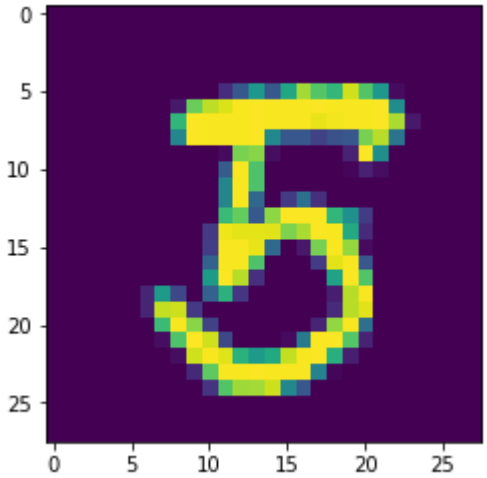
```
Python 3.6.2 |Anaconda custom (64-bit)| (default, Sep 19 2017, 08:03:39) [MSC v.1900 64 bit
(AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 6.1.0 -- An enhanced Interactive Python.

In [1]:

In [1]: from keras.datasets import mnist
   ...: from keras.models import Sequential
   ...: from keras.layers import Dense, Dropout, Activation, Flatten
   ...: from keras.layers import Convolution2D, MaxPooling2D, Dense, Dropout, Flatten, Conv2D,
MaxPool2D, BatchNormalization
   ...: from keras.utils import np_utils
   ...: from keras import backend as K
   ...: K.set_image_dim_ordering('th')
   ...: import matplotlib
   ...: import matplotlib.pyplot as plt
   ...: import numpy as np
   ...: %matplotlib inline
   ...: from sklearn.model_selection import train_test_split
   ...: from sklearn.metrics import confusion_matrix
   ...: from keras.utils.np_utils import to_categorical
   ...: from keras.models import Sequential
   ...: from keras.optimizers import Adam
   ...: from keras.preprocessing.image import ImageDataGenerator
   ...: from keras.callbacks import LearningRateScheduler
   ...:
   ...: #batch size to train
   ...: batch_size = 128
   ...: #number of output classes
   ...: nb_classes = 10
   ...: #numbe of epochs to train
   ...: nb_epoch = 12
   ...:
   ...: # input image dimensions
   ...: img_rows, img_cols = 28, 28
   ...: # number of convolutional filters to use
   ...: nb_filters = 32
   ...: #size of pooling area for max pooling
   ...: nb_pool = 2
   ...: #convolution kernel size
   ...: nb_conv = 3
   ...:
   ...: # the data, shuffled and split between train and test sets
   ...: (X_train, y_train), (X_test, y_test) = mnist.load_data()
   ...:
   ...: #Reshape the data
   ...: 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)
   ...:
   ...: X_train = X_train.astype('float32')
   ...: X_test = X_test.astype('float32')
   ...: #normalize
   ...: 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 = np_utils.to_categorical(y_train, nb_classes)
   ...: Y_test = np_utils.to_categorical(y_test, nb_classes)
   ...:
```

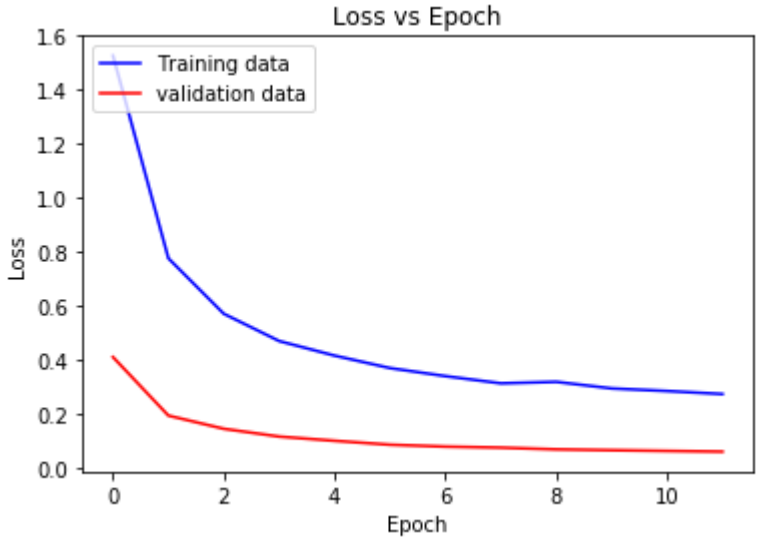
```
   ...: #Consider an instance and check the image mapping
   ...: i = 4600
   ...: plt.imshow(X_train[i, 0], interpolation="nearest")
   ...: print('Label:', Y_train[i:])
Using Theano backend.
X_train shape: (60000, 1, 28, 28)
60000 train samples
10000 test samples
Label: [[ 0.  0.  0. ...,  0.  0.  0.]
 [ 0.  0.  0. ...,  0.  0.  0.]
 [ 0.  1.  0. ...,  0.  0.  0.]
 ...,
 [ 0.  0.  0. ...,  0.  0.  0.]
 [ 0.  0.  0. ...,  0.  0.  0.]
 [ 0.  0.  0. ...,  0.  1.  0.]]
```

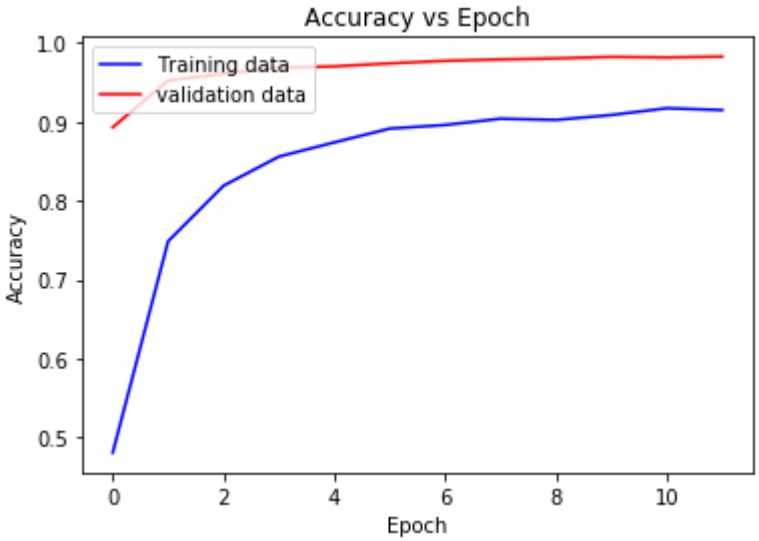



```
In [2]: model = Sequential()
   ...: model.add(Convolution2D(nb_filters, nb_conv, nb_conv,
   ...:                         border_mode='valid',
   ...:                         input_shape=(1,img_rows,img_cols)))
   ...: convout1 = Activation('relu')
   ...: model.add(Activation('relu'))
   ...: model.add(Convolution2D(nb_filters, nb_conv, nb_conv))
   ...: convout2 = Activation('relu')
   ...: model.add(Activation('relu'))
   ...: model.add(MaxPooling2D(pool_size=(nb_pool, nb_pool)))
   ...: model.add(Dropout(0.25))
   ...: model.add(Flatten())
   ...: model.add(Dense(128))
   ...: model.add(Activation('relu'))
   ...: model.add(Dropout(0.5))
   ...: model.add(Dense(nb_classes))
   ...: model.add(Activation('softmax'))
   ...: model.compile(loss='categorical_crossentropy',optimizer='adadelta')
   ...:
   ...: #Augmentation used for regularization
   ...: datagen = ImageDataGenerator(zoom_range = 0.1,height_shift_range = 0.1,width_shift_range =
0.1,rotation_range = 10)
   ...: #model compilation: optimizing using adams
   ...: model.compile(loss='categorical_crossentropy', optimizer = Adam(lr=1e-4), metrics=
["accuracy"])
   ...: #Reduce the learning rate by 10% after every epoch
   ...: annealer = LearningRateScheduler(lambda x: 1e-3 * 0.9 ** x)
   ...: #Trainnig the model with a small validation set(steps per epoch= 500)
   ...: hist = model.fit_generator(datagen.flow(X_train, Y_train, batch_size=batch_size),
   ...:                            steps_per_epoch=50,
   ...:                            epochs=nb_epoch, #Increase this when not on Kaggle kernel
```

```
   ...:                             verbose=1,  #1 for ETA, 0 for silent
   ...:                             validation_data= (X_test, Y_test), #For speed
   ...:                             callbacks=[annealer])
__main__:4: UserWarning: Update your `Conv2D` call to the Keras 2 API: `Conv2D(32, (3, 3),
input_shape=(1, 28, 28..., padding="valid")`
__main__:7: UserWarning: Update your `Conv2D` call to the Keras 2 API: `Conv2D(32, (3, 3))`
Epoch 1/12
50/50 [==============================] - 16s - loss: 1.5275 - acc: 0.4808 - val_loss: 0.4097 -
val_acc: 0.8928
Epoch 2/12
50/50 [==============================] - 16s - loss: 0.7759 - acc: 0.7486 - val_loss: 0.1926 -
val_acc: 0.9521
Epoch 3/12
50/50 [==============================] - 16s - loss: 0.5702 - acc: 0.8189 - val_loss: 0.1437 -
val_acc: 0.9608
Epoch 4/12
50/50 [==============================] - 16s - loss: 0.4690 - acc: 0.8558 - val_loss: 0.1148 -
val_acc: 0.9683
Epoch 5/12
50/50 [==============================] - 16s - loss: 0.4150 - acc: 0.8738 - val_loss: 0.0991 -
val_acc: 0.9700
Epoch 6/12
50/50 [==============================] - 16s - loss: 0.3688 - acc: 0.8912 - val_loss: 0.0846 -
val_acc: 0.9738
Epoch 7/12
50/50 [==============================] - 16s - loss: 0.3390 - acc: 0.8958 - val_loss: 0.0776 -
val_acc: 0.9771
Epoch 8/12
50/50 [==============================] - 16s - loss: 0.3122 - acc: 0.9038 - val_loss: 0.0737 -
val_acc: 0.9788
Epoch 9/12
50/50 [==============================] - 16s - loss: 0.3177 - acc: 0.9022 - val_loss: 0.0674 -
val_acc: 0.9801
Epoch 10/12
50/50 [==============================] - 17s - loss: 0.2932 - acc: 0.9083 - val_loss: 0.0643 -
val_acc: 0.9821
Epoch 11/12
50/50 [==============================] - 16s - loss: 0.2837 - acc: 0.9170 - val_loss: 0.0615 -
val_acc: 0.9813
Epoch 12/12
50/50 [==============================] - 16s - loss: 0.2726 - acc: 0.9147 - val_loss: 0.0590 -
val_acc: 0.9824

In [3]: final_loss, final_acc = model.evaluate(X_test, Y_test, verbose=0)
   ...: print("Final loss: {0:.4f}, final accuracy: {1:.4f}".format(final_loss, final_acc))
Final loss: 0.0590, final accuracy: 0.9824

In [5]: y_hat = model.predict(X_test)
   ...: y_pred = np.argmax(y_hat, axis=1)
   ...: y_true = np.argmax(Y_test, axis=1)
   ...: cm = confusion_matrix(y_true, y_pred)
   ...: print(cm)

[[ 973    0    0    0    0    1    3    1    1    1]
 [   0 1125    4    2    0    0    1    1    2    0]
 [   3    1 1019    0    1    0    0    5    3    0]
 [   0    0    0 1001    0    2    0    3    3    1]
 [   0    0    0    0  962    0    6    1    0   13]
 [   0    0    0    8    0  875    3    2    3    1]
 [   3    2    0    0    2    3  946    0    2    0]
 [   1    2   11    4    0    0    0  995    3   12]
 [   7    0    3    0    3    7    2    1  949    2]
 [   3    6    0    5    5    1    0    3    7  979]]
```

```
In [9]: plt.plot(hist.history['loss'], color='b')
   ...: plt.plot(hist.history['val_loss'], color='r')
   ...: plt.legend(['Training data', 'validation data'], loc='upper left')
   ...: plt.title('Loss vs Epoch')
   ...: plt.xlabel('Epoch')
   ...: plt.ylabel('Loss')
   ...: plt.show()
   ...: plt.plot(hist.history['acc'], color='b')
   ...: plt.plot(hist.history['val_acc'], color='r')
   ...: plt.legend(['Training data','validation data'], loc='upper left')
   ...: plt.title('Accuracy vs Epoch')
   ...: plt.xlabel('Epoch')
   ...: plt.ylabel('Accuracy')
   ...: plt.show()
```






```
In [10]:
```