Project: Basic Image Classification

In [8]:
```
#importing libraries
import tensorflow as tf
import keras
import numpy as np
import matplotlib.pyplot as plt
```

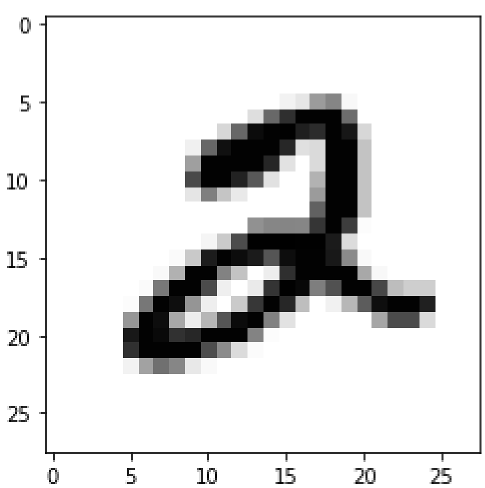
```
Using TensorFlow backend.
```

In [9]:
```
#importing data from keras.datasets
from tensorflow.keras.datasets import mnist

#data in the form of training as well as testing
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

In [10]:
```
#checking the shapes of dataset
print('x_train shape: ', x_train.shape)
print('y_train shape: ', y_train.shape)
print('x_test shape: ', x_test.shape)
print('y_test shape: ', y_test.shape)
```

```
x_train shape:  (60000, 28, 28)
y_train shape:  (60000,)
x_test shape:  (10000, 28, 28)
y_test shape:  (10000,)
```

In [11]:
```
#ploting one of the image of dataset
%matplotlib inline

plt.imshow(x_train[5], cmap = 'binary')
plt.show()
```



In [12]:
```
#cehecking y_train set
print(set(y_train))
```

```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

In [13]:
```
#As this y_train and y_test are the numpy arrays that represents the digit in x set
#now our task is to ENCODE each value into a 10-dimensional vector
#that represents each of the digit
from tensorflow.keras.utils import to_categorical
```

```
y_train_encoded = to_categorical(y_train)
y_test_encoded = to_categorical(y_test)
```

In [14]:

```
print('y_train shape: ', y_train_encoded.shape)
print('y_test shape: ', y_test_encoded.shape)
```

```
y_train shape:  (60000, 10)
y_test shape:  (10000, 10)
```

In [20]:

```
#checking how y_train and y_train_encoded are related
for i in range(5):
    print(y_train[i],y_train_encoded[i])
```

```
5 [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
0 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
4 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
1 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
9 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
```

In [23]:

```
#Preprocessing
#now we create a neural network
#unwrapping the x-(28,28) to x-(784,1)
x_train_reshaped = np.reshape(x_train,(60000,784))
x_test_reshaped = np.reshape(x_test,(10000,784))

print('x_train_reshaped shape: ', x_train_reshaped.shape)
print('x_test_reshaped shape: ', x_test_reshaped.shape)
```

```
x_train_reshaped shape:  (60000, 784)
x_test_reshaped shape:  (10000, 784)
```

In [25]:

```
#Pixel values range from 0 to 255
print(set(x_train_reshaped[0]))
```

```
{0, 1, 2, 3, 9, 11, 14, 16, 18, 23, 24, 25, 26, 27, 30, 35, 36, 39, 43, 45, 46, 49, 55,
56, 64, 66, 70, 78, 80, 81, 82, 90, 93, 94, 107, 108, 114, 119, 126, 127, 130, 132, 133,
135, 136, 139, 148, 150, 154, 156, 160, 166, 170, 171, 172, 175, 182, 183, 186, 187, 19
0, 195, 198, 201, 205, 207, 212, 213, 219, 221, 225, 226, 229, 238, 240, 241, 242, 244,
247, 249, 250, 251, 252, 253, 255}
```

In [30]:

```
#We normalize these values to fit in the model
x_mean = np.mean(x_train_reshaped)
x_std = np.std(x_train_reshaped)

print('mean: ', x_mean)
print('std: ', x_std)

epsilon = 1e-10
x_train_norm = (x_train_reshaped - x_mean)/(x_std+epsilon)
x_test_norm = (x_test_reshaped - x_mean)/(x_std+epsilon)

print(set(x_train_norm[0]))
```

```
mean:  33.318421449829934
std:  78.56748998339798
{-0.38589016215482896, 1.306921966983251, 1.17964285952926, 1.803310486053816, 1.6887592
893452241, 2.8215433456857437, 2.719720059722551, 1.1923707702746593, 1.739670932326820
5, 2.057868700961798, 2.3633385588513764, 2.096052433197995, 1.7651267538176187, 2.79608
75241949457, 2.7451758812133495, 2.45243393406917, 0.02140298169794222, -0.2204273224646
4067, 1.2305545025108566, 0.2759611966059242, 2.210603629906587, 2.6560805059955555, 2.6
051688630139593, -0.4240738943910262, 0.4668798577869107, 0.1486820891519332, 0.39051239
33145161, 1.0905474843114664, -0.09314821501064967, 1.4851127174188385, 2.75790379195874
```

```
86, 1.5360243604004349, 0.07231462467953861, -0.13133194724684696, 1.29419405623785
2, 0.03413089244334132, 1.3451056992194483, 2.274243183633583, -0.24588314395543887, 0.772349715676489, 0.75962180493109, 0.7214380726948927, 0.1995937321335296, -0.41134598364562713, 0.5687031437501034, 0.5941589652409017, 0.9378125553666773, 0.9505404661120763, 0.6068868759863008, 0.4159682148053143, -0.042236572029053274, 2.7706317027041476, 2.1342361654341926, 0.12322626766113501, -0.08042030426525057, 0.16140999989733232, 1.8924058612716097, 1.2560103240016547, 2.185147808415789, 0.6196147867316999, 1.943317504253206, -0.11860403650144787, -0.30952269768243434, 1.9942291472348024, -0.2840668761916362, 2.6306246845047574, 2.286971094378982, -0.19497150097384247, -0.39861807290022805, 0.2886891073513233, 1.7523988430722195, 2.3887943803421745, 2.681536327486354, 1.4596568959280403, 2.439706023323771, 2.7833596134495466, 2.490617666305367, -0.10587612575604877, 1.5614801818912332, 1.9051337720170087, 1.6123918248728295, 1.268738234747054, 1.9560454149986053, 2.6433525952501564, 1.026907930584471}
```

Creating a model

In [31]:

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential([
    Dense(128, activation = 'relu', input_shape = (784,)),
    Dense(128, activation= 'relu'),
    Dense(10, activation = 'softmax')
])
```

In [32]:

```python
model.compile(
    optimizer = 'sgd',
    loss = 'categorical_crossentropy',
    metrics = ['accuracy']
)

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense (Dense)                (None, 128)               100480    
_________________________________________________________________
dense_1 (Dense)              (None, 128)               16512     
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290      
=================================================================
Total params: 118,282
Trainable params: 118,282
Non-trainable params: 0
_________________________________________________________________
```

In [33]:

```python
h = model.fit(
    x_train_norm,
    y_train_encoded,
    epochs = 5
)
```

```
Train on 60000 samples
Epoch 1/5
60000/60000 [==============================] - 5s 82us/sample - loss: 0.3605 - accuracy: 0.8953
Epoch 2/5
60000/60000 [==============================] - 4s 74us/sample - loss: 0.1812 - accuracy: 0.9468
```

```
Epoch 3/5
60000/60000 [==============================] - 5s 76us/sample - loss: 0.1381 - accuracy:
0.9597
Epoch 4/5
60000/60000 [==============================] - 4s 74us/sample - loss: 0.1118 - accuracy:
0.9676
Epoch 5/5
60000/60000 [==============================] - 5s 77us/sample - loss: 0.0947 - accuracy:
0.9725
```

In [34]:

```
#testing on test data
loss, accuracy = model.evaluate(x_test_norm, y_test_encoded)
print('test set accuracy: ', accuracy*100)
```

```
10000/10000 [==============================] - 1s 59us/sample - loss: 0.1017 - accuracy:
0.9699
test set accuracy:  96.99000120162964
```

In [36]:

```
#making predictions
preds = model.predict(x_test_norm)
print('shape of preds: ', preds.shape)
```

```
shape of preds:  (10000, 10)
```

In [40]:

```
#plot

plt.figure(figsize= (12,12))

start_index = 0

for i in range(36):
    plt.subplot(6,6,i+1)
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])
    pred = np.argmax(preds[start_index + i])
    actual = np.argmax(y_test_encoded[start_index + i])
    col = 'g'
    if pred != actual:
        col = 'r'
    plt.xlabel('i={} | pred={} | true={}'.format(start_index +i,pred,actual),color = co
    plt.imshow(x_test[start_index + i],cmap = 'binary')
plt.show()
```

In [46]:

```
count = 0
for i in range(9999):
    pred = np.argmax(preds[i])
    actual = np.argmax(y_test_encoded[i])
    if pred== actual:
        count += 1
print(count)
#INFERENCE
#just to check how many predictions went wrong
#and we can see the accuracy. out of 10000, 9698 are correct
#model has almost 97% accuracy.
```

9698

In [ ]: