0940 - acc: 0.9777 - va
Epoch 20/25
60000/60000 [==============================] - 7s 120us/sample - loss: 0.0950 - acc: 0.9776 - va
Epoch 21/25
```

```
60000/60000 [==============================] - 7s 121us/sample - loss: 0.0926 - acc: 0.9780 - va
Epoch 22/25
60000/60000 [==============================] - 7s 121us/sample - loss: 0.0896 - acc: 0.9793 - va
Epoch 23/25
60000/60000 [==============================] - 7s 122us/sample - loss: 0.0885 - acc: 0.9792 - va
Epoch 24/25
60000/60000 [==============================] - 7s 122us/sample - loss: 0.0907 - acc: 0.9795 - va
Epoch 25/25
60000/60000 [==============================] - 7s 122us/sample - loss: 0.0853 - acc: 0.9804 - va
```

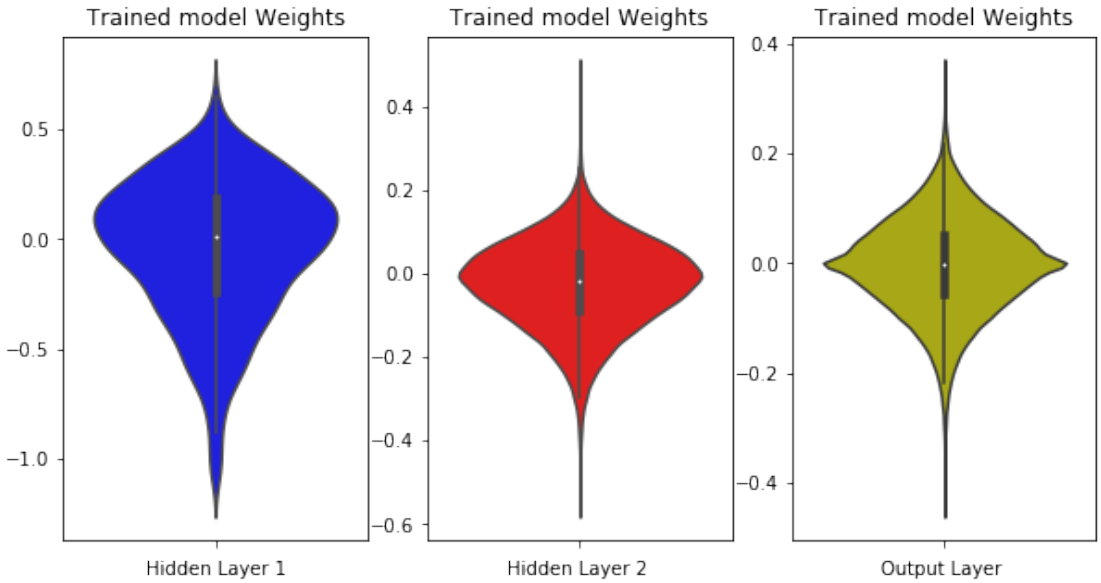
```
In [20]: print("************************************************")
         print("Printing the Model Summary")
         print(m24_model.summary())
         print("************************************************")

         score = m24_model.evaluate(x_test_new, y_test)

         print('Test score:', score[0])
         print('Test accuracy:', score[1])

         final_output = final_output.append({"#ConvNets": 2,
                                             "#Kernels/Filters": '5x5',
                                             "Padding": '-',
                                             "Stride": '2x2',
                                             "Dropout": True,
                                             "BatchNormalization": True,
                                             "Regularization": 'L2 (0.00001)',
                                             "TRAIN_LOSS": '{:.5f}'.format(model.history["loss"]
                                             "TEST_LOSS": '{:.5f}'.format(model.history["val_los
                                             "TRAIN_ACC": '{:.5f}'.format(model.history["acc"][n
                                             "TEST_ACC": '{:.5f}'.format(model.history["val_acc"

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

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

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

```
************************************************
Printing the Model Summary
_________________________________________________________________
Layer (type)                 Output Shape              Param #
```

```
=================================================================
conv2d_6 (Conv2D)            (None, 24, 24, 32)        832
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 12, 12, 32)        0
_________________________________________________________________
dropout_4 (Dropout)          (None, 12, 12, 32)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 8, 8, 64)          51264
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 4, 4, 64)          0
_________________________________________________________________
dropout_5 (Dropout)          (None, 4, 4, 64)          0
_________________________________________________________________
flatten_3 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_9 (Dense)              (None, 128)               131200
_________________________________________________________________
batch_normalization_v1_2 (Ba (None, 128)               512
_________________________________________________________________
dropout_6 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_10 (Dense)             (None, 64)                8256
_________________________________________________________________
batch_normalization_v1_3 (Ba (None, 64)                256
_________________________________________________________________
dropout_7 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_11 (Dense)             (None, 10)                650
=================================================================
Total params: 192,970
Trainable params: 192,586
Non-trainable params: 384
_________________________________________________________________
None
**********************************************
10000/10000 [==============================] - 1s 107us/sample - loss: 0.0497 - acc: 0.9899
Test score: 0.049702025166153 91
Test accuracy: 0.9899
```



In [21]:
```
w_after = m24_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(figsize=(10, 5))
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()
```



2.2 3 ConvNet Architecture

In this type of architecture, we will be trying out 3 convolution layer along with 3 maxpooling layer for each.

2.2.1 3x3 Filter/Kernel

In [22]:
```
m31_model = Sequential()

# First Convolutional layer with MaxPooling
m31_model.add(Conv2D(32, kernel_size=(3,3), activation=tf.nn.relu, input_shape=(28,28,1
m31_model.add(MaxPooling2D(pool_size=(2,2)))

# Second Convolutional layer with MaxPooling
m31_model.add(Conv2D(64, kernel_size=(3,3), activation=tf.nn.relu))
m31_model.add(MaxPooling2D(pool_size=(2,2)))

# Third Convolutional layer with MaxPooling
m31_model.add(Conv2D(32, kernel_size=(3,3), activation=tf.nn.relu))
m31_model.add(MaxPooling2D(pool_size=(2,2)))

# Three dense layers in MLP
m31_model.add(Flatten())
m31_model.add(Dense(128, activation=tf.nn.relu))
m31_model.add(Dense(64, activation=tf.nn.relu))
m31_model.add(Dense(10, activation=tf.nn.softmax))
```

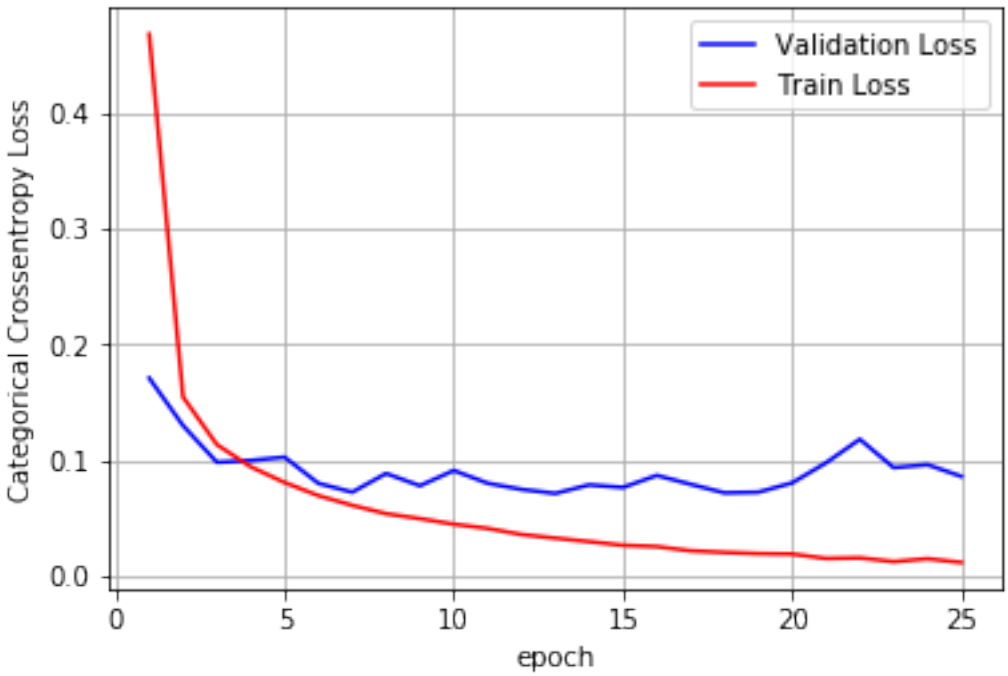
```
        m31_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy
        model = m31_model.fit(x_train_new, y_train, epochs=n_epochs, batch_size=batchsize, verb

Train on 60000 samples, validate on 10000 samples
Epoch 1/25
60000/60000 [==============================] - 7s 110us/sample - loss: 0.4676 - acc: 0.8478 - va
Epoch 2/25
60000/60000 [==============================] - 6s 104us/sample - loss: 0.1543 - acc: 0.9522 - va
Epoch 3/25
60000/60000 [==============================] - 6s 104us/sample - loss: 0.1131 - acc: 0.9648 - va
Epoch 4/25
60000/60000 [==============================] - 6s 102us/sample - loss: 0.0940 - acc: 0.9704 - va
Epoch 5/25
60000/60000 [==============================] - 6s 102us/sample - loss: 0.0804 - acc: 0.9750 - va
Epoch 6/25
60000/60000 [==============================] - 6s 101us/sample - loss: 0.0691 - acc: 0.9785 - va
Epoch 7/25
60000/60000 [==============================] - 6s 101us/sample - loss: 0.0607 - acc: 0.9808 - va
Epoch 8/25
60000/60000 [==============================] - 6s 102us/sample - loss: 0.0535 - acc: 0.9830 - va
Epoch 9/25
60000/60000 [==============================] - 6s 101us/sample - loss: 0.0492 - acc: 0.9846 - va
Epoch 10/25
60000/60000 [==============================] - 6s 101us/sample - loss: 0.0444 - acc: 0.9861 - va
Epoch 11/25
60000/60000 [==============================] - 6s 101us/sample - loss: 0.0408 - acc: 0.9863 - va
Epoch 12/25
60000/60000 [==============================] - 6s 100us/sample - loss: 0.0355 - acc: 0.9886 - va
Epoch 13/25
60000/60000 [==============================] - 6s 100us/sample - loss: 0.0324 - acc: 0.9895 - va
Epoch 14/25
60000/60000 [==============================] - 6s 99us/sample - loss: 0.0294 - acc: 0.9904 - val
Epoch 15/25
60000/60000 [==============================] - 6s 100us/sample - loss: 0.0262 - acc: 0.9912 - va
Epoch 16/25
60000/60000 [==============================] - 6s 100us/sample - loss: 0.0251 - acc: 0.9921 - va
Epoch 17/25
60000/60000 [==============================] - 6s 100us/sample - loss: 0.0214 - acc: 0.9930 - va
Epoch 18/25
60000/60000 [==============================] - 6s 100us/sample - loss: 0.0200 - acc: 0.9932 - va
Epoch 19/25
60000/60000 [==============================] - 6s 99us/sample - loss: 0.0190 - acc: 0.9934 - val
Epoch 20/25
60000/60000 [==============================] - 6s 99us/sample - loss: 0.0186 - acc: 0.9937 - val
Epoch 21/25
60000/60000 [==============================] - 6s 99us/sample - loss: 0.0148 - acc: 0.9952 - val
Epoch 22/25
60000/60000 [==============================] - 6s 99us/sample - loss: 0.0153 - acc: 0.9946 - val
```

```
Epoch 23/25
60000/60000 [==============================] - 6s 99us/sample - loss: 0.0121 - acc: 0.9961 - val
Epoch 24/25
60000/60000 [==============================] - 6s 99us/sample - loss: 0.0146 - acc: 0.9947 - val
Epoch 25/25
60000/60000 [==============================] - 6s 99us/sample - loss: 0.0114 - acc: 0.9961 - val
```

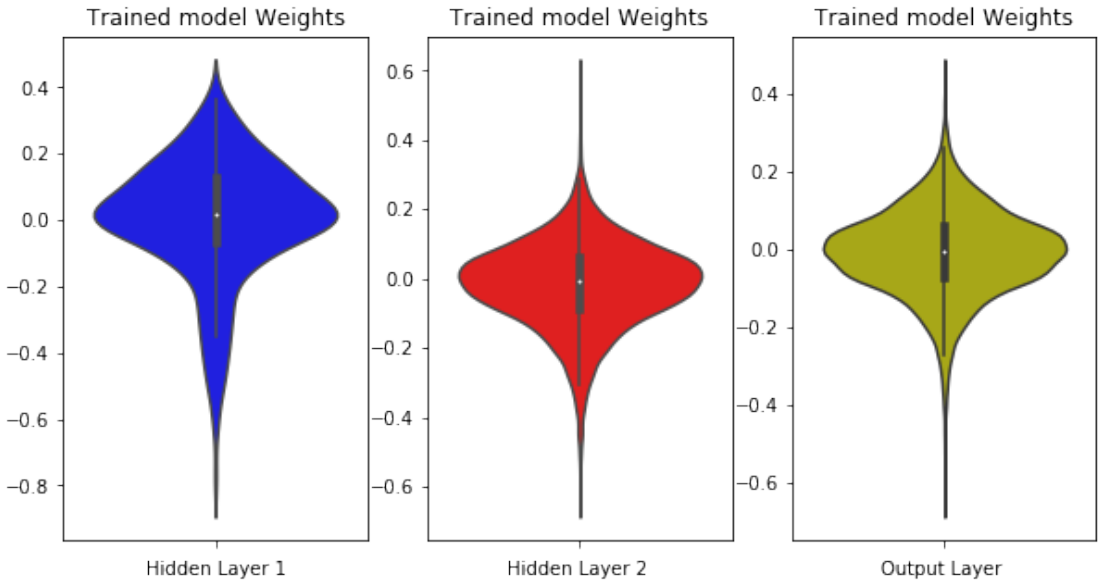
```
In [23]: print("************************************************")
         print("Printing the Model Summary")
         print(m31_model.summary())
         print("************************************************")

         score = m31_model.evaluate(x_test_new, y_test)

         print('Test score:', score[0])
         print('Test accuracy:', score[1])

         final_output = final_output.append({"#ConvNets": 3,
                                             "#Kernels/Filters": '3x3',
                                             "Padding": '-',
                                             "Stride": '2x2',
                                             "Dropout": '-',
                                             "BatchNormalization": False,
                                             "Regularization": '-',
                                             "TRAIN_LOSS": '{:.5f}'.format(model.history["loss"]
                                             "TEST_LOSS": '{:.5f}'.format(model.history["val_los
                                             "TRAIN_ACC": '{:.5f}'.format(model.history["acc"][n
                                             "TEST_ACC": '{:.5f}'.format(model.history["val_acc"


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

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

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

```
**********************************************
Printing the Model Summary
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_8 (Conv2D)            (None, 26, 26, 32)        320
```

```
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 3, 3, 32)          18464
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 1, 1, 32)          0
_________________________________________________________________
flatten_4 (Flatten)          (None, 32)                0
_________________________________________________________________
dense_12 (Dense)             (None, 128)               4224
_________________________________________________________________
dense_13 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_14 (Dense)             (None, 10)                650
=================================================================
Total params: 50,410
Trainable params: 50,410
Non-trainable params: 0
_________________________________________________________________
None
**********************************************
10000/10000 [==============================] - 1s 85us/sample - loss: 0.0857 - acc: 0.9812
Test score: 0.08567447326574075
Test accuracy: 0.9812
```



In [24]:
```
w_after = m31_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(figsize=(10, 5))
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()
```



2.2.2 5x5 Kernel/Filter

```
In [25]: m32_model = Sequential()

         # First Convolutional layer with MaxPooling
         m32_model.add(Conv2D(32, kernel_size=(5,5), padding="same",  activation=tf.nn.relu, inp
         m32_model.add(MaxPooling2D(pool_size=(2,2)))

         # Second Convolutional layer with MaxPooling
         m32_model.add(Conv2D(64, kernel_size=(5,5), padding="same", activation=tf.nn.relu))
         m32_model.add(MaxPooling2D(pool_size=(2,2)))

         # Third Convolutional layer with MaxPooling
         m32_model.add(Conv2D(32, kernel_size=(5,5), padding="same", activation=tf.nn.relu))
         m32_model.add(MaxPooling2D(pool_size=(2,2)))

         # Three dense layers in MLP
         m32_model.add(Flatten())
         m32_model.add(Dense(128, activation=tf.nn.relu))
         m32_model.add(Dense(64, activation=tf.nn.relu))
         m32_model.add(Dense(10, activation=tf.nn.softmax))


         m32_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy
         model = m32_model.fit(x_train_new, y_train, epochs=n_epochs, batch_size=batchsize, verb

Train on 60000 samples, validate on 10000 samples
Epoch 1/25
```

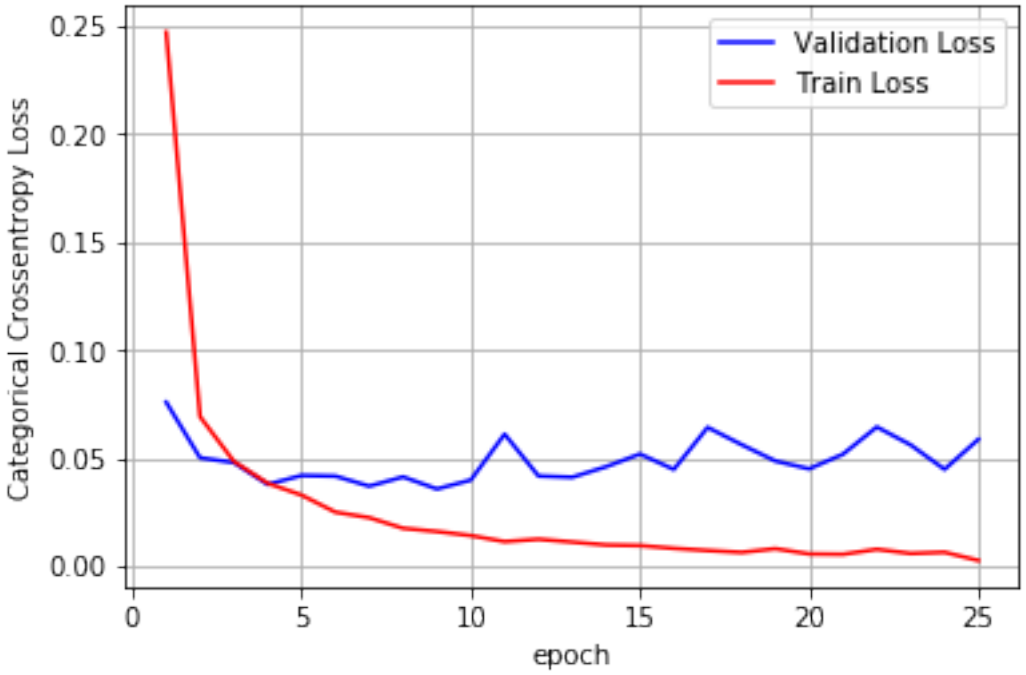
```
60000/60000 [==============================] - 8s 138us/sample - loss: 0.2470 - acc: 0.9249 - va
Epoch 2/25
60000/60000 [==============================] - 8s 127us/sample - loss: 0.0691 - acc: 0.9786 - va
Epoch 3/25
60000/60000 [==============================] - 8s 129us/sample - loss: 0.0485 - acc: 0.9851 - va
Epoch 4/25
60000/60000 [==============================] - 8s 126us/sample - loss: 0.0383 - acc: 0.9875 - va
Epoch 5/25
60000/60000 [==============================] - 8s 129us/sample - loss: 0.0329 - acc: 0.9895 - va
Epoch 6/25
60000/60000 [==============================] - 8s 126us/sample - loss: 0.0250 - acc: 0.9916 - va
Epoch 7/25
60000/60000 [==============================] - 8s 127us/sample - loss: 0.0224 - acc: 0.9926 - va
Epoch 8/25
60000/60000 [==============================] - 8s 126us/sample - loss: 0.0175 - acc: 0.9944 - va
Epoch 9/25
60000/60000 [==============================] - 8s 126us/sample - loss: 0.0160 - acc: 0.9944 - va
Epoch 10/25
60000/60000 [==============================] - 8s 126us/sample - loss: 0.0141 - acc: 0.9952 - va
Epoch 11/25
60000/60000 [==============================] - 8s 126us/sample - loss: 0.0114 - acc: 0.9962 - va
Epoch 12/25
60000/60000 [==============================] - 8s 126us/sample - loss: 0.0125 - acc: 0.9960 - va
Epoch 13/25
60000/60000 [==============================] - 8s 126us/sample - loss: 0.0111 - acc: 0.9961 - va
Epoch 14/25
60000/60000 [==============================] - 8s 126us/sample - loss: 0.0098 - acc: 0.9969 - va
Epoch 15/25
60000/60000 [==============================] - 8s 126us/sample - loss: 0.0095 - acc: 0.9967 - va
Epoch 16/25
60000/60000 [==============================] - 8s 127us/sample - loss: 0.0082 - acc: 0.9972 - va
Epoch 17/25
60000/60000 [==============================] - 8s 126us/sample - loss: 0.0072 - acc: 0.9976 - va
Epoch 18/25
60000/60000 [==============================] - 8s 126us/sample - loss: 0.0064 - acc: 0.9979 - va
Epoch 19/25
60000/60000 [==============================] - 8s 126us/sample - loss: 0.0081 - acc: 0.9974 - va
Epoch 20/25
60000/60000 [==============================] - 8s 126us/sample - loss: 0.0056 - acc: 0.9982 - va
Epoch 21/25
60000/60000 [==============================] - 8s 125us/sample - loss: 0.0055 - acc: 0.9982 - va
Epoch 22/25
60000/60000 [==============================] - 8s 125us/sample - loss: 0.0078 - acc: 0.9976 - va
Epoch 23/25
60000/60000 [==============================] - 8s 126us/sample - loss: 0.0059 - acc: 0.9981 - va
Epoch 24/25
60000/60000 [==============================] - 8s 126us/sample - loss: 0.0064 - acc: 0.9980 - va
Epoch 25/25
```

```
60000/60000 [==============================] - 8s 126us/sample - loss: 0.0026 - acc: 0.9994 - va
```

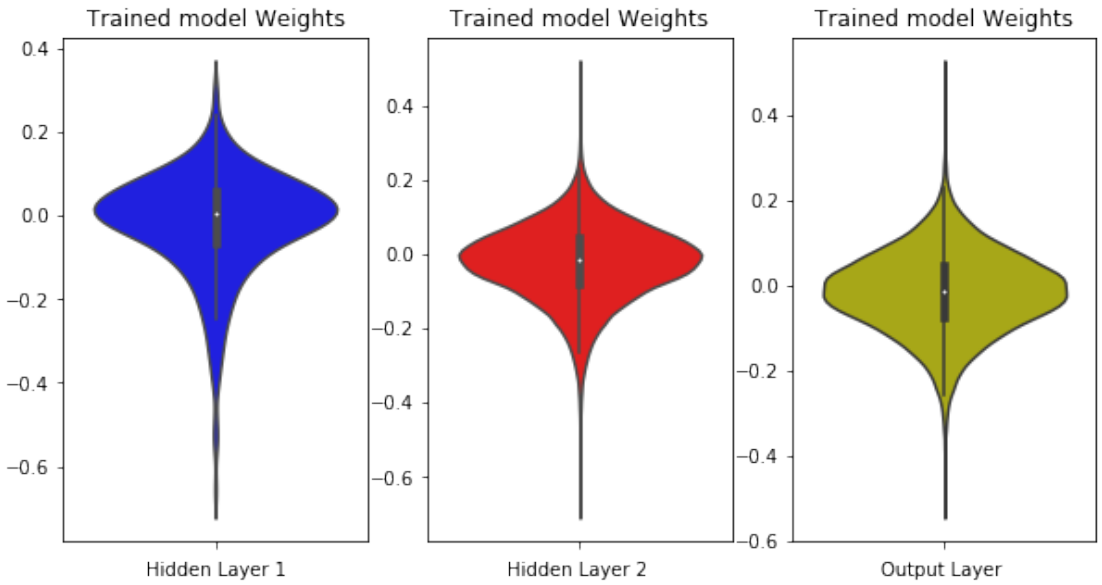
```
In [26]: print("************************************************")
         print("Printing the Model Summary")
         print(m32_model.summary())
         print("************************************************")

         score = m32_model.evaluate(x_test_new, y_test)

         print('Test score:', score[0])
         print('Test accuracy:', score[1])


         final_output = final_output.append({"#ConvNets": 3,
                                             "#Kernels/Filters": '5x5',
                                             "Padding": 'same',
                                             "Stride": '2x2',
                                             "Dropout": False,
                                             "BatchNormalization": False,
                                             "Regularization": '-',
                                             "TRAIN_LOSS": '{:.5f}'.format(model.history["loss"]
                                             "TEST_LOSS": '{:.5f}'.format(model.history["val_los
                                             "TRAIN_ACC": '{:.5f}'.format(model.history["acc"][n
                                             "TEST_ACC": '{:.5f}'.format(model.history["val_acc"

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

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

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

```
************************************************
Printing the Model Summary
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_11 (Conv2D)           (None, 28, 28, 32)        832
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 14, 14, 64)        51264
_________________________________________________________________
```

```
max_pooling2d_12 (MaxPooling (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 7, 7, 32)          51232
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 3, 3, 32)          0
_________________________________________________________________
flatten_5 (Flatten)          (None, 288)               0
_________________________________________________________________
dense_15 (Dense)             (None, 128)               36992
_________________________________________________________________
dense_16 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_17 (Dense)             (None, 10)                650
=================================================================
Total params: 149,226
Trainable params: 149,226
Non-trainable params: 0
_________________________________________________________________
None
**********************************************
10000/10000 [==============================] - 1s 106us/sample - loss: 0.0587 - acc: 0.9890
Test score: 0.05866653555549251
Test accuracy: 0.989
```



In [27]:
```
w_after = m32_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(figsize=(10, 5))
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()
```



2.2.3 3x3 Kernel/Filter with Dropout and Weights Regularization & Initialization and Batch Normalization

```
In [28]: m33_model = Sequential()

         # First Convolutional layer with MaxPooling
         m33_model.add(Conv2D(32, kernel_size=(3,3), activation=tf.nn.relu, input_shape=(28,28,1
                              kernel_initializer='he_normal', kernel_regularizer=tf.keras.regula
         m33_model.add(MaxPooling2D(pool_size=(2,2)))

         m33_model.add(Dropout(rate=0.5))

         # Second Convolutional layer with MaxPooling
         m33_model.add(Conv2D(64, kernel_size=(3,3), activation=tf.nn.relu,
                              kernel_initializer='he_normal', kernel_regularizer=tf.keras.regular
         m33_model.add(MaxPooling2D(pool_size=(2,2)))

         m33_model.add(Dropout(rate=0.5))

         # Third Convolutional layer with MaxPooling
         m33_model.add(Conv2D(32, kernel_size=(3,3), activation=tf.nn.relu,
                              kernel_initializer='he_normal', kernel_regularizer=tf.keras.regular
         m33_model.add(MaxPooling2D(pool_size=(2,2)))

         m33_model.add(Dropout(rate=0.5))

         # Three dense layers in MLP
         m33_model.add(Flatten())
         m33_model.add(Dense(128, activation=tf.nn.relu, kernel_initializer='he_normal', kernel_
         m33_model.add(BatchNormalization())
         m33_model.add(Dropout(rate=0.5))
         m33_model.add(Dense(64, activation=tf.nn.relu, kernel_initializer='he_normal', kernel_r
         m33_model.add(BatchNormalization())
         m33_model.add(Dropout(rate=0.5))
         m33_model.add(Dense(10, activation=tf.nn.softmax))


         m33_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy
         model = m33_model.fit(x_train_new, y_train, epochs=n_epochs, batch_size=batchsize, verb
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/25
60000/60000 [==============================] - 10s 161us/sample - loss: 2.4852 - acc: 0.1963 - v
Epoch 2/25
60000/60000 [==============================] - 8s 140us/sample - loss: 1.5537 - acc: 0.4574 - va
Epoch 3/25
60000/60000 [==============================] - 8s 140us/sample - loss: 1.1184 - acc: 0.6223 - va
Epoch 4/25
60000/60000 [==============================] - 8s 139us/sample - loss: 0.9041 - acc: 0.7059 - va
```

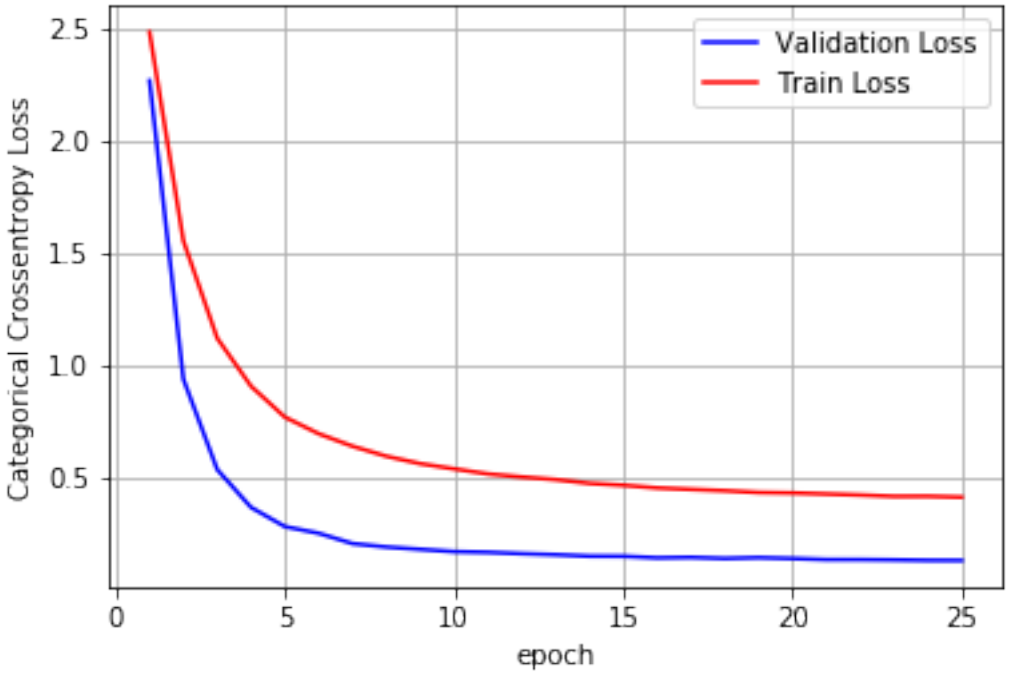
```
Epoch 5/25
60000/60000 [==============================] - 8s 138us/sample - loss: 0.7661 - acc: 0.7604 - va
Epoch 6/25
60000/60000 [==============================] - 8s 139us/sample - loss: 0.6918 - acc: 0.7858 - va
Epoch 7/25
60000/60000 [==============================] - 8s 139us/sample - loss: 0.6360 - acc: 0.8064 - va
Epoch 8/25
60000/60000 [==============================] - 8s 139us/sample - loss: 0.5914 - acc: 0.8265 - va
Epoch 9/25
60000/60000 [==============================] - 8s 137us/sample - loss: 0.5582 - acc: 0.8349 - va
Epoch 10/25
60000/60000 [==============================] - 8s 136us/sample - loss: 0.5350 - acc: 0.8443 - va
Epoch 11/25
60000/60000 [==============================] - 8s 136us/sample - loss: 0.5124 - acc: 0.8513 - va
Epoch 12/25
60000/60000 [==============================] - 8s 139us/sample - loss: 0.4988 - acc: 0.8566 - va
Epoch 13/25
60000/60000 [==============================] - 8s 139us/sample - loss: 0.4872 - acc: 0.8616 - va
Epoch 14/25
60000/60000 [==============================] - 8s 139us/sample - loss: 0.4703 - acc: 0.8667 - va
Epoch 15/25
60000/60000 [==============================] - 8s 138us/sample - loss: 0.4619 - acc: 0.8695 - va
Epoch 16/25
60000/60000 [==============================] - 8s 137us/sample - loss: 0.4499 - acc: 0.8731 - va
Epoch 17/25
60000/60000 [==============================] - 8s 140us/sample - loss: 0.4434 - acc: 0.8752 - va
Epoch 18/25
60000/60000 [==============================] - 8s 140us/sample - loss: 0.4376 - acc: 0.8776 - va
Epoch 19/25
60000/60000 [==============================] - 8s 139us/sample - loss: 0.4296 - acc: 0.8812 - va
Epoch 20/25
60000/60000 [==============================] - 8s 139us/sample - loss: 0.4274 - acc: 0.8802 - va
Epoch 21/25
60000/60000 [==============================] - 9s 142us/sample - loss: 0.4229 - acc: 0.8820 - va
Epoch 22/25
60000/60000 [==============================] - 8s 139us/sample - loss: 0.4182 - acc: 0.8842 - va
Epoch 23/25
60000/60000 [==============================] - 8s 139us/sample - loss: 0.4122 - acc: 0.8852 - va
Epoch 24/25
60000/60000 [==============================] - 8s 139us/sample - loss: 0.4126 - acc: 0.8870 - va
Epoch 25/25
60000/60000 [==============================] - 8s 139us/sample - loss: 0.4088 - acc: 0.8870 - va
```

```
In [29]: print("************************************************")
         print("Printing the Model Summary")
         print(m33_model.summary())
         print("************************************************")
```

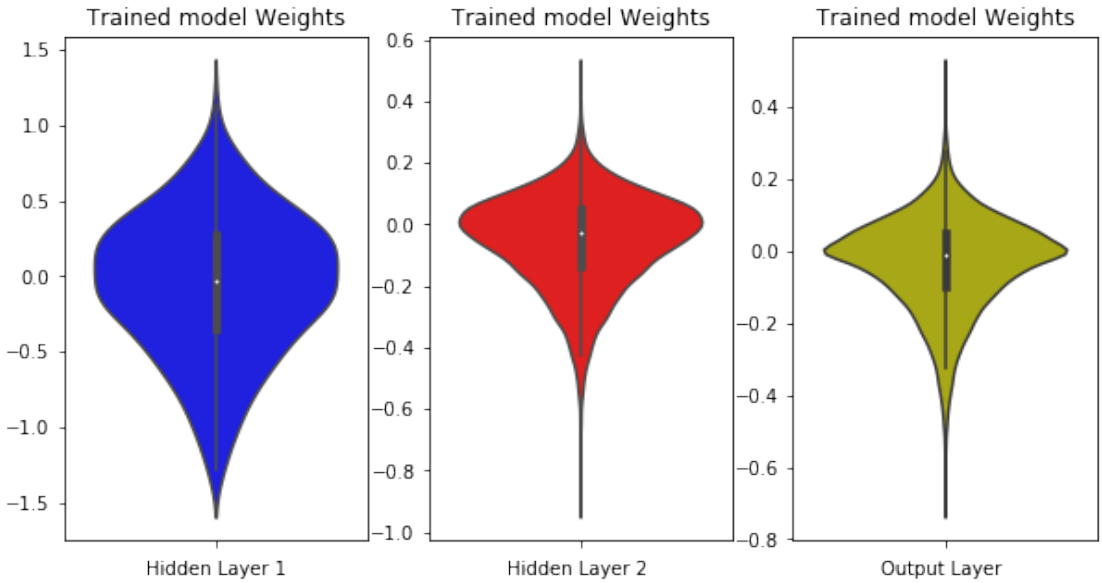
```
        score = m33_model.evaluate(x_test_new, y_test)

        print('Test score:', score[0])
        print('Test accuracy:', score[1])

        final_output = final_output.append({"#ConvNets": 3,
                                            "#Kernels/Filters": '3x3',
                                            "Padding": '-',
                                            "Stride": '2x2',
                                            "Dropout": True,
                                            "BatchNormalization": True,
                                            "Regularization": 'L2 (0.00001)',
                                            "TRAIN_LOSS": '{:.5f}'.format(model.history["loss"]
                                            "TEST_LOSS": '{:.5f}'.format(model.history["val_los
                                            "TRAIN_ACC": '{:.5f}'.format(model.history["acc"][n
                                            "TEST_ACC": '{:.5f}'.format(model.history["val_acc"

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

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

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

```
**********************************************
Printing the Model Summary
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_14 (Conv2D)           (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 13, 13, 32)        0
_________________________________________________________________
dropout_8 (Dropout)          (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 5, 5, 64)          0
_________________________________________________________________
dropout_9 (Dropout)          (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 3, 3, 32)          18464
_________________________________________________________________
```

```
max_pooling2d_16 (MaxPooling (None, 1, 1, 32)          0
_________________________________________________________________
dropout_10 (Dropout)         (None, 1, 1, 32)          0
_________________________________________________________________
flatten_6 (Flatten)          (None, 32)                0
_________________________________________________________________
dense_18 (Dense)             (None, 128)               4224
_________________________________________________________________
batch_normalization_v1_4 (Ba (None, 128)               512
_________________________________________________________________
dropout_11 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_19 (Dense)             (None, 64)                8256
_________________________________________________________________
batch_normalization_v1_5 (Ba (None, 64)                256
_________________________________________________________________
dropout_12 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_20 (Dense)             (None, 10)                650
=================================================================
Total params: 51,178
Trainable params: 50,794
Non-trainable params: 384
_________________________________________________________________
None
**********************************************
10000/10000 [==============================] - 1s 112us/sample - loss: 0.1261 - acc: 0.9679
Test score: 0.1261051001906395
Test accuracy: 0.9679
```



In [30]:
```
w_after = m33_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(figsize=(10, 5))
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()
```



2.2.4 5x5 Kernel/Filter with Dropout, Weight Regularization/ initialization and Batch Normalization

In [31]:
```
m34_model = Sequential()

    # First Convolutional layer with MaxPooling
    m34_model.add(Conv2D(32, kernel_size=(5,5), activation=tf.nn.relu, padding="same", inpu
                         kernel_initializer='he_normal', kernel_regularizer=tf.keras.regula
    m34_model.add(MaxPooling2D(pool_size=(2,2), padding="same"))

    m34_model.add(Dropout(rate=0.5))

    # Second Convolutional layer with MaxPooling
    m34_model.add(Conv2D(64, kernel_size=(5,5), activation=tf.nn.relu, padding="same",
                         kernel_initializer='he_normal', kernel_regularizer=tf.keras.regular
    m34_model.add(MaxPooling2D(pool_size=(2,2), padding="same"))

    m34_model.add(Dropout(rate=0.5))

    # Third Convolutional layer with MaxPooling
    m34_model.add(Conv2D(32, kernel_size=(5,5), activation=tf.nn.relu, padding="same",
                         kernel_initializer='he_normal', kernel_regularizer=tf.keras.regular
    m34_model.add(MaxPooling2D(pool_size=(2,2), padding="same"))

    m34_model.add(Dropout(rate=0.5))

    # Fourth Convolutional layer with MaxPooling
```

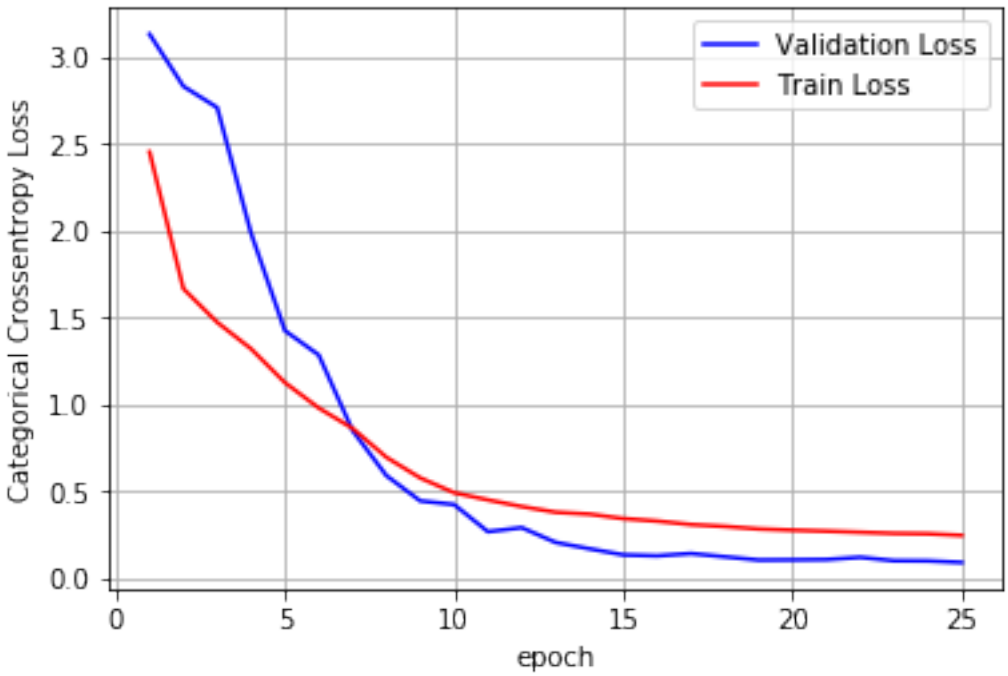
```
        m34_model.add(Conv2D(32, kernel_size=(5,5), activation=tf.nn.relu, padding="same",
                             kernel_initializer='he_normal', kernel_regularizer=tf.keras.regular
        m34_model.add(MaxPooling2D(pool_size=(2,2)))

        m34_model.add(Dropout(rate=0.5))

        # Fifth Convolutional layer with MaxPooling
        m34_model.add(Conv2D(32, kernel_size=(5,5), activation=tf.nn.relu, padding="same",
                             kernel_initializer='he_normal', kernel_regularizer=tf.keras.regular
        m34_model.add(MaxPooling2D(pool_size=(2,2), padding="same"))

        m34_model.add(Dropout(rate=0.5))

        # Three dense layers in MLP
        m34_model.add(Flatten())
        m34_model.add(Dense(128, activation=tf.nn.relu, kernel_initializer='he_normal', kernel_
        m34_model.add(BatchNormalization())
        m34_model.add(Dropout(rate=0.5))
        m34_model.add(Dense(64, activation=tf.nn.relu, kernel_initializer='he_normal', kernel_r
        m34_model.add(BatchNormalization())
        m34_model.add(Dropout(rate=0.5))
        m34_model.add(Dense(10, activation=tf.nn.softmax))


        m34_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy
        model = m34_model.fit(x_train_new, y_train, epochs=n_epochs, batch_size=batchsize, verb

Train on 60000 samples, validate on 10000 samples
Epoch 1/25
60000/60000 [==============================] - 12s 199us/sample - loss: 2.4554 - acc: 0.1821 - v
Epoch 2/25
60000/60000 [==============================] - 11s 178us/sample - loss: 1.6683 - acc: 0.3192 - v
Epoch 3/25
60000/60000 [==============================] - 11s 178us/sample - loss: 1.4748 - acc: 0.3810 - v
Epoch 4/25
60000/60000 [==============================] - 11s 178us/sample - loss: 1.3223 - acc: 0.4378 - v
Epoch 5/25
60000/60000 [==============================] - 11s 178us/sample - loss: 1.1281 - acc: 0.5334 - v
Epoch 6/25
60000/60000 [==============================] - 11s 178us/sample - loss: 0.9818 - acc: 0.6104 - v
Epoch 7/25
60000/60000 [==============================] - 11s 178us/sample - loss: 0.8646 - acc: 0.6875 - v
Epoch 8/25
60000/60000 [==============================] - 11s 179us/sample - loss: 0.6982 - acc: 0.7766 - v
Epoch 9/25
60000/60000 [==============================] - 11s 179us/sample - loss: 0.5794 - acc: 0.8276 - v
Epoch 10/25
60000/60000 [==============================] - 11s 179us/sample - loss: 0.4943 - acc: 0.8582 - v
```

```
Epoch 11/25
60000/60000 [==============================] - 11s 178us/sample - loss: 0.4528 - acc: 0.8742 - v
Epoch 12/25
60000/60000 [==============================] - 11s 178us/sample - loss: 0.4146 - acc: 0.8893 - v
Epoch 13/25
60000/60000 [==============================] - 11s 180us/sample - loss: 0.3819 - acc: 0.8985 - v
Epoch 14/25
60000/60000 [==============================] - 11s 178us/sample - loss: 0.3710 - acc: 0.9036 - v
Epoch 15/25
60000/60000 [==============================] - 11s 178us/sample - loss: 0.3453 - acc: 0.9120 - v
Epoch 16/25
60000/60000 [==============================] - 11s 178us/sample - loss: 0.3310 - acc: 0.9179 - v
Epoch 17/25
60000/60000 [==============================] - 11s 178us/sample - loss: 0.3109 - acc: 0.9244 - v
Epoch 18/25
60000/60000 [==============================] - 11s 180us/sample - loss: 0.3003 - acc: 0.9275 - v
Epoch 19/25
60000/60000 [==============================] - 11s 179us/sample - loss: 0.2858 - acc: 0.9311 - v
Epoch 20/25
60000/60000 [==============================] - 11s 179us/sample - loss: 0.2788 - acc: 0.9354 - v
Epoch 21/25
60000/60000 [==============================] - 11s 179us/sample - loss: 0.2730 - acc: 0.9369 - v
Epoch 22/25
60000/60000 [==============================] - 11s 179us/sample - loss: 0.2666 - acc: 0.9392 - v
Epoch 23/25
60000/60000 [==============================] - 11s 179us/sample - loss: 0.2602 - acc: 0.9403 - v
Epoch 24/25
60000/60000 [==============================] - 11s 180us/sample - loss: 0.2580 - acc: 0.9419 - v
Epoch 25/25
60000/60000 [==============================] - 11s 179us/sample - loss: 0.2488 - acc: 0.9449 - v
```

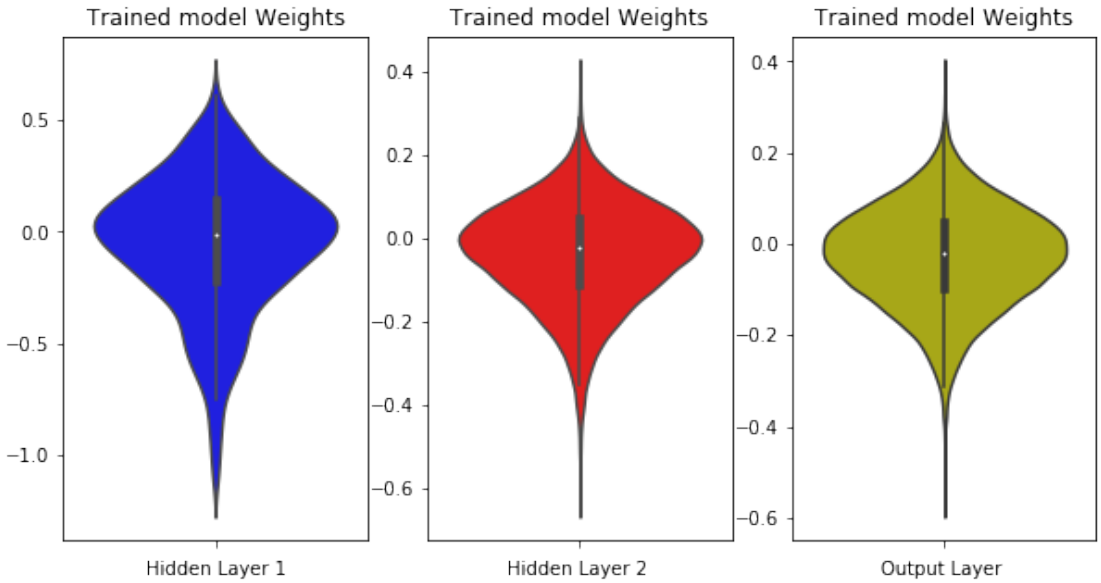
```
In [32]: print("************************************************")
         print("Printing the Model Summary")
         print(m34_model.summary())
         print("************************************************")

         score = m34_model.evaluate(x_test_new, y_test)

         print('Test score:', score[0])
         print('Test accuracy:', score[1])

         final_output = final_output.append({"#ConvNets": 3,
                                             "#Kernels/Filters": '5x5',
                                             "Padding": 'same',
                                             "Stride": '2x2',
                                             "Dropout": True,
                                             "BatchNormalization": True,
```

```
                                                "Regularization": 'L2 (0.00001)',
                                                "TRAIN_LOSS": '{:.5f}'.format(model.history["loss"]
                                                "TEST_LOSS": '{:.5f}'.format(model.history["val_los
                                                "TRAIN_ACC": '{:.5f}'.format(model.history["acc"][n
                                                "TEST_ACC": '{:.5f}'.format(model.history["val_acc"

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

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

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

**********************************************
Printing the Model Summary
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_17 (Conv2D)           (None, 28, 28, 32)        832
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 14, 14, 32)        0
_________________________________________________________________
dropout_13 (Dropout)         (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 14, 14, 64)        51264
_________________________________________________________________
max_pooling2d_18 (MaxPooling (None, 7, 7, 64)          0
_________________________________________________________________
dropout_14 (Dropout)         (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 7, 7, 32)          51232
_________________________________________________________________
max_pooling2d_19 (MaxPooling (None, 4, 4, 32)          0
_________________________________________________________________
dropout_15 (Dropout)         (None, 4, 4, 32)          0
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 4, 4, 32)          25632
_________________________________________________________________
max_pooling2d_20 (MaxPooling (None, 2, 2, 32)          0
_________________________________________________________________
dropout_16 (Dropout)         (None, 2, 2, 32)          0
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 2, 2, 32)          25632
_________________________________________________________________
```

```
max_pooling2d_21 (MaxPooling (None, 1, 1, 32)          0
_________________________________________________________________
dropout_17 (Dropout)         (None, 1, 1, 32)          0
_________________________________________________________________
flatten_7 (Flatten)          (None, 32)                0
_________________________________________________________________
dense_21 (Dense)             (None, 128)               4224
_________________________________________________________________
batch_normalization_v1_6 (Ba (None, 128)               512
_________________________________________________________________
dropout_18 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_22 (Dense)             (None, 64)                8256
_________________________________________________________________
batch_normalization_v1_7 (Ba (None, 64)                256
_________________________________________________________________
dropout_19 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_23 (Dense)             (None, 10)                650
=================================================================
Total params: 168,490
Trainable params: 168,106
Non-trainable params: 384
_________________________________________________________________
None
**********************************************
10000/10000 [==============================] - 1s 143us/sample - loss: 0.0930 - acc: 0.9821
Test score: 0.0929575339615345
Test accuracy: 0.9821
```



In [33]:
```
w_after = m34_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(figsize=(10, 5))
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()
```



2.3 5 ConvNet Architecture

In this type of architecture, we will be trying out 5 convolution layers along with 5 maxpooling layer for each.

2.3.1 3x3 Kernel/Filter

In [34]:
```
m51_model = Sequential()

        # First Convolutional layer with MaxPooling
        m51_model.add(Conv2D(32, kernel_size=(3,3), padding="same", activation=tf.nn.relu, inpu
        m51_model.add(MaxPooling2D(pool_size=(2,2), padding="same"))

        # Second Convolutional layer with MaxPooling
        m51_model.add(Conv2D(64, kernel_size=(3,3), padding="same", activation=tf.nn.relu))
        m51_model.add(MaxPooling2D(pool_size=(2,2), padding="same"))

        # Third Convolutional layer with MaxPooling
        m51_model.add(Conv2D(64, kernel_size=(3,3), padding="same", activation=tf.nn.relu))
        m51_model.add(MaxPooling2D(pool_size=(2,2), padding="same"))

        # Fourth Convolutional layer with MaxPooling
        m51_model.add(Conv2D(64, kernel_size=(3,3), padding="same", activation=tf.nn.relu))
        m51_model.add(MaxPooling2D(pool_size=(2,2), padding="same"))

        # Fifth Convolutional layer with MaxPooling
        m51_model.add(Conv2D(32, kernel_size=(3,3), padding="same", activation=tf.nn.relu))
        m51_model.add(MaxPooling2D(pool_size=(2,2), padding="same"))
```

```
        # Three dense layers in MLP
        m51_model.add(Flatten())
        m51_model.add(Dense(128, activation=tf.nn.relu))
        m51_model.add(BatchNormalization())
        m51_model.add(Dropout(rate=0.5))
        m51_model.add(Dense(64, activation=tf.nn.relu))
        m51_model.add(BatchNormalization())
        m51_model.add(Dropout(rate=0.5))
        m51_model.add(Dense(10, activation=tf.nn.softmax))


        m51_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy
        model = m51_model.fit(x_train_new, y_train, epochs=n_epochs, batch_size=batchsize, verb
```

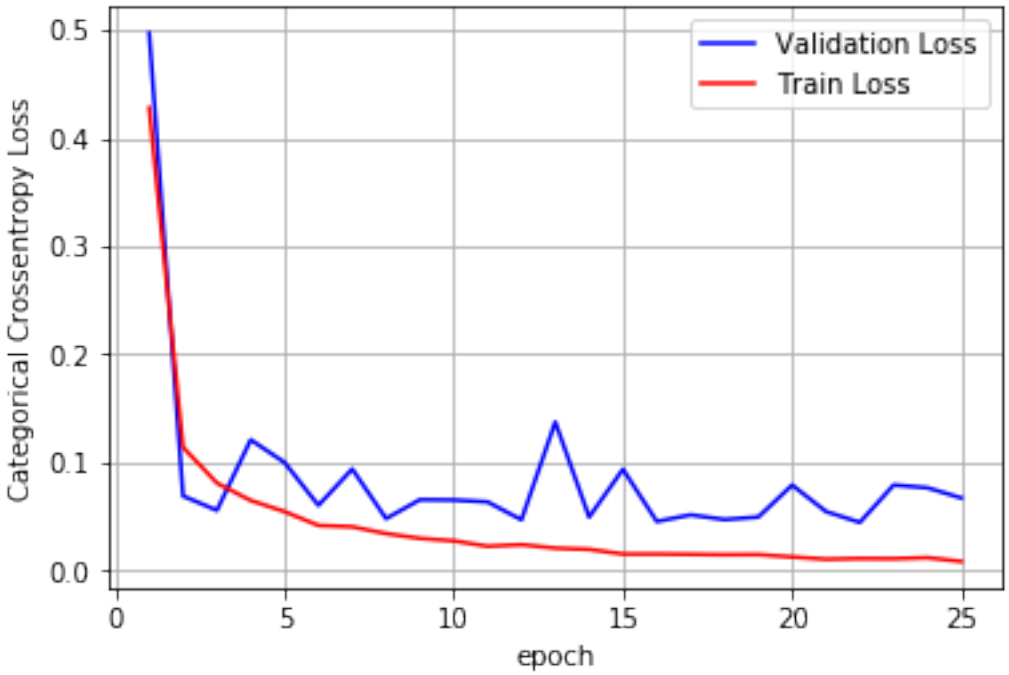
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/25
60000/60000 [==============================] - 11s 191us/sample - loss: 0.4281 - acc: 0.8713 - v
Epoch 2/25
60000/60000 [==============================] - 10s 169us/sample - loss: 0.1139 - acc: 0.9693 - v
Epoch 3/25
60000/60000 [==============================] - 10s 169us/sample - loss: 0.0808 - acc: 0.9776 - v
Epoch 4/25
60000/60000 [==============================] - 10s 169us/sample - loss: 0.0646 - acc: 0.9819 - v
Epoch 5/25
60000/60000 [==============================] - 10s 169us/sample - loss: 0.0543 - acc: 0.9851 - v
Epoch 6/25
60000/60000 [==============================] - 10s 168us/sample - loss: 0.0415 - acc: 0.9882 - v
Epoch 7/25
60000/60000 [==============================] - 10s 166us/sample - loss: 0.0403 - acc: 0.9884 - v
Epoch 8/25
60000/60000 [==============================] - 10s 165us/sample - loss: 0.0339 - acc: 0.9904 - v
Epoch 9/25
60000/60000 [==============================] - 10s 166us/sample - loss: 0.0296 - acc: 0.9913 - v
Epoch 10/25
60000/60000 [==============================] - 10s 166us/sample - loss: 0.0272 - acc: 0.9923 - v
Epoch 11/25
60000/60000 [==============================] - 10s 167us/sample - loss: 0.0222 - acc: 0.9931 - v
Epoch 12/25
60000/60000 [==============================] - 10s 166us/sample - loss: 0.0237 - acc: 0.9932 - v
Epoch 13/25
60000/60000 [==============================] - 10s 167us/sample - loss: 0.0205 - acc: 0.9935 - v
Epoch 14/25
60000/60000 [==============================] - 10s 167us/sample - loss: 0.0193 - acc: 0.9945 - v
Epoch 15/25
60000/60000 [==============================] - 10s 167us/sample - loss: 0.0151 - acc: 0.9956 - v
Epoch 16/25
60000/60000 [==============================] - 10s 167us/sample - loss: 0.0151 - acc: 0.9955 - v
```

```
Epoch 17/25
60000/60000 [==============================] - 10s 166us/sample - loss: 0.0147 - acc: 0.9954 - v
Epoch 18/25
60000/60000 [==============================] - 10s 167us/sample - loss: 0.0144 - acc: 0.9954 - v
Epoch 19/25
60000/60000 [==============================] - 10s 167us/sample - loss: 0.0145 - acc: 0.9956 - v
Epoch 20/25
60000/60000 [==============================] - 10s 166us/sample - loss: 0.0123 - acc: 0.9965 - v
Epoch 21/25
60000/60000 [==============================] - 10s 167us/sample - loss: 0.0103 - acc: 0.9969 - v
Epoch 22/25
60000/60000 [==============================] - 10s 167us/sample - loss: 0.0109 - acc: 0.9967 - v
Epoch 23/25
60000/60000 [==============================] - 10s 167us/sample - loss: 0.0107 - acc: 0.9966 - v
Epoch 24/25
60000/60000 [==============================] - 10s 167us/sample - loss: 0.0117 - acc: 0.9967 - v
Epoch 25/25
60000/60000 [==============================] - 10s 167us/sample - loss: 0.0082 - acc: 0.9976 - v
```

```
In [35]: print("************************************************")
         print("Printing the Model Summary")
         print(m51_model.summary())
         print("************************************************")

         score = m51_model.evaluate(x_test_new, y_test)

         print('Test score:', score[0])
         print('Test accuracy:', score[1])

         final_output = final_output.append({"#ConvNets": 5,
                                             "#Kernels/Filters": '3x3',
                                             "Padding": 'same',
                                             "Stride": '2x2',
                                             "Dropout": False,
                                             "BatchNormalization": False,
                                             "Regularization": '-',
                                             "TRAIN_LOSS": '{:.5f}'.format(model.history["loss"]
                                             "TEST_LOSS": '{:.5f}'.format(model.history["val_los
                                             "TRAIN_ACC": '{:.5f}'.format(model.history["acc"][n
                                             "TEST_ACC": '{:.5f}'.format(model.history["val_acc"

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

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

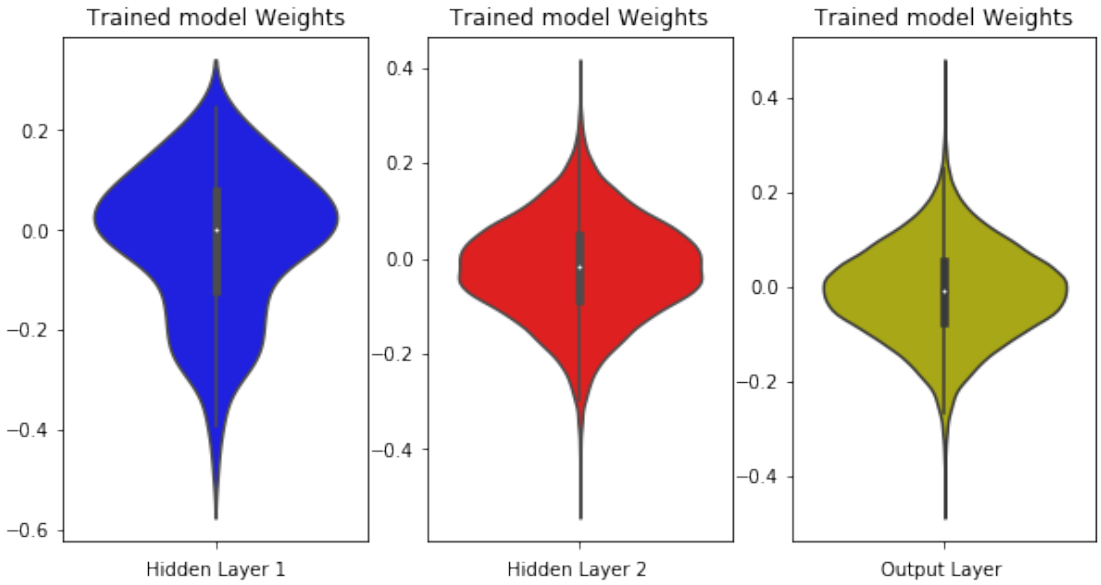
```
        vy = model.history['val_loss']
        ty = model.history['loss']
        plt_dynamic(x, vy, ty, ax)

**********************************************
Printing the Model Summary
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_22 (Conv2D)           (None, 28, 28, 32)        320
_________________________________________________________________
max_pooling2d_22 (MaxPooling (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 14, 14, 64)        18496
_________________________________________________________________
max_pooling2d_23 (MaxPooling (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_24 (Conv2D)           (None, 7, 7, 64)          36928
_________________________________________________________________
max_pooling2d_24 (MaxPooling (None, 4, 4, 64)          0
_________________________________________________________________
conv2d_25 (Conv2D)           (None, 4, 4, 64)          36928
_________________________________________________________________
max_pooling2d_25 (MaxPooling (None, 2, 2, 64)          0
_________________________________________________________________
conv2d_26 (Conv2D)           (None, 2, 2, 32)          18464
_________________________________________________________________
max_pooling2d_26 (MaxPooling (None, 1, 1, 32)          0
_________________________________________________________________
flatten_8 (Flatten)          (None, 32)                0
_________________________________________________________________
dense_24 (Dense)             (None, 128)               4224
_________________________________________________________________
batch_normalization_v1_8 (Ba (None, 128)               512
_________________________________________________________________
dropout_20 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_25 (Dense)             (None, 64)                8256
_________________________________________________________________
batch_normalization_v1_9 (Ba (None, 64)                256
_________________________________________________________________
dropout_21 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_26 (Dense)             (None, 10)                650
=================================================================
Total params: 125,034
Trainable params: 124,650
```

```
Non-trainable params: 384
_________________________________________________________________
None
**********************************************
10000/10000 [==============================] - 1s 123us/sample - loss: 0.0669 - acc: 0.9879
Test score: 0.06689615963574229
Test accuracy: 0.9879
```



In [36]:
```
w_after = m51_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(figsize=(10, 5))
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()
```



2.3.2 5x5 Kernel/Filter

In [37]:
```
m52_model = Sequential()

# First Convolutional layer with MaxPooling
m52_model.add(Conv2D(32, kernel_size=(5,5), padding="same", activation=tf.nn.relu, inpu
m52_model.add(MaxPooling2D(pool_size=(2,2), padding="same"))

# Second Convolutional layer with MaxPooling
m52_model.add(Conv2D(64, kernel_size=(5,5), padding="same", activation=tf.nn.relu))
m52_model.add(MaxPooling2D(pool_size=(2,2), padding="same"))

# Third Convolutional layer with MaxPooling
m52_model.add(Conv2D(64, kernel_size=(5,5), padding="same", activation=tf.nn.relu))
m52_model.add(MaxPooling2D(pool_size=(2,2), padding="same"))

# Fourth Convolutional layer with MaxPooling
m52_model.add(Conv2D(64, kernel_size=(5,5), padding="same", activation=tf.nn.relu))
m52_model.add(MaxPooling2D(pool_size=(2,2), padding="same"))
```

```
        # Fifth Convolutional layer with MaxPooling
        m52_model.add(Conv2D(32, kernel_size=(5,5), padding="same", activation=tf.nn.relu))
        m52_model.add(MaxPooling2D(pool_size=(2,2), padding="same"))

        # Three dense layers in MLP
        m52_model.add(Flatten())
        m52_model.add(Dense(128, activation=tf.nn.relu))
        m52_model.add(BatchNormalization())
        m52_model.add(Dropout(rate=0.5))
        m52_model.add(Dense(64, activation=tf.nn.relu))
        m52_model.add(BatchNormalization())
        m52_model.add(Dropout(rate=0.5))
        m52_model.add(Dense(10, activation=tf.nn.softmax))


        m52_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy
        model = m52_model.fit(x_train_new, y_train, epochs=n_epochs, batch_size=batchsize, verb
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/25
60000/60000 [==============================] - 12s 203us/sample - loss: 0.4521 - acc: 0.8658 - v
Epoch 2/25
60000/60000 [==============================] - 11s 180us/sample - loss: 0.1033 - acc: 0.9724 - v
Epoch 3/25
60000/60000 [==============================] - 11s 181us/sample - loss: 0.0690 - acc: 0.9812 - v
Epoch 4/25
60000/60000 [==============================] - 11s 183us/sample - loss: 0.0572 - acc: 0.9846 - v
Epoch 5/25
60000/60000 [==============================] - 11s 181us/sample - loss: 0.0458 - acc: 0.9878 - v
Epoch 6/25
60000/60000 [==============================] - 11s 180us/sample - loss: 0.0402 - acc: 0.9888 - v
Epoch 7/25
60000/60000 [==============================] - 11s 181us/sample - loss: 0.0326 - acc: 0.9909 - v
Epoch 8/25
60000/60000 [==============================] - 11s 181us/sample - loss: 0.0287 - acc: 0.9921 - v
Epoch 9/25
60000/60000 [==============================] - 11s 180us/sample - loss: 0.0272 - acc: 0.9922 - v
Epoch 10/25
60000/60000 [==============================] - 11s 181us/sample - loss: 0.0218 - acc: 0.9939 - v
Epoch 11/25
60000/60000 [==============================] - 11s 181us/sample - loss: 0.0188 - acc: 0.9945 - v
Epoch 12/25
60000/60000 [==============================] - 11s 181us/sample - loss: 0.0171 - acc: 0.9950 - v
Epoch 13/25
60000/60000 [==============================] - 11s 181us/sample - loss: 0.0152 - acc: 0.9959 - v
Epoch 14/25
60000/60000 [==============================] - 11s 180us/sample - loss: 0.0146 - acc: 0.9958 - v
```

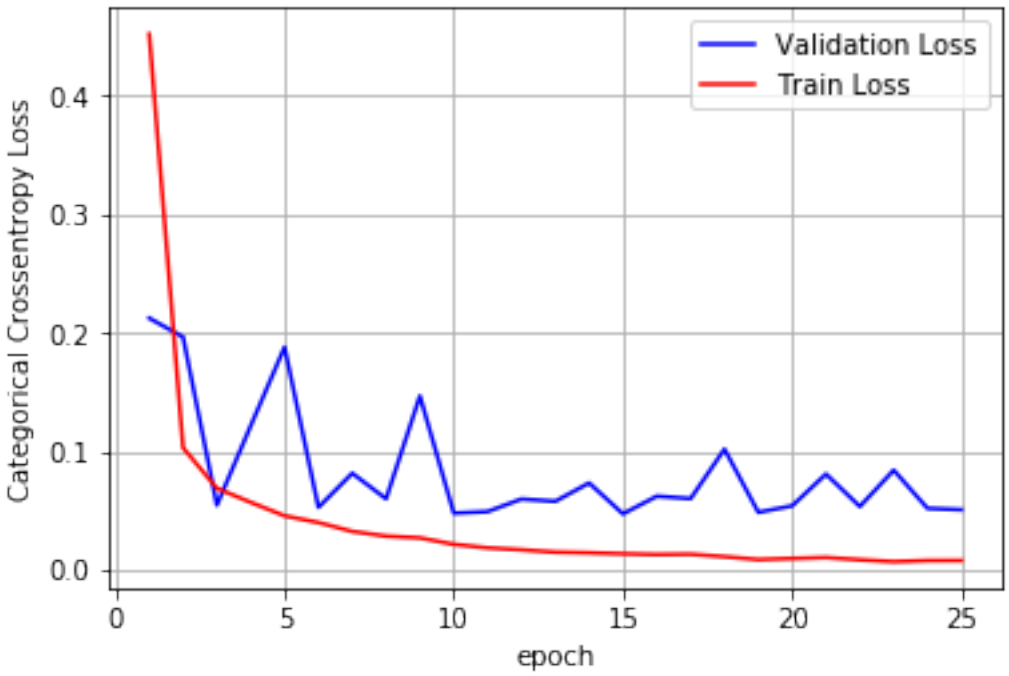
```
Epoch 15/25
60000/60000 [==============================] - 11s 181us/sample - loss: 0.0138 - acc: 0.9958 - v
Epoch 16/25
60000/60000 [==============================] - 11s 181us/sample - loss: 0.0132 - acc: 0.9961 - v
Epoch 17/25
60000/60000 [==============================] - 11s 180us/sample - loss: 0.0134 - acc: 0.9964 - v
Epoch 18/25
60000/60000 [==============================] - 11s 181us/sample - loss: 0.0113 - acc: 0.9969 - v
Epoch 19/25
60000/60000 [==============================] - 11s 181us/sample - loss: 0.0089 - acc: 0.9974 - v
Epoch 20/25
60000/60000 [==============================] - 11s 181us/sample - loss: 0.0098 - acc: 0.9972 - v
Epoch 21/25
60000/60000 [==============================] - 11s 180us/sample - loss: 0.0106 - acc: 0.9972 - v
Epoch 22/25
60000/60000 [==============================] - 11s 181us/sample - loss: 0.0088 - acc: 0.9975 - v
Epoch 23/25
60000/60000 [==============================] - 11s 181us/sample - loss: 0.0072 - acc: 0.9979 - v
Epoch 24/25
60000/60000 [==============================] - 11s 181us/sample - loss: 0.0081 - acc: 0.9976 - v
Epoch 25/25
60000/60000 [==============================] - 11s 181us/sample - loss: 0.0081 - acc: 0.9976 - v
```

```
In [38]: print("************************************************")
         print("Printing the Model Summary")
         print(m52_model.summary())
         print("************************************************")

         score = m52_model.evaluate(x_test_new, y_test)

         print('Test score:', score[0])
         print('Test accuracy:', score[1])

         final_output = final_output.append({"#ConvNets": 5,
                                             "#Kernels/Filters": '5x5',
                                             "Padding": 'same',
                                             "Stride": '2x2',
                                             "Dropout": False,
                                             "BatchNormalization": False,
                                             "Regularization": '-',
                                             "TRAIN_LOSS": '{:.5f}'.format(model.history["loss"]
                                             "TEST_LOSS": '{:.5f}'.format(model.history["val_los
                                             "TRAIN_ACC": '{:.5f}'.format(model.history["acc"][n
                                             "TEST_ACC": '{:.5f}'.format(model.history["val_acc"

         fig,ax = plt.subplots(1,1)
         ax.set_xlabel('epoch')
```

```
        ax.set_ylabel('Categorical Crossentropy Loss')

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

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

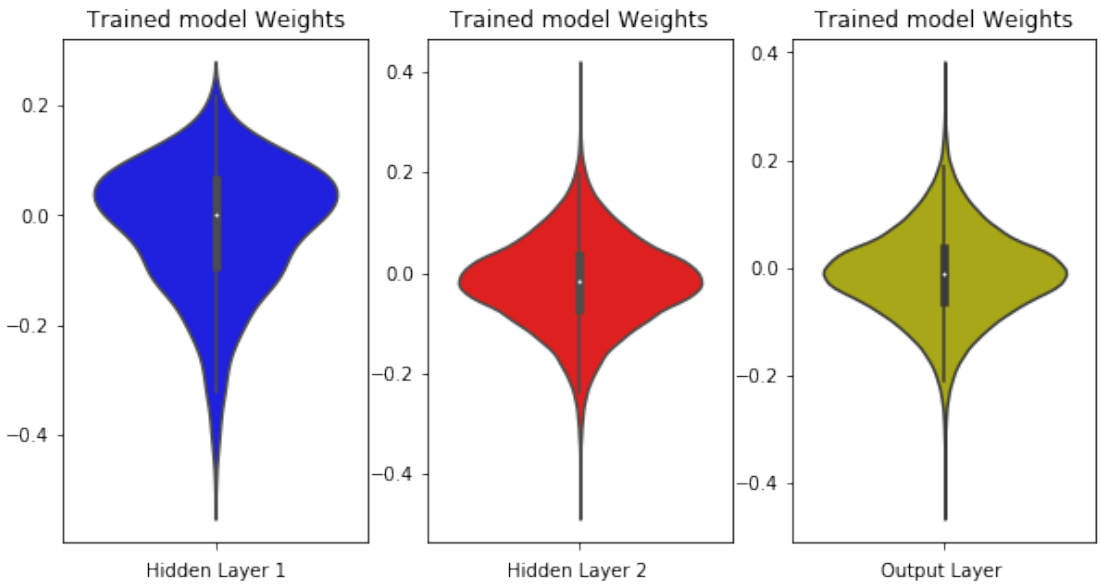
```
*********************************************
Printing the Model Summary
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_27 (Conv2D)           (None, 28, 28, 32)        832
_________________________________________________________________
max_pooling2d_27 (MaxPooling (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_28 (Conv2D)           (None, 14, 14, 64)        51264
_________________________________________________________________
max_pooling2d_28 (MaxPooling (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_29 (Conv2D)           (None, 7, 7, 64)          102464
_________________________________________________________________
max_pooling2d_29 (MaxPooling (None, 4, 4, 64)          0
_________________________________________________________________
conv2d_30 (Conv2D)           (None, 4, 4, 64)          102464
_________________________________________________________________
max_pooling2d_30 (MaxPooling (None, 2, 2, 64)          0
_________________________________________________________________
conv2d_31 (Conv2D)           (None, 2, 2, 32)          51232
_________________________________________________________________
max_pooling2d_31 (MaxPooling (None, 1, 1, 32)          0
_________________________________________________________________
flatten_9 (Flatten)          (None, 32)                0
_________________________________________________________________
dense_27 (Dense)             (None, 128)               4224
_________________________________________________________________
batch_normalization_v1_10 (B (None, 128)               512
_________________________________________________________________
dropout_22 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_28 (Dense)             (None, 64)                8256
_________________________________________________________________
batch_normalization_v1_11 (B (None, 64)                256
_________________________________________________________________
dropout_23 (Dropout)         (None, 64)                0
_________________________________________________________________
```

```
dense_29 (Dense)             (None, 10)                650
=================================================================
Total params: 322,154
Trainable params: 321,770
Non-trainable params: 384
_________________________________________________________________
None
**********************************************
10000/10000 [==============================] - 1s 145us/sample - loss: 0.0510 - acc: 0.9899
Test score: 0.0509517434184585
Test accuracy: 0.9899
```




```
In [39]: w_after = m52_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(figsize=(10, 5))
         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()
```



2.3.3 3x3 Filter/Kernel with Dropout and Weight Regularization & Initialization and Batch Normalization

In [40]:
```
m53_model = Sequential()

# First Convolutional layer with MaxPooling
m53_model.add(Conv2D(32, kernel_size=(3,3), padding="same", activation=tf.nn.relu, inpu
                     kernel_initializer='he_normal', kernel_regularizer=tf.keras.regular
m53_model.add(MaxPooling2D(pool_size=(2,2), padding="same"))

m53_model.add(Dropout(rate=0.5))

# Second Convolutional layer with MaxPooling
m53_model.add(Conv2D(64, kernel_size=(3,3), padding="same", activation=tf.nn.relu,
                     kernel_initializer='he_normal', kernel_regularizer=tf.keras.regular
```

```
        m53_model.add(MaxPooling2D(pool_size=(2,2), padding="same"))

        m53_model.add(Dropout(rate=0.5))

        # Third Convolutional layer with MaxPooling
        m53_model.add(Conv2D(64, kernel_size=(3,3), padding="same", activation=tf.nn.relu,
                             kernel_initializer='he_normal', kernel_regularizer=tf.keras.regular
        m53_model.add(MaxPooling2D(pool_size=(2,2), padding="same"))

        m53_model.add(Dropout(rate=0.5))

        # Fourth Convolutional layer with MaxPooling
        m53_model.add(Conv2D(64, kernel_size=(3,3), padding="same", activation=tf.nn.relu,
                             kernel_initializer='he_normal', kernel_regularizer=tf.keras.regular
        m53_model.add(MaxPooling2D(pool_size=(2,2), padding="same"))

        m53_model.add(Dropout(rate=0.5))

        # Fifth Convolutional layer with MaxPooling
        m53_model.add(Conv2D(32, kernel_size=(3,3), padding="same", activation=tf.nn.relu,
                             kernel_initializer='he_normal', kernel_regularizer=tf.keras.regular
        m53_model.add(MaxPooling2D(pool_size=(2,2), padding="same"))

        m53_model.add(Dropout(rate=0.5))

        # Three dense layers in MLP
        m53_model.add(Flatten())
        m53_model.add(Dense(128, activation=tf.nn.relu, kernel_initializer='he_normal', kernel_
        m53_model.add(BatchNormalization())
        m53_model.add(Dropout(rate=0.5))
        m53_model.add(Dense(64, activation=tf.nn.relu, kernel_initializer='he_normal', kernel_r
        m53_model.add(BatchNormalization())
        m53_model.add(Dropout(rate=0.5))
        m53_model.add(Dense(10, activation=tf.nn.softmax))


        m53_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy
        model = m53_model.fit(x_train_new, y_train, epochs=n_epochs, batch_size=batchsize, verb

Train on 60000 samples, validate on 10000 samples
Epoch 1/25
60000/60000 [==============================] - 13s 212us/sample - loss: 2.5089 - acc: 0.1661 - v
Epoch 2/25
60000/60000 [==============================] - 11s 186us/sample - loss: 1.8547 - acc: 0.2749 - v
Epoch 3/25
60000/60000 [==============================] - 11s 186us/sample - loss: 1.5292 - acc: 0.3967 - v
Epoch 4/25
60000/60000 [==============================] - 11s 186us/sample - loss: 1.2578 - acc: 0.5101 - v
```

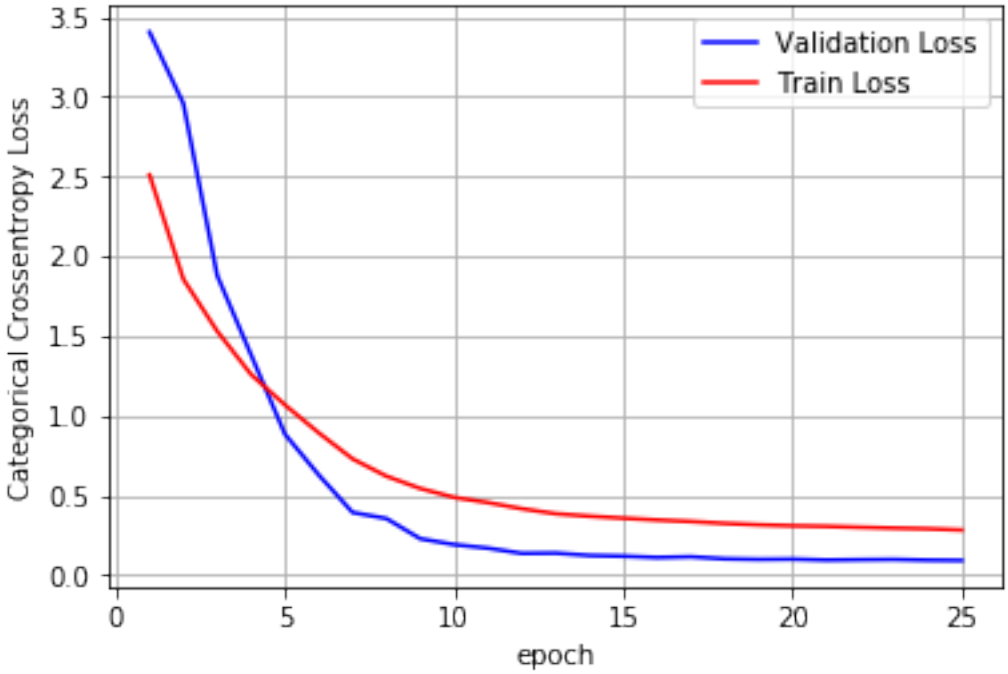
```
Epoch 5/25
60000/60000 [==============================] - 11s 186us/sample - loss: 1.0671 - acc: 0.6069 - v
Epoch 6/25
60000/60000 [==============================] - 11s 186us/sample - loss: 0.8918 - acc: 0.6987 - v
Epoch 7/25
60000/60000 [==============================] - 11s 184us/sample - loss: 0.7286 - acc: 0.7781 - v
Epoch 8/25
60000/60000 [==============================] - 11s 183us/sample - loss: 0.6200 - acc: 0.8219 - v
Epoch 9/25
60000/60000 [==============================] - 11s 182us/sample - loss: 0.5430 - acc: 0.8506 - v
Epoch 10/25
60000/60000 [==============================] - 11s 183us/sample - loss: 0.4870 - acc: 0.8700 - v
Epoch 11/25
60000/60000 [==============================] - 11s 183us/sample - loss: 0.4545 - acc: 0.8814 - v
Epoch 12/25
60000/60000 [==============================] - 11s 183us/sample - loss: 0.4160 - acc: 0.8951 - v
Epoch 13/25
60000/60000 [==============================] - 11s 183us/sample - loss: 0.3847 - acc: 0.9025 - v
Epoch 14/25
60000/60000 [==============================] - 11s 184us/sample - loss: 0.3691 - acc: 0.9079 - v
Epoch 15/25
60000/60000 [==============================] - 11s 182us/sample - loss: 0.3567 - acc: 0.9126 - v
Epoch 16/25
60000/60000 [==============================] - 11s 183us/sample - loss: 0.3454 - acc: 0.9169 - v
Epoch 17/25
60000/60000 [==============================] - 11s 183us/sample - loss: 0.3358 - acc: 0.9183 - v
Epoch 18/25
60000/60000 [==============================] - 11s 183us/sample - loss: 0.3239 - acc: 0.9235 - v
Epoch 19/25
60000/60000 [==============================] - 11s 182us/sample - loss: 0.3151 - acc: 0.9244 - v
Epoch 20/25
60000/60000 [==============================] - 11s 182us/sample - loss: 0.3090 - acc: 0.9269 - v
Epoch 21/25
60000/60000 [==============================] - 11s 183us/sample - loss: 0.3054 - acc: 0.9293 - v
Epoch 22/25
60000/60000 [==============================] - 11s 184us/sample - loss: 0.2996 - acc: 0.9308 - v
Epoch 23/25
60000/60000 [==============================] - 11s 184us/sample - loss: 0.2945 - acc: 0.9324 - v
Epoch 24/25
60000/60000 [==============================] - 11s 184us/sample - loss: 0.2903 - acc: 0.9322 - v
Epoch 25/25
60000/60000 [==============================] - 11s 184us/sample - loss: 0.2826 - acc: 0.9348 - v
```

```
In [41]: print("************************************************")
         print("Printing the Model Summary")
         print(m53_model.summary())
         print("************************************************")
```

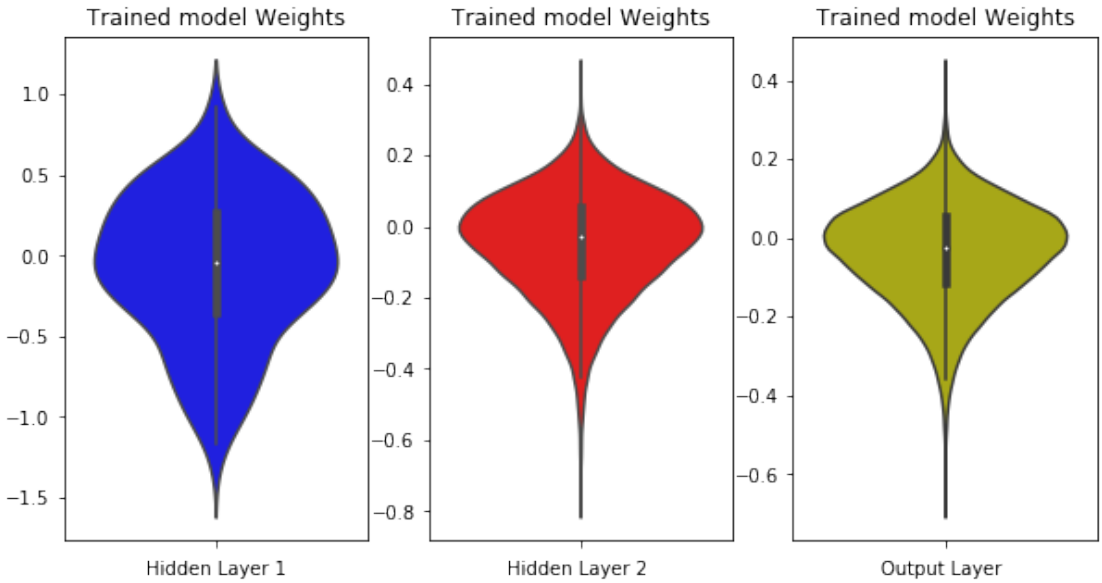
```
        score = m53_model.evaluate(x_test_new, y_test)

        print('Test score:', score[0])
        print('Test accuracy:', score[1])

        final_output = final_output.append({"#ConvNets": 5,
                                            "#Kernels/Filters": '3x3',
                                            "Padding": 'same',
                                            "Stride": '2x2',
                                            "Dropout": True,
                                            "BatchNormalization": True,
                                            "Regularization": 'L2 (0.00001)',
                                            "TRAIN_LOSS": '{:.5f}'.format(model.history["loss"]
                                            "TEST_LOSS": '{:.5f}'.format(model.history["val_los
                                            "TRAIN_ACC": '{:.5f}'.format(model.history["acc"][n
                                            "TEST_ACC": '{:.5f}'.format(model.history["val_acc"

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

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

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

```
**********************************************
Printing the Model Summary
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_32 (Conv2D)           (None, 28, 28, 32)        320
_________________________________________________________________
max_pooling2d_32 (MaxPooling (None, 14, 14, 32)        0
_________________________________________________________________
dropout_24 (Dropout)         (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_33 (Conv2D)           (None, 14, 14, 64)        18496
_________________________________________________________________
max_pooling2d_33 (MaxPooling (None, 7, 7, 64)          0
_________________________________________________________________
dropout_25 (Dropout)         (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_34 (Conv2D)           (None, 7, 7, 64)          36928
_________________________________________________________________
```

```
max_pooling2d_34 (MaxPooling (None, 4, 4, 64)          0
_________________________________________________________________
dropout_26 (Dropout)         (None, 4, 4, 64)          0
_________________________________________________________________
conv2d_35 (Conv2D)           (None, 4, 4, 64)          36928
_________________________________________________________________
max_pooling2d_35 (MaxPooling (None, 2, 2, 64)          0
_________________________________________________________________
dropout_27 (Dropout)         (None, 2, 2, 64)          0
_________________________________________________________________
conv2d_36 (Conv2D)           (None, 2, 2, 32)          18464
_________________________________________________________________
max_pooling2d_36 (MaxPooling (None, 1, 1, 32)          0
_________________________________________________________________
dropout_28 (Dropout)         (None, 1, 1, 32)          0
_________________________________________________________________
flatten_10 (Flatten)         (None, 32)                0
_________________________________________________________________
dense_30 (Dense)             (None, 128)               4224
_________________________________________________________________
batch_normalization_v1_12 (B (None, 128)               512
_________________________________________________________________
dropout_29 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_31 (Dense)             (None, 64)                8256
_________________________________________________________________
batch_normalization_v1_13 (B (None, 64)                256
_________________________________________________________________
dropout_30 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_32 (Dense)             (None, 10)                650
=================================================================
Total params: 125,034
Trainable params: 124,650
Non-trainable params: 384
_________________________________________________________________
None
**********************************************
10000/10000 [==============================] - 1s 136us/sample - loss: 0.0917 - acc: 0.9828
Test score: 0.09166334240734578
Test accuracy: 0.9828
```



In [42]:
```
w_after = m53_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(figsize=(10, 5))
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()
```



2.3.4 5x5 Kernel/Filter with Dropout, Weight Regularization/ initialization and Batch Normalization

```
In [43]: m54_model = Sequential()

         # First Convolutional layer with MaxPooling
         m54_model.add(Conv2D(32, kernel_size=(5,5), padding="same", activation=tf.nn.relu, inpu
                              kernel_initializer='he_normal', kernel_regularizer=tf.keras.regular
         m54_model.add(MaxPooling2D(pool_size=(2,2), padding="same"))

         m54_model.add(Dropout(rate=0.5))

         # Second Convolutional layer with MaxPooling
         m54_model.add(Conv2D(64, kernel_size=(5,5), padding="same", activation=tf.nn.relu,
                              kernel_initializer='he_normal', kernel_regularizer=tf.keras.regular
         m54_model.add(MaxPooling2D(pool_size=(2,2), padding="same"))

         m54_model.add(Dropout(rate=0.5))

         # Third Convolutional layer with MaxPooling
         m54_model.add(Conv2D(64, kernel_size=(5,5), padding="same", activation=tf.nn.relu,
                              kernel_initializer='he_normal', kernel_regularizer=tf.keras.regular
         m54_model.add(MaxPooling2D(pool_size=(2,2), padding="same"))

         m54_model.add(Dropout(rate=0.5))

         # Fourth Convolutional layer with MaxPooling
```

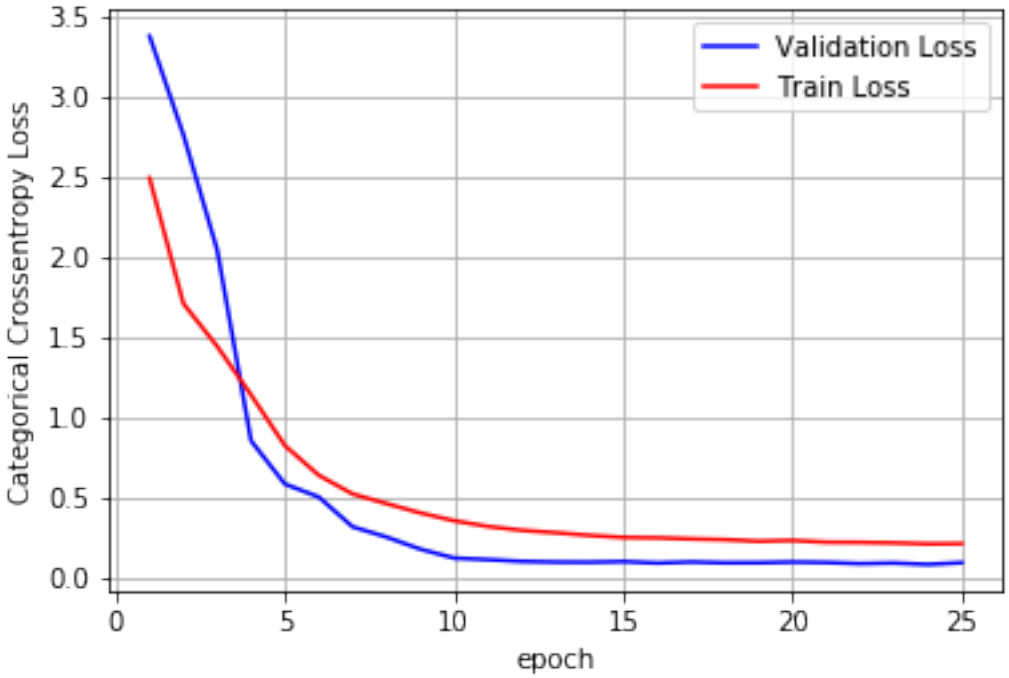
```
        m54_model.add(Conv2D(64, kernel_size=(5,5), padding="same", activation=tf.nn.relu,
                             kernel_initializer='he_normal', kernel_regularizer=tf.keras.regular
        m54_model.add(MaxPooling2D(pool_size=(2,2), padding="same"))

        m54_model.add(Dropout(rate=0.5))

        # Fifth Convolutional layer with MaxPooling
        m54_model.add(Conv2D(32, kernel_size=(5,5), padding="same", activation=tf.nn.relu,
                             kernel_initializer='he_normal', kernel_regularizer=tf.keras.regular
        m54_model.add(MaxPooling2D(pool_size=(2,2), padding="same"))

        m54_model.add(Dropout(rate=0.5))

        # Three dense layers in MLP
        m54_model.add(Flatten())
        m54_model.add(Dense(128, activation=tf.nn.relu, kernel_initializer='he_normal', kernel_
        m54_model.add(BatchNormalization())
        m54_model.add(Dropout(rate=0.5))
        m54_model.add(Dense(64, activation=tf.nn.relu, kernel_initializer='he_normal', kernel_r
        m54_model.add(BatchNormalization())
        m54_model.add(Dropout(rate=0.5))
        m54_model.add(Dense(10, activation=tf.nn.softmax))


        m54_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy
        model = m54_model.fit(x_train_new, y_train, epochs=n_epochs, batch_size=batchsize, verb

Train on 60000 samples, validate on 10000 samples
Epoch 1/25
60000/60000 [==============================] - 13s 225us/sample - loss: 2.4886 - acc: 0.1789 - v
Epoch 2/25
60000/60000 [==============================] - 12s 198us/sample - loss: 1.7058 - acc: 0.3115 - v
Epoch 3/25
60000/60000 [==============================] - 12s 198us/sample - loss: 1.4386 - acc: 0.4155 - v
Epoch 4/25
60000/60000 [==============================] - 12s 198us/sample - loss: 1.1361 - acc: 0.5479 - v
Epoch 5/25
60000/60000 [==============================] - 12s 199us/sample - loss: 0.8199 - acc: 0.7095 - v
Epoch 6/25
60000/60000 [==============================] - 12s 198us/sample - loss: 0.6351 - acc: 0.7962 - v
Epoch 7/25
60000/60000 [==============================] - 12s 198us/sample - loss: 0.5185 - acc: 0.8469 - v
Epoch 8/25
60000/60000 [==============================] - 12s 198us/sample - loss: 0.4583 - acc: 0.8717 - v
Epoch 9/25
60000/60000 [==============================] - 12s 196us/sample - loss: 0.4002 - acc: 0.8941 - v
Epoch 10/25
60000/60000 [==============================] - 12s 198us/sample - loss: 0.3514 - acc: 0.9144 - v
```

```
Epoch 11/25
60000/60000 [==============================] - 12s 197us/sample - loss: 0.3157 - acc: 0.9266 - v
Epoch 12/25
60000/60000 [==============================] - 12s 197us/sample - loss: 0.2918 - acc: 0.9331 - v
Epoch 13/25
60000/60000 [==============================] - 12s 197us/sample - loss: 0.2768 - acc: 0.9384 - v
Epoch 14/25
60000/60000 [==============================] - 12s 198us/sample - loss: 0.2599 - acc: 0.9420 - v
Epoch 15/25
60000/60000 [==============================] - 12s 198us/sample - loss: 0.2473 - acc: 0.9459 - v
Epoch 16/25
60000/60000 [==============================] - 12s 197us/sample - loss: 0.2449 - acc: 0.9464 - v
Epoch 17/25
60000/60000 [==============================] - 12s 198us/sample - loss: 0.2383 - acc: 0.9498 - v
Epoch 18/25
60000/60000 [==============================] - 12s 198us/sample - loss: 0.2336 - acc: 0.9503 - v
Epoch 19/25
60000/60000 [==============================] - 12s 198us/sample - loss: 0.2237 - acc: 0.9534 - v
Epoch 20/25
60000/60000 [==============================] - 12s 197us/sample - loss: 0.2286 - acc: 0.9526 - v
Epoch 21/25
60000/60000 [==============================] - 12s 197us/sample - loss: 0.2173 - acc: 0.9558 - v
Epoch 22/25
60000/60000 [==============================] - 12s 198us/sample - loss: 0.2166 - acc: 0.9561 - v
Epoch 23/25
60000/60000 [==============================] - 12s 198us/sample - loss: 0.2131 - acc: 0.9567 - v
Epoch 24/25
60000/60000 [==============================] - 12s 198us/sample - loss: 0.2074 - acc: 0.9584 - v
Epoch 25/25
60000/60000 [==============================] - 12s 197us/sample - loss: 0.2086 - acc: 0.9590 - v
```

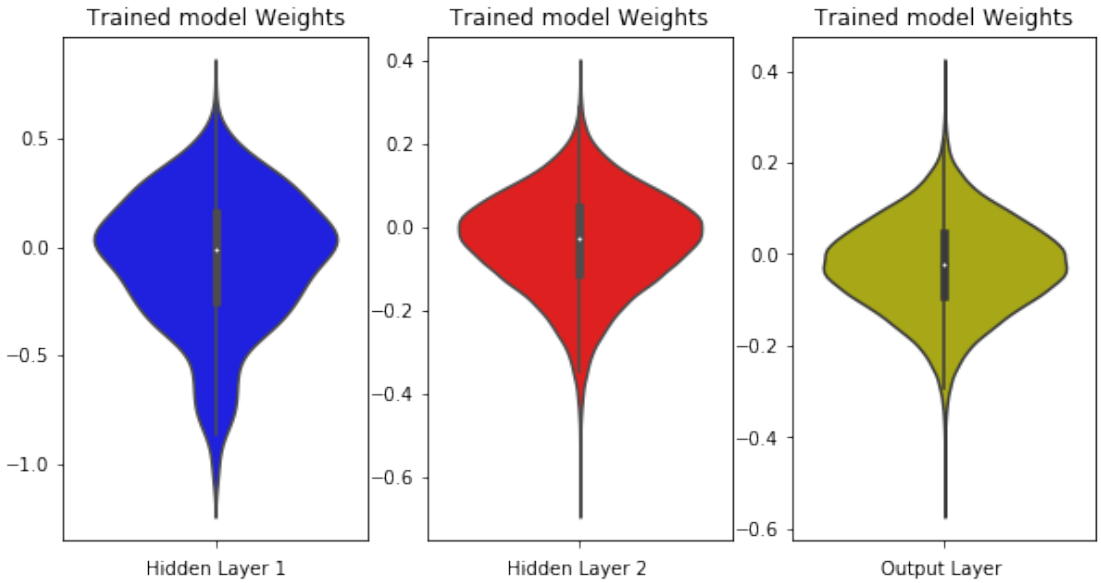
```
In [44]: print("************************************************")
         print("Printing the Model Summary")
         print(m54_model.summary())
         print("************************************************")

         score = m54_model.evaluate(x_test_new, y_test)

         print('Test score:', score[0])
         print('Test accuracy:', score[1])

         final_output = final_output.append({"#ConvNets": 5,
                                             "#Kernels/Filters": '5x5',
                                             "Padding": 'same',
                                             "Stride": '2x2',
                                             "Dropout": True,
                                             "BatchNormalization": True,
```

```
                                                "Regularization": 'L2 (0.00001)',
                                                "TRAIN_LOSS": '{:.5f}'.format(model.history["loss"]
                                                "TEST_LOSS": '{:.5f}'.format(model.history["val_los
                                                "TRAIN_ACC": '{:.5f}'.format(model.history["acc"][n
                                                "TEST_ACC": '{:.5f}'.format(model.history["val_acc"

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

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

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

**********************************************
Printing the Model Summary
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_37 (Conv2D)           (None, 28, 28, 32)        832
_________________________________________________________________
max_pooling2d_37 (MaxPooling (None, 14, 14, 32)        0
_________________________________________________________________
dropout_31 (Dropout)         (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_38 (Conv2D)           (None, 14, 14, 64)        51264
_________________________________________________________________
max_pooling2d_38 (MaxPooling (None, 7, 7, 64)          0
_________________________________________________________________
dropout_32 (Dropout)         (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_39 (Conv2D)           (None, 7, 7, 64)          102464
_________________________________________________________________
max_pooling2d_39 (MaxPooling (None, 4, 4, 64)          0
_________________________________________________________________
dropout_33 (Dropout)         (None, 4, 4, 64)          0
_________________________________________________________________
conv2d_40 (Conv2D)           (None, 4, 4, 64)          102464
_________________________________________________________________
max_pooling2d_40 (MaxPooling (None, 2, 2, 64)          0
_________________________________________________________________
dropout_34 (Dropout)         (None, 2, 2, 64)          0
_________________________________________________________________
conv2d_41 (Conv2D)           (None, 2, 2, 32)          51232
_________________________________________________________________
```

```
max_pooling2d_41 (MaxPooling (None, 1, 1, 32)          0
_________________________________________________________________
dropout_35 (Dropout)         (None, 1, 1, 32)          0
_________________________________________________________________
flatten_11 (Flatten)         (None, 32)                0
_________________________________________________________________
dense_33 (Dense)             (None, 128)               4224
_________________________________________________________________
batch_normalization_v1_14 (B (None, 128)               512
_________________________________________________________________
dropout_36 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_34 (Dense)             (None, 64)                8256
_________________________________________________________________
batch_normalization_v1_15 (B (None, 64)                256
_________________________________________________________________
dropout_37 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_35 (Dense)             (None, 10)                650
=================================================================
Total params: 322,154
Trainable params: 321,770
Non-trainable params: 384
_________________________________________________________________
None
**********************************************
10000/10000 [==============================] - 2s 162us/sample - loss: 0.0901 - acc: 0.9868
Test score: 0.0901069624900818
Test accuracy: 0.9868
```



In [45]:
```
w_after = m54_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(figsize=(10, 5))
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()
```



2.4 7 Convnet

In this type of architecture, we will be trying out 7 convolution layer along with 2 maxpooling layer after 3 convolution layers.

2.4.1 3x3 Filter/Kernel with Dropout and Weight Regularization & Initialization and Batch Normalization

In [46]:
```
m71_model = Sequential()

# First Convolutional layer
m71_model.add(Conv2D(32, kernel_size=(3,3), padding="same", activation=tf.nn.relu, inpu
                     kernel_initializer='he_normal', kernel_regularizer=tf.keras.regular

m71_model.add(Dropout(rate=0.5))

# Second Convolutional layer
m71_model.add(Conv2D(64, kernel_size=(3,3), padding="same", activation=tf.nn.relu,
                     kernel_initializer='he_normal', kernel_regularizer=tf.keras.regular

m71_model.add(Dropout(rate=0.5))

# Third Convolutional layer with MaxPooling
m71_model.add(Conv2D(64, kernel_size=(3,3), padding="same", activation=tf.nn.relu,
                     kernel_initializer='he_normal', kernel_regularizer=tf.keras.regular

m71_model.add(MaxPooling2D(pool_size=(2,2), padding="same"))
```

```
        m71_model.add(Dropout(rate=0.5))

        # Fourth Convolutional layer
        m71_model.add(Conv2D(64, kernel_size=(3,3), padding="same", activation=tf.nn.relu,
                             kernel_initializer='he_normal', kernel_regularizer=tf.keras.regular

        m71_model.add(Dropout(rate=0.5))

        # Fifth Convolutional layer
        m71_model.add(Conv2D(64, kernel_size=(3,3), padding="same", activation=tf.nn.relu,
                             kernel_initializer='he_normal', kernel_regularizer=tf.keras.regular

        m71_model.add(Dropout(rate=0.5))

        # Sixth Convolutional layer with MaxPooling
        m71_model.add(Conv2D(32, kernel_size=(3,3), padding="same", activation=tf.nn.relu,
                             kernel_initializer='he_normal', kernel_regularizer=tf.keras.regular

        m71_model.add(Dropout(rate=0.5))

        m71_model.add(MaxPooling2D(pool_size=(2,2), padding="same"))

        # Seventh Convolutional layer
        m71_model.add(Conv2D(32, kernel_size=(3,3), padding="same", activation=tf.nn.relu,
                             kernel_initializer='he_normal', kernel_regularizer=tf.keras.regular

        m71_model.add(Dropout(rate=0.5))

        # Three dense layers in MLP
        m71_model.add(Flatten())
        m71_model.add(Dense(128, activation=tf.nn.relu, kernel_initializer='he_normal', kernel_
        m71_model.add(BatchNormalization())
        m71_model.add(Dropout(rate=0.5))
        m71_model.add(Dense(64, activation=tf.nn.relu, kernel_initializer='he_normal', kernel_r
        m71_model.add(BatchNormalization())
        m71_model.add(Dropout(rate=0.5))
        m71_model.add(Dense(10, activation=tf.nn.softmax))


        m71_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy
        model = m71_model.fit(x_train_new, y_train, epochs=n_epochs, batch_size=batchsize, verb

Train on 60000 samples, validate on 10000 samples
Epoch 1/25
60000/60000 [==============================] - 30s 501us/sample - loss: 1.4084 - acc: 0.5541 - v
Epoch 2/25
60000/60000 [==============================] - 28s 464us/sample - loss: 0.5161 - acc: 0.8457 - v
Epoch 3/25
```

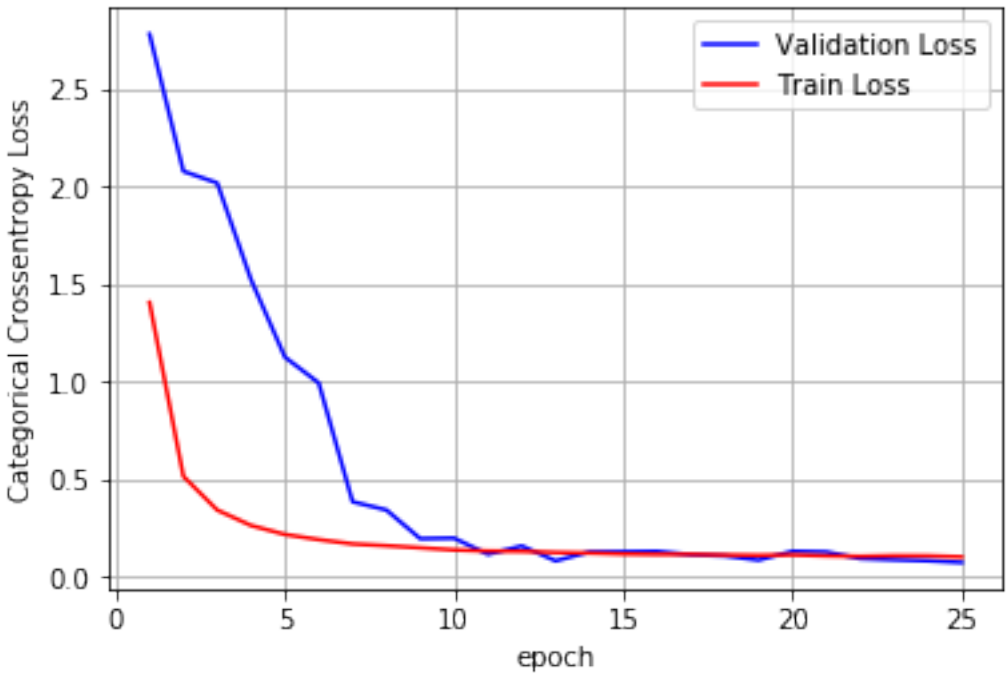
```
60000/60000 [==============================] - 28s 462us/sample - loss: 0.3433 - acc: 0.9033 - v
Epoch 4/25
60000/60000 [==============================] - 28s 462us/sample - loss: 0.2640 - acc: 0.9283 - v
Epoch 5/25
60000/60000 [==============================] - 28s 463us/sample - loss: 0.2180 - acc: 0.9416 - v
Epoch 6/25
60000/60000 [==============================] - 28s 464us/sample - loss: 0.1915 - acc: 0.9501 - v
Epoch 7/25
60000/60000 [==============================] - 28s 461us/sample - loss: 0.1694 - acc: 0.9563 - v
Epoch 8/25
60000/60000 [==============================] - 28s 463us/sample - loss: 0.1603 - acc: 0.9595 - v
Epoch 9/25
60000/60000 [==============================] - 28s 460us/sample - loss: 0.1495 - acc: 0.9631 - v
Epoch 10/25
60000/60000 [==============================] - 28s 464us/sample - loss: 0.1387 - acc: 0.9662 - v
Epoch 11/25
60000/60000 [==============================] - 28s 462us/sample - loss: 0.1329 - acc: 0.9676 - v
Epoch 12/25
60000/60000 [==============================] - 28s 462us/sample - loss: 0.1314 - acc: 0.9697 - v
Epoch 13/25
60000/60000 [==============================] - 28s 463us/sample - loss: 0.1255 - acc: 0.9706 - v
Epoch 14/25
60000/60000 [==============================] - 28s 464us/sample - loss: 0.1228 - acc: 0.9716 - v
Epoch 15/25
60000/60000 [==============================] - 28s 462us/sample - loss: 0.1180 - acc: 0.9728 - v
Epoch 16/25
60000/60000 [==============================] - 28s 462us/sample - loss: 0.1159 - acc: 0.9739 - v
Epoch 17/25
60000/60000 [==============================] - 28s 462us/sample - loss: 0.1161 - acc: 0.9737 - v
Epoch 18/25
60000/60000 [==============================] - 28s 463us/sample - loss: 0.1127 - acc: 0.9750 - v
Epoch 19/25
60000/60000 [==============================] - 28s 462us/sample - loss: 0.1108 - acc: 0.9763 - v
Epoch 20/25
60000/60000 [==============================] - 28s 461us/sample - loss: 0.1133 - acc: 0.9757 - v
Epoch 21/25
60000/60000 [==============================] - 28s 462us/sample - loss: 0.1084 - acc: 0.9777 - v
Epoch 22/25
60000/60000 [==============================] - 28s 461us/sample - loss: 0.1056 - acc: 0.9779 - v
Epoch 23/25
60000/60000 [==============================] - 28s 461us/sample - loss: 0.1087 - acc: 0.9772 - v
Epoch 24/25
60000/60000 [==============================] - 28s 462us/sample - loss: 0.1081 - acc: 0.9783 - v
Epoch 25/25
60000/60000 [==============================] - 28s 460us/sample - loss: 0.1029 - acc: 0.9798 - v
```

```
In [47]: print("**********************************************")
```

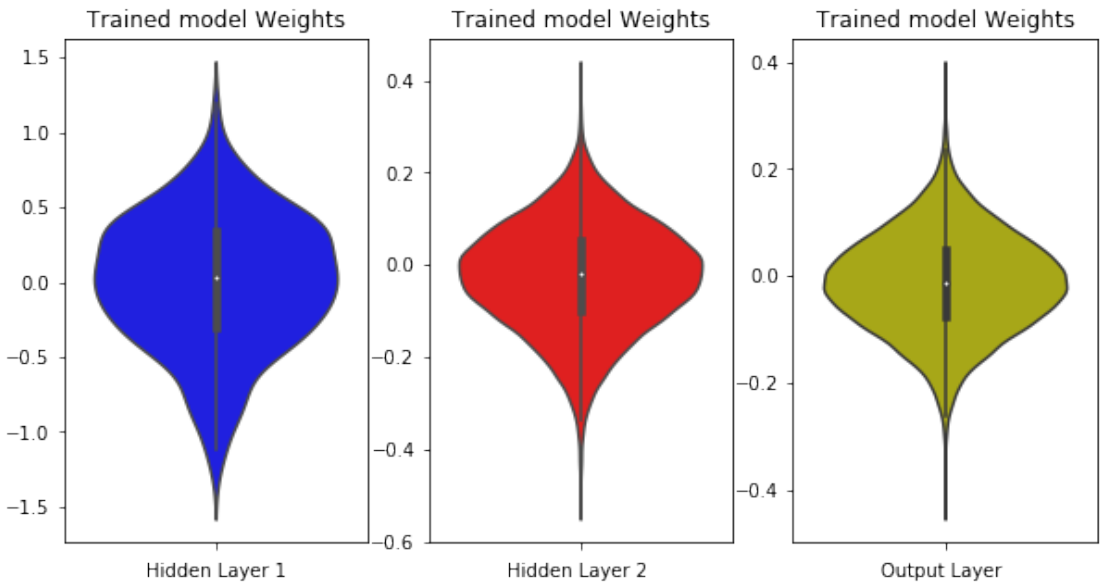
```
        print("Printing the Model Summary")
        print(m71_model.summary())
        print("***********************************************")

        score = m71_model.evaluate(x_test_new, y_test)

        print('Test score:', score[0])
        print('Test accuracy:', score[1])

        final_output = final_output.append({"#ConvNets": 7,
                                            "#Kernels/Filters": '3x3',
                                            "Padding": 'same',
                                            "Stride": '2x2',
                                            "Dropout": True,
                                            "BatchNormalization": True,
                                            "Regularization": 'L2 (0.00001)',
                                            "TRAIN_LOSS": '{:.5f}'.format(model.history["loss"]
                                            "TEST_LOSS": '{:.5f}'.format(model.history["val_los
                                            "TRAIN_ACC": '{:.5f}'.format(model.history["acc"][n
                                            "TEST_ACC": '{:.5f}'.format(model.history["val_acc"

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

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

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

```
***********************************************
Printing the Model Summary
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_42 (Conv2D)           (None, 28, 28, 32)        320
_________________________________________________________________
dropout_38 (Dropout)         (None, 28, 28, 32)        0
_________________________________________________________________
conv2d_43 (Conv2D)           (None, 28, 28, 64)        18496
_________________________________________________________________
dropout_39 (Dropout)         (None, 28, 28, 64)        0
_________________________________________________________________
conv2d_44 (Conv2D)           (None, 28, 28, 64)        36928
_________________________________________________________________
max_pooling2d_42 (MaxPooling (None, 14, 14, 64)        0
```

```
_________________________________________________________________
dropout_40 (Dropout)         (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_45 (Conv2D)           (None, 14, 14, 64)        36928
_________________________________________________________________
dropout_41 (Dropout)         (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_46 (Conv2D)           (None, 14, 14, 64)        36928
_________________________________________________________________
dropout_42 (Dropout)         (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_47 (Conv2D)           (None, 14, 14, 32)        18464
_________________________________________________________________
dropout_43 (Dropout)         (None, 14, 14, 32)        0
_________________________________________________________________
max_pooling2d_43 (MaxPooling (None, 7, 7, 32)          0
_________________________________________________________________
conv2d_48 (Conv2D)           (None, 7, 7, 32)          9248
_________________________________________________________________
dropout_44 (Dropout)         (None, 7, 7, 32)          0
_________________________________________________________________
flatten_12 (Flatten)         (None, 1568)              0
_________________________________________________________________
dense_36 (Dense)             (None, 128)               200832
_________________________________________________________________
batch_normalization_v1_16 (B (None, 128)               512
_________________________________________________________________
dropout_45 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_37 (Dense)             (None, 64)                8256
_________________________________________________________________
batch_normalization_v1_17 (B (None, 64)                256
_________________________________________________________________
dropout_46 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_38 (Dense)             (None, 10)                650
=================================================================
Total params: 367,818
Trainable params: 367,434
Non-trainable params: 384
_________________________________________________________________
None
**********************************************
10000/10000 [==============================] - 2s 218us/sample - loss: 0.0743 - acc: 0.9879
Test score: 0.07434590795636177
Test accuracy: 0.9879
```



In [48]:
```
w_after = m71_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(figsize=(10, 5))
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()
```



2.4.2 5x5 Filter/Kernel with Dropout and Weight Regularization & Initialization and Batch Normalization

In [49]:
```
m72_model = Sequential()

    # First Convolutional layer
    m72_model.add(Conv2D(32, kernel_size=(5,5), padding="same", activation=tf.nn.relu, inpu
                         kernel_initializer='he_normal', kernel_regularizer=tf.keras.regular

    m72_model.add(Dropout(rate=0.5))

    # Second Convolutional layer
    m72_model.add(Conv2D(64, kernel_size=(5,5), padding="same", activation=tf.nn.relu,
                         kernel_initializer='he_normal', kernel_regularizer=tf.keras.regular

    m72_model.add(Dropout(rate=0.5))

    # Third Convolutional layer with MaxPooling
    m72_model.add(Conv2D(64, kernel_size=(5,5), padding="same", activation=tf.nn.relu,
                         kernel_initializer='he_normal', kernel_regularizer=tf.keras.regular


    m72_model.add(MaxPooling2D(pool_size=(2,2), padding="same"))

    m72_model.add(Dropout(rate=0.5))

    # Fourth Convolutional layer
```

```
        m72_model.add(Conv2D(64, kernel_size=(5,5), padding="same", activation=tf.nn.relu,
                             kernel_initializer='he_normal', kernel_regularizer=tf.keras.regular

        m72_model.add(Dropout(rate=0.5))

        # Fifth Convolutional layer
        m72_model.add(Conv2D(64, kernel_size=(5,5), padding="same", activation=tf.nn.relu,
                             kernel_initializer='he_normal', kernel_regularizer=tf.keras.regular

        m72_model.add(Dropout(rate=0.5))

        # Sixth Convolutional layer with MaxPooling
        m72_model.add(Conv2D(32, kernel_size=(5,5), padding="same", activation=tf.nn.relu,
                             kernel_initializer='he_normal', kernel_regularizer=tf.keras.regular

        m72_model.add(Dropout(rate=0.5))

        m72_model.add(MaxPooling2D(pool_size=(2,2), padding="same"))

        # Seventh Convolutional layer
        m72_model.add(Conv2D(32, kernel_size=(5,5), padding="same", activation=tf.nn.relu,
                             kernel_initializer='he_normal', kernel_regularizer=tf.keras.regular

        m72_model.add(Dropout(rate=0.5))

        # Three dense layers in MLP
        m72_model.add(Flatten())
        m72_model.add(Dense(128, activation=tf.nn.relu, kernel_initializer='he_normal', kernel_
        m72_model.add(BatchNormalization())
        m72_model.add(Dropout(rate=0.5))
        m72_model.add(Dense(64, activation=tf.nn.relu, kernel_initializer='he_normal', kernel_r
        m72_model.add(BatchNormalization())
        m72_model.add(Dropout(rate=0.5))
        m72_model.add(Dense(10, activation=tf.nn.softmax))


        m72_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy
        model = m72_model.fit(x_train_new, y_train, epochs=n_epochs, batch_size=batchsize, verb

Train on 60000 samples, validate on 10000 samples
Epoch 1/25
60000/60000 [==============================] - 35s 582us/sample - loss: 1.7129 - acc: 0.4547 - v
Epoch 2/25
60000/60000 [==============================] - 32s 536us/sample - loss: 0.4255 - acc: 0.8813 - v
Epoch 3/25
60000/60000 [==============================] - 32s 536us/sample - loss: 0.2254 - acc: 0.9433 - v
Epoch 4/25
60000/60000 [==============================] - 32s 536us/sample - loss: 0.1759 - acc: 0.9552 - v
```

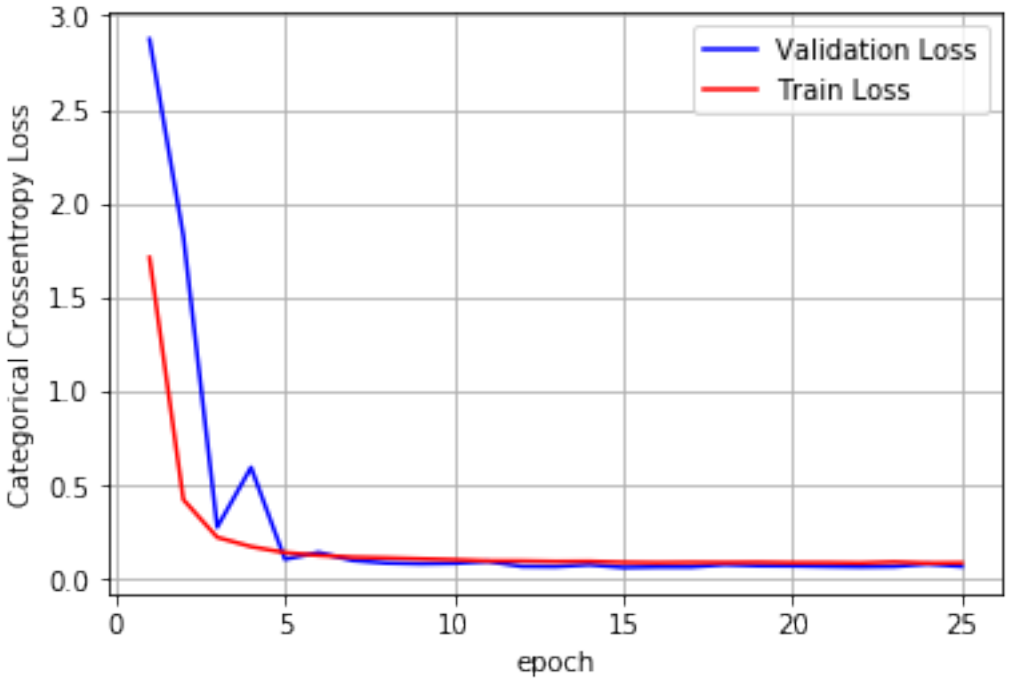
```
Epoch 5/25
60000/60000 [==============================] - 32s 534us/sample - loss: 0.1442 - acc: 0.9658 - v
Epoch 6/25
60000/60000 [==============================] - 32s 535us/sample - loss: 0.1293 - acc: 0.9703 - v
Epoch 7/25
60000/60000 [==============================] - 32s 535us/sample - loss: 0.1197 - acc: 0.9730 - v
Epoch 8/25
60000/60000 [==============================] - 32s 537us/sample - loss: 0.1161 - acc: 0.9736 - v
Epoch 9/25
60000/60000 [==============================] - 32s 536us/sample - loss: 0.1110 - acc: 0.9765 - v
Epoch 10/25
60000/60000 [==============================] - 32s 536us/sample - loss: 0.1062 - acc: 0.9778 - v
Epoch 11/25
60000/60000 [==============================] - 32s 535us/sample - loss: 0.1000 - acc: 0.9793 - v
Epoch 12/25
60000/60000 [==============================] - 32s 536us/sample - loss: 0.0998 - acc: 0.9799 - v
Epoch 13/25
60000/60000 [==============================] - 32s 537us/sample - loss: 0.0968 - acc: 0.9808 - v
Epoch 14/25
60000/60000 [==============================] - 32s 537us/sample - loss: 0.0976 - acc: 0.9808 - v
Epoch 15/25
60000/60000 [==============================] - 32s 535us/sample - loss: 0.0924 - acc: 0.9830 - v
Epoch 16/25
60000/60000 [==============================] - 32s 536us/sample - loss: 0.0904 - acc: 0.9834 - v
Epoch 17/25
60000/60000 [==============================] - 32s 536us/sample - loss: 0.0919 - acc: 0.9831 - v
Epoch 18/25
60000/60000 [==============================] - 32s 538us/sample - loss: 0.0914 - acc: 0.9841 - v
Epoch 19/25
60000/60000 [==============================] - 32s 536us/sample - loss: 0.0925 - acc: 0.9835 - v
Epoch 20/25
60000/60000 [==============================] - 32s 537us/sample - loss: 0.0902 - acc: 0.9845 - v
Epoch 21/25
60000/60000 [==============================] - 32s 538us/sample - loss: 0.0904 - acc: 0.9857 - v
Epoch 22/25
60000/60000 [==============================] - 32s 540us/sample - loss: 0.0884 - acc: 0.9856 - v
Epoch 23/25
60000/60000 [==============================] - 32s 538us/sample - loss: 0.0942 - acc: 0.9851 - v
Epoch 24/25
60000/60000 [==============================] - 32s 537us/sample - loss: 0.0881 - acc: 0.9866 - v
Epoch 25/25
60000/60000 [==============================] - 32s 537us/sample - loss: 0.0895 - acc: 0.9860 - v
```

```
In [50]: print("************************************************")
         print("Printing the Model Summary")
         print(m72_model.summary())
         print("************************************************")
```

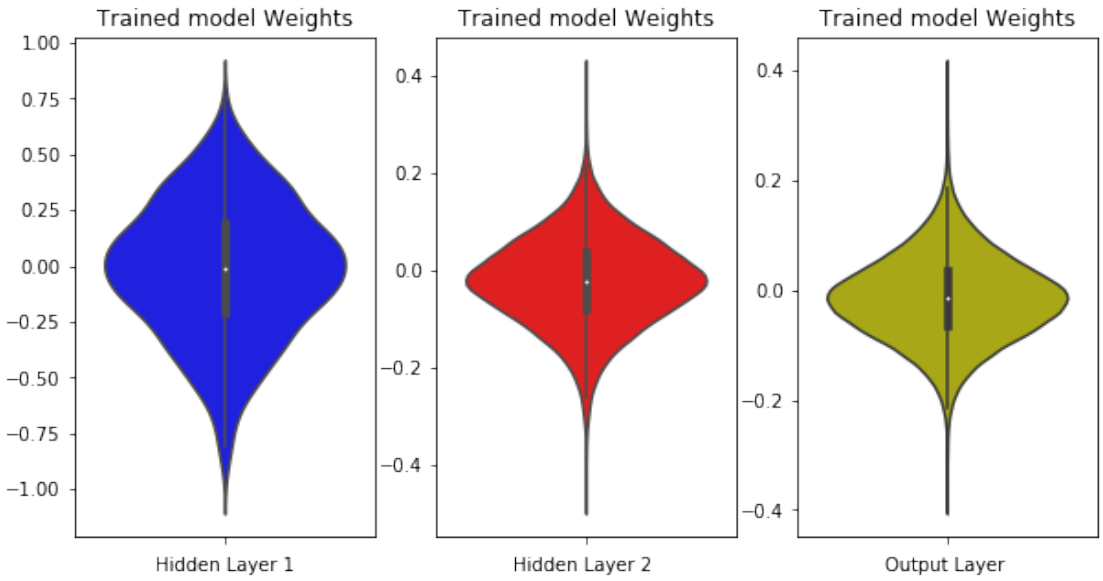
```
        score = m72_model.evaluate(x_test_new, y_test)

        print('Test score:', score[0])
        print('Test accuracy:', score[1])

        final_output = final_output.append({"#ConvNets": 7,
                                            "#Kernels/Filters": '5x5',
                                            "Padding": 'same',
                                            "Stride": '2x2',
                                            "Dropout": True,
                                            "BatchNormalization": True,
                                            "Regularization": 'L2 (0.00001)',
                                            "TRAIN_LOSS": '{:.5f}'.format(model.history["loss"]
                                            "TEST_LOSS": '{:.5f}'.format(model.history["val_los
                                            "TRAIN_ACC": '{:.5f}'.format(model.history["acc"][n
                                            "TEST_ACC": '{:.5f}'.format(model.history["val_acc"

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

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

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

```
**********************************************
Printing the Model Summary
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_49 (Conv2D)           (None, 28, 28, 32)        832
_________________________________________________________________
dropout_47 (Dropout)         (None, 28, 28, 32)        0
_________________________________________________________________
conv2d_50 (Conv2D)           (None, 28, 28, 64)        51264
_________________________________________________________________
dropout_48 (Dropout)         (None, 28, 28, 64)        0
_________________________________________________________________
conv2d_51 (Conv2D)           (None, 28, 28, 64)        102464
_________________________________________________________________
max_pooling2d_44 (MaxPooling (None, 14, 14, 64)        0
_________________________________________________________________
dropout_49 (Dropout)         (None, 14, 14, 64)        0
_________________________________________________________________
```

```
conv2d_52 (Conv2D)           (None, 14, 14, 64)        102464
_________________________________________________________________
dropout_50 (Dropout)         (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_53 (Conv2D)           (None, 14, 14, 64)        102464
_________________________________________________________________
dropout_51 (Dropout)         (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_54 (Conv2D)           (None, 14, 14, 32)        51232
_________________________________________________________________
dropout_52 (Dropout)         (None, 14, 14, 32)        0
_________________________________________________________________
max_pooling2d_45 (MaxPooling (None, 7, 7, 32)          0
_________________________________________________________________
conv2d_55 (Conv2D)           (None, 7, 7, 32)          25632
_________________________________________________________________
dropout_53 (Dropout)         (None, 7, 7, 32)          0
_________________________________________________________________
flatten_13 (Flatten)         (None, 1568)              0
_________________________________________________________________
dense_39 (Dense)             (None, 128)               200832
_________________________________________________________________
batch_normalization_v1_18 (B (None, 128)               512
_________________________________________________________________
dropout_54 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_40 (Dense)             (None, 64)                8256
_________________________________________________________________
batch_normalization_v1_19 (B (None, 64)                256
_________________________________________________________________
dropout_55 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_41 (Dense)             (None, 10)                650
=================================================================
Total params: 646,858
Trainable params: 646,474
Non-trainable params: 384
_________________________________________________________________
None
**********************************************
10000/10000 [==============================] - 3s 278us/sample - loss: 0.0706 - acc: 0.9911
Test score: 0.07064299584031106
Test accuracy: 0.9911
```



In [51]:
```
w_after = m72_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(figsize=(10, 5))
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 Conclusion

```
In [52]: final_output
```

Out[52]:

	#ConvNets	#Kernels/Filters	Padding	...	TEST_LOSS	TRAIN_ACC	TEST_ACC
0	2	3x3	-	...	0.05639	0.99870	0.98830
1	2	5x5	-	...	0.06200	0.99898	0.98870
2	2	3x3	-	...	0.05771	0.97355	0.98810
3	2	5x5	-	...	0.04970	0.98045	0.98990
4	3	3x3	-	...	0.08567	0.99605	0.98120
5	3	5x5	same	...	0.05867	0.99938	0.98900
6	3	3x3	-	...	0.12611	0.88702	0.96790
7	3	5x5	same	...	0.09296	0.94485	0.98210
8	5	3x3	same	...	0.06690	0.99763	0.98790
9	5	5x5	same	...	0.05095	0.99762	0.98990
10	5	3x3	same	...	0.09166	0.93482	0.98280
11	5	5x5	same	...	0.09011	0.95897	0.98680
12	7	3x3	same	...	0.07435	0.97978	0.98790
13	7	5x5	same	...	0.07064	0.98602	0.99110

[14 rows x 11 columns]

In this task, We tried Convolution Neural Networks on MNIST Dataset with Keras.

There are a couple different CNN-architecture that we tried - *2 ConvNets, 3 ConvNets, 5 ConvNets, and 7 ConvNets*.

For each ConvNet, 3x3 and 5x5 kernels/filters were used. Also, dropouts, batch normalization, weight regularization were also used to differentiate the effect of these methods.

```
Note -
Dropout was set to rate = 0.5 which meant onyl 50% of the neurons will remain active at a partic
Weight regularization was set to L2 (Ridge) = 0.00001.
Weight intialization was set to he-normal
Number of epochs was set to 25
Batch size was set to 128
```

3.0.1 The main conclusion that we can draw from the above table and train-test-loss plots is that without dropout and regularization, the architecture tends to overfit after 10-15 epochs. However, with dropout(0.5) and L2-regularization we can see that the overall loss i.e train loss and test loss is somewhat converging to be equal. Though we tried only 25 epochs in this task, that is only reason the train and validation accuracy is less for the models with regularization than that of the model without regularization.

3.0.2 Thus we can guarantee that with the use of regularization in CNN where the ConvNets are more, we are sure that the model is generalizing better.