```
# Name: Samsher Bahadur Rana
```

Student ID: 611060

In [0]:

```
import numpy as np
import matplotlib.pyplot as plt
import keras
from keras.models import Sequential
from keras.datasets import mnist
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D
from keras.optimizers import Adam
import seaborn as cbn
from keras.optimizers import RMSprop
from keras.preprocessing.image import ImageDataGenerator
import random
import os
from keras.utils.np_utils import to_categorical # convert to one-hot-encoding
os.environ['KERAS_BACKEND'] = "tensorflow" #or "theano"
cbn.set(rc={'figure.figsize':(17,6)})
np.random.seed(2)
```

In [443]:

```
#loading images data from mnist data set
(x_train, y_train), (x_test, y_test) = mnist.load_data()
print(x_train.shape)
print(x_test.shape)
print(y_train.shape[0])
```

```
(60000, 28, 28)
(10000, 28, 28)
60000
```

In [444]:

```
x_train_sum = np.sum(x_train)
x_train_nan = np.isnan(x_train_sum)
print("x_train contains null data: %s" %x_train_nan)

y_train_sum = np.sum(x_train)
y_train_nan = np.isnan(y_train_sum)
print("y_train contains null data: %s" %y_train_nan)


x_test_sum = np.sum(x_test)
x_test_nan = np.isnan(x_test_sum)
print("x_test contains null data: %s" %x_test_nan)

y_test_sum = np.sum(x_test)
y_test_nan = np.isnan(y_test_sum)
print("y_test contains null data: %s" %y_test_nan)
```

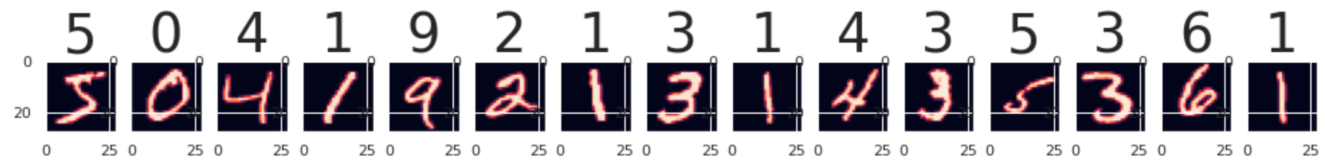
```
x_train contains null data: False
y_train contains null data: False
x_test contains null data: False
y_test contains null data: False
```

In [520]:

```
#We have 60000 images in training set. let's check with 15 elements how they look like
fig = plt.figure()
number_samples = 15
plt.figtext(.16, 0.3, "Images with labeled values")
data_index = 0;
```

```
for x in x_train:
    if(data_index >= number_samples):
        break
    container = fig.add_subplot(1,number_samples,data_index+1)
    imgplot = plt.imshow(x)
    container.set_title(y_train[data_index],fontsize = 40)

    data_index = data_index + 1
```



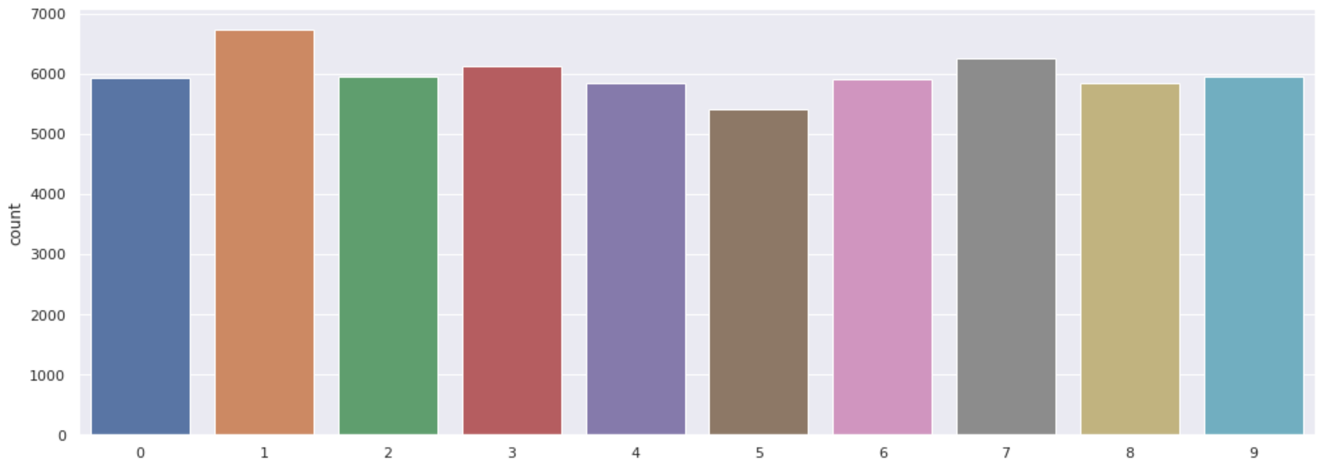
Images with labeled values

In [446]:

```
cbn.countplot(y_train)
```

Out[446]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc8f048b630>
```



In [447]:

```
# lets see how many samples for each digits
unique, counts = np.unique(y_train, return_counts=True)
dict(zip(unique, counts))
```

Out[447]:

```
{0: 5923,
 1: 6742,
 2: 5958,
 3: 6131,
 4: 5842,
 5: 5421,
 6: 5918,
 7: 6265,
 8: 5851,
 9: 5949}
```

In [0]:

```
#now possible output is 0-9. converting these outputs into binary set with 10 digits ex: 2 into [0
,0,0,0,0,0,0,1,0,0]
y_trainNew = to_categorical(y_train, 10)
y_trainNew = to_categorical(y_train, 10)
y_testNew = to_categorical(y_test, 10)
```

In [0]:

```
#Normalization by dividing 255 with maximum value of color
x_trainNew = x_train/255
x_testNew = x_test/255
```

In [458]:

```
#we must change the shape of the images to 1d array(28*28)

num_pixels = 784 #num_pixels: 28*28 to 1*784
x_trainNew = x_trainNew.reshape(x_trainNew.shape[0],28,28,1)
x_testNew = x_testNew.reshape(x_testNew.shape[0],28,28,1)
print(x_testNew.shape)
```

```
(10000, 28, 28, 1)
```

In [0]:

```
num_of_classes = 10 #each digit total: 10
num_pixels = 784 #num_pixels: 28*28
def createCNNmodel():
  model = Sequential()

  model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',
                activation ='relu', input_shape = (28,28,1)))
  model.add(Conv2D(filters = 32, kernel_size = (3,3),padding = 'Same',
                 activation ='relu'))
  model.add(MaxPool2D(pool_size=(2,2), strides=(2,2)))
  model.add(Dropout(0.25))
  model.add(Flatten())
  model.add(Dense(256, activation = "relu"))
  model.add(Dropout(0.5))
  model.add(Dense(10, activation = "softmax"))
  model.compile(Adam(lr=0.01), #lr: learning rate
               loss='categorical_crossentropy', #loss function
              metrics=['accuracy'])
  return model
```

In [465]:

```
history = createCNNmodel().fit(x_trainNew, y_trainNew, validation_split=0.1, epochs=10, batch_size=
200,
                    verbose=1, shuffle=1)
```

```
Train on 54000 samples, validate on 6000 samples
Epoch 1/10
54000/54000 [==============================] - 125s 2ms/step - loss: 0.2283 - accuracy: 0.9299 - v
al_loss: 0.0493 - val_accuracy: 0.9862
Epoch 2/10
54000/54000 [==============================] - 121s 2ms/step - loss: 0.1010 - accuracy: 0.9704 - v
al_loss: 0.0466 - val_accuracy: 0.9877
Epoch 3/10
54000/54000 [==============================] - 119s 2ms/step - loss: 0.0886 - accuracy: 0.9737 - v
al_loss: 0.0536 - val_accuracy: 0.9838
Epoch 4/10
54000/54000 [==============================] - 119s 2ms/step - loss: 0.0832 - accuracy: 0.9754 - v
al_loss: 0.0422 - val_accuracy: 0.9890
Epoch 5/10
54000/54000 [==============================] - 119s 2ms/step - loss: 0.0697 - accuracy: 0.9794 - v
al_loss: 0.0351 - val_accuracy: 0.9905
Epoch 6/10
54000/54000 [==============================] - 124s 2ms/step - loss: 0.0742 - accuracy: 0.9788 - v
al_loss: 0.0427 - val_accuracy: 0.9873
Epoch 7/10
54000/54000 [==============================] - 119s 2ms/step - loss: 0.0748 - accuracy: 0.9782 - v
al_loss: 0.0379 - val_accuracy: 0.9907
Epoch 8/10
54000/54000 [==============================] - 119s 2ms/step - loss: 0.0673 - accuracy: 0.9807 - v
```

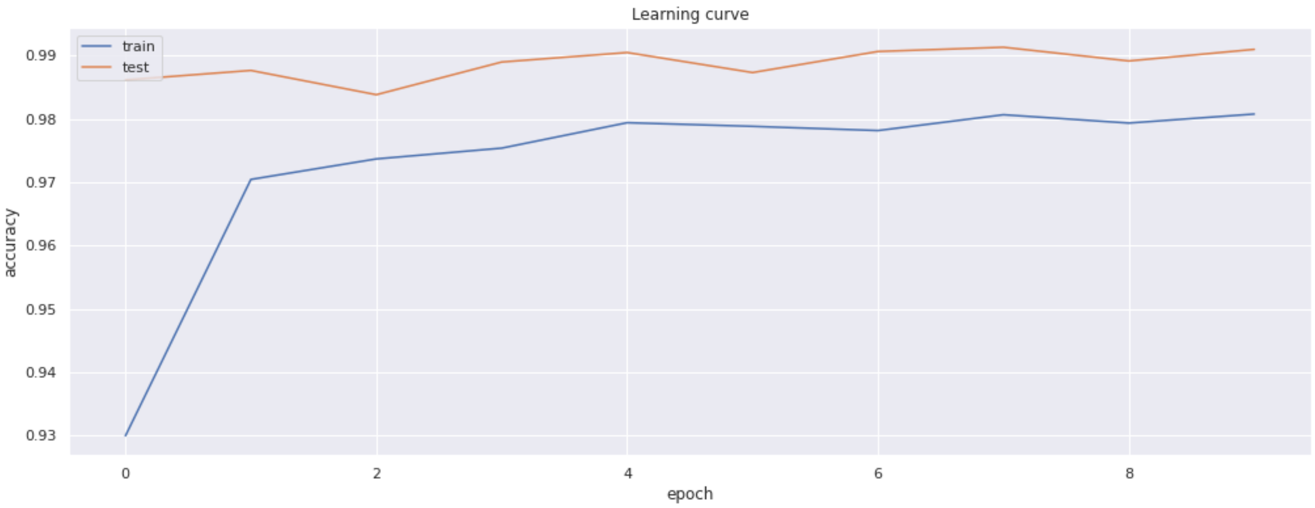
```
al_loss: 0.0428 - val_accuracy: 0.9913
Epoch 9/10
54000/54000 [==============================] - 120s 2ms/step - loss: 0.0745 - accuracy: 0.9794 - v
al_loss: 0.0415 - val_accuracy: 0.9892
Epoch 10/10
54000/54000 [==============================] - 120s 2ms/step - loss: 0.0653 - accuracy: 0.9808 - v
al_loss: 0.0342 - val_accuracy: 0.9910
```

In [0]:

In [0]:

In [475]:

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Learning curve')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



In [0]:

```
score = model.evaluate(x_testNew, y_testNew, verbose=0)
```

In [471]:

```
print(type(score))
print('Test Score:', score[0])
print('Test Accuracy:', score[1])
```

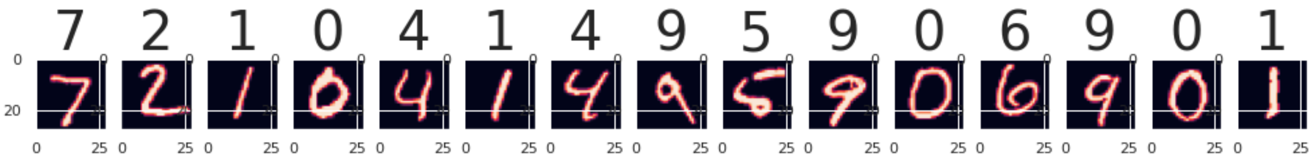
```
<class 'list'>
Test Score: 0.04739607303261291
Test Accuracy: 0.9847999811172485
```

In [519]:

```
# now taking a test with only 15 sample datas
prediction = model.predict_classes(x_testNew[:15])
fig = plt.figure()
plt.figtext(.16, 0.3, "Images with predicted values")
number_samples = 15

data_index = 0;
```

```
for x in x_test:
    if(data_index >= number_samples):
        break
    container = fig.add_subplot(1,number_samples,data_index+1)
    imgplot = plt.imshow(x)
    container.set_title(prediction[data_index],fontsize = 40)
    data_index = data_index + 1
```



```
Images with predicted values
```

In [0]:

```
```