12-Copy1

November 18, 2018

1 Assignment 12 - Neural Networks image recognition

Use both MLNN and the ConvNet to solve the following problem.

1. Add random noise (i.e. `np.random.normal`) to the images in training and testing. Make sure each image gets a different noise feature added to it. Inspect by printing out an image.
2. Compare the loss/accuracy (train, val) after N epochs for both MLNN and ConvNet with and without noise.
3. Vary the amount of noise (multiply `np.random.normal` by a factor) and keep track of the accuracy and loss (for training and validation) and plot these results.

2 Neural Networks - Image Recognition

```
In [1]: import keras
        from keras.datasets import mnist
        from keras.models import Sequential
        from keras.optimizers import RMSprop
        from keras.layers import Dense, Dropout, Flatten
        from keras.layers import Conv2D, MaxPooling2D
        from keras import backend
        from keras import utils as np_utils

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

```
Using TensorFlow backend.
```

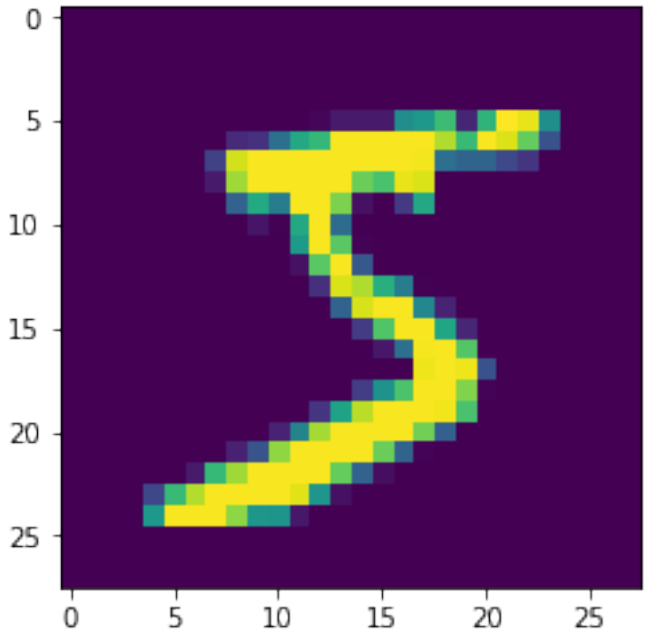
2.1 Multi Layer Neural Network

Trains a simple deep NN on the MNIST dataset. Gets to 98.40% test accuracy after 20 epochs (there is *a lot* of margin for parameter tuning).

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

In [3]:
```
plt.imshow(x_train[0])
```

Out[3]:
```
<matplotlib.image.AxesImage at 0x10ed86780>
```



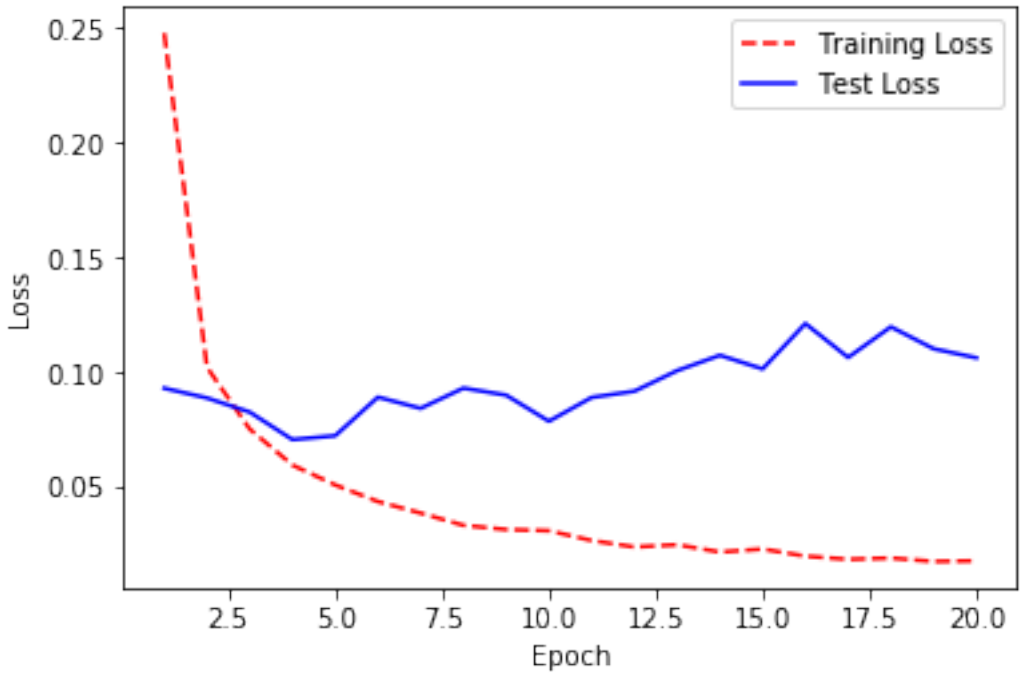
In [4]:
```
# Graph without noise
(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

batch_size = 128
num_classes = 10
epochs = 20

# 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(Dense(512, activation='relu', input_shape=(784,)))
```

```
        model.add(Dropout(0.2))
        model.add(Dense(512, activation='relu'))
        model.add(Dropout(0.2))
        model.add(Dense(10, activation='softmax'))

        model.summary()

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

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

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

        # Create count of the number of epochs
        epoch_count = range(1, len(training_loss) + 1)

        # Visualize loss history
        plt.plot(epoch_count, training_loss, 'r--')
        plt.plot(epoch_count, test_loss, 'b-')
        plt.legend(['Training Loss', 'Test Loss'])
        plt.xlabel('Epoch')
        plt.ylabel('Loss')
        plt.show();
```

```
60000 train samples
10000 test samples
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 512)               401920
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               262656
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
```

```
dense_3 (Dense)              (None, 10)                5130
=================================================================
Total params: 669,706
Trainable params: 669,706
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 8s 126us/step - loss: 0.2479 - acc: 0.9236 - val_
Epoch 2/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.1023 - acc: 0.9695 - val_
Epoch 3/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.0753 - acc: 0.9774 - val_
Epoch 4/20
60000/60000 [==============================] - 8s 125us/step - loss: 0.0596 - acc: 0.9826 - val_
Epoch 5/20
60000/60000 [==============================] - 8s 126us/step - loss: 0.0510 - acc: 0.9842 - val_
Epoch 6/20
60000/60000 [==============================] - 8s 126us/step - loss: 0.0437 - acc: 0.9865 - val_
Epoch 7/20
60000/60000 [==============================] - 9s 142us/step - loss: 0.0387 - acc: 0.9883 - val_
Epoch 8/20
60000/60000 [==============================] - 9s 157us/step - loss: 0.0334 - acc: 0.9902 - val_
Epoch 9/20
60000/60000 [==============================] - 9s 144us/step - loss: 0.0316 - acc: 0.9909 - val_
Epoch 10/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.0310 - acc: 0.9910 - val_
Epoch 11/20
60000/60000 [==============================] - 10s 165us/step - loss: 0.0268 - acc: 0.9924 - val
Epoch 12/20
60000/60000 [==============================] - 9s 151us/step - loss: 0.0239 - acc: 0.9930 - val_
Epoch 13/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.0249 - acc: 0.9929 - val_
Epoch 14/20
60000/60000 [==============================] - 7s 124us/step - loss: 0.0217 - acc: 0.9939 - val_
Epoch 15/20
60000/60000 [==============================] - 8s 125us/step - loss: 0.0231 - acc: 0.9937 - val_
Epoch 16/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.0199 - acc: 0.9941 - val_
Epoch 17/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.0186 - acc: 0.9949 - val_
Epoch 18/20
60000/60000 [==============================] - 7s 123us/step - loss: 0.0191 - acc: 0.9945 - val_
Epoch 19/20
60000/60000 [==============================] - 7s 123us/step - loss: 0.0176 - acc: 0.9953 - val_
Epoch 20/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.0179 - acc: 0.9953 - val_
Test loss: 0.10639518911132022
```

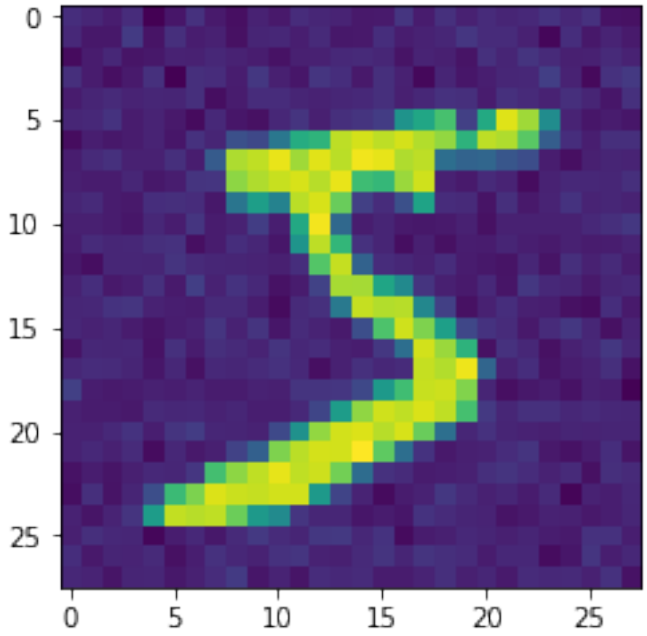
```
Test accuracy: 0.9837
```




```
In [ ]: # 98.37% accuracy
```

```
In [5]: # Noise sigma = 10
        (x_train, y_train), (x_test, y_test) = mnist.load_data()

        # x_train values after the noise
        mu, sigma = 0, 10
        noise = np.random.normal(mu, sigma, [28,28])
        x_train = x_train - noise

        plt.imshow(x_train[0])
```

```
Out[5]: <matplotlib.image.AxesImage at 0xb28b30d68>
```

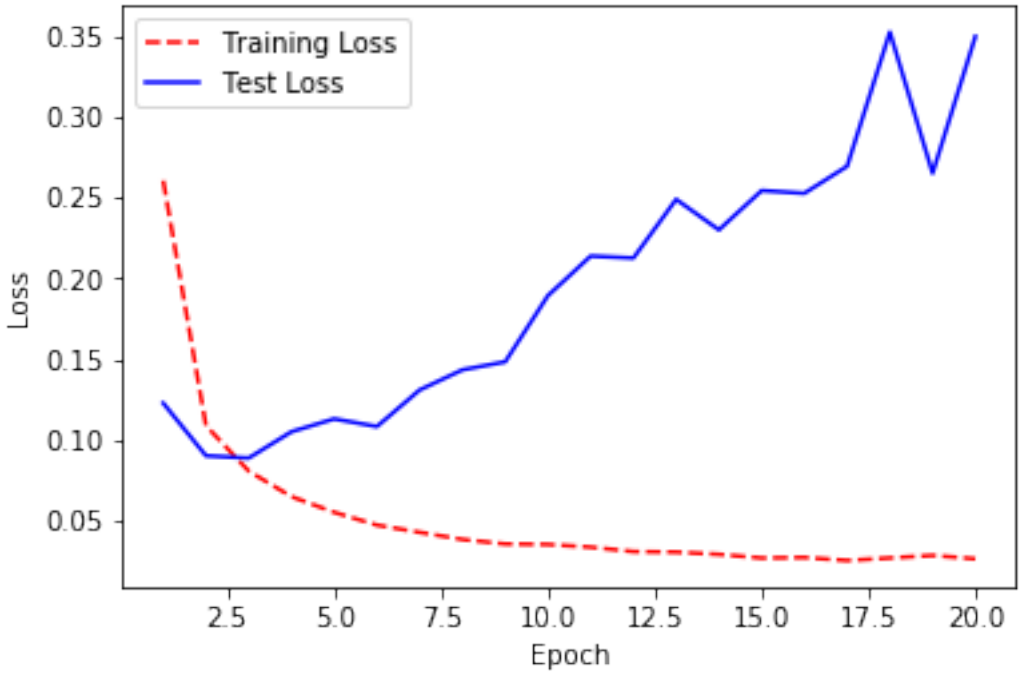


In [6]:
```
x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255

# 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)

batch_size = 128
num_classes = 10
epochs = 20

model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation='softmax'))

model.summary()

model.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(),
```

```
                    metrics=['accuracy'])

        history = model.fit(x_train, y_train,
                            batch_size=batch_size,
                            epochs=epochs,
                            verbose=1,
                            validation_data=(x_test, y_test))

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

        # Create count of the number of epochs
        epoch_count = range(1, len(training_loss) + 1)

        # Visualize loss history
        plt.plot(epoch_count, training_loss, 'r--')
        plt.plot(epoch_count, test_loss, 'b-')
        plt.legend(['Training Loss', 'Test Loss'])
        plt.xlabel('Epoch')
        plt.ylabel('Loss')
        plt.show();
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 512)               401920
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_5 (Dense)              (None, 512)               262656
_________________________________________________________________
dropout_4 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_6 (Dense)              (None, 10)                5130
=================================================================
Total params: 669,706
Trainable params: 669,706
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 8s 131us/step - loss: 0.2603 - acc: 0.9196 - val_
Epoch 2/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.1087 - acc: 0.9670 - val_
Epoch 3/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.0806 - acc: 0.9760 - val_
Epoch 4/20
```

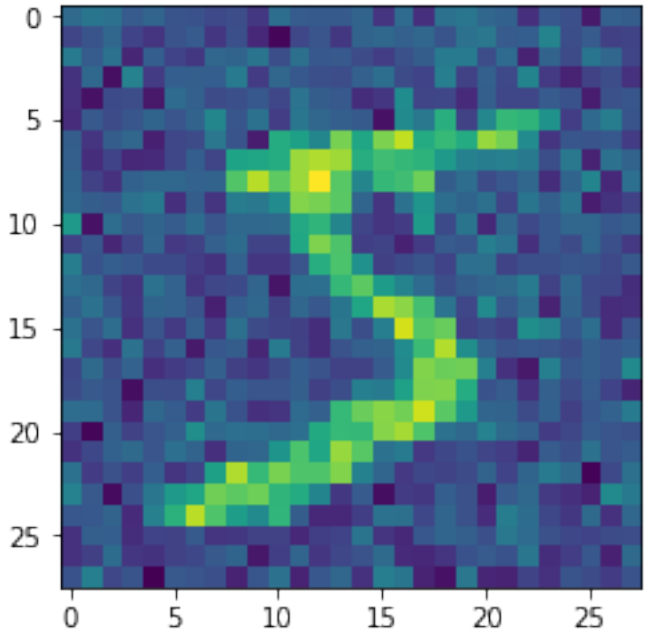
```
60000/60000 [==============================] - 7s 123us/step - loss: 0.0648 - acc: 0.9815 - val_
Epoch 5/20
60000/60000 [==============================] - 8s 127us/step - loss: 0.0551 - acc: 0.9834 - val_
Epoch 6/20
60000/60000 [==============================] - 8s 127us/step - loss: 0.0472 - acc: 0.9857 - val_
Epoch 7/20
60000/60000 [==============================] - 7s 123us/step - loss: 0.0427 - acc: 0.9874 - val_
Epoch 8/20
60000/60000 [==============================] - 8s 128us/step - loss: 0.0383 - acc: 0.9888 - val_
Epoch 9/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.0355 - acc: 0.9898 - val_
Epoch 10/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.0352 - acc: 0.9904 - val_
Epoch 11/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.0336 - acc: 0.9908 - val_
Epoch 12/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.0308 - acc: 0.9919 - val_
Epoch 13/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.0305 - acc: 0.9920 - val_
Epoch 14/20
60000/60000 [==============================] - 7s 123us/step - loss: 0.0291 - acc: 0.9923 - val_
Epoch 15/20
60000/60000 [==============================] - 7s 123us/step - loss: 0.0268 - acc: 0.9931 - val_
Epoch 16/20
60000/60000 [==============================] - 8s 126us/step - loss: 0.0272 - acc: 0.9930 - val_
Epoch 17/20
60000/60000 [==============================] - 8s 131us/step - loss: 0.0253 - acc: 0.9935 - val_
Epoch 18/20
60000/60000 [==============================] - 8s 126us/step - loss: 0.0269 - acc: 0.9934 - val_
Epoch 19/20
60000/60000 [==============================] - 7s 125us/step - loss: 0.0284 - acc: 0.9934 - val_
Epoch 20/20
60000/60000 [==============================] - 8s 130us/step - loss: 0.0264 - acc: 0.9938 - val_
```



In [ ]:
```
# 96.91 percent accuracy
```

In [8]:
```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
mu, sigma = 0, 50
noise = np.random.normal(mu, sigma, [28,28])
x_train = x_train - noise
x_train[0]
plt.imshow(x_train[0])
```

Out[8]:
```
<matplotlib.image.AxesImage at 0xb29240e10>
```



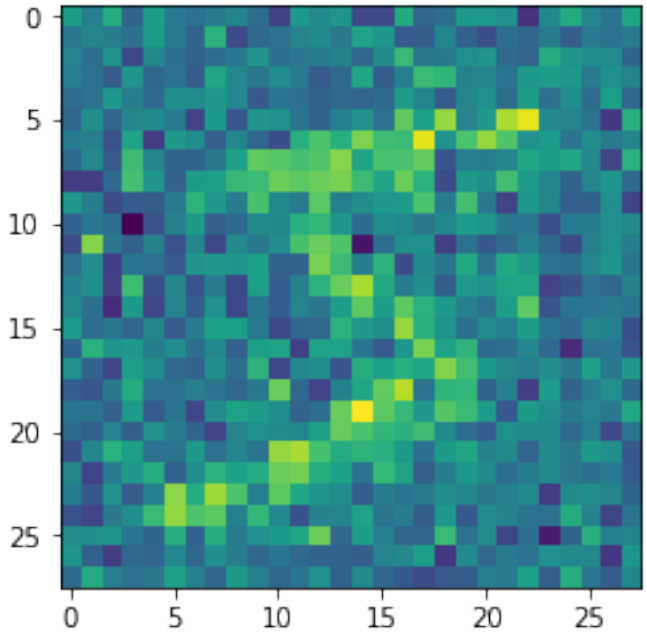
In [9]:
```
x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255

# 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)

batch_size = 128
num_classes = 10
epochs = 20

model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation='softmax'))

model.summary()

model.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(),
```

```
                    metrics=['accuracy'])

        history = model.fit(x_train, y_train,
                            batch_size=batch_size,
                            epochs=epochs,
                            verbose=1,
                            validation_data=(x_test, y_test))

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

        # Create count of the number of epochs
        epoch_count = range(1, len(training_loss) + 1)

        # Visualize loss history
        plt.plot(epoch_count, training_loss, 'r--')
        plt.plot(epoch_count, test_loss, 'b-')
        plt.legend(['Training Loss', 'Test Loss'])
        plt.xlabel('Epoch')
        plt.ylabel('Loss')
        plt.show();
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_7 (Dense)              (None, 512)               401920
_________________________________________________________________
dropout_5 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_8 (Dense)              (None, 512)               262656
_________________________________________________________________
dropout_6 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_9 (Dense)              (None, 10)                5130
=================================================================
Total params: 669,706
Trainable params: 669,706
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 9s 152us/step - loss: 0.3090 - acc: 0.9022 - val_
Epoch 2/20
60000/60000 [==============================] - 8s 137us/step - loss: 0.1346 - acc: 0.9590 - val_
Epoch 3/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.1017 - acc: 0.9685 - val_
Epoch 4/20
```

```
60000/60000 [==============================] - 7s 120us/step - loss: 0.0871 - acc: 0.9739 - val_
Epoch 5/20
60000/60000 [==============================] - 7s 125us/step - loss: 0.0784 - acc: 0.9768 - val_
Epoch 6/20
60000/60000 [==============================] - 7s 125us/step - loss: 0.0722 - acc: 0.9792 - val_
Epoch 7/20
60000/60000 [==============================] - 8s 131us/step - loss: 0.0670 - acc: 0.9807 - val_
Epoch 8/20
60000/60000 [==============================] - 8s 128us/step - loss: 0.0617 - acc: 0.9824 - val_
Epoch 9/20
60000/60000 [==============================] - 8s 130us/step - loss: 0.0610 - acc: 0.9833 - val_
Epoch 10/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.0592 - acc: 0.9842 - val_
Epoch 11/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.0593 - acc: 0.9848 - val_
Epoch 12/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.0549 - acc: 0.9852 - val_
Epoch 13/20
60000/60000 [==============================] - 8s 138us/step - loss: 0.0558 - acc: 0.9862 - val_
Epoch 14/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.0471 - acc: 0.9877 - val_
Epoch 15/20
60000/60000 [==============================] - 8s 131us/step - loss: 0.0528 - acc: 0.9873 - val_
Epoch 16/20
60000/60000 [==============================] - 8s 131us/step - loss: 0.0509 - acc: 0.9878 - val_
Epoch 17/20
60000/60000 [==============================] - 8s 131us/step - loss: 0.0488 - acc: 0.9880 - val_
Epoch 18/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.0499 - acc: 0.9879 - val_
Epoch 19/20
60000/60000 [==============================] - 8s 128us/step - loss: 0.0496 - acc: 0.9882 - val_
Epoch 20/20
60000/60000 [==============================] - 8s 126us/step - loss: 0.0495 - acc: 0.9883 - val_
```


```
In [ ]: # 95.64 percent accuracy

In [12]: (x_train, y_train), (x_test, y_test) = mnist.load_data()
         mu, sigma = 0, 99
         noise = np.random.normal(mu, sigma, [28,28])
         x_train = x_train - noise
         x_train[0]
         plt.imshow(x_train[0])

Out[12]: <matplotlib.image.AxesImage at 0xb2a907518>
```

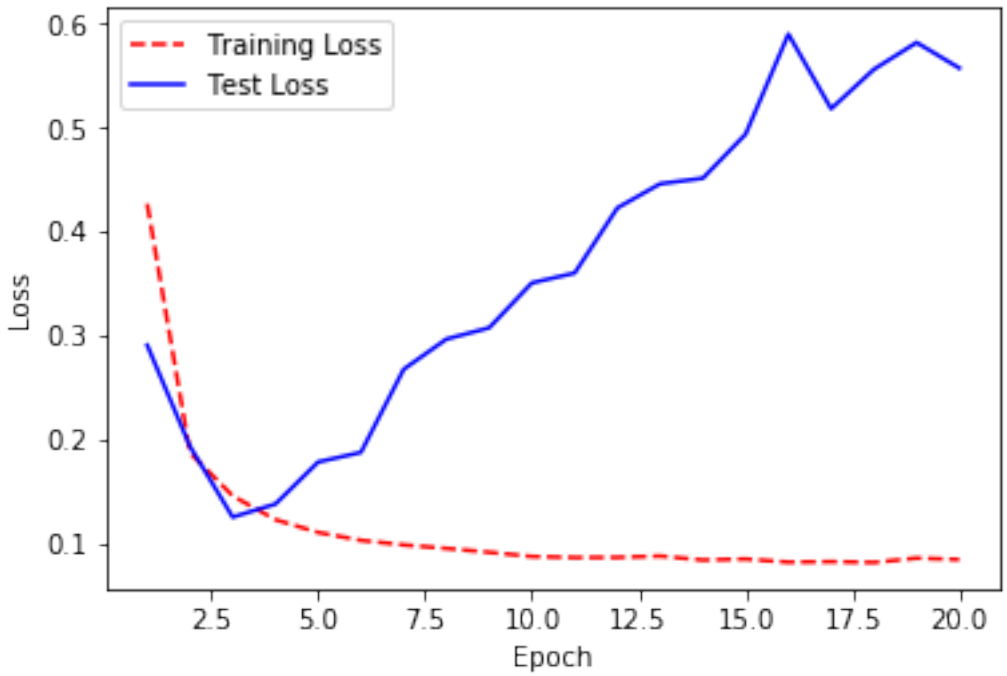


In [13]:
```
x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255

# 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)

batch_size = 128
num_classes = 10
epochs = 20

model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation='softmax'))

model.summary()

model.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(),
```

```
                    metrics=['accuracy'])

        history = model.fit(x_train, y_train,
                            batch_size=batch_size,
                            epochs=epochs,
                            verbose=1,
                            validation_data=(x_test, y_test))

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

        # Create count of the number of epochs
        epoch_count = range(1, len(training_loss) + 1)

        # Visualize loss history
        plt.plot(epoch_count, training_loss, 'r--')
        plt.plot(epoch_count, test_loss, 'b-')
        plt.legend(['Training Loss', 'Test Loss'])
        plt.xlabel('Epoch')
        plt.ylabel('Loss')
        plt.show();
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_10 (Dense)             (None, 512)               401920
_________________________________________________________________
dropout_7 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_11 (Dense)             (None, 512)               262656
_________________________________________________________________
dropout_8 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_12 (Dense)             (None, 10)                5130
=================================================================
Total params: 669,706
Trainable params: 669,706
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.4268 - acc: 0.8630 - val_
Epoch 2/20
60000/60000 [==============================] - 7s 116us/step - loss: 0.1871 - acc: 0.9421 - val_
Epoch 3/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.1456 - acc: 0.9556 - val_
Epoch 4/20
```

```
60000/60000 [==============================] - 8s 126us/step - loss: 0.1228 - acc: 0.9621 - val_
Epoch 5/20
60000/60000 [==============================] - 8s 126us/step - loss: 0.1105 - acc: 0.9673 - val_
Epoch 6/20
60000/60000 [==============================] - 8s 127us/step - loss: 0.1029 - acc: 0.9692 - val_
Epoch 7/20
60000/60000 [==============================] - 8s 128us/step - loss: 0.0985 - acc: 0.9711 - val_
Epoch 8/20
60000/60000 [==============================] - 8s 129us/step - loss: 0.0951 - acc: 0.9729 - val_
Epoch 9/20
60000/60000 [==============================] - 8s 129us/step - loss: 0.0914 - acc: 0.9740 - val_
Epoch 10/20
60000/60000 [==============================] - 8s 129us/step - loss: 0.0873 - acc: 0.9756 - val_
Epoch 11/20
60000/60000 [==============================] - 8s 128us/step - loss: 0.0864 - acc: 0.9756 - val_
Epoch 12/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.0865 - acc: 0.9764 - val_
Epoch 13/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.0878 - acc: 0.9761 - val_
Epoch 14/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.0839 - acc: 0.9776 - val_
Epoch 15/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.0849 - acc: 0.9777 - val_
Epoch 16/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.0819 - acc: 0.9783 - val_
Epoch 17/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.0824 - acc: 0.9788 - val_
Epoch 18/20
60000/60000 [==============================] - 8s 129us/step - loss: 0.0816 - acc: 0.9792 - val_
Epoch 19/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.0858 - acc: 0.9794 - val_
Epoch 20/20
60000/60000 [==============================] - 8s 128us/step - loss: 0.0844 - acc: 0.9795 - val_
```



2.2 Conv Net

Trains a simple convnet on the MNIST dataset. Gets to 99.25% test accuracy after 12 epochs (there is still a lot of margin for parameter tuning).

```
In [14]: # input image dimensions
         img_rows, img_cols = 28, 28

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

         if backend.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')
```

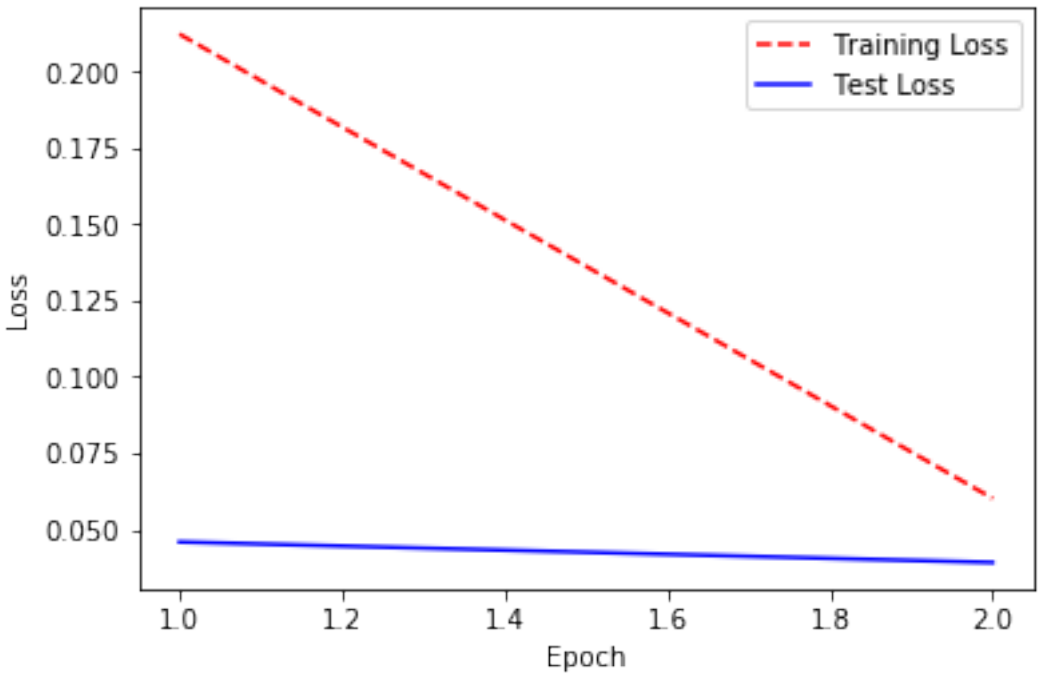
```
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
```

```
In [15]: # 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)
```

```
In [18]: batch_size = 128
         num_classes = 10
         epochs = 2

         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(0.2))
         model.add(Dense(num_classes, activation='softmax'))

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

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/2
60000/60000 [==============================] - 151s 3ms/step - loss: 0.2123 - acc: 0.9348 - val_
Epoch 2/2
60000/60000 [==============================] - 150s 2ms/step - loss: 0.0602 - acc: 0.9821 - val_
Test loss: 0.03919108632341958
Test accuracy: 0.9869
```

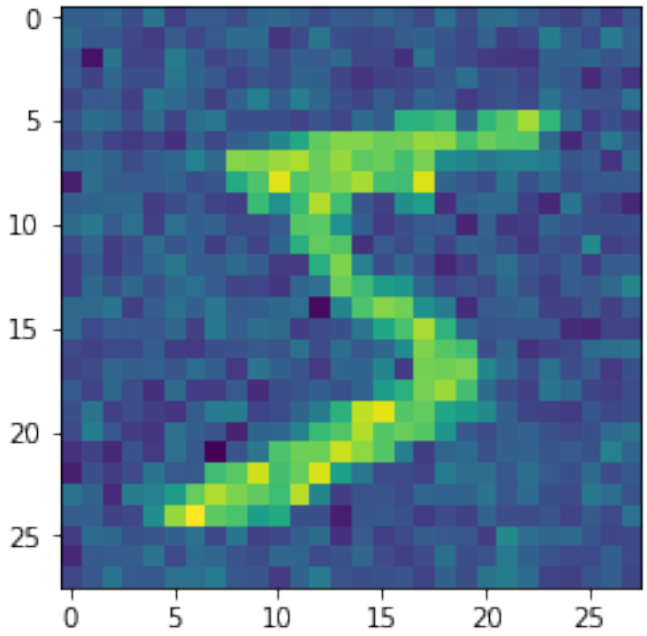
In [20]:
```
# Get training and test loss histories
training_loss = history.history['loss']
test_loss = history.history['val_loss']

# Create count of the number of epochs
epoch_count = range(1, len(training_loss) + 1)

# Visualize loss history
plt.plot(epoch_count, training_loss, 'r--')
plt.plot(epoch_count, test_loss, 'b-')
plt.legend(['Training Loss', 'Test Loss'])
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.show();
```



In [22]:
```
# input image dimensions
img_rows, img_cols = 28, 28

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

mu, sigma = 0, 35
noise = np.random.normal(mu, sigma, [28,28])
x_train = x_train - noise
```

```
x_train[0]
plt.imshow(x_train[0])
```

Out[22]: <matplotlib.image.AxesImage at 0xb2e3fa2b0>

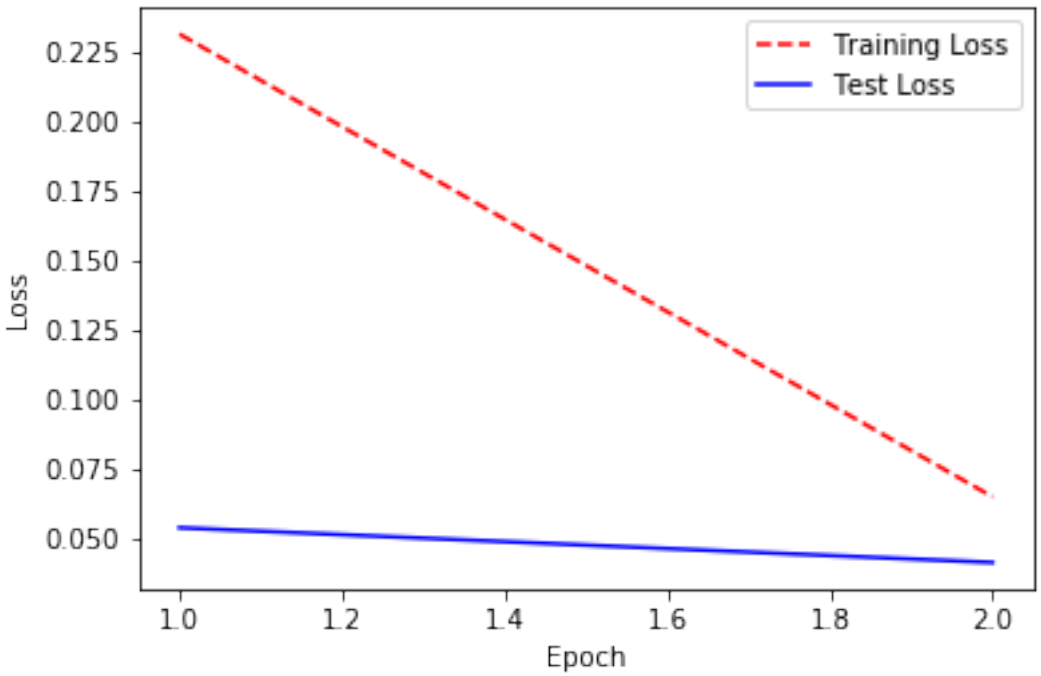


In [23]:
```
if backend.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, 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

# 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)

batch_size = 128
num_classes = 10
epochs = 2
```

```
        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(0.2))
        model.add(Dense(num_classes, activation='softmax'))

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

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

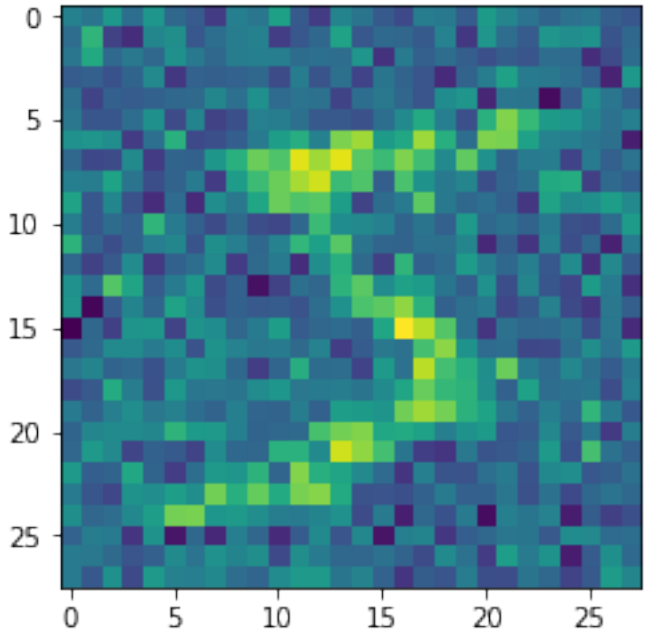
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/2
60000/60000 [==============================] - 161s 3ms/step - loss: 0.2315 - acc: 0.9286 - val_
Epoch 2/2
60000/60000 [==============================] - 160s 3ms/step - loss: 0.0648 - acc: 0.9800 - val_
Test loss: 0.04118220631625736
Test accuracy: 0.9857
```

```
In [24]: # Get training and test loss histories
         training_loss = history.history['loss']
         test_loss = history.history['val_loss']

         # Create count of the number of epochs
         epoch_count = range(1, len(training_loss) + 1)

         # Visualize loss history
         plt.plot(epoch_count, training_loss, 'r--')
         plt.plot(epoch_count, test_loss, 'b-')
         plt.legend(['Training Loss', 'Test Loss'])
         plt.xlabel('Epoch')
         plt.ylabel('Loss')
         plt.show();
```



In [25]:
```
# input image dimensions
img_rows, img_cols = 28, 28

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

mu, sigma = 0, 85
noise = np.random.normal(mu, sigma, [28,28])
x_train = x_train - noise
x_train[0]
plt.imshow(x_train[0])
```

Out[25]: <matplotlib.image.AxesImage at 0xb356d7eb8>




```
In [26]: if backend.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

         # 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)

         batch_size = 128
         num_classes = 10
         epochs = 8

         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'])

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/8
60000/60000 [==============================] - 169s 3ms/step - loss: 0.3801 - acc: 0.8812 - val_
Epoch 2/8
60000/60000 [==============================] - 160s 3ms/step - loss: 0.1280 - acc: 0.9623 - val_
Epoch 3/8
60000/60000 [==============================] - 163s 3ms/step - loss: 0.0966 - acc: 0.9716 - val_
Epoch 4/8
60000/60000 [==============================] - 154s 3ms/step - loss: 0.0792 - acc: 0.9766 - val_
Epoch 5/8
60000/60000 [==============================] - 151s 3ms/step - loss: 0.0707 - acc: 0.9789 - val_
Epoch 6/8
60000/60000 [==============================] - 151s 3ms/step - loss: 0.0620 - acc: 0.9825 - val_
Epoch 7/8
60000/60000 [==============================] - 153s 3ms/step - loss: 0.0560 - acc: 0.9828 - val_
Epoch 8/8
60000/60000 [==============================] - 160s 3ms/step - loss: 0.0534 - acc: 0.9839 - val_
Test loss: 0.04573370984120675
Test accuracy: 0.9885
```

In [27]:
```
        # Get training and test loss histories
        training_loss = history.history['loss']
        test_loss = history.history['val_loss']

        # Create count of the number of epochs
        epoch_count = range(1, len(training_loss) + 1)
```

```
# Visualize loss history
plt.plot(epoch_count, training_loss, 'r--')
plt.plot(epoch_count, test_loss, 'b-')
plt.legend(['Training Loss', 'Test Loss'])
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.show();
```