CNN's on MNIST

In [0]:
```
# Credits: https://github.com/keras-team/keras/blob/master/examples/mnist_cnn.py


from __future__ import print_function
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K

batch_size = 128
num_classes = 10
epochs = 12

# input image dimensions
img_rows, img_cols = 28, 28

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

if K.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

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

model.compile(loss=keras.losses.categorical_crossentropy,
```

```
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

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

In [1]:
```
# if you keras is not using tensorflow as backend set "KERAS_BACKEND=tensor
flow" use this command
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense,Activation
from keras.layers.normalization import BatchNormalization
from keras.layers import Dropout,Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K
```

```
Using TensorFlow backend.
```

In [2]:
```
# the data, shuffled and split between train and test sets
(X_train, y_train), (x_test, y_test) = mnist.load_data()
```

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

In [0]:
```
# some model parameters
input_dim = X_train.shape[1]
num_clas = 10
batch_size = 128
no_epoch = 20
img_row, img_col = 28,28
```

In [0]:
```
if K.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_row, img_col)
    x_test = x_test.reshape(x_test.shape[0], 1, img_row, img_col)
    input_shape = (1, img_row, img_col)
else:
    X_train = X_train.reshape(X_train.shape[0], img_row, img_col, 1)
    x_test = x_test.reshape(x_test.shape[0], img_row, img_col, 1)
    input_shape = (img_row, img_col, 1)
```

In [5]:
```
print("Number of training examples :", X_train.shape[0], X_train.shape[1],
X_train.shape[2])
```

```
Number of training examples : 60000 28 28
```

In [6]:
```
print("Number of training examples :", X_train.shape[0], "and each image is
of shape (%d, %d)"%(X_train.shape[1], X_train.shape[2]))
print("Number of training examples :", x_test.shape[0], "and each image is
 of shape (%d, %d)"%(x_test.shape[1], x_test.shape[2]))
```

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

In [7]:
```
# after converting the input images from 3d to 2d vectors

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

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

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

X_train = X_train/255
x_test = x_test/255
```

In [9]:
```
# here we are having a class number for each image
print("Class label of first image :", y_train[0])

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

y_train = keras.utils.to_categorical(y_train, num_clas)
y_test = keras.utils.to_categorical(y_test, num_clas)

print("After converting the output into a vector : ", type(y_train))
```

```
Class label of first image : 5
After converting the output into a vector :  <class 'numpy.ndarray'>
```

In [10]:
```
print(y_train.shape)
print(y_test.shape)
```

```
(60000, 10)
(10000, 10)
```

3-layer ConvNet + Relu + adam

No Batch normalization and Dropout

In [0]:
```
%matplotlib notebook
import matplotlib.pyplot as plt
import numpy as np
import time
# https://gist.github.com/greydanus/f6eee59eaf1d90fcb3b534a25362cea4
# https://stackoverflow.com/a/14434334
# this function is used to update the plots for each epoch and error
def plt_mdl_res(x_val, trn_los, tst_los, tst_scr, tst_acc):
    # Visualize loss history
    plt.figure(figsize=(16,16))
    plt.plot(x_val, trn_los, 'r--')
    plt.plot(x_val, tst_los, 'b-')
    plt.legend(['Training Loss', 'Test Loss'])
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.grid()
    plt.show();

    print('Test score:', tst_scr)
    print('Test accuracy:', tst_acc)
```

In [0]:
```
#ConvNet
mdl_1 = Sequential()
mdl_1.add(Conv2D(32,kernel_size=(7,7), activation='relu',input_shape=(input
_shape)))
mdl_1.add(Conv2D(64,kernel_size=(5,5), activation='relu'))
mdl_1.add(MaxPooling2D(pool_size=(2,2)))
mdl_1.add(Conv2D(128,kernel_size=(5,5), activation='relu'))
mdl_1.add(Flatten())
mdl_1.add(Dense(128, activation='relu'))
mdl_1.add(Dense(num_clas,activation='softmax'))
mdl_1.summary()
```

```
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_28 (Conv2D)           (None, 22, 22, 32)        1600
_________________________________________________________________
conv2d_29 (Conv2D)           (None, 18, 18, 64)        51264
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 9, 9, 64)          0
_________________________________________________________________
conv2d_30 (Conv2D)           (None, 5, 5, 128)         204928
_________________________________________________________________
flatten_10 (Flatten)         (None, 3200)              0
_________________________________________________________________
dense_19 (Dense)             (None, 128)               409728
_________________________________________________________________
dense_20 (Dense)             (None, 10)                1290
=================================================================
Total params: 668,810
Trainable params: 668,810
Non-trainable params: 0
_________________________________________________________________
```

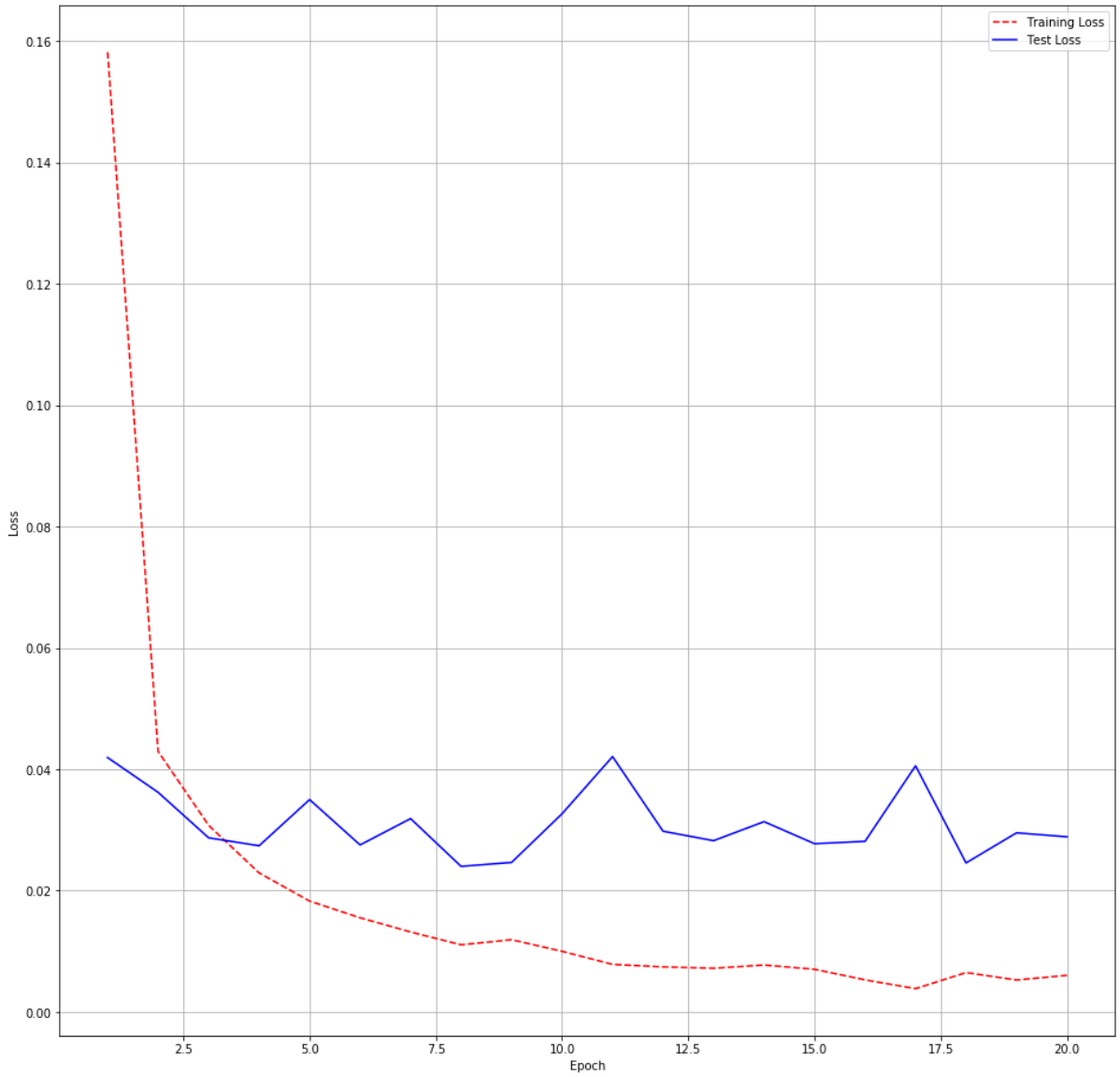
In [0]:
```
#set optimizer and loss
mdl_1.compile(optimizer=keras.optimizers.Adam(),loss=keras.losses.categoric
al_crossentropy, metrics=['accuracy'])
```

In [0]:
```
history = mdl_1.fit(X_train,y_train,batch_size=batch_size,epochs=no_epoch,v
erbose=1,validation_data=(x_test,y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 10s 166us/step - loss: 0.158
2 - acc: 0.9500 - val_loss: 0.0419 - val_acc: 0.9869
Epoch 2/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.0429
- acc: 0.9873 - val_loss: 0.0362 - val_acc: 0.9878
Epoch 3/20
60000/60000 [==============================] - 9s 148us/step - loss: 0.0308
- acc: 0.9905 - val_loss: 0.0287 - val_acc: 0.9912
Epoch 4/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.0229
- acc: 0.9925 - val_loss: 0.0274 - val_acc: 0.9917
Epoch 5/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.0183
- acc: 0.9941 - val_loss: 0.0350 - val_acc: 0.9890
Epoch 6/20
60000/60000 [==============================] - 9s 148us/step - loss: 0.0155
- acc: 0.9948 - val_loss: 0.0275 - val_acc: 0.9912
Epoch 7/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.0132
- acc: 0.9960 - val_loss: 0.0319 - val_acc: 0.9918
Epoch 8/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.0111
- acc: 0.9966 - val_loss: 0.0240 - val_acc: 0.9933
Epoch 9/20
60000/60000 [==============================] - 9s 145us/step - loss: 0.0119
- acc: 0.9960 - val_loss: 0.0246 - val_acc: 0.9921
Epoch 10/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.0100
- acc: 0.9968 - val_loss: 0.0327 - val_acc: 0.9914
Epoch 11/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.0078
- acc: 0.9973 - val_loss: 0.0421 - val_acc: 0.9907
Epoch 12/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.0074
- acc: 0.9976 - val_loss: 0.0298 - val_acc: 0.9924
Epoch 13/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.0072
- acc: 0.9977 - val_loss: 0.0282 - val_acc: 0.9938
Epoch 14/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.0077
- acc: 0.9975 - val_loss: 0.0314 - val_acc: 0.9931
Epoch 15/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.0070
- acc: 0.9978 - val_loss: 0.0277 - val_acc: 0.9929
Epoch 16/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.0053
- acc: 0.9984 - val_loss: 0.0281 - val_acc: 0.9925
Epoch 17/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.0038
- acc: 0.9989 - val_loss: 0.0406 - val_acc: 0.9920
Epoch 18/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.0065
- acc: 0.9981 - val_loss: 0.0246 - val_acc: 0.9936
Epoch 19/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.0053
```

```
- acc: 0.9983 - val_loss: 0.0295 - val_acc: 0.9933
Epoch 20/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.0060
- acc: 0.9983 - val_loss: 0.0289 - val_acc: 0.9938
```

In [0]:
```
%matplotlib inline
import keras
from matplotlib import pyplot as plt

epoch_count = list(range(1,no_epoch+1))

# Get training and test loss histories
training_loss = history.history['loss']
test_loss = history.history['val_loss']

# Create count of the number of epochs
score = mdl_1.evaluate(x_test, y_test, verbose=0)
plt_mdl_res(epoch_count, training_loss, test_loss, score[0], score[1])
```




```
Test score: 0.028870063201270432
Test accuracy: 0.9938
```

Hyper Parameter tuning Dropout rate for 3-Layer convNNet

In [ ]:
```
no_cnvnets = 8
mdl = [0] *no_cnvnets
history = [0] * no_cnvnets
nme = ["Dropout=0.0","Dropout=0.1","Dropout=0.2","Dropout=0.3","Dropout=0.
4","Dropout=0.5","Dropout=0.6","Dropout=0.7"]

for i in range(no_cnvnets):
    #ConvNet
    mdl[i] = Sequential()
    mdl[i].add(Conv2D(32,kernel_size=(7,7), activation='relu',input_shape=(
input_shape)))
    mdl[i].add(Conv2D(64,kernel_size=(5,5), activation='relu'))
    mdl[i].add(MaxPooling2D(pool_size=(2,2)))
    mdl[i].add(Conv2D(128,kernel_size=(5,5), activation='relu'))
    mdl[i].add(Flatten())
    mdl[i].add(Dense(128, activation='relu'))
    mdl[i].add(Dropout(rate=(i*0.01)))
    mdl[i].add(Dense(num_clas,activation='softmax'))
    mdl[i].compile(optimizer="adam", loss="categorical_crossentropy", metri
cs=["accuracy"])

for j in range(no_cnvnets):
    history[j] = mdl[j].fit(X_train,y_train, batch_size=batch_size, epochs
= no_epoch,
    validation_data = (x_test,y_test), verbose=1)

for k in range(no_cnvnets):
    print("CNN {0}: Epochs={1:d}, Train accuracy={2:.5f}, Validation accura
cy={3:.5f}".format(
        nme[k],no_epoch,max(history[k].history['acc']),max(history[k].histo
ry['val_acc']) ))
```

CNN Dropout=0.0: Epochs=20, Train accuracy=0.99913, Validation accuracy=0.99410 CNN Dropout=0.1: Epochs=20, Train accuracy=0.99910, Validation accuracy=0.99360 CNN Dropout=0.2: Epochs=20, Train accuracy=0.99882, Validation accuracy=0.99400 CNN Dropout=0.3: Epochs=20, Train accuracy=0.99883, Validation accuracy=0.99290 CNN Dropout=0.4: Epochs=20, Train accuracy=0.99878, Validation accuracy=0.99330 CNN Dropout=0.5: Epochs=20, Train accuracy=0.99910, Validation accuracy=0.99320 CNN Dropout=0.6: Epochs=20, Train accuracy=0.99915, Validation accuracy=0.99340 CNN Dropout=0.7: Epochs=20, Train accuracy=0.99865, Validation accuracy=0.99340

3-Layer ConvNet + ReLu + Adam + Dropout

No Batch Normalization with Dropout rate 0.0

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

output_dim = 10
input_dim = X_train.shape[1]

batch_size = 128
no_epoch = 20
```

In [0]:
```
#ConvNet
mdl_2 = Sequential()
mdl_2.add(Conv2D(32,kernel_size=(7,7), activation='relu',input_shape=(input
_shape)))
mdl_2.add(Conv2D(64,kernel_size=(5,5), activation='relu'))
mdl_2.add(MaxPooling2D(pool_size=(2,2)))
mdl_2.add(Conv2D(128,kernel_size=(5,5), activation='relu'))
mdl_2.add(Dropout(rate=0.0))
mdl_2.add(Flatten())
mdl_2.add(Dense(128, activation='relu'))
mdl_2.add(Dropout(rate=0.0))
mdl_2.add(Dense(num_clas,activation='softmax'))
mdl_2.summary()
```

```
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_31 (Conv2D)           (None, 22, 22, 32)        1600
_________________________________________________________________
conv2d_32 (Conv2D)           (None, 18, 18, 64)        51264
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 9, 9, 64)          0
_________________________________________________________________
conv2d_33 (Conv2D)           (None, 5, 5, 128)         204928
_________________________________________________________________
dropout_10 (Dropout)         (None, 5, 5, 128)         0
_________________________________________________________________
flatten_11 (Flatten)         (None, 3200)              0
_________________________________________________________________
dense_21 (Dense)             (None, 128)               409728
_________________________________________________________________
dropout_11 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_22 (Dense)             (None, 10)                1290
=================================================================
Total params: 668,810
Trainable params: 668,810
Non-trainable params: 0
_________________________________________________________________
```

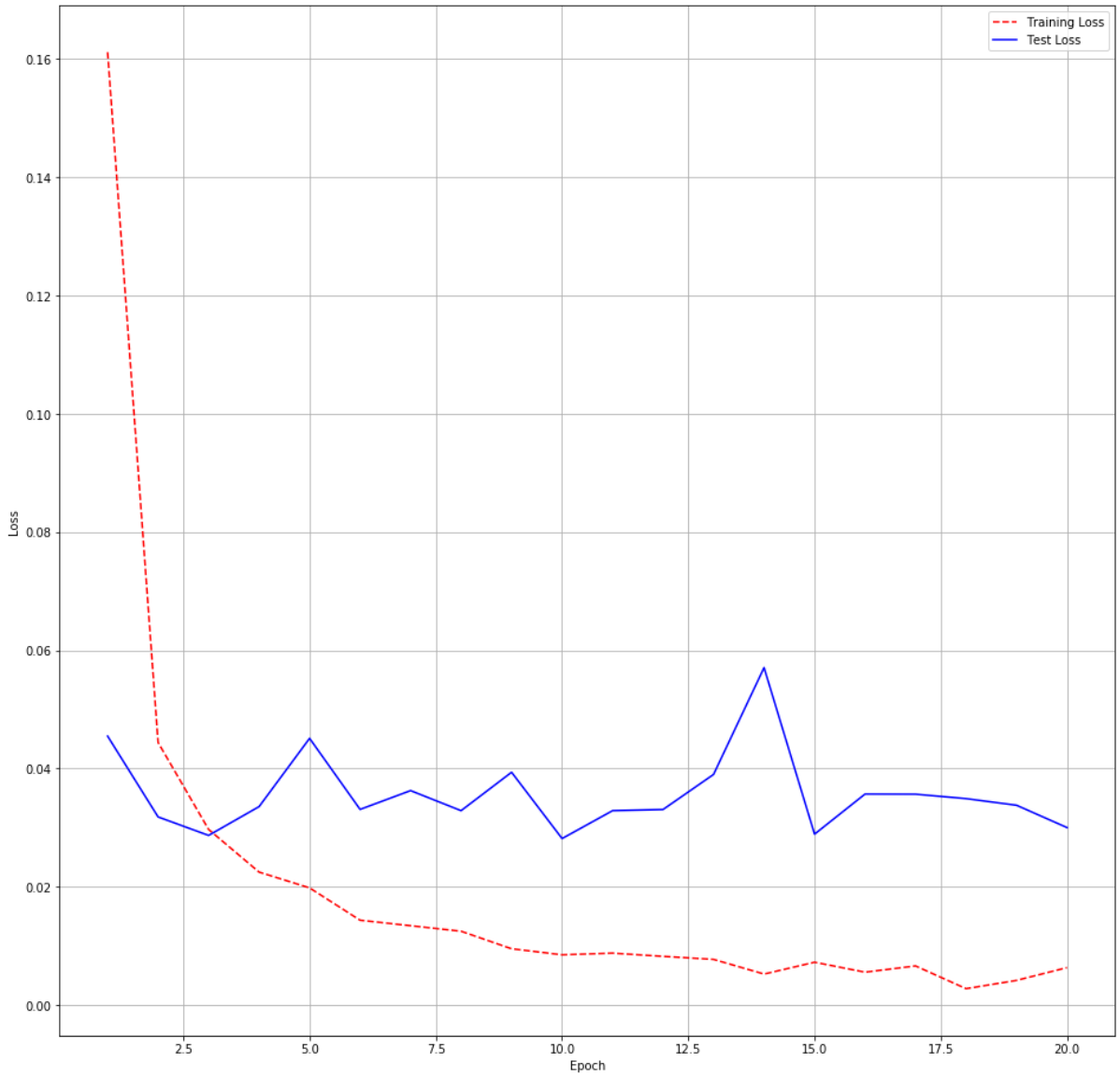
In [0]:
```
#set optimizer and loss
mdl_2.compile(optimizer='adam',loss='categorical_crossentropy', metrics=['a
ccuracy'])
```

In [0]:
```
history_2 = mdl_2.fit(X_train,y_train,batch_size=batch_size,epochs=no_epoch
,verbose=1,validation_data=(x_test,y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 10s 170us/step - loss: 0.161
2 - acc: 0.9502 - val_loss: 0.0455 - val_acc: 0.9856
Epoch 2/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.0444
- acc: 0.9859 - val_loss: 0.0318 - val_acc: 0.9895
Epoch 3/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.0297
- acc: 0.9906 - val_loss: 0.0287 - val_acc: 0.9913
Epoch 4/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.0225
- acc: 0.9932 - val_loss: 0.0336 - val_acc: 0.9882
Epoch 5/20
60000/60000 [==============================] - 9s 145us/step - loss: 0.0198
- acc: 0.9938 - val_loss: 0.0451 - val_acc: 0.9870
Epoch 6/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.0144
- acc: 0.9956 - val_loss: 0.0331 - val_acc: 0.9904
Epoch 7/20
60000/60000 [==============================] - 9s 145us/step - loss: 0.0134
- acc: 0.9959 - val_loss: 0.0363 - val_acc: 0.9888
Epoch 8/20
60000/60000 [==============================] - 9s 145us/step - loss: 0.0125
- acc: 0.9960 - val_loss: 0.0329 - val_acc: 0.9909
Epoch 9/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.0095
- acc: 0.9969 - val_loss: 0.0394 - val_acc: 0.9899
Epoch 10/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.0085
- acc: 0.9974 - val_loss: 0.0282 - val_acc: 0.9924
Epoch 11/20
60000/60000 [==============================] - 9s 145us/step - loss: 0.0088
- acc: 0.9971 - val_loss: 0.0329 - val_acc: 0.9912
Epoch 12/20
60000/60000 [==============================] - 9s 145us/step - loss: 0.0083
- acc: 0.9974 - val_loss: 0.0331 - val_acc: 0.9927
Epoch 13/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.0077
- acc: 0.9974 - val_loss: 0.0390 - val_acc: 0.9910
Epoch 14/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.0053
- acc: 0.9986 - val_loss: 0.0571 - val_acc: 0.9889
Epoch 15/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.0073
- acc: 0.9978 - val_loss: 0.0289 - val_acc: 0.9934
Epoch 16/20
60000/60000 [==============================] - 9s 145us/step - loss: 0.0056
- acc: 0.9982 - val_loss: 0.0357 - val_acc: 0.9917
Epoch 17/20
60000/60000 [==============================] - 9s 145us/step - loss: 0.0066
- acc: 0.9979 - val_loss: 0.0357 - val_acc: 0.9929
Epoch 18/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.0028
- acc: 0.9992 - val_loss: 0.0349 - val_acc: 0.9925
Epoch 19/20
60000/60000 [==============================] - 9s 144us/step - loss: 0.0042
```

```
- acc: 0.9987 - val_loss: 0.0338 - val_acc: 0.9924
Epoch 20/20
60000/60000 [==============================] - 9s 145us/step - loss: 0.0064
- acc: 0.9982 - val_loss: 0.0300 - val_acc: 0.9928
```

In [0]:
```
%matplotlib inline
import keras
from matplotlib import pyplot as plt

epoch_count = list(range(1,no_epoch+1))

# Get training and test loss histories
training_loss = history_2.history['loss']
test_loss = history_2.history['val_loss']

# Create count of the number of epochs
score = mdl_2.evaluate(x_test, y_test, verbose=0)
plt_mdl_res(epoch_count, training_loss, test_loss, score[0], score[1])
```




```
Test score: 0.030026011430384324
Test accuracy: 0.9928
```

5-Layer ConvNNet + ReLu + Adam

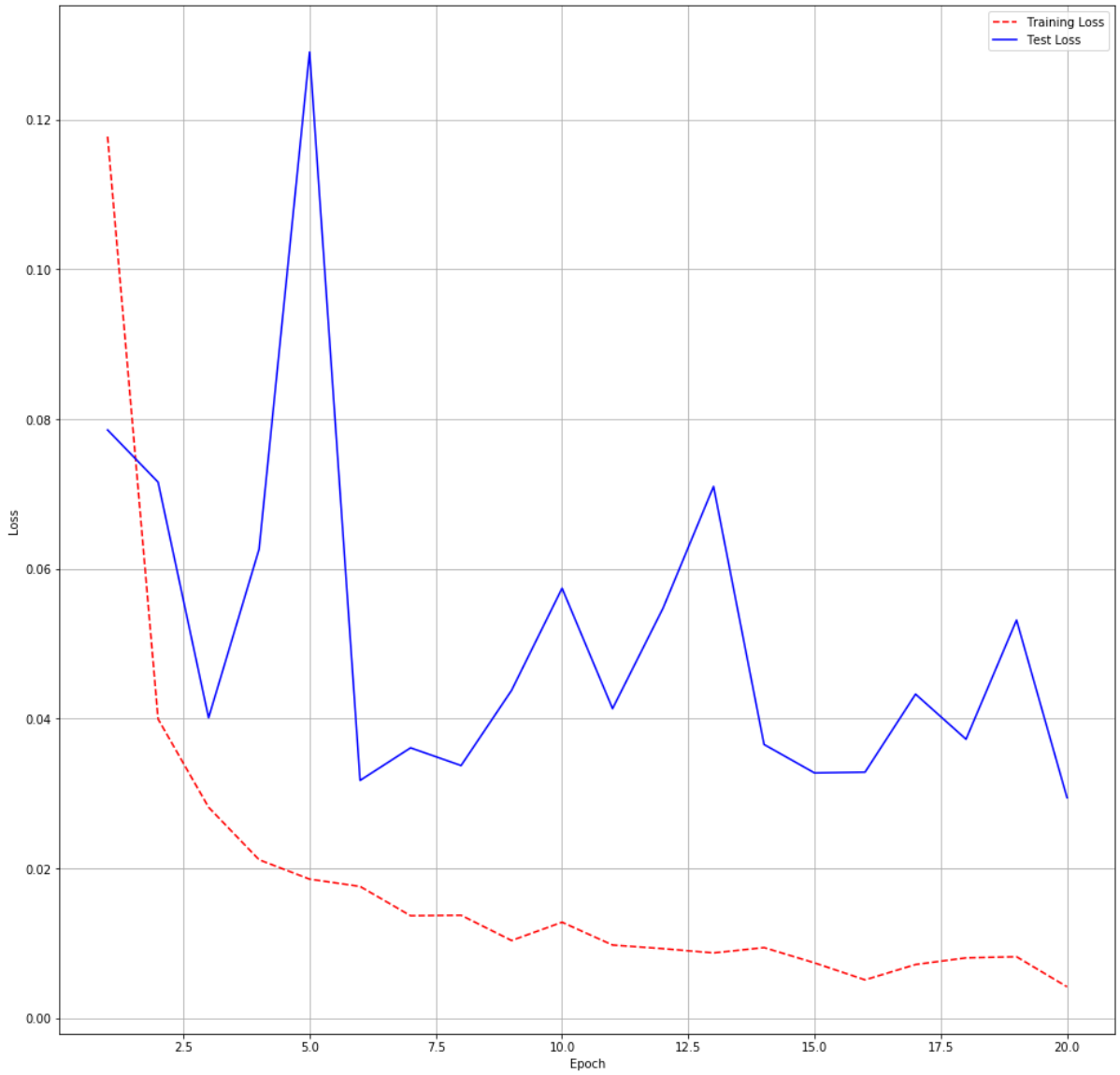
Batch Normalization with No DropOut

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

output_dim = 10
input_dim = X_train.shape[1]

batch_size = 128
no_epoch = 20

#ConvNet
mdl_3 = Sequential()
mdl_3.add(Conv2D(32,kernel_size=(3,3), activation='relu',input_shape=(input
_shape)))
mdl_3.add(Conv2D(32,kernel_size=(3,3), activation='relu'))
mdl_3.add(MaxPooling2D(pool_size=(2,2)))
mdl_3.add(Conv2D(32,kernel_size=(5,5), activation='relu'))
mdl_3.add(BatchNormalization())
mdl_3.add(Conv2D(64,kernel_size=(3,3), activation='relu'))
mdl_3.add(MaxPooling2D(pool_size=(2,2)))
mdl_3.add(Conv2D(128,kernel_size=(2,2), activation='relu'))
mdl_3.add(BatchNormalization())
mdl_3.add(Flatten())
mdl_3.add(Dense(128, activation='relu'))
mdl_3.add(Dense(num_clas,activation='softmax'))
mdl_3.summary()

mdl_3.compile(optimizer='adam',loss='categorical_crossentropy', metrics=['a
ccuracy'])
history_3 = mdl_3.fit(X_train,y_train,batch_size=batch_size,epochs=no_epoch
,verbose=1,validation_data=(x_test,y_test))
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backen
d/tensorflow_backend.py:2041: The name tf.nn.fused_batch_norm is deprecate
d. Please use tf.compat.v1.nn.fused_batch_norm instead.

Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_34 (Conv2D)           (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_35 (Conv2D)           (None, 24, 24, 32)        9248
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 12, 12, 32)        0
_________________________________________________________________
conv2d_36 (Conv2D)           (None, 8, 8, 32)          25632
_________________________________________________________________
batch_normalization_1 (Batch (None, 8, 8, 32)          128
_________________________________________________________________
conv2d_37 (Conv2D)           (None, 6, 6, 64)          18496
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 3, 3, 64)          0
_________________________________________________________________
conv2d_38 (Conv2D)           (None, 2, 2, 128)         32896
_________________________________________________________________
batch_normalization_2 (Batch (None, 2, 2, 128)         512
_________________________________________________________________
flatten_12 (Flatten)         (None, 512)               0
_________________________________________________________________
dense_23 (Dense)             (None, 128)               65664
_________________________________________________________________
dense_24 (Dense)             (None, 10)                1290
=================================================================
Total params: 154,186
Trainable params: 153,866
Non-trainable params: 320
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 11s 184us/step - loss: 0.117
7 - acc: 0.9638 - val_loss: 0.0785 - val_acc: 0.9763
Epoch 2/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.0399
- acc: 0.9873 - val_loss: 0.0716 - val_acc: 0.9784
Epoch 3/20
60000/60000 [==============================] - 9s 148us/step - loss: 0.0281
- acc: 0.9908 - val_loss: 0.0401 - val_acc: 0.9883
Epoch 4/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.0211
- acc: 0.9936 - val_loss: 0.0626 - val_acc: 0.9827
Epoch 5/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.0185
- acc: 0.9940 - val_loss: 0.1290 - val_acc: 0.9661
Epoch 6/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.0176
- acc: 0.9942 - val_loss: 0.0317 - val_acc: 0.9899
Epoch 7/20
60000/60000 [==============================] - 9s 148us/step - loss: 0.0136
```

```
- acc: 0.9953 - val_loss: 0.0361 - val_acc: 0.9890
Epoch 8/20
60000/60000 [==============================] - 9s 148us/step - loss: 0.0137
- acc: 0.9956 - val_loss: 0.0337 - val_acc: 0.9902
Epoch 9/20
60000/60000 [==============================] - 9s 148us/step - loss: 0.0103
- acc: 0.9967 - val_loss: 0.0438 - val_acc: 0.9891
Epoch 10/20
60000/60000 [==============================] - 9s 148us/step - loss: 0.0128
- acc: 0.9957 - val_loss: 0.0574 - val_acc: 0.9842
Epoch 11/20
60000/60000 [==============================] - 9s 148us/step - loss: 0.0097
- acc: 0.9965 - val_loss: 0.0413 - val_acc: 0.9882
Epoch 12/20
60000/60000 [==============================] - 9s 149us/step - loss: 0.0092
- acc: 0.9970 - val_loss: 0.0547 - val_acc: 0.9873
Epoch 13/20
60000/60000 [==============================] - 9s 148us/step - loss: 0.0087
- acc: 0.9974 - val_loss: 0.0710 - val_acc: 0.9851
Epoch 14/20
60000/60000 [==============================] - 9s 150us/step - loss: 0.0094
- acc: 0.9968 - val_loss: 0.0365 - val_acc: 0.9916
Epoch 15/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.0073
- acc: 0.9976 - val_loss: 0.0327 - val_acc: 0.9922
Epoch 16/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.0051
- acc: 0.9983 - val_loss: 0.0328 - val_acc: 0.9922
Epoch 17/20
60000/60000 [==============================] - 9s 148us/step - loss: 0.0071
- acc: 0.9978 - val_loss: 0.0433 - val_acc: 0.9907
Epoch 18/20
60000/60000 [==============================] - 9s 148us/step - loss: 0.0080
- acc: 0.9973 - val_loss: 0.0372 - val_acc: 0.9915
Epoch 19/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.0082
- acc: 0.9972 - val_loss: 0.0532 - val_acc: 0.9882
Epoch 20/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.0041
- acc: 0.9986 - val_loss: 0.0294 - val_acc: 0.9934
```

In [0]:
```
%matplotlib inline
import keras
from matplotlib import pyplot as plt

epoch_count = list(range(1,no_epoch+1))

# Get training and test loss histories
training_loss = history_3.history['loss']
test_loss = history_3.history['val_loss']

# Create count of the number of epochs
score = mdl_3.evaluate(x_test, y_test, verbose=0)
plt_mdl_res(epoch_count, training_loss, test_loss, score[0], score[1])
```



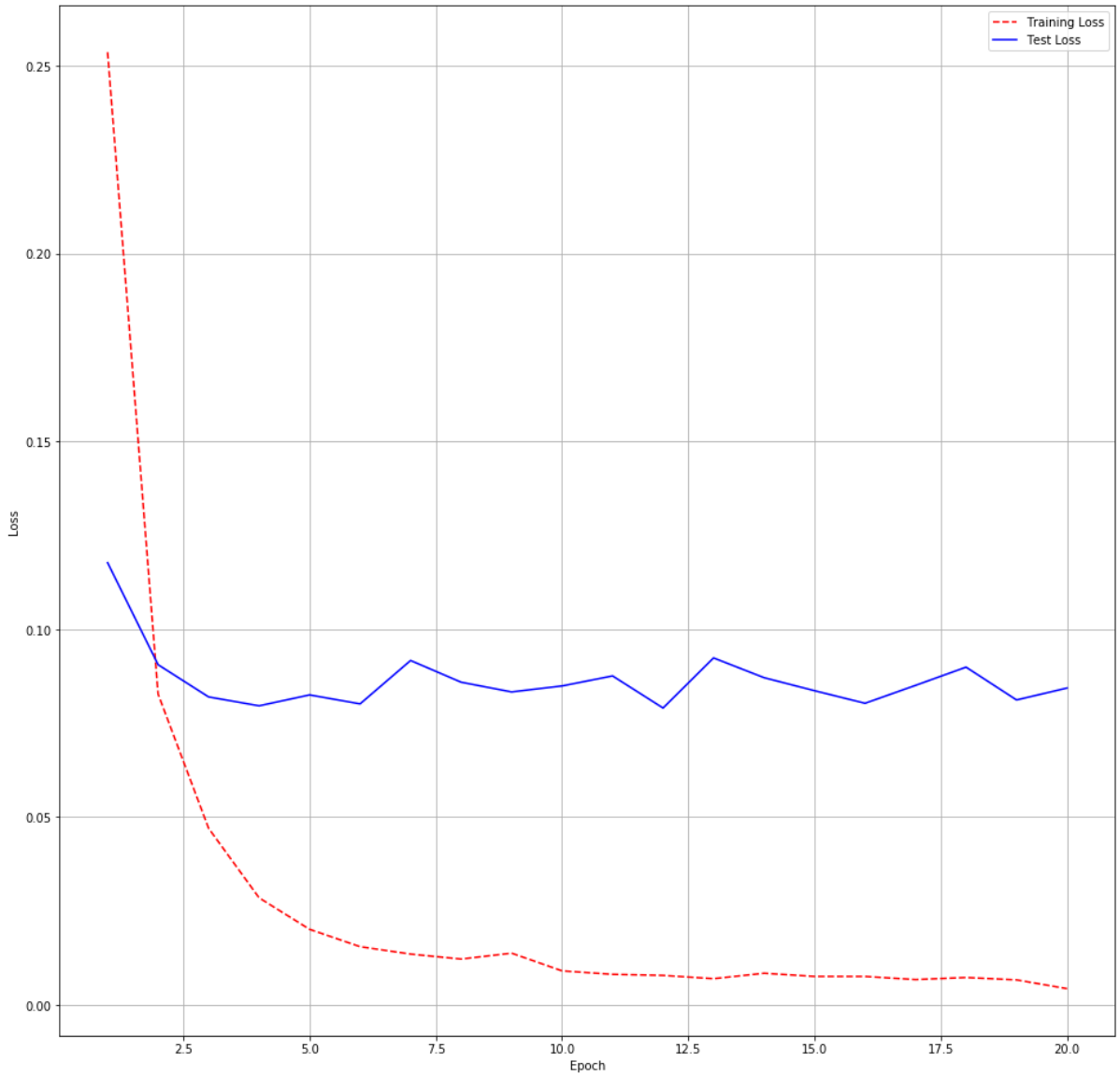

```
Test score: 0.029422459162015413
Test accuracy: 0.9934
```

7-Layer ConvNNet + Adam + Relu

Batch Normalization with Dropout rate 0.0

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

output_dim = 10
input_dim = X_train.shape[1]

batch_size = 128
no_epoch = 20

#ConvNet
mdl_4 = Sequential()
mdl_4.add(Conv2D(32,kernel_size=(3,3), activation='relu',input_shape=(input
_shape)))
mdl_4.add(Conv2D(32,kernel_size=(3,3), activation='relu'))
mdl_4.add(Conv2D(32,kernel_size=(3,3), activation='relu'))
mdl_4.add(Dropout(rate=0.0))

mdl_4.add(Conv2D(64,kernel_size=(3,3), activation='relu'))
mdl_4.add(Conv2D(64,kernel_size=(3,3), activation='relu'))

mdl_4.add(Dropout(rate=0.0))
mdl_4.add(BatchNormalization())

mdl_4.add(Conv2D(64,kernel_size=(5,5), activation='relu'))
mdl_4.add(Conv2D(128,kernel_size=(2,2), activation='relu'))

mdl_4.add(BatchNormalization())
mdl_4.add(Flatten())
mdl_4.add(Dense(128, activation='relu'))
mdl_4.add(Dense(num_clas,activation='softmax'))

mdl_4.summary()


mdl_4.compile(optimizer='adam',loss='categorical_crossentropy', metrics=['a
ccuracy'])
history_4 = mdl_4.fit(X_train,y_train,batch_size=batch_size,epochs=no_epoch
,verbose=1,validation_data=(x_test,y_test))
```

In [0]:
```
%matplotlib inline
import keras
from matplotlib import pyplot as plt

epoch_count = list(range(1,no_epoch+1))

# Get training and test loss histories
training_loss = history_4.history['loss']
test_loss = history_4.history['val_loss']

# Create count of the number of epochs
score = mdl_4.evaluate(x_test, y_test, verbose=0)
plt_mdl_res(epoch_count, training_loss, test_loss, score[0], score[1])
```




```
Test score: 0.08437914842601021
Test accuracy: 0.981
```

Hyperparameter tuning Dropout rate for 7-Layer ConvNNet

In [ ]:
```
no_cnvnets = 8
mdl = [0] *no_cnvnets
history = [0] * no_cnvnets
nme = ["Dropout=0.0","Dropout=0.1","Dropout=0.2","Dropout=0.3","Dropout=0.
4","Dropout=0.5","Dropout=0.6","Dropout=0.7"]

for i in range(no_cnvnets):
    #ConvNet
    mdl[i] = Sequential()
    mdl[i].add(Conv2D(32,kernel_size=(3,3), activation='relu',input_shape=(
input_shape)))
    mdl[i].add(Conv2D(32,kernel_size=(3,3), activation='relu'))
    mdl[i].add(Conv2D(32,kernel_size=(3,3), activation='relu'))
    mdl[i].add(Dropout(rate=(i*0.01)))

    mdl[i].add(Conv2D(64,kernel_size=(3,3), activation='relu'))
    mdl[i].add(Conv2D(64,kernel_size=(3,3), activation='relu'))

    mdl[i].add(Dropout(rate=(i*0.01)))
    mdl[i].add(BatchNormalization())

    mdl[i].add(Conv2D(64,kernel_size=(5,5), activation='relu'))
    mdl[i].add(Conv2D(128,kernel_size=(2,2), activation='relu'))

    mdl[i].add(BatchNormalization())
    mdl[i].add(Flatten())
    mdl[i].add(Dense(128, activation='relu'))
    mdl[i].add(Dense(num_clas,activation='softmax'))
    mdl[i].compile(optimizer="adam", loss="categorical_crossentropy", metri
cs=["accuracy"])

for j in range(no_cnvnets):
    history[j] = mdl[j].fit(X_train,y_train, batch_size=batch_size, epochs
= no_epoch,
    validation_data = (x_test,y_test), verbose=1)
    print('Processed model:',j)

for k in range(no_cnvnets):
    print("CNN {0}: Epochs={1:d}, Train accuracy={2:.5f}, Validation accura
cy={3:.5f}".format(
        nme[k],no_epoch,max(history[k].history['acc']),max(history[k].histo
ry['val_acc']) ))
```

CNN Dropout=0.0: Epochs=20, Train accuracy=0.99800, Validation accuracy=0.99200 CNN Dropout=0.1: Epochs=20, Train accuracy=0.99852, Validation accuracy=0.99160 CNN Dropout=0.2: Epochs=20, Train accuracy=0.99865, Validation accuracy=0.99290 CNN Dropout=0.3: Epochs=20, Train accuracy=0.99782, Validation accuracy=0.99240 CNN Dropout=0.4: Epochs=20, Train accuracy=0.99820, Validation accuracy=0.99130 CNN Dropout=0.5: Epochs=20, Train accuracy=0.99792, Validation accuracy=0.99090 CNN Dropout=0.6: Epochs=20, Train accuracy=0.99817, Validation accuracy=0.99080 CNN Dropout=0.7: Epochs=20, Train accuracy=0.99783, Validation accuracy=0.99270

7-Layer ConvNNet + Adam + Relu

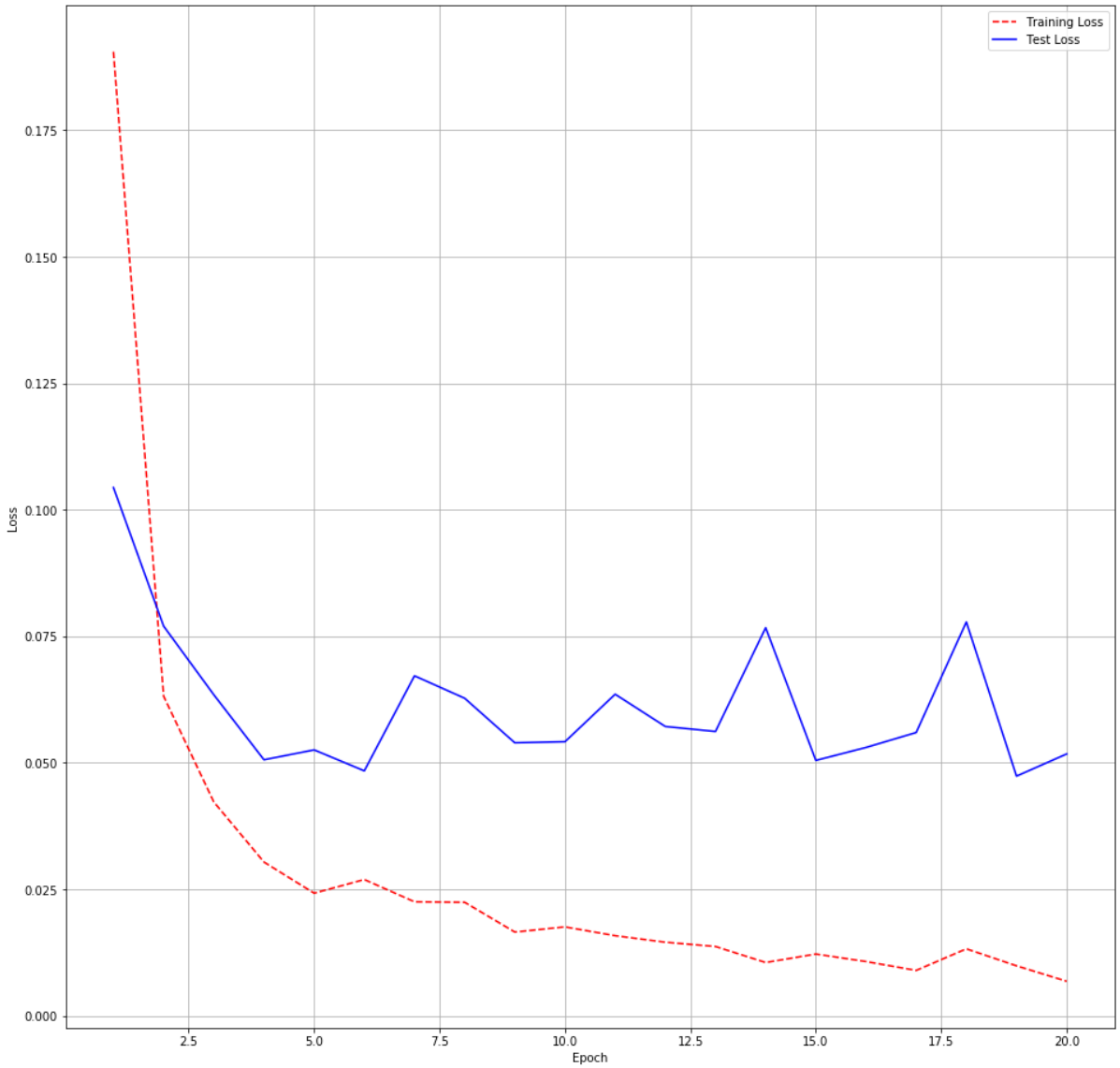
Batch Normalization with Dropout rate 0.2

In [19]:
```
# some model parameters

output_dim = 10
input_dim = X_train.shape[1]

batch_size = 128
no_epoch = 20

#ConvNet
mdl_5 = Sequential()
mdl_5.add(Conv2D(32,kernel_size=(3,3), activation='relu',input_shape=(input
_shape)))
mdl_5.add(Conv2D(32,kernel_size=(3,3), activation='relu'))
mdl_5.add(Conv2D(32,kernel_size=(3,3), activation='relu'))
mdl_5.add(Dropout(rate=0.2))

mdl_5.add(Conv2D(64,kernel_size=(3,3), activation='relu'))
mdl_5.add(Conv2D(64,kernel_size=(3,3), activation='relu'))

mdl_5.add(Dropout(rate=0.2))
mdl_5.add(BatchNormalization())

mdl_5.add(Conv2D(64,kernel_size=(5,5), activation='relu'))
mdl_5.add(Conv2D(128,kernel_size=(2,2), activation='relu'))

mdl_5.add(BatchNormalization())
mdl_5.add(Flatten())
mdl_5.add(Dense(128, activation='relu'))
mdl_5.add(Dense(num_clas,activation='softmax'))

mdl_5.summary()


mdl_5.compile(optimizer='adam',loss='categorical_crossentropy', metrics=['a
ccuracy'])
history_5 = mdl_5.fit(X_train,y_train,batch_size=batch_size,epochs=no_epoch
,verbose=1,validation_data=(x_test,y_test))
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_22 (Conv2D)           (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 24, 24, 32)        9248
_________________________________________________________________
conv2d_24 (Conv2D)           (None, 22, 22, 32)        9248
_________________________________________________________________
dropout_7 (Dropout)          (None, 22, 22, 32)        0
_________________________________________________________________
conv2d_25 (Conv2D)           (None, 20, 20, 64)        18496
_________________________________________________________________
conv2d_26 (Conv2D)           (None, 18, 18, 64)        36928
_________________________________________________________________
dropout_8 (Dropout)          (None, 18, 18, 64)        0
_________________________________________________________________
batch_normalization_7 (Batch (None, 18, 18, 64)        256
_________________________________________________________________
conv2d_27 (Conv2D)           (None, 14, 14, 64)        102464
_________________________________________________________________
conv2d_28 (Conv2D)           (None, 13, 13, 128)       32896
_________________________________________________________________
batch_normalization_8 (Batch (None, 13, 13, 128)       512
_________________________________________________________________
flatten_4 (Flatten)          (None, 21632)             0
_________________________________________________________________
dense_7 (Dense)              (None, 128)               2769024
_________________________________________________________________
dense_8 (Dense)              (None, 10)                1290
=================================================================
Total params: 2,980,682
Trainable params: 2,980,298
Non-trainable params: 384
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 30s 498us/step - loss: 0.190
5 - acc: 0.9527 - val_loss: 0.1044 - val_acc: 0.9759
Epoch 2/20
60000/60000 [==============================] - 28s 473us/step - loss: 0.063
2 - acc: 0.9831 - val_loss: 0.0770 - val_acc: 0.9785
Epoch 3/20
60000/60000 [==============================] - 28s 470us/step - loss: 0.042
3 - acc: 0.9881 - val_loss: 0.0634 - val_acc: 0.9839
Epoch 4/20
60000/60000 [==============================] - 28s 473us/step - loss: 0.030
4 - acc: 0.9916 - val_loss: 0.0506 - val_acc: 0.9887
Epoch 5/20
60000/60000 [==============================] - 28s 472us/step - loss: 0.024
2 - acc: 0.9928 - val_loss: 0.0525 - val_acc: 0.9858
Epoch 6/20
60000/60000 [==============================] - 28s 470us/step - loss: 0.026
9 - acc: 0.9927 - val_loss: 0.0484 - val_acc: 0.9876
Epoch 7/20
60000/60000 [==============================] - 28s 471us/step - loss: 0.022
```

```
5 - acc: 0.9935 - val_loss: 0.0672 - val_acc: 0.9865
Epoch 8/20
60000/60000 [==============================] - 28s 472us/step - loss: 0.022
4 - acc: 0.9935 - val_loss: 0.0628 - val_acc: 0.9876
Epoch 9/20
60000/60000 [==============================] - 28s 472us/step - loss: 0.016
6 - acc: 0.9952 - val_loss: 0.0540 - val_acc: 0.9886
Epoch 10/20
60000/60000 [==============================] - 28s 471us/step - loss: 0.017
6 - acc: 0.9949 - val_loss: 0.0542 - val_acc: 0.9900
Epoch 11/20
60000/60000 [==============================] - 28s 472us/step - loss: 0.015
8 - acc: 0.9957 - val_loss: 0.0635 - val_acc: 0.9866
Epoch 12/20
60000/60000 [==============================] - 28s 472us/step - loss: 0.014
5 - acc: 0.9959 - val_loss: 0.0572 - val_acc: 0.9879
Epoch 13/20
60000/60000 [==============================] - 29s 476us/step - loss: 0.013
7 - acc: 0.9962 - val_loss: 0.0562 - val_acc: 0.9898
Epoch 14/20
60000/60000 [==============================] - 29s 475us/step - loss: 0.010
5 - acc: 0.9973 - val_loss: 0.0767 - val_acc: 0.9854
Epoch 15/20
60000/60000 [==============================] - 29s 477us/step - loss: 0.012
2 - acc: 0.9969 - val_loss: 0.0505 - val_acc: 0.9898
Epoch 16/20
60000/60000 [==============================] - 29s 480us/step - loss: 0.010
7 - acc: 0.9971 - val_loss: 0.0530 - val_acc: 0.9904
Epoch 17/20
60000/60000 [==============================] - 28s 472us/step - loss: 0.009
0 - acc: 0.9975 - val_loss: 0.0560 - val_acc: 0.9906
Epoch 18/20
60000/60000 [==============================] - 28s 472us/step - loss: 0.013
2 - acc: 0.9965 - val_loss: 0.0778 - val_acc: 0.9850
Epoch 19/20
60000/60000 [==============================] - 28s 472us/step - loss: 0.009
9 - acc: 0.9975 - val_loss: 0.0474 - val_acc: 0.9917
Epoch 20/20
60000/60000 [==============================] - 28s 471us/step - loss: 0.006
8 - acc: 0.9981 - val_loss: 0.0517 - val_acc: 0.9919
```

In [20]:
```
%matplotlib inline
import keras
from matplotlib import pyplot as plt

epoch_count = list(range(1,no_epoch+1))

# Get training and test loss histories
training_loss = history_5.history['loss']
test_loss = history_5.history['val_loss']

# Create count of the number of epochs
score = mdl_5.evaluate(x_test, y_test, verbose=0)
plt_mdl_res(epoch_count, training_loss, test_loss, score[0], score[1])
```




```
Test score: 0.05174927458126408
Test accuracy: 0.9919
```

Conclusion

In [3]:
```
import tabulate
res_tab = [['Layer','Batch \nNormalization','Dropout','Dropout \nRate','Tes
t \nAccuracy'],
           [3,'No','No',0.0,0.9938],
           [3,'No','Yes',0.0,0.9928],
           [5,'Yes','No',0.0,0.9934],
           [7,'Yes','Yes',0.0,0.9810],
           [7,'Yes','Yes',0.2,0.9919]
          ]
print(tabulate.tabulate(res_tab,tablefmt='fancy_grid'))
```

Layer	Batch Normalization	Dropout	Dropout Rate	Test Accuracy
3	No	No	0.0	0.9938
3	No	Yes	0.0	0.9928
5	Yes	No	0.0	0.9934
7	Yes	Yes	0.0	0.981
7	Yes	Yes	0.2	0.9919