In this case study, we have been provided with images of traffic signs and the goal is to train a Deep Network to classify them

The dataset contains 43 different classes of images.

Classes are as listed below:

- (0, b'Speed limit (20km/h)') (1, b'Speed limit (30km/h)')
- (2, b'Speed limit (50km/h)') (3, b'Speed limit (60km/h)')
- (4, b'Speed limit (70km/h)') (5, b'Speed limit (80km/h)')
- (6, b'End of speed limit (80km/h)') (7, b'Speed limit (100km/h)')
- (8, b'Speed limit (120km/h)') (9, b'No passing')
- (10, b'No passing for vehicles over 3.5 metric tons')
- (11, b'Right-of-way at the next intersection') (12, b'Priority road')
- (13, b'Yield') (14, b'Stop') (15, b'No vehicles')
- (16, b'Vehicles over 3.5 metric tons prohibited') (17, b'No entry')
- (18, b'General caution') (19, b'Dangerous curve to the left')
- (20, b'Dangerous curve to the right') (21, b'Double curve')
- (22, b'Bumpy road') (23, b'Slippery road')
- (24, b'Road narrows on the right') (25, b'Road work')
- (26, b'Traffic signals') (27, b'Pedestrians') (28, b'Children crossing')
- (29, b'Bicycles crossing') (30, b'Beware of ice/snow')
- (31, b'Wild animals crossing')
- (32, b'End of all speed and passing limits') (33, b'Turn right ahead')
- (34, b'Turn left ahead') (35, b'Ahead only') (36, b'Go straight or right')
- (37, b'Go straight or left') (38, b'Keep right') (39, b'Keep left')
- (40, b'Roundabout mandatory') (41, b'End of no passing')
- (42, b'End of no passing by vehicles over 3.5 metric tons')

Importing Libraries

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import pickle
```

In [2]:
```
# The pickle module implements binary protocols for serializing and de-
serializing a Python object structure.
with open("./traffic-signs-data/train.p", mode='rb') as training_data:
    train = pickle.load(training_data)
with open("./traffic-signs-data/valid.p", mode='rb') as validation_data
:
    valid = pickle.load(validation_data)
with open("./traffic-signs-data/test.p", mode='rb') as testing_data:
    test = pickle.load(testing_data)
```

In [3]:
```
X_train, y_train = train['features'], train['labels']
X_validation, y_validation = valid['features'], valid['labels']
X_test, y_test = test['features'], test['labels']
```

In [4]:
```
X_train.shape
```

Out[4]: (34799, 32, 32, 3)

In [5]:
```
y_train.shape
```

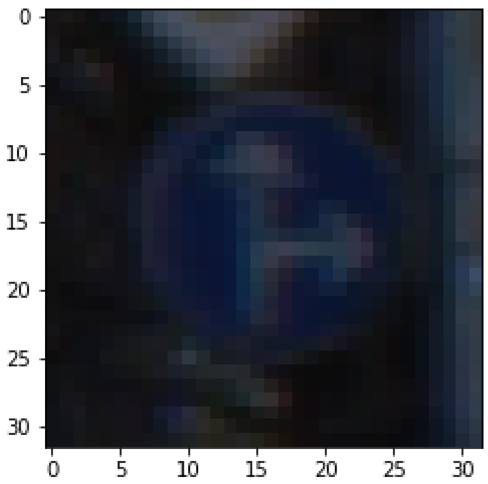
Out[5]: (34799,)

Image exploration

In [6]:
```
i = 1001
plt.imshow(X_train[i]) # Show images are not shuffled
```

```
y_train[i]
```

Out[6]: 36



Data Preparation

In [7]:
```
## Shuffle the dataset for changing the order
from sklearn.utils import shuffle
X_train, y_train = shuffle(X_train, y_train)
```

In [8]:
```
X_train_gray = np.sum(X_train/3, axis=3, keepdims=True)
X_test_gray  = np.sum(X_test/3, axis=3, keepdims=True)
X_validation_gray  = np.sum(X_validation/3, axis=3, keepdims=True)
```

In [9]:
```
X_train_gray_norm = (X_train_gray - 128)/128
X_test_gray_norm = (X_test_gray - 128)/128
X_validation_gray_norm = (X_validation_gray - 128)/128
```

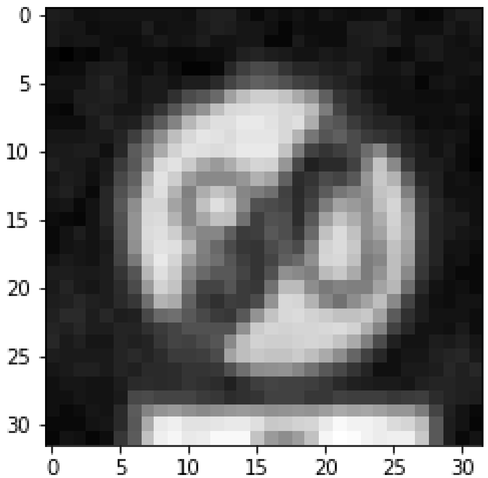
In [10]:
```
X_train_gray.shape
```

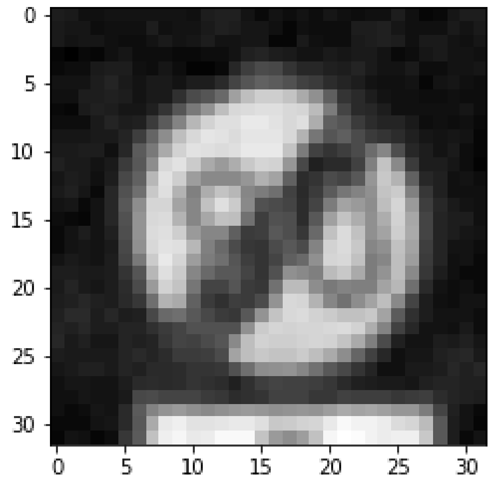
Out[10]: (34799, 32, 32, 1)

Verifying image not broken after normalization

In [26]:
```
i = 610
plt.imshow(X_train_gray[i].squeeze(), cmap='gray')
plt.figure()
plt.imshow(X_train[i])
plt.imshow(X_train_gray_norm[i].squeeze(), cmap='gray')
```

Out[26]: <matplotlib.image.AxesImage at 0x11890a444a8>





Model Training

The model consists of the following layers:

STEP 1: THE FIRST CONVOLUTIONAL LAYER #1

- Input = 32x32x1
- Output = 28x28x6
- Output = (Input-filter+1)/Stride* => (32-5+1)/1=28
- Used a 5x5 Filter with input depth of 3 and output depth of 6
- Apply a RELU Activation function to the output pooling for input, Input = 28x28x6 and Output = 14x14x6
- Stride is the amount by which the kernel is shifted when the kernel is passed over the image.

STEP 2: THE SECOND CONVOLUTIONAL LAYER #2

- Input = 14x14x6
- Output = 10x10x16
- Layer 2: Convolutional layer with Output = 10x10x16

- Output = (Input-filter+1)/strides => 10 = 14-5+1/1

Apply a RELU Activation function to the output Pooling with Input = 10x10x16 and Output = 5x5x16

STEP 3: FLATTENING THE NETWORK

- Flatten the network with Input = 5x5x16 and Output = 400

STEP 4: FULLY CONNECTED LAYER

Layer 3: Fully Connected layer with Input = 400 and Output = 120

- Apply a RELU Activation function to the output

STEP 5: ANOTHER FULLY CONNECTED LAYER

Layer 4: Fully Connected Layer with Input = 120 and Output = 84

- Apply a RELU Activation function to the output STEP 6: FULLY CONNECTED LAYER

Layer 5: Fully Connected layer with Input = 84 and Output = 43

In [13]:
```
# Import train_test_split from scikit library

from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, AveragePooling2D, Dense,
 Flatten, Dropout
from keras.optimizers import Adam
from keras.callbacks import TensorBoard

from sklearn.model_selection import train_test_split
```

In [14]:
```
image_shape = X_train_gray[i].shape
```

In [15]:
```
cnn_model = Sequential()

cnn_model.add(Conv2D(filters=6, kernel_size=(5, 5), activation='relu',
```

```
input_shape=(32,32,1)))
cnn_model.add(AveragePooling2D())

cnn_model.add(Conv2D(filters=16, kernel_size=(5, 5), activation='relu'
))
cnn_model.add(AveragePooling2D())

cnn_model.add(Flatten())

cnn_model.add(Dense(units=120, activation='relu'))

cnn_model.add(Dense(units=84, activation='relu'))

cnn_model.add(Dense(units=43, activation = 'softmax'))
```

In [16]:

```
cnn_model.compile(loss ='sparse_categorical_crossentropy', optimizer=Ad
am(lr=0.001),metrics =['accuracy'])
```

In [17]:

```
history = cnn_model.fit(X_train_gray_norm,
                        y_train,
                        batch_size=500,
                        nb_epoch=50,
                        verbose=1,
                        validation_data = (X_validation_gray_norm,y_val
idation))
```

```
C:\Users\Soumyansh\Anaconda3\envs\tensorflow\lib\site-packages\ipykerne
l_launcher.py:6: UserWarning: The `nb_epoch` argument in `fit` has been
renamed `epochs`.
```

```
Train on 34799 samples, validate on 4410 samples
Epoch 1/50
34799/34799 [==============================] - 33s 954us/step - loss:
3.1640 - acc: 0.1734 - val_loss: 2.7258 - val_acc: 0.2955
Epoch 2/50
34799/34799 [==============================] - 32s 930us/step - loss:
1.7118 - acc: 0.5256 - val_loss: 1.4861 - val_acc: 0.5868
Epoch 3/50
```

```
34799/34799 [==============================] - 32s 926us/step - loss:
0.9745 - acc: 0.7211 - val_loss: 1.0987 - val_acc: 0.6667
Epoch 4/50
34799/34799 [==============================] - 32s 921us/step - loss:
0.6984 - acc: 0.8010 - val_loss: 0.9238 - val_acc: 0.7243
Epoch 5/50
34799/34799 [==============================] - 32s 929us/step - loss:
0.5595 - acc: 0.8448 - val_loss: 0.8392 - val_acc: 0.7483
Epoch 6/50
34799/34799 [==============================] - 32s 929us/step - loss:
0.4553 - acc: 0.8788 - val_loss: 0.7928 - val_acc: 0.7667
Epoch 7/50
34799/34799 [==============================] - 32s 926us/step - loss:
0.3939 - acc: 0.8948 - val_loss: 0.7761 - val_acc: 0.7696
Epoch 8/50
34799/34799 [==============================] - 32s 923us/step - loss:
0.3464 - acc: 0.9076 - val_loss: 0.7101 - val_acc: 0.7909
Epoch 9/50
34799/34799 [==============================] - 32s 909us/step - loss:
0.3089 - acc: 0.9191 - val_loss: 0.6820 - val_acc: 0.8059
Epoch 10/50
34799/34799 [==============================] - 32s 911us/step - loss:
0.2665 - acc: 0.9303 - val_loss: 0.6600 - val_acc: 0.8195
Epoch 11/50
34799/34799 [==============================] - 32s 925us/step - loss:
0.2474 - acc: 0.9349 - val_loss: 0.6983 - val_acc: 0.8161
Epoch 12/50
34799/34799 [==============================] - 32s 928us/step - loss:
0.2198 - acc: 0.9434 - val_loss: 0.7405 - val_acc: 0.8209
Epoch 13/50
34799/34799 [==============================] - 32s 929us/step - loss:
0.2039 - acc: 0.9469 - val_loss: 0.7278 - val_acc: 0.8082
Epoch 14/50
34799/34799 [==============================] - 32s 931us/step - loss:
0.1832 - acc: 0.9532 - val_loss: 0.6932 - val_acc: 0.8224
Epoch 15/50
34799/34799 [==============================] - 32s 934us/step - loss:
0.1649 - acc: 0.9589 - val_loss: 0.7772 - val_acc: 0.8168
Epoch 16/50
34799/34799 [==============================] - 32s 917us/step - loss:
```

```
0.1562 - acc: 0.9605 - val_loss: 0.7591 - val_acc: 0.8125
Epoch 17/50
34799/34799 [==============================] - 32s 931us/step - loss:
0.1476 - acc: 0.9614 - val_loss: 0.6834 - val_acc: 0.8329
Epoch 18/50
34799/34799 [==============================] - 33s 936us/step - loss:
0.1309 - acc: 0.9673 - val_loss: 0.6867 - val_acc: 0.8290
Epoch 19/50
34799/34799 [==============================] - 32s 922us/step - loss:
0.1250 - acc: 0.9686 - val_loss: 0.7991 - val_acc: 0.8256
Epoch 20/50
34799/34799 [==============================] - 32s 923us/step - loss:
0.1176 - acc: 0.9700 - val_loss: 0.7620 - val_acc: 0.8331
Epoch 21/50
34799/34799 [==============================] - 32s 925us/step - loss:
0.1082 - acc: 0.9736 - val_loss: 0.7815 - val_acc: 0.8295
Epoch 22/50
34799/34799 [==============================] - 32s 918us/step - loss:
0.1021 - acc: 0.9749 - val_loss: 0.7680 - val_acc: 0.8347
Epoch 23/50
34799/34799 [==============================] - 32s 912us/step - loss:
0.0946 - acc: 0.9765 - val_loss: 0.7552 - val_acc: 0.8268
Epoch 24/50
34799/34799 [==============================] - 31s 896us/step - loss:
0.0977 - acc: 0.9753 - val_loss: 0.7580 - val_acc: 0.8433
Epoch 25/50
34799/34799 [==============================] - 32s 925us/step - loss:
0.0820 - acc: 0.9798 - val_loss: 0.7699 - val_acc: 0.8322
Epoch 26/50
34799/34799 [==============================] - 32s 922us/step - loss:
0.0767 - acc: 0.9821 - val_loss: 0.7995 - val_acc: 0.8410
Epoch 27/50
34799/34799 [==============================] - 32s 923us/step - loss:
0.0734 - acc: 0.9826 - val_loss: 0.7458 - val_acc: 0.8420
Epoch 28/50
34799/34799 [==============================] - 32s 914us/step - loss:
0.0679 - acc: 0.9841 - val_loss: 0.7693 - val_acc: 0.8376
Epoch 29/50
34799/34799 [==============================] - 31s 901us/step - loss:
```

```
0.0655 - acc: 0.9852 - val_loss: 0.7866 - val_acc: 0.8438
Epoch 30/50
34799/34799 [==============================] - 32s 906us/step - loss:
0.0616 - acc: 0.9859 - val_loss: 0.8202 - val_acc: 0.8395
Epoch 31/50
34799/34799 [==============================] - 31s 892us/step - loss:
0.0578 - acc: 0.9862 - val_loss: 0.7709 - val_acc: 0.8383
Epoch 32/50
34799/34799 [==============================] - 32s 919us/step - loss:
0.0579 - acc: 0.9856 - val_loss: 0.7758 - val_acc: 0.8458
Epoch 33/50
34799/34799 [==============================] - 32s 918us/step - loss:
0.0507 - acc: 0.9890 - val_loss: 0.7282 - val_acc: 0.8546
Epoch 34/50
34799/34799 [==============================] - 32s 909us/step - loss:
0.0545 - acc: 0.9867 - val_loss: 0.8062 - val_acc: 0.8537
Epoch 35/50
34799/34799 [==============================] - 32s 928us/step - loss:
0.0516 - acc: 0.9875 - val_loss: 0.7730 - val_acc: 0.8517
Epoch 36/50
34799/34799 [==============================] - 32s 926us/step - loss:
0.0530 - acc: 0.9871 - val_loss: 0.7893 - val_acc: 0.8370
Epoch 37/50
34799/34799 [==============================] - 32s 925us/step - loss:
0.0529 - acc: 0.9864 - val_loss: 0.8187 - val_acc: 0.8422
Epoch 38/50
34799/34799 [==============================] - 32s 923us/step - loss:
0.0468 - acc: 0.9886 - val_loss: 0.7619 - val_acc: 0.8506
Epoch 39/50
34799/34799 [==============================] - 32s 912us/step - loss:
0.0390 - acc: 0.9913 - val_loss: 0.7641 - val_acc: 0.8512
Epoch 40/50
34799/34799 [==============================] - 32s 928us/step - loss:
0.0401 - acc: 0.9910 - val_loss: 0.7775 - val_acc: 0.8501
Epoch 41/50
34799/34799 [==============================] - 32s 912us/step - loss:
0.0351 - acc: 0.9927 - val_loss: 0.8170 - val_acc: 0.8517
Epoch 42/50
34799/34799 [==============================] - 32s 925us/step - loss:
```

```
0.0353 - acc: 0.9926 - val_loss: 0.7606 - val_acc: 0.8610
Epoch 43/50
34799/34799 [==============================] - 32s 927us/step - loss:
0.0380 - acc: 0.9907 - val_loss: 0.7652 - val_acc: 0.8560
Epoch 44/50
34799/34799 [==============================] - 32s 911us/step - loss:
0.0366 - acc: 0.9918 - val_loss: 0.8865 - val_acc: 0.8408
Epoch 45/50
34799/34799 [==============================] - 32s 925us/step - loss:
0.0349 - acc: 0.9923 - val_loss: 0.8071 - val_acc: 0.8524
Epoch 46/50
34799/34799 [==============================] - 32s 923us/step - loss:
0.0303 - acc: 0.9933 - val_loss: 0.7749 - val_acc: 0.8639
Epoch 47/50
34799/34799 [==============================] - 32s 924us/step - loss:
0.0308 - acc: 0.9930 - val_loss: 0.7972 - val_acc: 0.8540
Epoch 48/50
34799/34799 [==============================] - 32s 920us/step - loss:
0.0345 - acc: 0.9920 - val_loss: 0.7618 - val_acc: 0.8537
Epoch 49/50
34799/34799 [==============================] - 32s 915us/step - loss:
0.0362 - acc: 0.9911 - val_loss: 0.7296 - val_acc: 0.8662
Epoch 50/50
34799/34799 [==============================] - 32s 916us/step - loss:
0.0340 - acc: 0.9916 - val_loss: 0.8333 - val_acc: 0.8469
```

In [19]:

```
score = cnn_model.evaluate(X_test_gray_norm, y_test,verbose=0)
print('Test Accuracy : {:.4f}'.format(score[1]))
```

```
Test Accuracy : 0.8411
```

In [20]:

```
history.history.keys()
```

Out[20]:

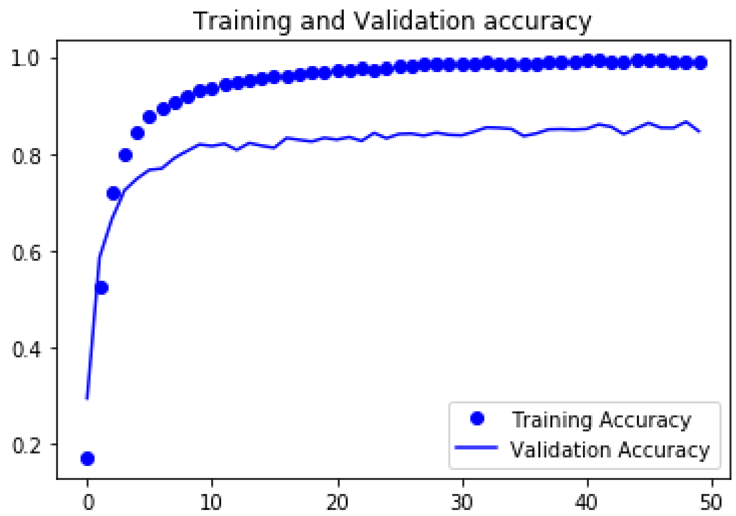
```
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```

In [21]:

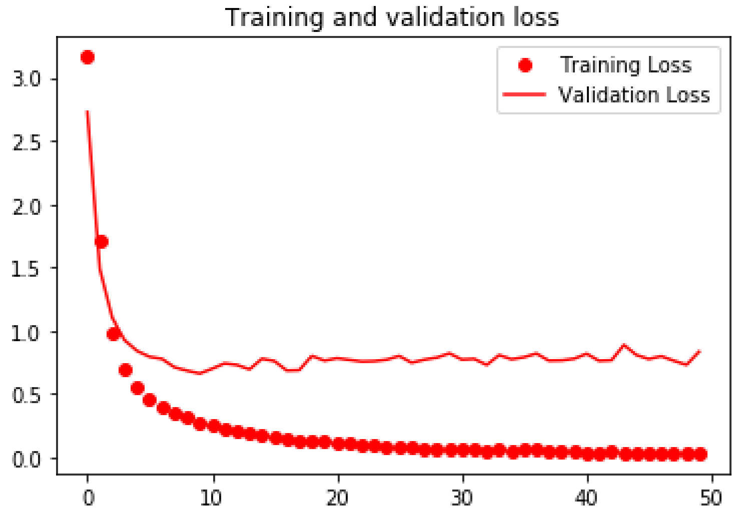
```
accuracy = history.history['acc']
val_accuracy = history.history['val_acc']
```

```python
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(accuracy))

plt.plot(epochs, accuracy, 'bo', label='Training Accuracy')
plt.plot(epochs, val_accuracy, 'b', label='Validation Accuracy')
plt.title('Training and Validation accuracy')
plt.legend()
```

Out[21]: <matplotlib.legend.Legend at 0x1188c76f400>



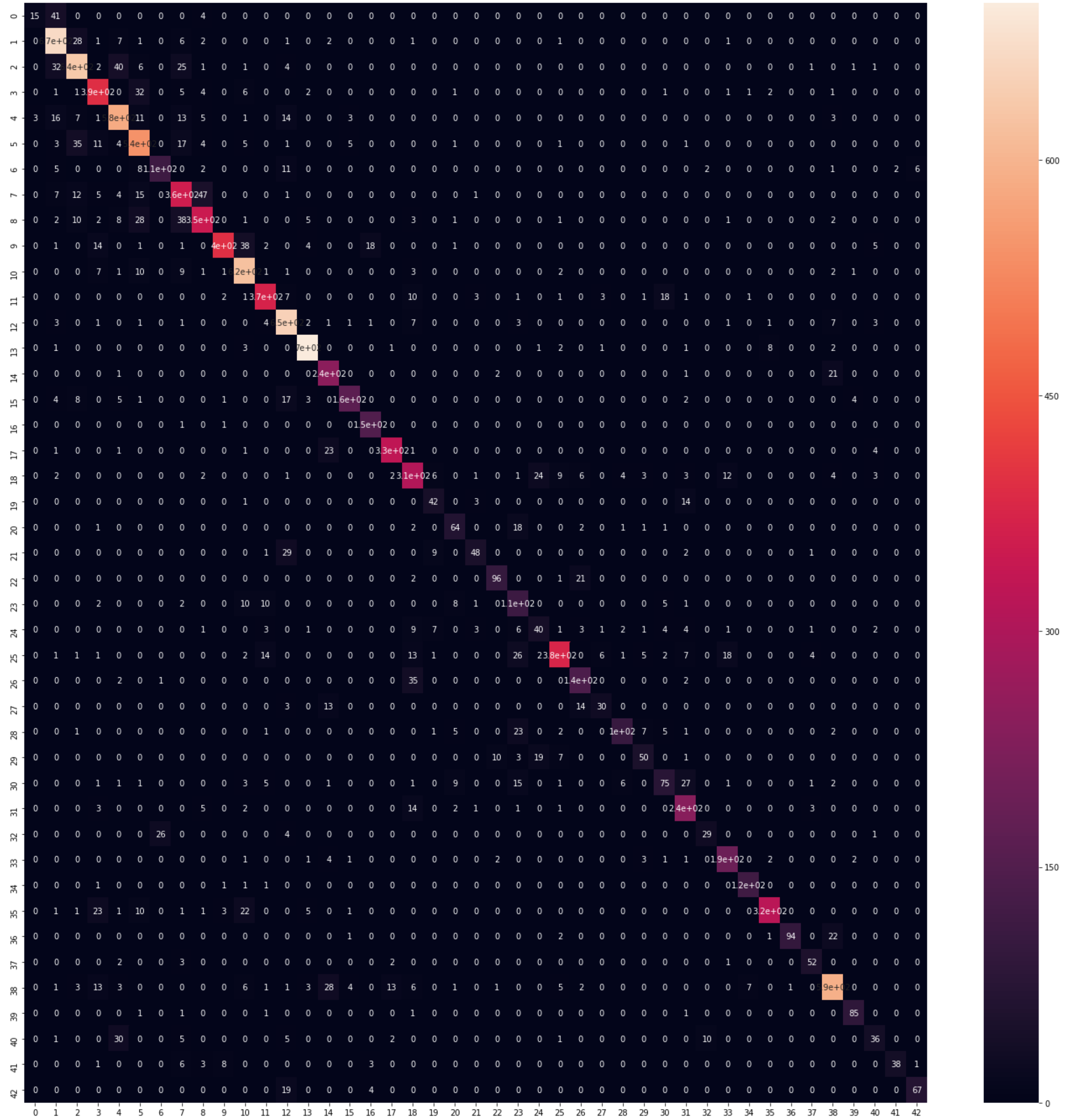
In [22]:
```python
plt.plot(epochs, loss, 'ro', label='Training Loss')
plt.plot(epochs, val_loss, 'r', label='Validation Loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```



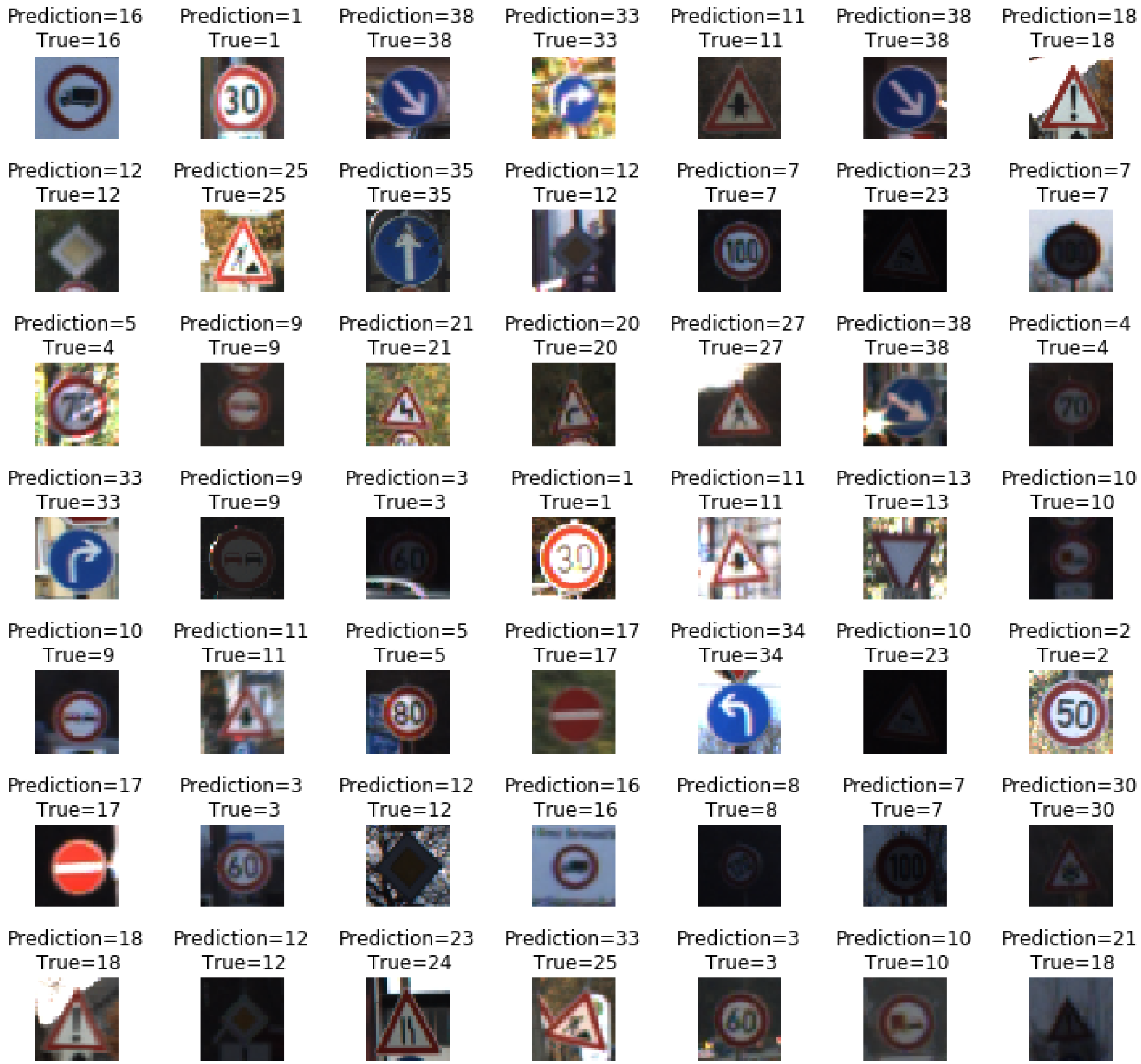
In [23]:
```
#get the predictions for the test data
predicted_classes = cnn_model.predict_classes(X_test_gray_norm)
#get the indices to be plotted
y_true = y_test
```

In [24]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_true, predicted_classes)
plt.figure(figsize = (25,25))
sns.heatmap(cm, annot=True)
```

Out[24]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1188c7b0780>
```


600
450
300
150
0

In [25]:
```
L = 7
W = 7
fig, axes = plt.subplots(L, W, figsize = (12,12))
axes = axes.ravel() #

for i in np.arange(0, L * W):
    axes[i].imshow(X_test[i])
    axes[i].set_title("Prediction={}\n True={}".format(predicted_classe
s[i], y_true[i]))
    axes[i].axis('off')

plt.subplots_adjust(wspace=1)
```



Prediction=16 True=16
Prediction=1 True=1
Prediction=38 True=38
Prediction=33 True=33
Prediction=11 True=11
Prediction=38 True=38
Prediction=18 True=18
Prediction=12 True=12
Prediction=25 True=25
Prediction=35 True=35
Prediction=12 True=12
Prediction=7 True=7
Prediction=23 True=23
Prediction=7 True=7
Prediction=5 True=4
Prediction=9 True=9
Prediction=21 True=21
Prediction=20 True=20
Prediction=27 True=27
Prediction=38 True=38
Prediction=4 True=4
Prediction=33 True=33
Prediction=9 True=9
Prediction=3 True=3
Prediction=1 True=1
Prediction=11 True=11
Prediction=13 True=13
Prediction=10 True=10
Prediction=10 True=9
Prediction=11 True=11
Prediction=5 True=5
Prediction=17 True=17
Prediction=34 True=34
Prediction=10 True=23
Prediction=2 True=2
Prediction=17 True=17
Prediction=3 True=3
Prediction=12 True=12
Prediction=16 True=16
Prediction=8 True=8
Prediction=7 True=7
Prediction=30 True=30
Prediction=18 True=18
Prediction=12 True=12
Prediction=23 True=24
Prediction=33 True=25
Prediction=3 True=3
Prediction=10 True=10
Prediction=21 True=18

In [ ]: