In [7]:
```
# Traffic Sign Detection Using CNN
```

In [8]:
```
#initializations
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
import cv2
from PIL import Image
import os
from tensorflow import keras
from keras.datasets import mnist
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten
from tensorflow.keras.optimizers import SGD
from tensorflow.keras import backend as K
```

```
Using TensorFlow backend.
```

In [9]:
```
## Reading images and putting in numpy
data = []
labels = []

height = 30
width = 30
channels = 3
classes = 43
n_inputs = height * width * channels

for i in range(classes) :
    path = "C:/Users/HP/Desktop/TrafficSign/Train/{0}/".format(i)
    print(path)
    Class=os.listdir(path)
    for a in Class:
        try:
            image=cv2.imread(path+a)
            image_from_array = Image.fromarray(image, 'RGB')
            size_image = image_from_array.resize((height, width))
            data.append(np.array(size_image))
            labels.append(i)
        except AttributeError:
            print(" ")

Cells=np.array(data)
labels=np.array(labels)
#Randomize the order of the input images because accuracy was too high and valida
s=np.arange(Cells.shape[0])
np.random.seed(43)
np.random.shuffle(s)

Cells=Cells[s]
labels=labels[s]
```

```
C:/Users/HP/Desktop/TrafficSign/Train/0/
C:/Users/HP/Desktop/TrafficSign/Train/1/
C:/Users/HP/Desktop/TrafficSign/Train/2/
C:/Users/HP/Desktop/TrafficSign/Train/3/
C:/Users/HP/Desktop/TrafficSign/Train/4/
C:/Users/HP/Desktop/TrafficSign/Train/5/
C:/Users/HP/Desktop/TrafficSign/Train/6/
C:/Users/HP/Desktop/TrafficSign/Train/7/
C:/Users/HP/Desktop/TrafficSign/Train/8/
C:/Users/HP/Desktop/TrafficSign/Train/9/
C:/Users/HP/Desktop/TrafficSign/Train/10/
C:/Users/HP/Desktop/TrafficSign/Train/11/
C:/Users/HP/Desktop/TrafficSign/Train/12/
C:/Users/HP/Desktop/TrafficSign/Train/13/
C:/Users/HP/Desktop/TrafficSign/Train/14/
C:/Users/HP/Desktop/TrafficSign/Train/15/
C:/Users/HP/Desktop/TrafficSign/Train/16/
C:/Users/HP/Desktop/TrafficSign/Train/17/
C:/Users/HP/Desktop/TrafficSign/Train/18/
C:/Users/HP/Desktop/TrafficSign/Train/19/
C:/Users/HP/Desktop/TrafficSign/Train/20/
C:/Users/HP/Desktop/TrafficSign/Train/21/
```

```
C:/Users/HP/Desktop/TrafficSign/Train/22/
C:/Users/HP/Desktop/TrafficSign/Train/23/
C:/Users/HP/Desktop/TrafficSign/Train/24/
C:/Users/HP/Desktop/TrafficSign/Train/25/
C:/Users/HP/Desktop/TrafficSign/Train/26/
C:/Users/HP/Desktop/TrafficSign/Train/27/
C:/Users/HP/Desktop/TrafficSign/Train/28/
C:/Users/HP/Desktop/TrafficSign/Train/29/
C:/Users/HP/Desktop/TrafficSign/Train/30/
C:/Users/HP/Desktop/TrafficSign/Train/31/
C:/Users/HP/Desktop/TrafficSign/Train/32/
C:/Users/HP/Desktop/TrafficSign/Train/33/
C:/Users/HP/Desktop/TrafficSign/Train/34/
C:/Users/HP/Desktop/TrafficSign/Train/35/
C:/Users/HP/Desktop/TrafficSign/Train/36/
C:/Users/HP/Desktop/TrafficSign/Train/37/
C:/Users/HP/Desktop/TrafficSign/Train/38/
C:/Users/HP/Desktop/TrafficSign/Train/39/
C:/Users/HP/Desktop/TrafficSign/Train/40/
C:/Users/HP/Desktop/TrafficSign/Train/41/
C:/Users/HP/Desktop/TrafficSign/Train/42/
```

In [10]:

```
## Train Test Split 80-20

(x_train,x_test) = Cells[(int)(0.2*len(labels)):],Cells[:(int)(0.2*len(labels))]
(y_train,y_test) = labels[(int)(0.2*len(labels)):],labels[:(int)(0.2*len(labels))
```

In [11]:

```
## Train-test Shapes
print(x_train.shape)  #train images
print(y_train.shape)  #train labels
print(x_test.shape)  #test images
print(y_test.shape)  #test labels
```

```
(31368, 30, 30, 3)
(31368,)
(7841, 30, 30, 3)
(7841,)
```

In [12]:

```
## Setting input image shape

#train[0] gives 1st dimension and train[1] gives second dimension
img_rows = x_train[0].shape[0]
img_cols = x_train[1].shape[0]

#storing shape of a single image
input_shape = (img_rows, img_cols, 3)
print(input_shape)
```

```
(30, 30, 3)
```

In [13]:
```
## Preprocessing of image data

#keras want float32 data type
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
#normalization of image data ... as images range from 0 to 255 and normalize them
x_train = x_train / 255
x_test = x_test / 255
```

In [14]:
```
#hot one encoding

from keras.utils import np_utils

y_train = np_utils.to_categorical(y_train,classes)
#print(y_train)
y_test = np_utils.to_categorical(y_test,classes)
#print(y_test)
```

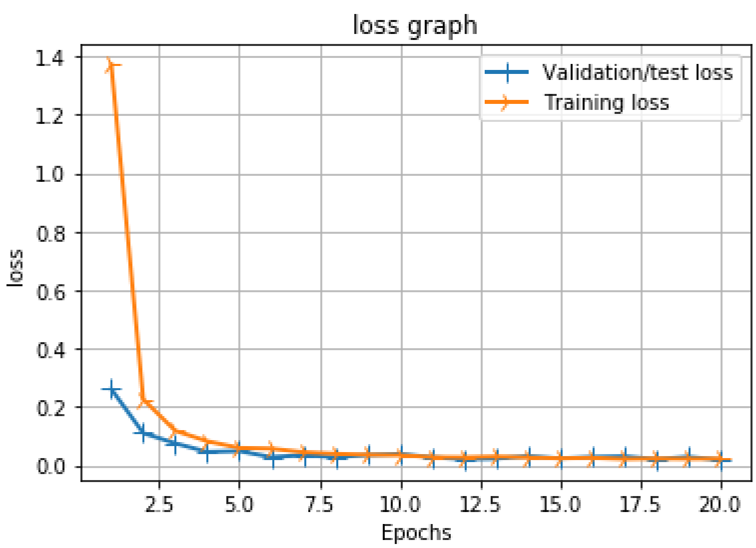
In [15]:
```
#model creation
model = Sequential()

#conv layer1
model.add(Conv2D(32, kernel_size=(3,3), activation='relu',input_shape = input_sha
model.add(MaxPooling2D(pool_size = (2,2)))


#conv layer2
model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Dropout(0.25))

#conv layer3
model.add(Conv2D(128, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size = (2,2)))


#flatten
model.add(Flatten())

#dense layer 1
model.add(Dense(256,activation = 'relu'))
model.add(Dropout(0.25))

#dense layer 2
model.add(Dense(classes, activation = 'softmax'))
#compiling model
model.compile(loss = 'categorical_crossentropy',
              optimizer = 'adam',
              metrics = ['accuracy'])
print(model.summary())

#training begins here
#verbose is how much information we wanna see in training
batch_size = 32
epochs = 20
#batch size 16 for large images and 32 for small
history = model.fit(x_train, y_train,
                    batch_size = batch_size,
                    epochs = epochs,
                    verbose = 1,
                    validation_data = (x_test,y_test))

#evaluate model
score = model.evaluate(x_test, y_test, verbose = 0)
print('Test loss',score[0])
print('Test accuracy',score[1])
```

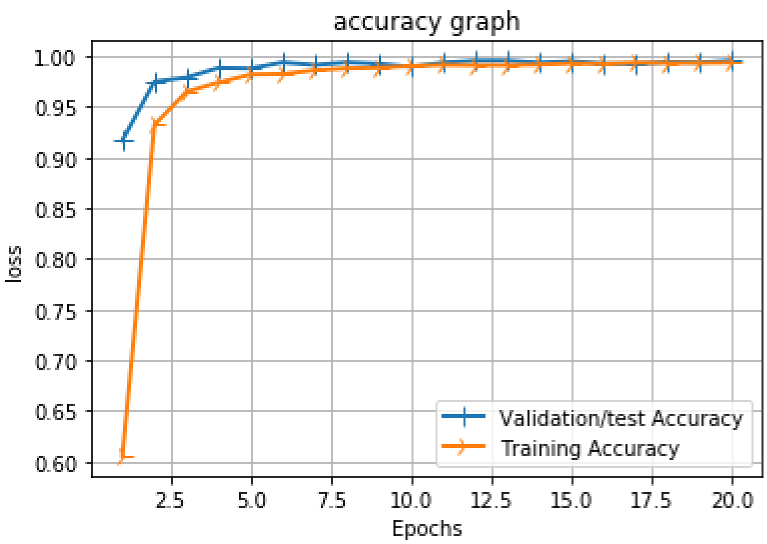
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 28, 28, 32)        896       
_________________________________________________________________
```

```
max_pooling2d (MaxPooling2D) (None, 14, 14, 32)       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 12, 12, 64)       18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 64)         0
_________________________________________________________________
dropout (Dropout)            (None, 6, 6, 64)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 4, 4, 128)        73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 2, 2, 128)        0
_________________________________________________________________
flatten (Flatten)            (None, 512)              0
_________________________________________________________________
dense (Dense)                (None, 256)              131328
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)              0
_________________________________________________________________
dense_1 (Dense)              (None, 43)               11051
=================================================================
Total params: 235,627
Trainable params: 235,627
Non-trainable params: 0
_________________________________________________________________
None
Train on 31368 samples, validate on 7841 samples
Epoch 1/20
31368/31368 [==============================] - 31s 974us/sample - loss: 1.3730
- accuracy: 0.6057 - val_loss: 0.2610 - val_accuracy: 0.9168
Epoch 2/20
31368/31368 [==============================] - 30s 945us/sample - loss: 0.2257
- accuracy: 0.9322 - val_loss: 0.1099 - val_accuracy: 0.9749
Epoch 3/20
31368/31368 [==============================] - 30s 954us/sample - loss: 0.1179
- accuracy: 0.9649 - val_loss: 0.0748 - val_accuracy: 0.9790
Epoch 4/20
31368/31368 [==============================] - 30s 959us/sample - loss: 0.0820
- accuracy: 0.9741 - val_loss: 0.0454 - val_accuracy: 0.9884
Epoch 5/20
31368/31368 [==============================] - 32s 1ms/sample - loss: 0.0590 -
accuracy: 0.9820 - val_loss: 0.0482 - val_accuracy: 0.9879
Epoch 6/20
31368/31368 [==============================] - 35s 1ms/sample - loss: 0.0567 -
accuracy: 0.9824 - val_loss: 0.0275 - val_accuracy: 0.9939
Epoch 7/20
31368/31368 [==============================] - 36s 1ms/sample - loss: 0.0443 -
accuracy: 0.9861 - val_loss: 0.0341 - val_accuracy: 0.9913
Epoch 8/20
31368/31368 [==============================] - 37s 1ms/sample - loss: 0.0388 -
accuracy: 0.9880 - val_loss: 0.0273 - val_accuracy: 0.9939
Epoch 9/20
31368/31368 [==============================] - 36s 1ms/sample - loss: 0.0355 -
accuracy: 0.9886 - val_loss: 0.0347 - val_accuracy: 0.9925
Epoch 10/20
31368/31368 [==============================] - 36s 1ms/sample - loss: 0.0343 -
accuracy: 0.9904 - val_loss: 0.0382 - val_accuracy: 0.9895
Epoch 11/20
```

```
31368/31368 [==============================] - 35s 1ms/sample - loss: 0.0273 -
accuracy: 0.9914 - val_loss: 0.0268 - val_accuracy: 0.9936
Epoch 12/20
31368/31368 [==============================] - 34s 1ms/sample - loss: 0.0267 -
accuracy: 0.9911 - val_loss: 0.0203 - val_accuracy: 0.9955
Epoch 13/20
31368/31368 [==============================] - 34s 1ms/sample - loss: 0.0295 -
accuracy: 0.9911 - val_loss: 0.0223 - val_accuracy: 0.9957
Epoch 14/20
31368/31368 [==============================] - 34s 1ms/sample - loss: 0.0261 -
accuracy: 0.9917 - val_loss: 0.0308 - val_accuracy: 0.9935
Epoch 15/20
31368/31368 [==============================] - 35s 1ms/sample - loss: 0.0233 -
accuracy: 0.9930 - val_loss: 0.0230 - val_accuracy: 0.9950
Epoch 16/20
31368/31368 [==============================] - 36s 1ms/sample - loss: 0.0250 -
accuracy: 0.9924 - val_loss: 0.0278 - val_accuracy: 0.9926
Epoch 17/20
31368/31368 [==============================] - 37s 1ms/sample - loss: 0.0211 -
accuracy: 0.9936 - val_loss: 0.0302 - val_accuracy: 0.9925
Epoch 18/20
31368/31368 [==============================] - 37s 1ms/sample - loss: 0.0227 -
accuracy: 0.9932 - val_loss: 0.0203 - val_accuracy: 0.9941
Epoch 19/20
31368/31368 [==============================] - 38s 1ms/sample - loss: 0.0215 -
accuracy: 0.9932 - val_loss: 0.0283 - val_accuracy: 0.9938
Epoch 20/20
31368/31368 [==============================] - 35s 1ms/sample - loss: 0.0224 -
accuracy: 0.9936 - val_loss: 0.0179 - val_accuracy: 0.9957
Test loss 0.017889902631156914
Test accuracy 0.9956638
```

In [16]:
```
#plot the loss charts
history_dict = history.history

loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
epochs = range(1, len(loss_values)+1)

line1 = plt.plot(epochs,val_loss_values, label = 'Validation/test loss')
line2 = plt.plot(epochs,loss_values, label = 'Training loss')
plt.setp(line1, linewidth = 2.0, marker = '+', markersize = 10.0 )
plt.setp(line2, linewidth = 2.0, marker = '4', markersize = 10.0 )
plt.title('loss graph')
plt.xlabel('Epochs')
plt.ylabel('loss')
plt.grid(True)
plt.legend()
plt.show()

acc_values = history_dict['accuracy']
val_acc_values = history_dict['val_accuracy']

line1 = plt.plot(epochs,val_acc_values, label = 'Validation/test Accuracy')
line2 = plt.plot(epochs,acc_values, label = 'Training Accuracy')
plt.setp(line1, linewidth = 2.0, marker = '+', markersize = 10.0 )
plt.setp(line2, linewidth = 2.0, marker = '4', markersize = 10.0 )
plt.title('accuracy graph')
plt.xlabel('Epochs')
plt.ylabel('loss')
plt.grid(True)
plt.legend()
plt.show()
```





In [17]:
```
#save the model
from keras.models import model_from_json
model.save("C:/Users/HP/Desktop/TrafficSign/cnn_signal.h5")
print('model saved')
```

```
model saved
```

In [18]:
```
#load model
from tensorflow.keras.models import load_model
classifier = load_model("C:/Users/HP/Desktop/TrafficSign/cnn_signal.h5")
```

In [19]:
```
#Saving the history File of our model
import pickle

pickle_out = open("trafficSign_pickle","wb")
pickle.dump(history.history, pickle_out)
pickle_out.close()
```

In [20]:
```
#To load the save history of model

pickle_in = open("trafficSign_pickle","rb")
saved_history = pickle.load(pickle_in)
print(saved_history)
```

```
{'loss': [1.3729721551204754, 0.22566022191674573, 0.11794098528390391, 0.08201
463327231728, 0.0589599325390496, 0.056652773496607814, 0.044309441444429096,
0.03876699338290267, 0.03550584499654482, 0.03428600539905851, 0.02733587903378
9118, 0.026672158697717382, 0.029452142662001104, 0.026137571665771287, 0.02333
7168610882073, 0.024977875898564407, 0.021051364324426232, 0.02266189936173432,
0.021489173710362245, 0.022433864524114167], 'accuracy': [0.60568094, 0.932223
9, 0.96490055, 0.9740819, 0.9819561, 0.98237056, 0.9861005, 0.98801327, 0.98861
897, 0.99040425, 0.9913606, 0.9910737, 0.99113744, 0.9917432, 0.9929546, 0.9924
445, 0.9935603, 0.9931778, 0.99320966, 0.9935922], 'val_loss': [0.2610037137431
677, 0.10991166436332545, 0.07476574632861425, 0.04535789232614401, 0.048167949
88281464, 0.027491415433260915, 0.03408292307128107, 0.02727517366098761, 0.034
71781250152566, 0.03822880742013202, 0.026828537457087392, 0.02030718326056428
5, 0.02225077899036963, 0.030787936957357487, 0.023046976764631088, 0.027837193
084593752, 0.030226810949427106, 0.020292335452397316, 0.02829721574097263, 0.0
17889902631156914], 'val_accuracy': [0.91684735, 0.9748756, 0.97895676, 0.98839
43, 0.9878842, 0.9938783, 0.99132764, 0.9938783, 0.99247545, 0.9895421, 0.99362
326, 0.99553627, 0.9956638, 0.9934957, 0.9950262, 0.992603, 0.99247545, 0.99413
34, 0.9937508, 0.9956638]}
```

In [26]:
```
import sklearn
import scipy
from sklearn.metrics import classification_report,confusion_matrix
import numpy as np

y_pred = classifier.predict_classes(x_test)

#classification report
print(classification_report(np.argmax(y_test,axis=1), y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        39
           1       1.00      0.99      0.99       451
           2       0.99      0.99      0.99       481
           3       0.97      1.00      0.98       274
           4       1.00      1.00      1.00       411
           5       0.99      0.97      0.98       346
           6       0.98      1.00      0.99        91
           7       1.00      0.99      1.00       318
           8       1.00      1.00      1.00       280
           9       1.00      1.00      1.00       291
          10       1.00      1.00      1.00       379
          11       1.00      1.00      1.00       249
          12       1.00      1.00      1.00       366
          13       1.00      1.00      1.00       449
          14       1.00      0.99      1.00       145
          15       0.99      1.00      1.00       146
          16       1.00      1.00      1.00        95
          17       1.00      1.00      1.00       208
          18       0.98      1.00      0.99       249
          19       0.98      1.00      0.99        42
          20       1.00      0.99      0.99        73
          21       1.00      0.98      0.99        63
          22       1.00      1.00      1.00        90
          23       1.00      1.00      1.00        93
          24       1.00      0.98      0.99        64
          25       1.00      1.00      1.00       304
          26       1.00      0.97      0.99       117
          27       1.00      0.96      0.98        57
          28       1.00      1.00      1.00        95
          29       1.00      1.00      1.00        57
          30       1.00      0.99      0.99        98
          31       0.99      1.00      1.00       168
          32       1.00      1.00      1.00        55
          33       1.00      1.00      1.00       143
          34       1.00      1.00      1.00        78
          35       1.00      1.00      1.00       241
          36       1.00      1.00      1.00        68
          37       1.00      1.00      1.00        37
          38       1.00      1.00      1.00       411
          39       1.00      1.00      1.00        53
          40       0.99      1.00      0.99        71
          41       1.00      1.00      1.00        45
          42       1.00      1.00      1.00        50
```

```
    accuracy                           1.00      7841
   macro avg       1.00      1.00      1.00      7841
weighted avg       1.00      1.00      1.00      7841
```

In [27]:

```
#Confusion Matrix
print(confusion_matrix(np.argmax(y_test,axis=1),y_pred))
```

```
[[ 39   0   0 ...   0   0   0]
 [  0 448   1 ...   0   0   0]
 [  0   2 476 ...   0   0   0]
 ...
 [  0   0   0 ...  71   0   0]
 [  0   0   0 ...   0  45   0]
 [  0   0   0 ...   0   0  50]]
```

In [ ]: