Self-Driving Car Engineer Nanodegree

Deep Learning

Project: Build a Traffic Sign Recognition Classifier ¶

In this notebook, a template is provided for you to implement your functionality in stages, which is required to successfully complete this project. If additional code is required that cannot be included in the notebook, be sure that the Python code is successfully imported and included in your submission if necessary.

> **Note**: Once you have completed all of the code implementations, you need to finalize your work by exporting the iPython Notebook as an HTML document. Before exporting the notebook to html, all of the code cells need to have been run so that reviewers can see the final implementation and output. You can then export the notebook by using the menu above and navigating to \n", "**File -> Download as -> HTML (.html)**. Include the finished document along with this notebook as your submission.

In addition to implementing code, there is a writeup to complete. The writeup should be completed in a separate file, which can be either a markdown file or a pdf document. There is a write up template (https://github.com/udacity/CarND-Traffic-Sign-Classifier-Project/blob/master/writeup_template.md) that can be used to guide the writing process. Completing the code template and writeup template will cover all of the rubric points (https://review.udacity.com/#!/rubrics/481/view) for this project.

The rubric (https://review.udacity.com/#!/rubrics/481/view) contains "Stand Out Suggestions" for enhancing the project beyond the minimum requirements. The stand out suggestions are optional. If you decide to pursue the "stand out suggestions", you can include the code in this Ipython notebook and also discuss the results in the writeup file.

> **Note:** Code and Markdown cells can be executed using the **Shift + Enter** keyboard shortcut. In addition, Markdown cells can be edited by typically double-clicking the cell to enter edit mode.

Step 0: Load The Data

Step 1: Dataset Summary & Exploration

The pickled data is a dictionary with 4 key/value pairs:

- `'features'` is a 4D array containing raw pixel data of the traffic sign images, (num examples, width, height, channels).
- `'labels'` is a 1D array containing the label/class id of the traffic sign. The file `signnames.csv` contains id -> name mappings for each id.
- `'sizes'` is a list containing tuples, (width, height) representing the original width and height the image.
- `'coords'` is a list containing tuples, (x1, y1, x2, y2) representing coordinates of a bounding box around the sign in the image. **THESE COORDINATES ASSUME THE ORIGINAL IMAGE. THE PICKLED DATA CONTAINS RESIZED VERSIONS (32 by 32) OF THESE IMAGES**

Complete the basic data summary below. Use python, numpy and/or pandas methods to calculate the data summary rather than hard coding the results. For example, the pandas shape method (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.shape.html) might be useful for calculating some of the summary results.

In [1]:

```
# coding: UTF-8

########## Function (1) :Load 3 data of training, validation, testing##################
import os
print(os.sys.path)

# Load pickled data

import platform
print('python_version',platform.python_version())

# Load pickled data
import pickle


# TODO: Fill this in based on where you saved the training and testing data

training_file = './train.p'
validation_file='./valid.p'
testing_file =  './test.p'

with open(training_file, mode='rb') as f:
    train = pickle.load(f)
with open(validation_file, mode='rb') as f:
    valid = pickle.load(f)
with open(testing_file, mode='rb') as f:
    test = pickle.load(f)

X_train, y_train = train['features'], train['labels']
X_valid, y_valid = valid['features'], valid['labels']
X_test, y_test = test['features'], test['labels']

print('X_train',X_train.shape,type(X_train))
print('y_train',y_train.shape,type(y_train))
print('X_valid',X_valid.shape,type(X_valid))

print('y_valid',y_valid.shape,type(y_valid))
print('X_test',X_test.shape,type(X_test))
print('y_test',y_test.shape,type(y_test))
```

```
['', '/home/uda/miniconda3/lib/python3.6/site-packages', '/home/uda/uda/CarND-Traf
fic-Sign-Classifier-Project', '/home/uda/.conda/envs/IntroToTensorFlow/lib/python3
6.zip', '/home/uda/.conda/envs/IntroToTensorFlow/lib/python3.6', '/home/uda/.cond
a/envs/IntroToTensorFlow/lib/python3.6/lib-dynload', '/home/uda/.conda/envs/IntroT
oTensorFlow/lib/python3.6/site-packages', '/home/uda/.conda/envs/IntroToTensorFlo
w/lib/python3.6/site-packages/IPython/extensions', '/home/uda/.ipython']
python_version 3.6.3
X_train (34799, 32, 32, 3) <class 'numpy.ndarray'>
y_train (34799,) <class 'numpy.ndarray'>
X_valid (4410, 32, 32, 3) <class 'numpy.ndarray'>
y_valid (4410,) <class 'numpy.ndarray'>
X_test (12630, 32, 32, 3) <class 'numpy.ndarray'>
y_test (12630,) <class 'numpy.ndarray'>
```

In [2]:

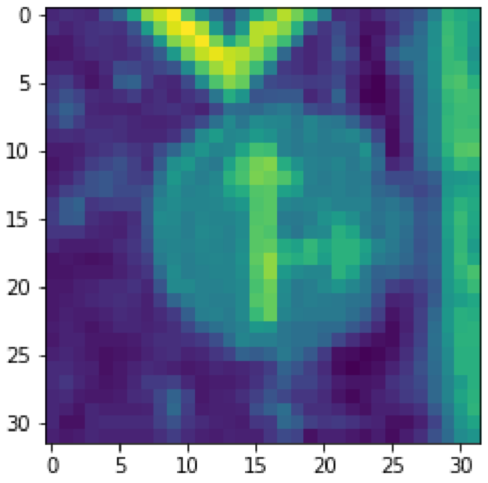
```
########## Function (2): Load signnames data ##################
import numpy as np
import csv

xlist=[]
with open('signnames.csv', 'r') as f:
    reader = csv.reader(f) # リストで読まれる
    header = next(reader)  # ヘッダーを読み飛ばしたい時

    for row in reader:
        xlist.append(row)
        #print(row[0],row[1])
        #print(row)

print(xlist[y_train[1000]][1],'&',xlist[y_valid[1000]][1])
```

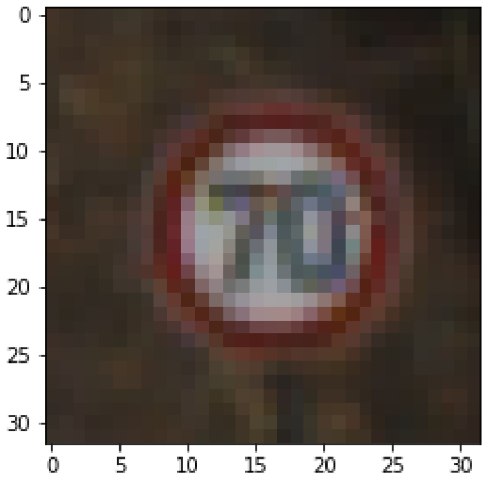
```
Go straight or right & Speed limit (70km/h)
```

In [3]:

```
##########  Check My Code  for Debug (Optional)#################
import pickle
import matplotlib.pyplot as plt

import sys
##sys.path.append('/usr/local/lib/python2.7/site-packages')
##sys.path.append('/home/kshiba/conda/lib/python3.6/site-packages')
import cv2
print('cv2.version',cv2.__version__)

#print(len(train))#print(train)
#print(len(valid))#print(valid)
#print(len(test))#print(test)
#print(len(train['features']) )

#print(X_train[:,:,:,:])
#X_train[0,:,:,:].shape[2]

print('X_train[0].shape',X_train[0].shape)
print('X_train.shape' ,X_train.shape )
print(' len(X_train.shape)', len(X_train.shape))

print('################')
for i in range(len(X_train.shape) ):
    print('X_train.shape[',i,']',X_train.shape[i] )

#print(X_train[0,:,:,0])
plt.imshow(X_train[1000,:,:,2])
plt.show()

plt.imshow(X_valid[1000,:,:,:])
plt.show()



print(X_train.shape[1:4])
if  X_train.shape[1:3]==(32,32):
     print('tapple')
else:
     print('other')
print('################')
print(train.keys())
for i in range(len(train.keys())):
    print(sorted(train.keys())[i])

print(valid.keys())
print(test.keys())
print('################')


print(X_train.shape,X_valid.shape,X_test.shape)
print(y_train.shape,y_valid.shape,y_test.shape)
print(y_train[1000],y_valid[1000],)


print(xlist[y_train[1000]][1],'&',xlist[y_valid[1000]][1])
```

```
cv2.version 3.1.0
X_train[0].shape (32, 32, 3)
X_train.shape (34799, 32, 32, 3)
 len(X_train.shape) 4
################
X_train.shape[ 0 ] 34799
X_train.shape[ 1 ] 32
X_train.shape[ 2 ] 32
X_train.shape[ 3 ] 3
```





```
(32, 32, 3)
tapple
################
dict_keys(['coords', 'labels', 'features', 'sizes'])
coords
features
labels
sizes
dict_keys(['coords', 'labels', 'features', 'sizes'])
dict_keys(['sizes', 'coords', 'features', 'labels'])
################
(34799, 32, 32, 3) (4410, 32, 32, 3) (12630, 32, 32, 3)
(34799,) (4410,) (12630,)
36 4
Go straight or right & Speed limit (70km/h)
```

Provide a Basic Summary of the Data Set Using Python, Numpy and/or Pandas

In [4]:

```
########## Function (3)   Prepare system parameters and basic summary of the Data###############
###

import numpy as np
### Replace each question mark with the appropriate value.
### Use python, pandas or numpy methods rather than hard coding the results

# TODO: Number of training examples
n_train = X_train.shape[0]

# TODO: Number of validation examples
n_validation = X_valid.shape[0]

# TODO: Number of testing examples.
n_test = X_test.shape[0]

# TODO: What's the shape of an traffic sign image?
image_shape = X_test.shape[1:4]

# TODO: How many unique classes/labels there are in the dataset.
n_classes = len(np.unique(y_train))    #label'num = class's num

print("Number of training examples =", n_train)
print("Number of testing examples =", n_test)
print("Image data shape =", image_shape)
print("Number of classes =", n_classes)
```

```
Number of training examples = 34799
Number of testing examples = 12630
Image data shape = (32, 32, 3)
Number of classes = 43
```

Include an exploratory visualization of the dataset

Visualize the German Traffic Signs Dataset using the pickled file(s). This is open ended, suggestions include: plotting traffic sign images, plotting the count of each sign, etc.

The Matplotlib (http://matplotlib.org/) examples (http://matplotlib.org/examples/index.html) and gallery (http://matplotlib.org/gallery.html) pages are a great resource for doing visualizations in Python.

NOTE: It's recommended you start with something simple first. If you wish to do more, come back to it after you've completed the rest of the sections. It can be interesting to look at the distribution of classes in the training, validation and test set. Is the distribution the same? Are there more examples of some classes than others?

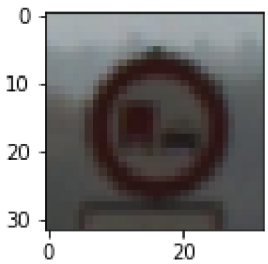
In [5]:

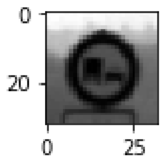
```
########## My Check Code Debug  (Optional)#################
# 各データの内訳確認
### Data exploration visualization code goes here.
### Feel free to use as many code cells as needed.
import matplotlib.pyplot as plt
# Visualizations will be shown in the notebook.
%matplotlib inline


import random
index = random.randint(0, len(X_train))
image = X_train[index].squeeze()

plt.figure(figsize=(2,2))
plt.imshow(image)  ##plt.imshow(image, cmap="gray")

gray = cv2.cvtColor(X_train[index], cv2.COLOR_BGR2GRAY)
plt.figure(figsize=(1,2))
plt.imshow(gray, cmap="gray")



from PIL import Image
import numpy as np

print(index,'y_train[index],xlist[y_train[index]-------',y_train[index],xlist[y_train[index]])

print('X_train[index]',type(X_train[index]) )
pilImg = Image.fromarray(np.uint8(X_train[index]))

plt.figure(figsize=(1,1))
plt.imshow(pilImg)
print('pilImg',type(pilImg) )


plt.figure(figsize=(1,1))
plt.imshow(X_train[index])
print(X_train[index].shape)
gray2 = cv2.cvtColor(X_train[index], cv2.COLOR_BGR2GRAY)
plt.figure(figsize=(1,1))
plt.imshow(gray)
print(gray.shape)

gray= cv2.bilateralFilter(gray,9,55,55)


plt.figure(figsize=(1,1))
plt.imshow(gray,cmap='gray')
```

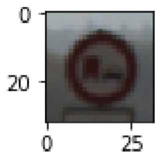
```
17566 y_train[index],xlist[y_train[index]------- 10 ['10', 'No passing for vehicle
s over 3.5 metric tons']
X_train[index] <class 'numpy.ndarray'>
pilImg <class 'PIL.Image.Image'>
(32, 32, 3)
(32, 32)
```

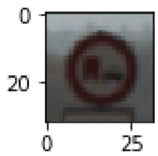
Out[5]:

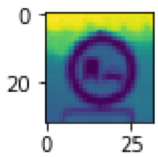
```
<matplotlib.image.AxesImage at 0x7f4bc15513c8>
```

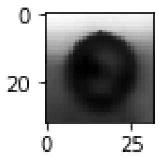












Step 2: Design and Test a Model Architecture

Design and implement a deep learning model that learns to recognize traffic signs. Train and test your model on the German Traffic Sign Dataset (http://benchmark.ini.rub.de/?section=gtsrb&subsection=dataset).

The LeNet-5 implementation shown in the classroom (https://classroom.udacity.com/nanodegrees/nd013/parts/fbf77062-5703-404e-b60c-95b78b2f3f9e/modules/6df7ae49-c61c-4bb2-a23e-6527e69209ec/lessons/601ae704-1035-4287-8b11-e2c2716217ad/concepts/d4aca031-508f-4e0b-b493-e7b706120f81) at the end of the CNN lesson is a solid starting point. You'll have to change the number of classes and possibly the preprocessing, but aside from that it's plug and play!

With the LeNet-5 solution from the lecture, you should expect a validation set accuracy of about 0.89. To meet specifications, the validation set accuracy will need to be at least 0.93. It is possible to get an even higher accuracy, but 0.93 is the minimum for a successful project submission.

There are various aspects to consider when thinking about this problem:

- Neural network architecture (is the network over or underfitting?)
- Play around preprocessing techniques (normalization, rgb to grayscale, etc)
- Number of examples per label (some have more than others).
- Generate fake data.

Here is an example of a published baseline model on this problem (http://yann.lecun.com/exdb/publis/pdf/sermanet-ijcnn-11.pdf). It's not required to be familiar with the approach used in the paper but, it's good practice to try to read papers like these.

Pre-process the Data Set (normalization, grayscale, etc.)

Minimally, the image data should be normalized so that the data has mean zero and equal variance. For image data, `(pixel - 128)/ 128` is a quick way to approximately normalize the data and can be used in this project.

Other pre-processing steps are optional. You can try different techniques to see if it improves performance.

Use the code cell (or multiple code cells, if necessary) to implement the first step of your project.

In [6]:

```
########## Function (4) :Define standard processing (grayscale image)  ##################
import numpy as np
def zscore(x, axis = None):
####  データから平均値をとって標準偏差で割ると正規化
    xmean = x.mean(axis=axis, keepdims=True)
    xstd  = np.std(x, axis=axis, keepdims=True)
    zscore = (x-xmean)/xstd
    return zscore
def min_max(x, axis=None):
####  データからを最大最小差で処理  →  階調補正
    min = x.min(axis=axis, keepdims=True)
    max = x.max(axis=axis, keepdims=True)
    result = (x-min)/(max-min)
    return result
###https://deepage.net/features/numpy-normalize.html
```

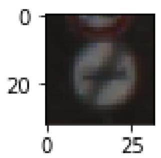
In [7]:

```
########## Check My Code  for Debug (Optional)##################
A=X_train[0]
B=X_train[1]
C=X_train[2]

print(A.shape,type(A))
print(B.shape,type(B))
print(C.shape,type(C))
D= A
print('D',D.shape)
D= np.array([D,B])
print('D',D.shape)
D= np.array([A,B,C])
print('D',D.shape)

##############################################
res=10
x_train = []
for row in range(res):
    x_train_temp =A
    #img_path = img_dir + row['id'] + '.png'
    #x_train_temp = cv2.imread(img_path)
    # x_train = np.append(x_train,x_train_temp,axis=0)
    x_train.append(x_train_temp)

print(type(x_train))
x_train = np.asarray(x_train) ##list to ndArray
print(type(x_train))

print('x_train',x_train.shape,type(x_train))

import matplotlib.pyplot as plt
# Visualizations will be shown in the notebook.
%matplotlib inline
plt.figure(figsize=(1,1))
plt.imshow(x_train[9])
#############################
```

```
(32, 32, 3) <class 'numpy.ndarray'>
(32, 32, 3) <class 'numpy.ndarray'>
(32, 32, 3) <class 'numpy.ndarray'>
D (32, 32, 3)
D (2, 32, 32, 3)
D (3, 32, 32, 3)
<class 'list'>
<class 'numpy.ndarray'>
x_train (10, 32, 32, 3) <class 'numpy.ndarray'>
```

Out[7]:

```
<matplotlib.image.AxesImage at 0x7f4bc15b26a0>
```



In [8]:

```
########## Function (5)   Define conversion function from RGB to grayscale ##################
from PIL import Image   #use pilImg
import cv2
import numpy as np
####$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

def MyConvertRGB2NRMGRAY(src,flt):
    #print(src.shape)
    gray= cv2.cvtColor(src, cv2.COLOR_RGB2GRAY)
    #print(gray.shape,gray.dtype)

    #GRAY[1:3]=gray
    if(flt&0x08):
        gray= cv2.bilateralFilter(gray,3,55,55)
        #print('do it')

    if(flt&0x04):
        gray= cv2.GaussianBlur(gray,(3,3),0)
        #print('do it')

    if(flt&0x02):
        gray = zscore(gray)

    if(flt&0x01):
        gray = min_max(gray)
        #print('do it')


    #print(gray.shape,gray.dtype)

    GRAY=np.reshape(gray.astype(np.float32),(32,32,1))
    #print(GRAY.shape,GRAY.dtype)

    return GRAY
```

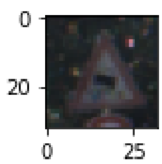
In [9]:

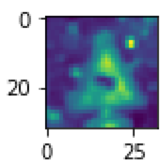
```
##########  Check My Code  for Debug (Optional)#################

x_temp=[]
gray= MyConvertRGB2NRMGRAY(X_test[0], 0)

print(type(gray),gray.shape)
x_temp.append(gray)
temp = np.asarray(x_temp) ##list to ndArray

print(type(temp),temp.shape)



test=MyConvertRGB2NRMGRAY(X_train[1999],7)
#print(type(test),test.shape)
#print(test)
import matplotlib.pyplot as plt
# Visualizations will be shown in the notebook.
%matplotlib inline

plt.figure(figsize=(1,5))
plt.imshow(X_train[1999].squeeze())
plt.figure(figsize=(1,5))
plt.imshow(test.squeeze())
```

```
<class 'numpy.ndarray'> (32, 32, 1)
<class 'numpy.ndarray'> (1, 32, 32, 1)
```

Out[9]:

```
<matplotlib.image.AxesImage at 0x7f4bd9a98f28>
```





In [10]:

```
########## Function (6)   Make a list of grayscale images (1K data at a time) ##################

########## [n,32,32,3]のデータを[n,32,32]に変換する
########## メモリの都合があるので1000データずつ処理してpickle配列に格納する

### Preprocess the data here. It is required to normalize the data. Other preprocessing steps co
uld include
### converting to grayscale, etc.
### Feel free to use as many code cells as needed.
import matplotlib.pyplot as plt
import numpy as np

# Dump (Save)  pickled data
import pickle

##http://blog.amedama.jp/entry/2015/12/05/132520
import io
global file


def conv_rgb2gray_to_pickleArray1kGroup(rgb_src_list,flag):

    file=np.array([ io.BytesIO() ])

 #################################################
    print(type(rgb_src_list))
    x_temp =[]

    total=rgb_src_list.shape[0]
    print("loop num total",total)
    for i in range( total) :
        gray= MyConvertRGB2NRMGRAY(rgb_src_list[i], flag)
        x_temp.append(gray)

        next=i+1
        if((next%1000)==0):
            ###ndarrayではなく  一旦listでpikle保存
            pickle.dump(x_temp, file[int(i/1000)])
            file=np.append(file,io.BytesIO())
            print("conv",next,"rate",float(next)/float(total))
            #debug用
            tmp=np.asarray(x_temp)
            print('x_temp',type(tmp),tmp.shape)
            #再度初期化リセット
            x_temp =  []
        if(next==total):
            pickle.dump(x_temp, file[int(i/1000)])

    return file
```

In [11]:

```
##########  Check My Code  for Debug (Optional)##################
### conv_rgb2gray_to_pickleArray1kGroup の動作を確認してみる
"""
global file
file=conv_rgb2gray_to_pickleArray1kGroup(X_train,0)

x_temp =  []
file[0].seek(0)
xlist=pickle.load(file[0])
x_temp.extend(xlist)
tmp= np.asarray(x_temp)
print(tmp.shape)#########

file[1].seek(0)
xlist=pickle.load(file[1])
x_temp.extend(xlist)
tmp= np.asarray(x_temp)
print(tmp.shape)#########

plt.figure(figsize=(1,5))
plt.imshow(x_temp[1999])

plt.figure(figsize=(1,5))
plt.imshow(X_train[1999])
"""
```

Out[11]:

```
'¥nglobal file¥nfile=conv_rgb2gray_to_pickleArray1kGroup(X_train,0)¥n¥nx_temp =
[] ¥nfile[0].seek(0)¥nxlist=pickle.load(file[0])¥nx_temp.extend(xlist)¥ntmp= np.as
array(x_temp)¥nprint(tmp.shape)#########¥n¥nfile[1].seek(0)¥nxlist=pickle.load(fil
e[1])¥nx_temp.extend(xlist)¥ntmp= np.asarray(x_temp)¥nprint(tmp.shape)#########¥n
¥nplt.figure(figsize=(1,5))¥nplt.imshow(x_temp[1999])¥n¥nplt.figure(figsize=(1,5))
¥nplt.imshow(X_train[1999])¥n'
```

In [12]:

```
########## Function(7) Concatenate the list to create an array(converted-image) & record the arr
ay
import pickle
import io

def encode_pickleArray1kGroup_to_grayImageArray(file,save_p_file_name):
    num=file.shape
    print(num)
    x_temp =  []
    for i in range(num[0]) :
        file[i].seek(0)
        tmp=pickle.load(file[i])
        x_temp.extend(tmp)


    print(type(x_temp))
    x_temp = np.asarray(x_temp) ##list to ndArray
    print(type(x_temp),x_temp.shape)


    with open(save_p_file_name, mode='wb') as f:
        pickle.dump(x_temp, f)
        print("complete");

    return
```

In [13]:

```
##########  Check My Code  for Debug (Optional)#################
### encode_pickleArray1kGroup_to_grayImageArray の動作を確認してみる
"""
encode_pickleArray1kGroup_to_grayImageArray(file,training_t_file)
train_cnv =[]
with open(training_t_file, mode='rb') as f:
        train_cnv = pickle.load(f)

print(type(train_cnv),train_cnv.shape)
plt.figure(figsize=(1,5))
plt.imshow(train_cnv[1999])
"""
```

Out[13]:

```
"¥nencode_pickleArray1kGroup_to_grayImageArray(file,training_t_file)¥ntrain_cnv =
[]¥nwith open(training_t_file, mode='rb') as f:¥n        train_cnv = pickle.load
(f)¥n¥nprint(type(train_cnv),train_cnv.shape)¥nplt.figure(figsize=(1,5))¥nplt.imsh
ow(train_cnv[1999])¥n"
```

In [14]:

```
###Function (8)   Load arrays and restore transformed images(train,valid,test + α)
##********* Restart Point *****************************
import os.path
import pickle
import io
```

```
training_t_file = './train_t.p'
validation_t_file='./valid_t.p'
testing_t_file =  './test_t.p'

path=[training_t_file,validation_t_file,testing_t_file]

#グレー化データのファイル有無を確認
flgExist=[os.path.isfile(path[0]),os.path.isfile(path[1]),os.path.isfile(path[2])]

x_train=[]
x_valid=[]
x_test=[]

global file

file=[]
if(flgExist[0]==False):
    file=conv_rgb2gray_to_pickleArray1kGroup(X_train,1+2+4)
    encode_pickleArray1kGroup_to_grayImageArray(file,path[0])
with open(path[0], mode='rb') as f:
    x_train = pickle.load(f)

file=[]
if(flgExist[1]==False):
    file=conv_rgb2gray_to_pickleArray1kGroup(X_valid,1+2+4)
    encode_pickleArray1kGroup_to_grayImageArray(file,path[1])
with open(path[1], mode='rb') as f:
    x_valid = pickle.load(f)

file=[]
if(flgExist[2]==False):
    file=conv_rgb2gray_to_pickleArray1kGroup(X_test,1+2+4)
    encode_pickleArray1kGroup_to_grayImageArray(file,path[2])
with open(path[2], mode='rb') as f:
    x_test = pickle.load(f)

print(type(X_test),X_test.shape ,"to ",type(x_test),x_test.shape)


##############################################
training_add1_file = './train_add1.p'
training_add2_file = './train_add2.p'

path.append(training_add1_file)
path.append(training_add2_file)

print(path)
flgExist.append(os.path.isfile(path[3]) )
flgExist.append(os.path.isfile(path[4]) )
print(flgExist)

file=[]
if(flgExist[3]==False):
    file=conv_rgb2gray_to_pickleArray1kGroup(X_train,1+2)
    encode_pickleArray1kGroup_to_grayImageArray(file,path[3])
with open(path[3], mode='rb') as f:
    x_add1 = pickle.load(f)
```

```
file=[]
if(flgExist[4]==False):
    file=conv_rgb2gray_to_pickleArray1kGroup(X_train,1+2+8)
    encode_pickleArray1kGroup_to_grayImageArray(file,path[4])
with open(path[4], mode='rb') as f:
    x_add2 = pickle.load(f)



print(x_train.shape)


xplus=[]
yplus=[]
y_add1=y_train
y_add2=y_train


for i in range(x_train.shape[0]) :
    xplus.append(x_train[i])
    yplus.append(y_train[i])

#from sklearn.utils import shuffle
#GX_train, Gy_train = shuffle(GX_train, Gy_train)
from sklearn.utils import shuffle
x_add1, y_add1 = shuffle(x_add1, y_add1)
x_add2, y_add2 = shuffle(x_add2, y_add2)

##add_num= int(x_train.shape[0]/4)
add_num= int(x_train.shape[0])

for i in range(add_num):
    xplus.append(x_add1[i])
    yplus.append(y_add1[i])

for i in range(add_num):
    xplus.append(x_add2[i])
    yplus.append(y_add2[i])

x_train_plus=np.asarray(xplus)
y_train_plus=np.asarray(yplus)

print(x_train.shape,y_train.shape)
print(x_train_plus.shape,y_train_plus.shape)
```

```
<class 'numpy.ndarray'>
loop num total 34799
conv 1000 rate 0.0287364579441938
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 2000 rate 0.0574729158883876
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 3000 rate 0.0862093738325814
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 4000 rate 0.1149458317767752
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 5000 rate 0.143682289720969
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 6000 rate 0.1724187476651628
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 7000 rate 0.2011552056093566
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 8000 rate 0.2298916635535504
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 9000 rate 0.25862812149774417
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 10000 rate 0.287364579441938
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 11000 rate 0.31610103738613177
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 12000 rate 0.3448374953303256
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 13000 rate 0.37357395327451937
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 14000 rate 0.4023104112187132
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 15000 rate 0.43104686916290696
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 16000 rate 0.4597833271071008
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 17000 rate 0.48851978505129456
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 18000 rate 0.5172562429954883
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 19000 rate 0.5459927009396822
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 20000 rate 0.574729158883876
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 21000 rate 0.6034656168280698
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 22000 rate 0.6322020747722635
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 23000 rate 0.6609385327164574
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 24000 rate 0.6896749906606512
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 25000 rate 0.718411448604845
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 26000 rate 0.7471479065490387
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 27000 rate 0.7758843644932326
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 28000 rate 0.8046208224374264
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
```

```
conv 29000 rate 0.8333572803816202
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 30000 rate 0.8620937383258139
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 31000 rate 0.8908301962700078
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 32000 rate 0.9195666542142016
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 33000 rate 0.9483031121583954
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 34000 rate 0.9770395701025891
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
(35,)
<class 'list'>
<class 'numpy.ndarray'> (34799, 32, 32, 1)
complete
<class 'numpy.ndarray'>
loop num total 4410
conv 1000 rate 0.22675736961451248
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 2000 rate 0.45351473922902497
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 3000 rate 0.6802721088435374
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 4000 rate 0.9070294784580499
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
(5,)
<class 'list'>
<class 'numpy.ndarray'> (4410, 32, 32, 1)
complete
<class 'numpy.ndarray'>
loop num total 12630
conv 1000 rate 0.0791765637371338
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 2000 rate 0.1583531274742676
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 3000 rate 0.2375296912114014
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 4000 rate 0.3167062549485352
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 5000 rate 0.395882818685669
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 6000 rate 0.4750593824228028
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 7000 rate 0.5542359461599367
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 8000 rate 0.6334125098970704
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 9000 rate 0.7125890736342043
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 10000 rate 0.7917656373713381
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 11000 rate 0.8709422011084719
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 12000 rate 0.9501187648456056
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
(13,)
<class 'list'>
<class 'numpy.ndarray'> (12630, 32, 32, 1)
```

```
complete
<class 'numpy.ndarray'> (12630, 32, 32, 3) to  <class 'numpy.ndarray'> (12630, 32, 
32, 1)
['./train_t.p', './valid_t.p', './test_t.p', './train_add1.p', './train_add2.p']
[False, False, False, False, False]
<class 'numpy.ndarray'>
loop num total 34799
conv 1000 rate 0.0287364579441938
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 2000 rate 0.0574729158883876
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 3000 rate 0.0862093738325814
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 4000 rate 0.1149458317767752
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 5000 rate 0.143682289720969
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 6000 rate 0.1724187476651628
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 7000 rate 0.2011552056093566
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 8000 rate 0.2298916635535504
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 9000 rate 0.25862812149774417
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 10000 rate 0.287364579441938
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 11000 rate 0.31610103738613177
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 12000 rate 0.3448374953303256
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 13000 rate 0.37357395327451937
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 14000 rate 0.4023104112187132
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 15000 rate 0.43104686916290696
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 16000 rate 0.4597833271071008
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 17000 rate 0.48851978505129456
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 18000 rate 0.5172562429954883
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 19000 rate 0.5459927009396822
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 20000 rate 0.574729158883876
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 21000 rate 0.6034656168280698
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 22000 rate 0.6322020747722635
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 23000 rate 0.6609385327164574
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 24000 rate 0.6896749906606512
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 25000 rate 0.718411448604845
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 26000 rate 0.7471479065490387
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
```

```
conv 27000 rate 0.7758843644932326
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 28000 rate 0.8046208224374264
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 29000 rate 0.8333572803816202
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 30000 rate 0.8620937383258139
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 31000 rate 0.8908301962700078
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 32000 rate 0.9195666542142016
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 33000 rate 0.9483031121583954
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 34000 rate 0.9770395701025891
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
(35,)
<class 'list'>
<class 'numpy.ndarray'> (34799, 32, 32, 1)
complete
<class 'numpy.ndarray'>
loop num total 34799
conv 1000 rate 0.0287364579441938
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 2000 rate 0.0574729158883876
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 3000 rate 0.0862093738325814
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 4000 rate 0.1149458317767752
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 5000 rate 0.143682289720969
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 6000 rate 0.1724187476651628
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 7000 rate 0.2011552056093566
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 8000 rate 0.2298916635535504
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 9000 rate 0.25862812149774417
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 10000 rate 0.287364579441938
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 11000 rate 0.31610103738613177
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 12000 rate 0.3448374953303256
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 13000 rate 0.37357395327451937
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 14000 rate 0.4023104112187132
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 15000 rate 0.43104686916290696
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 16000 rate 0.4597833271071008
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 17000 rate 0.48851978505129456
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 18000 rate 0.5172562429954883
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 19000 rate 0.5459927009396822
```

```
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 20000 rate 0.574729158883876
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 21000 rate 0.6034656168280698
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 22000 rate 0.6322020747722635
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 23000 rate 0.6609385327164574
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 24000 rate 0.6896749906606512
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 25000 rate 0.718411448604845
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 26000 rate 0.7471479065490387
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 27000 rate 0.7758843644932326
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 28000 rate 0.8046208224374264
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 29000 rate 0.8333572803816202
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 30000 rate 0.8620937383258139
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 31000 rate 0.8908301962700078
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 32000 rate 0.9195666542142016
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 33000 rate 0.9483031121583954
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
conv 34000 rate 0.9770395701025891
x_temp <class 'numpy.ndarray'> (1000, 32, 32, 1)
(35,)
<class 'list'>
<class 'numpy.ndarray'> (34799, 32, 32, 1)
complete
(34799, 32, 32, 1)
(34799, 32, 32, 1) (34799,)
(104397, 32, 32, 1) (104397,)
```

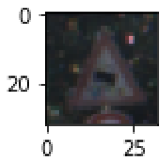
In [15]:

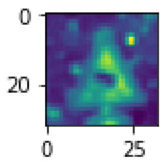
```
##########  Check My Code  for Debug (Optional)#################
print(type(X_train),X_train.shape ,"to ",type(x_train),x_train.shape)
plt.figure(figsize=(1,5))
plt.imshow(X_train[1999].squeeze())
plt.figure(figsize=(1,5))
plt.imshow(x_train[1999].squeeze())

print(type(X_valid),X_valid.shape ,"to ",type(x_valid),x_valid.shape)
plt.figure(figsize=(1,5))
plt.imshow(X_valid[1999].squeeze())
plt.figure(figsize=(1,5))
plt.imshow(x_valid[1999].squeeze())

print(type(X_test),X_test.shape ,"to ",type(x_test),x_test.shape)
plt.figure(figsize=(1,5))
plt.imshow(X_test[1999].squeeze())
plt.figure(figsize=(1,5))
plt.imshow(x_test[1999].squeeze())
```

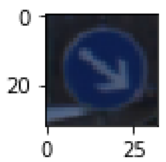
```
<class 'numpy.ndarray'> (34799, 32, 32, 3) to  <class 'numpy.ndarray'> (34799, 32,
32, 1)
<class 'numpy.ndarray'> (4410, 32, 32, 3) to  <class 'numpy.ndarray'> (4410, 32, 3
2, 1)
<class 'numpy.ndarray'> (12630, 32, 32, 3) to  <class 'numpy.ndarray'> (12630, 32,
32, 1)
```

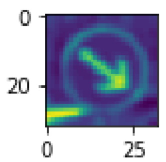
Out[15]:

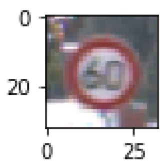
```
<matplotlib.image.AxesImage at 0x7f4ba60ea400>
```

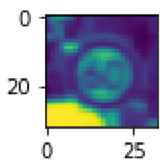












In [16]:

```
########## My Check Code Debug (Optional)##################
###leNet CNNサンプルで動作確認
from tensorflow.examples.tutorials.mnist import input_data

mnist = input_data.read_data_sets("MNIST_data/", reshape=False)

lX_train, ly_train           = mnist.train.images, mnist.train.labels
lX_validation, ly_validation = mnist.validation.images, mnist.validation.labels
lX_test, ly_test             = mnist.test.images, mnist.test.labels

assert(len(lX_train) == len(ly_train))
assert(len(lX_validation) == len(ly_validation))
assert(len(lX_test) == len(ly_test))

import numpy as np
#  パディングにより入力データを32x32に修正する （重要）

# Pad images with 0s
lX_train      = np.pad(lX_train, ((0,0),(2,2),(2,2),(0,0)), 'constant')
lX_validation = np.pad(lX_validation, ((0,0),(2,2),(2,2),(0,0)), 'constant')
lX_test       = np.pad(lX_test, ((0,0),(2,2),(2,2),(0,0)), 'constant')

print('lX_train[0]',lX_train[0].shape,lX_train[0].dtype)
print('x_train[0]',  x_train[0].shape ,x_train[0].dtype)

"""
x_train=  lX_train
y_train=  ly_train

x_valid=  lX_validation
y_valid=  ly_validation

x_test=  lX_test
y_test=  lX_test


"""
```

```
/home/uda/.conda/envs/IntroToTensorFlow/lib/python3.6/importlib/_bootstrap.py:219:
RuntimeWarning: compiletime version 3.5 of module 'tensorflow.python.framework.fas
t_tensor_util' does not match runtime version 3.6
  return f(*args, **kwds)
```

```
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
lX_train[0] (32, 32, 1) float32
x_train[0] (32, 32, 1) float32
```

Out[16]:

```
'    ¥nx_train=  lX_train¥ny_train=  ly_train¥n¥nx_valid=  lX_validation¥ny_valid=
  ly_validation¥n¥nx_test=  lX_test¥ny_test=  lX_test¥n¥n¥n'
```

In [17]:

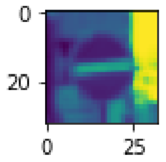
```
#####Function (9)   Replace training, validation, testing data with returned grayscale image
#####               Furthermore, initialize data used in TF

### Define your architecture here.
### Feel free to use as many code cells as needed.
global GX_train,Gy_train
global GX_valid,Gy_valid
global GX_test,Gy_test
global nTotalClasses

GX_train=x_train_plus
Gy_train=y_train_plus

#GX_train=x_train
#Gy_train=y_train

GX_valid=x_valid
Gy_valid=y_valid

GX_test=x_test
Gy_test=y_test

nTotalClasses= len(np.unique(Gy_test))    #label'num = class's num
print('nTotalClasses',nTotalClasses)


import tensorflow as tf

EPOCHS = 20
BATCH_SIZE = 128

global learningrate
learningrate=0.001

global nDROP
nDROP=[0, 0.9,0.8,0.5,0.5]
#tf.nn.dropoutへの指定は残すネットワークの割合
###Hinton氏の提案通り入力層他は0.8以上　全結合層を0.5としておく。
###トレーニング以外の時はDropoutを使用しない（100%残す＞＞＞tf.nn.dropout(1.0))
```

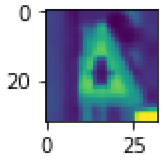
```
nTotalClasses 43
```

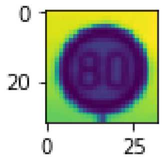
In [18]:

```
########## Function (10)  Shuffle training data for startup  ##########

from sklearn.utils import shuffle
global GX_train, Gy_train
GX_train, Gy_train = shuffle(GX_train, Gy_train)

for i in range(3):
    print(Gy_train[i*1000],'&',xlist[Gy_train[i*1000]][1])
    plt.figure(figsize=(1,5))
    plt.imshow(GX_train[i*1000].squeeze())
```

```
17 & No entry
11 & Right-of-way at the next intersection
5 & Speed limit (80km/h)
```







Model Architecture

In [19]:

```
########## Function (11) Define the leNet with Dropout ##########

from tensorflow.contrib.layers import flatten
global GX_train,Gy_train
global GX_valid,Gy_valid
global GX_test,Gy_test
global nTotalClasses
global nDROP

#①ネットワークを構築する
def LeNet(x):
    # Arguments used for tf.truncated_normal, randomly defines variables for the weights and bia
ses for each layer
    mu = 0
    sigma = 0.1

    # SOLUTION: Layer 1: Convolutional. Input = 32x32x1. Output = 28x28x6.
    # 5x5のフィルタを適用すると幅高さ32x32→幅高さ28x28となる
```

```
    # 加えて入力深度が1, 出力深度が6なので  shape=(5, 5, 1, 6)を適用する
    conv1_W = tf.Variable(tf.truncated_normal(shape=(5, 5, 1, 6), mean = mu, stddev = sigma))
    #出力深度が6なのでバイアスも6セット用意
    conv1_b = tf.Variable(tf.zeros(6))
    conv1   = tf.nn.conv2d(x, conv1_W, strides=[1, 1, 1, 1], padding='VALID') + conv1_b

    # SOLUTION: Activation.
    #出力結果にreluを適用する
    conv1 = tf.nn.relu(conv1)
    if((nDROP[0]!=0)&(nDROP[1]>0)):
        conv1 = tf.nn.dropout(nDROP[1])


    # SOLUTION: Pooling. Input = 28x28x6. Output = 14x14x6.
    #  maxプーリングを使ってサイズを縮小する
    #  入力が[index,幅,高さ,深度]なので第0th,3thは1  1th,2thにカーネルとストライドサイズを設定
    #  ksize=[1, w, h, 1]  strides=[1, hstride, vstride, 1]
    conv1 = tf.nn.max_pool(conv1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='VALID')

    # SOLUTION: Layer 2: Convolutional. Input = 14x14x6.  Output = 10x10x16.
    #  フィルタサイズが5x5  入力深度が6  出力深度が16
    conv2_W = tf.Variable(tf.truncated_normal(shape=(5, 5, 6, 16), mean = mu, stddev = sigma))
    conv2_b = tf.Variable(tf.zeros(16))
    conv2   = tf.nn.conv2d(conv1, conv2_W, strides=[1, 1, 1, 1], padding='VALID') + conv2_b

    # SOLUTION: Activation.
    conv2 = tf.nn.relu(conv2)
    if((nDROP[0]!=0)&(nDROP[2]>0)):
        conv1 = tf.nn.dropout(nDROP[2])

    # SOLUTION: Pooling. Input = 10x10x16. Output = 5x5x16.
    conv2 = tf.nn.max_pool(conv2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='VALID')

    # SOLUTION: Flatten. Input = 5x5x16. Output = 400.
    #  5x5x16=400  1次元に変換
    fc0   = flatten(conv2)

    # SOLUTION: Layer 3: Fully Connected. Input = 400. Output = 120.
    fc1_W = tf.Variable(tf.truncated_normal(shape=(400, 120), mean = mu, stddev = sigma))
    fc1_b = tf.Variable(tf.zeros(120))
    fc1   = tf.matmul(fc0, fc1_W) + fc1_b

    # SOLUTION: Activation.
    fc1    = tf.nn.relu(fc1)
    if((nDROP[0]!=0)&(nDROP[3]>0)):
        conv1 = tf.nn.dropout(nDROP[3])

    # SOLUTION: Layer 4: Fully Connected. Input = 120. Output = 84.
    fc2_W  = tf.Variable(tf.truncated_normal(shape=(120, 84), mean = mu, stddev = sigma))
    fc2_b  = tf.Variable(tf.zeros(84))
    fc2    = tf.matmul(fc1, fc2_W) + fc2_b

    # SOLUTION: Activation.
    fc2    = tf.nn.relu(fc2)
    if((nDROP[0]!=0)&(nDROP[4]>0)):
        conv1 = tf.nn.dropout(nDROP[4])

    # SOLUTION: Layer 5: Fully Connected. Input = 84. Output = nTotalClasses.
    #  tf.Variable(tf.Sessionのための準備)を実行
```

```
    fc3_W  = tf.Variable(tf.truncated_normal(shape=(84, nTotalClasses), mean = mu, stddev = sigm
a))
    fc3_b  = tf.Variable(tf.zeros(nTotalClasses))

    #fc3_W  = tf.Variable(tf.truncated_normal(shape=(84, 43), mean = mu, stddev = sigma))
    #fc3_b  = tf.Variable(tf.zeros(43))
    logits = tf.matmul(fc2, fc3_W) + fc3_b

    return logits
```

In [20]:

```
########## Function (12) Set placeholder & one_hot ##########

#ひな形を用意する
global nTotalClasses

#  32x32x1の画像を用意 (index=None :大きさを指定しない)
x = tf.placeholder(tf.float32, (None, 32, 32, 1))
y = tf.placeholder(tf.int32, (None))


#y(int32型ラベルデータ)をnTotalClassesパターン分  one_hotデータ(結果)として用意
one_hot_y = tf.one_hot(y, nTotalClasses)
```

In [21]:

```
########## Function (13) Set the TF function (group 1) ##########

#トレーニング内容を定義する
global learningrate

rate= learningrate
##rate = 0.001

#用意したLeNet関数に入力x (32x32x1型画像を適用)
# ※※※ xはplaceholderで定義　且つ　section内の「辞書」で入力実行※※
logits = LeNet(x)
#結果をソフトマックス・クロスエントロピーに適用
##########
#等価処理は               y = tf.nn.softmax(tf.matmul(x,W) + b)
##########  cross_entropy = -tf.reduce_sum(y_*tf.log(y))

# ※※※ y one_hot_yはplaceholderで定義したyより生成　且つ　section内の「辞書」で入力実行※※
cross_entropy = tf.nn.softmax_cross_entropy_with_logits(labels=one_hot_y, logits=logits)

#総和平均で得点を計算rate
loss_operation = tf.reduce_mean(cross_entropy)

#Adam法による最急降下を選択
optimizer = tf.train.AdamOptimizer(learning_rate = rate)
#他最急降下法の適用例：  optimizer = tf.train.GradientDescentOptimizer(0.01).minimize(cross_entr
opy)

#Adam法で評価
training_operation = optimizer.minimize(loss_operation)

#  「training_operation」が　本プロジェクトの骨子・核である「モデル生成および鍛錬」の為の処理で
ある
#    最急降下法に与える学習レートを調整しながら、とにかくtraining_operationを繰り返すほどモデル
精度が向上する
#    （ただし、過学習には注意　）
```

In [22]:

```
########## Function (14)  Set the TF function (group 2) ##########

###############後述
probrem_softmax=tf.nn.softmax(logits=logits)
#probrem_softmax=logits
```

In [23]:

```
########## Function (15)  Set the TF function (group 3)  ##########

import tensorflow as tf

#スコア評価方法を定義する

#tf.argmaxの引数
#tf.argmax (A,B)    Aが真のデータ、Bが評価されるデータ
#行列についてはdimensionに0を指定すると、行成分についての最大値をもつ要素（列成分）の添字を返却します。
```

```
#一方dimensionに1を指定すると、列成分についての最小値を持つ要素（行成分）の添字を返却します。

#tf.equal：ベクトルが一致しているか否か  True or False  （LeNet結果とOne_Hot値）
correct_prediction = tf.equal(tf.argmax(logits, 1), tf.argmax(one_hot_y, 1))
#総和平均で得点を計算 →  logitsとone_hotの件数分（例：55000件)のTRUE,FALSEが戻る  全部Trueならt
f.reduce_meanは100%一致と出力する
accuracy_operation = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))


tf_argmax=[[tf.argmax(logits, 1)] , [tf.argmax(one_hot_y, 1)]]

####################################################
#flg= tf.cast(correct_prediction, tf.int32)
#accuracy_operation_custom=   tf.cast(pow(-1,flg+1) ,tf.int32) *   tf.cast(tf.argmax(logits, 1)
 ,tf.int32)

#flg= tf.cast(correct_prediction, tf.float32)
#nn=   tf.cast(tf.argmax(logits, 1) ,tf.float32)
#accuracy_operation_custom=   pow(-1,flg+1)  * pow(10.0,nn )

accuracy_operation_custom= [ correct_prediction, tf.cast(tf.argmax(logits, 1)[0],tf.int32) ,tf.c
ast(tf.argmax(one_hot_y, 1),tf.int32)[0] ]

#######################################################

#accuracy_operation_custom = correct_prediction


#Tensorflowの学習パラーメータのsave, restoreにはtf.train.Saverを使用
#    →  tf.train.Saver()の引数を指定しない場合は全ての変数が保存
saver = tf.train.Saver()


def evaluatesoftmax(X_data, y_data):
    sess = tf.get_default_session()
    return  sess.run(probrem_softmax  , feed_dict={x:X_data, y:y_data})




def evaluate(X_data, y_data):
    num_examples = len(X_data)
    total_accuracy = 0

    #変数の初期化  global_variables_initializer
    #セッション中身を元の状態に戻す（再リセット）
    sess = tf.get_default_session()

    for offset in range(0, num_examples, BATCH_SIZE):
        batch_x, batch_y = X_data[offset:offset+BATCH_SIZE], y_data[offset:offset+BATCH_SIZE]
        accuracy = sess.run(accuracy_operation, feed_dict={x: batch_x, y: batch_y})
        total_accuracy += (accuracy * len(batch_x))
    return total_accuracy / num_examples

##############以降  追加のカスタムファンクション####################################################
def evaluatecustom(X_data, y_data):
    sess = tf.get_default_session()
    return  sess.run(accuracy_operation_custom  , feed_dict={x:X_data, y:y_data})

def show_debug(X_data, y_data):
```

```
    nDROP[0]=0
    sess = tf.get_default_session()
    print(sess.run(cross_entropy  , feed_dict={x:X_data, y:y_data}) )

    sess = tf.get_default_session()
    z_one_hot=sess.run(one_hot_y  , feed_dict={x:X_data, y:y_data})
    print('one_hot_y',z_one_hot,np.max(z_one_hot))

    sess = tf.get_default_session()
    zlogits=sess.run(logits  , feed_dict={x:X_data, y:y_data})
    print('logits',zlogits,np.max(zlogits) )

    sess = tf.get_default_session()
    zargmax=sess.run(tf_argmax  , feed_dict={x:X_data, y:y_data})
    print(zargmax)
    print('tf_argmax:logits',zargmax[0])
    print('tf_argmax:one_hot',zargmax[1])

    sess = tf.get_default_session()
    z_prediction=sess.run(correct_prediction  , feed_dict={x:X_data, y:y_data})
    print('correct_prediction',z_prediction)
    return
```

Train, Validate and Test the Model

A validation set can be used to assess how well the model is performing. A low accuracy on the training and validation sets imply underfitting. A high accuracy on the training set but low accuracy on the validation set implies overfitting.

In [24]:

```
########## Function (16)  Execute TF function & Record the Result (= Traffic-Sign Model) #######
###

### Train your model here.
### Calculate and report the accuracy on the training and validation set.
### Once a final model architecture is selected,
### the accuracy on the test set should be calculated and reported as well.
### Feel free to use as many code cells as needed.

#X_train訓練データ,Y_trainラベルデータを用いてモデルを作成
global GX_train,Gy_train
global GX_valid,Gy_valid
global GX_test,Gy_test
global nTotalClasses
global learningrate
global nDROP

import  os.path
exist_training = os.path.isfile('./learningrate.p')
# Dump (Save)  pickled data
import pickle

with tf.Session() as sess:
    if(exist_training==False):
        learningrate=0.001
```

```
        last_learningrate=learningrate

        sess.run(tf.global_variables_initializer())
        num_examples = len(GX_train)

        print("Training...")
        print()
        ##  EPOCHS=10 (パターンが数字0～9の10種類  →  1データずつ確認)
        for i in range(EPOCHS):

            #順序依存の誤教示を排除するためシャッフルする
            #  →  シャッフルにより  少数第２位水準で  実行のたびにモデル精度が変化する（シャッフル不要かも？）
            GX_train, Gy_train = shuffle(GX_train, Gy_train)

            #55000点データをBATCH_SIZE分ずつ処理
            for offset in range(0, num_examples, BATCH_SIZE):
                end = offset + BATCH_SIZE

                #batchはX_train、Y_trainの抜き出し部分（実入力データ）
                batch_x, batch_y = GX_train[offset:end], Gy_train[offset:end]

                nDROP[0]=1
                #先に定義したtraining_operation（Adam法）で入力データ[X_train,Y_train]を評価値に変換する
                sess.run(training_operation, feed_dict={x: batch_x, y: batch_y})

            print("EPOCH {} ...".format(i+1))

            """
            # evaluate は単なるデバッグのためだけの評価処理
            #    従ってevaluate は生成モデルの精度と全く無関係
            #    評価データX_validation, y_validation  は評価目的だけの変数(無駄)
            #            →X_test, y_test だけあれば動作評価できる

            """
            #evaluateでドロップアウトは使わない
            last_learningrate=learningrate
            nDROP[0]=0

            validation_accuracy = evaluate(GX_valid, Gy_valid)
            print("Validation Accuracy = {:.3f}".format(validation_accuracy))
            print()

            if (validation_accuracy>=0.935):
                learningrate=0.0000001
            elif (validation_accuracy>=0.93)&(validation_accuracy<0.935):
                learningrate=0.000001
            elif (validation_accuracy>=0.92)&(validation_accuracy<0.93):
                learningrate=0.00001
            elif (validation_accuracy>=0.91)&(validation_accuracy<0.92):
                learningrate=0.0001
            elif (validation_accuracy>=0.89)&(validation_accuracy<0.91):
                learningrate=0.001
            else:
                learningrate=0.01


        saver.save(sess, './lenet')
```

```
        print("Model saved")
        ##  tf.train.Saver.last_checkpoints

        with open( './learningrate.p', mode='wb') as f:
            pickle.dump( last_learningrate, f)
            print("complete");

    else :
        print('*********skip training************')


#saver.save(sess, './lenet')実行により
#①lenet.data-00000-of-00001②lenet.index③lenet.meta④checkpointの計４ファイルが記録される

#  saver.save(sess, './lenet')で保存されたモデルデータがあれば、Train the Modelは、以降　再実行
の必要ない！！
#  →  jupyterで　本ステップは省略・次項へ　スキップ可能
```

```
Training...

EPOCH 1 ...
Validation Accuracy = 0.863

EPOCH 2 ...
Validation Accuracy = 0.895

EPOCH 3 ...
Validation Accuracy = 0.895

EPOCH 4 ...
Validation Accuracy = 0.903

EPOCH 5 ...
Validation Accuracy = 0.916

EPOCH 6 ...
Validation Accuracy = 0.916

EPOCH 7 ...
Validation Accuracy = 0.910

EPOCH 8 ...
Validation Accuracy = 0.912

EPOCH 9 ...
Validation Accuracy = 0.905

EPOCH 10 ...
Validation Accuracy = 0.905

EPOCH 11 ...
Validation Accuracy = 0.920

EPOCH 12 ...
Validation Accuracy = 0.921

EPOCH 13 ...
Validation Accuracy = 0.935

EPOCH 14 ...
Validation Accuracy = 0.924

EPOCH 15 ...
Validation Accuracy = 0.924

EPOCH 16 ...
Validation Accuracy = 0.929

EPOCH 17 ...
Validation Accuracy = 0.930

EPOCH 18 ...
Validation Accuracy = 0.919

EPOCH 19 ...
Validation Accuracy = 0.924
```

```
EPOCH 20 ...
Validation Accuracy = 0.941

Model saved
complete
```

In [25]:

```
########## Function (17)  Load the last used learning rate ##########
import pickle
global learningrate
learningrate=0.0000001


def get_last_larning_rate(filename):
    with open(filename, mode='rb') as f:
        ret= pickle.load(f)
        print('load',filename,'=',ret)
        return  ret

learningrate=  get_last_larning_rate('./learningrate.p')
```

```
load ./learningrate.p = 1e-05
```

In [26]:

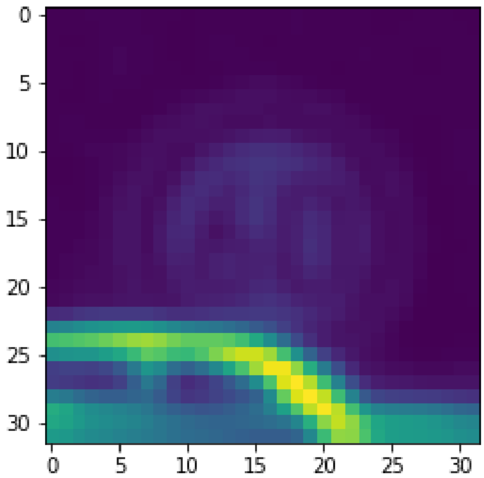
```
########## Function (18)  Load my Traffic-Sign Model＆Execute evaluation Test#########
#手法②  先に記録した定義モデルをrestoreでファイルからリロードする！


import tensorflow as tf
saver = tf.train.Saver()

print(GX_test.shape,Gy_test.shape)

global nDROP


with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    saver.restore(sess, './lenet')

    #X_test, y_testに対してevaluateの実行
    nDROP[0]=0
    test_accuracy = evaluate(GX_test, Gy_test)
    print("Test Accuracy = {:.3f}".format(test_accuracy))
```

```
(12630, 32, 32, 1) (12630,)
INFO:tensorflow:Restoring parameters from ./lenet
Test Accuracy = 0.912
```

In [27]:

```
########## My Check Code Debug (Optional)##################
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import tensorflow as tf
saver = tf.train.Saver()

global ErrorID
global nDROP


with tf.Session() as sess:
#先に記録した定義モデルをrestoreでファイルからリロードする！
    sess.run(tf.global_variables_initializer())
    saver.restore(sess, './lenet')
    nDROP[0]=0

    for i in range(GX_test.shape[0]) :
        X_try = np.array( [GX_test[i]])
        y_try = np.array( [Gy_test[i]])

        nDROP[0]=0
        n=( evaluatecustom(X_try, y_try) )

        #print(n)

        if(n[0]==False):
            print('i',i)
            print(GX_test.shape,type(GX_test))
            print(X_try.shape)

            print(Gy_test[i])
            print(n)
            print(xlist[n[1]],xlist[n[2]])

            plt.figure()
            plt.imshow(X_try.squeeze())
            ErrorID=i;
            if(i>2):
                break
```

```
INFO:tensorflow:Restoring parameters from ./lenet
i 23
(12630, 32, 32, 1) <class 'numpy.ndarray'>
(1, 32, 32, 1)
3
[array([False], dtype=bool), 5, 3]
['5', 'Speed limit (80km/h)'] ['3', 'Speed limit (60km/h)']
```



In [28]:

```
########## My Check Code Debug (Optional)##################
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import tensorflow as tf
saver = tf.train.Saver()

global ErrorID
global nDROP

learningrate = 0.0001
with tf.Session() as sess:
    OK=ErrorID-2
    sess.run(tf.global_variables_initializer())
    saver.restore(sess, './lenet')
    nDROP[0]=0

    X_try = np.array( [GX_test[OK]])
    y_try = np.array( [Gy_test[OK]])
    print('lbl',y_try)
    show_debug(X_try,y_try)

    OK=ErrorID-1
    sess.run(tf.global_variables_initializer())
    saver.restore(sess, './lenet')
    X_try = np.array( [GX_test[OK]])
    y_try = np.array( [Gy_test[OK]])
    print('lbl',y_try)
    show_debug(X_try,y_try)

    NG=ErrorID
    sess.run(tf.global_variables_initializer())
    saver.restore(sess, './lenet')
    X_try = np.array( [GX_test[NG]])
    y_try = np.array( [Gy_test[NG]])
    print('lbl',y_try)
    show_debug(X_try,y_try)
```

```
INFO:tensorflow:Restoring parameters from ./lenet
lbl [33]
[ 0.]
one_hot_y [[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
0.
   0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  1.  0.  0.
   0.  0.  0.  0.  0.  0.  0.]] 1.0
logits [[-16.75502205 -15.79017544 -25.23309898 -25.81983948 -19.94819641
  -19.27290726 -35.70846176 -26.44281578 -36.66640091  -0.26576775
  -22.52996445  -6.17677975  -3.21124458  -3.4057169  -12.14351845
   -6.20256662 -31.00487518  -9.44850159  -3.2102077  -18.39429092
   -5.46301317 -22.92830086 -12.90356922 -24.2758007  -28.85026169
    4.62926102   5.08789253 -30.25001144  -6.63922548 -12.82178211
  -14.26066113 -21.69800377 -20.22518921  36.89136505 -18.84747887
    4.64067745 -19.61193466  -1.85418999  -8.45915794  -2.05593181
  -15.34217358 -31.80074883 -39.96872711]] 36.8914
[[array([33])], [array([33])]]
tf_argmax:logits [array([33])]
tf_argmax:one_hot [array([33])]
correct_prediction [ True]
INFO:tensorflow:Restoring parameters from ./lenet
lbl [9]
[ 0.]
one_hot_y [[ 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.  0.  0.  0.  0.  0.  0.
   0.  0.  0.  0.  0.  0.  0.]] 1.0
logits [[-46.54110718 -24.40308952 -17.64025116  -8.29677582 -42.32734299
  -12.22011185 -34.45381927  -0.49815235 -11.25852966  22.56115913
    5.60808086  -8.32809639   2.13262272  -8.77393818 -27.76185036
   -8.04238892   2.62761545  -8.70656681 -22.91741371 -10.37016582
   -5.52493286 -32.59768295 -38.91967773  -5.93315315 -29.33203888
   -8.86202717 -16.24197388 -15.73644638   0.18236847 -13.23074436
  -14.84176254 -25.54574585 -15.70024014 -31.39100838 -15.14202023
   -2.55644822 -14.23315239 -27.53111267  -9.5635004  -25.76422882
   -8.74445057  -2.1864028  -14.1462307 ]] 22.5612
[[array([9])], [array([9])]]
tf_argmax:logits [array([9])]
tf_argmax:one_hot [array([9])]
correct_prediction [ True]
INFO:tensorflow:Restoring parameters from ./lenet
lbl [3]
[ 6.93992615]
one_hot_y [[ 0.  0.  0.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
0.
   0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
   0.  0.  0.  0.  0.  0.  0.]] 1.0
logits [[-49.09803772 -16.5016346  -19.31845093   0.47826594 -52.91858673
    7.41720629 -39.97531509  -8.09502316 -20.74891281 -14.33451462
   -3.63088822 -25.47299957 -13.31523132  -6.76805973 -57.82450485
  -26.57240868 -29.76734543 -37.91354752 -19.15711212 -24.79163361
  -10.42099953 -27.60579491 -57.6820755  -27.21976662 -57.03796387
  -17.00332069 -30.34791756 -35.22838593 -22.13203621 -29.50185013
  -45.21042633 -20.13088226 -29.40320778 -13.95530033 -50.99637222
  -23.85913086 -36.4317131  -55.84672928 -10.97224236 -49.19575119
  -53.93302155 -34.97703552 -49.88560486]] 7.41721
[[array([5])], [array([3])]]
tf_argmax:logits [array([5])]
```

```
tf_argmax:one_hot [array([3])]
correct_prediction [False]
```

Step 3: Test a Model on New Images

To give yourself more insight into how your model is working, download at least five pictures of German traffic signs from the web and use your model to predict the traffic sign type.

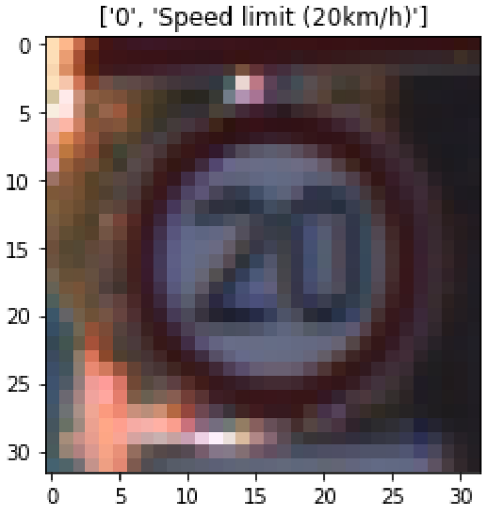
You may find `signnames.csv` useful as it contains mappings from the class id (integer) to the actual sign name.

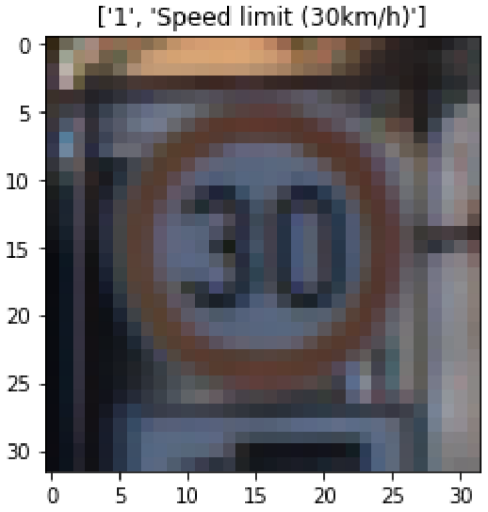
Load and Output the Images

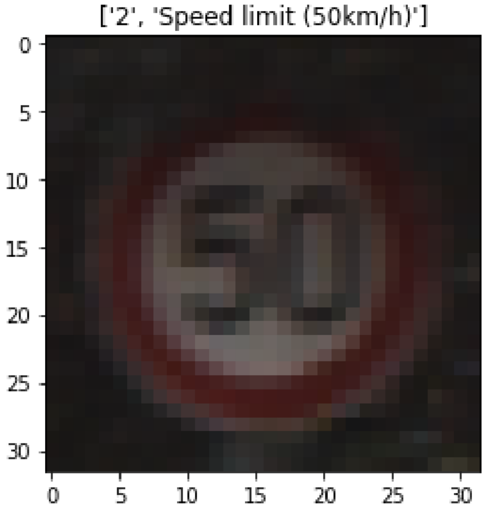
In [29]:

```
########## My Check Code Debug (Optional)##################
import matplotlib.pyplot as plt
%matplotlib inline

global nTotalClasses
n=0;

for i in range(X_test.shape[0]) :
    if(y_test[i]==n):
        plt.figure()
        plt.imshow(X_test[i].squeeze())
        plt.title(xlist[y_test[i]])
        n=n+1
        i=0
    if(n>=nTotalClasses):
        break
```

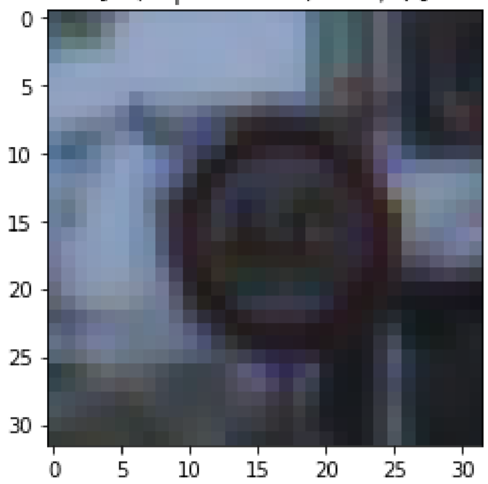
```
/home/uda/.conda/envs/IntroToTensorFlow/lib/python3.6/site-packages/matplotlib/pyp
lot.py:523: RuntimeWarning: More than 20 figures have been opened. Figures created
through the pyplot interface (`matplotlib.pyplot.figure`) are retained until expli
citly closed and may consume too much memory. (To control this warning, see the rc
Param `figure.max_open_warning`).
  max_open_warning, RuntimeWarning)
```



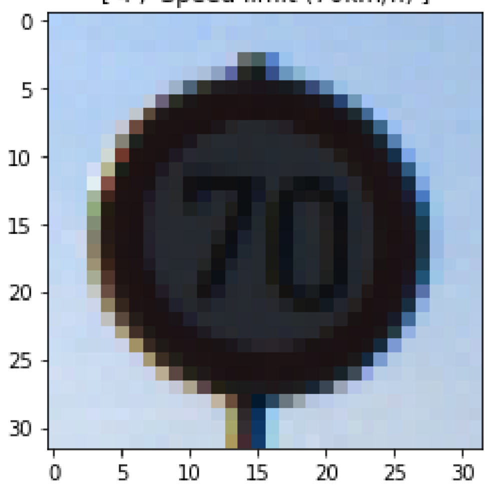




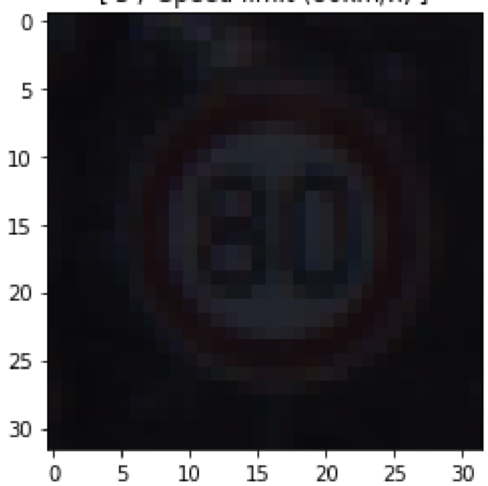
['3', 'Speed limit (60km/h)']



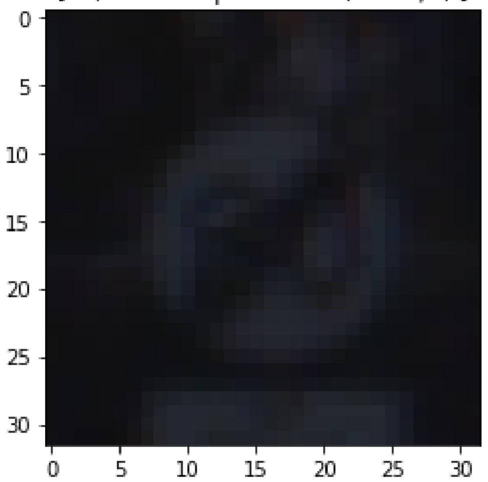
['4', 'Speed limit (70km/h)']



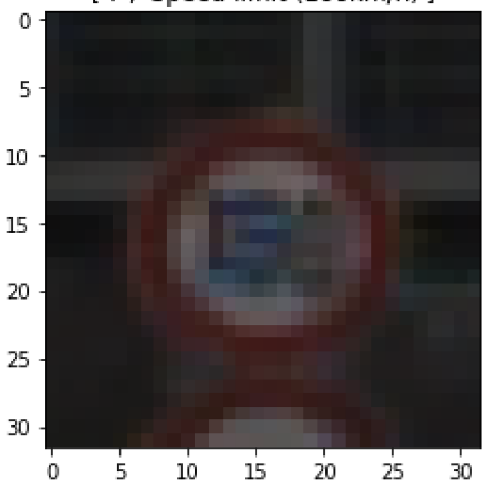
['5', 'Speed limit (80km/h)']



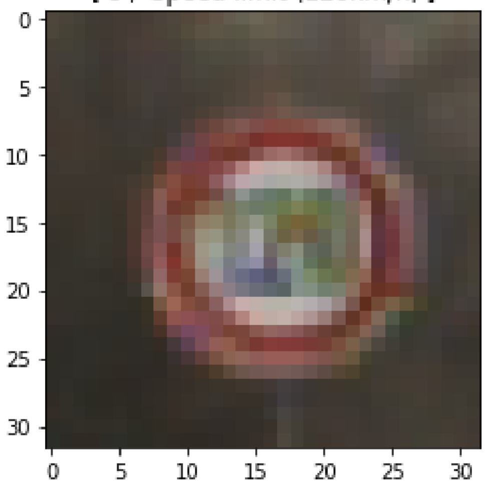
['6', 'End of speed limit (80km/h)']



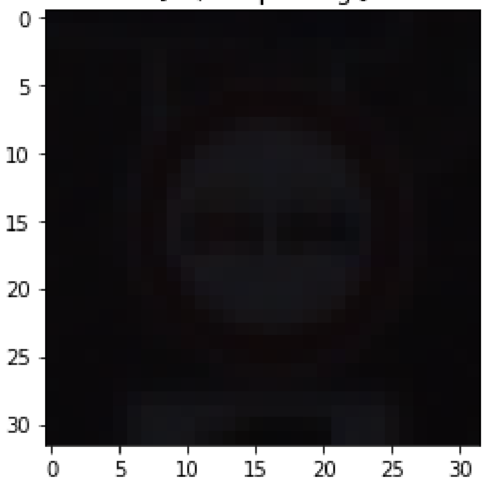
['7', 'Speed limit (100km/h)']



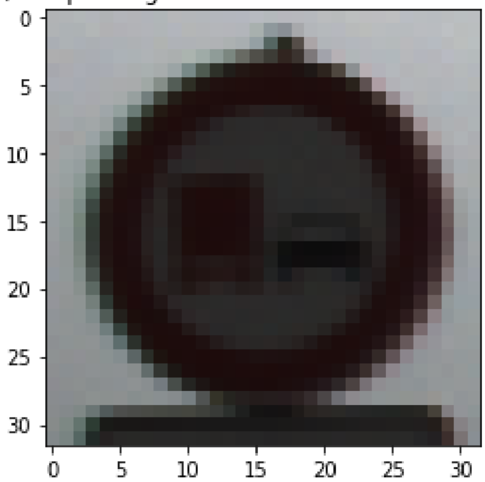
['8', 'Speed limit (120km/h)']



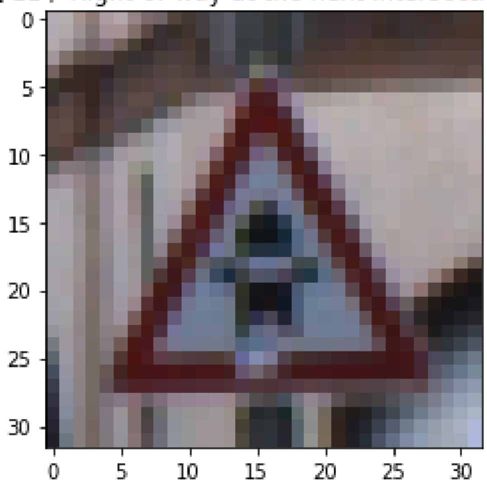
['9', 'No passing']



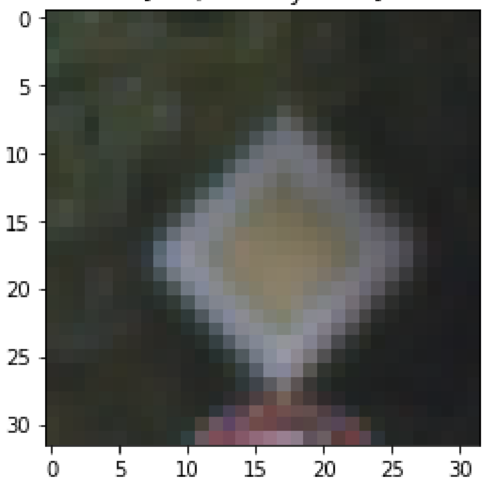
['10', 'No passing for vehicles over 3.5 metric tons']



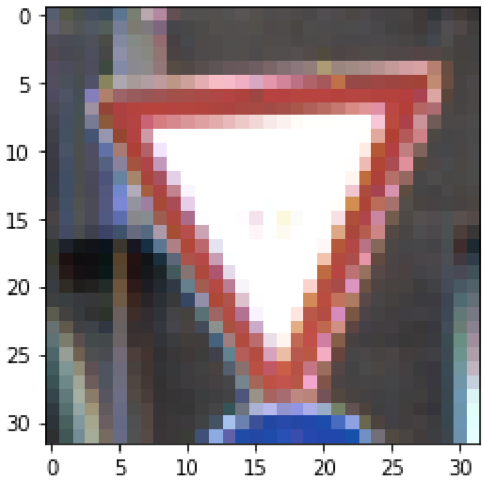
['11', 'Right-of-way at the next intersection']



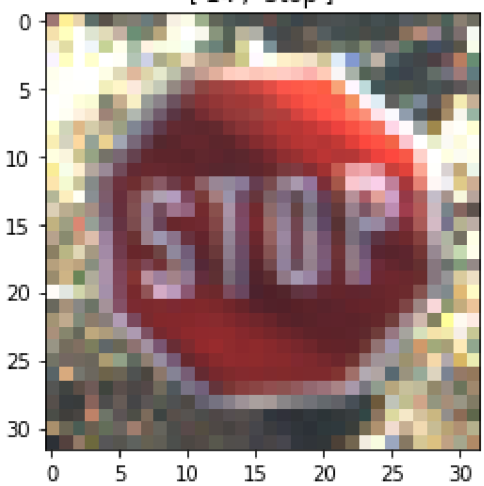
['12', 'Priority road']



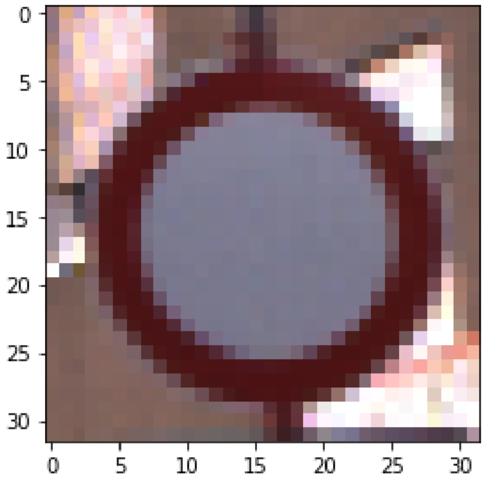
['13', 'Yield']



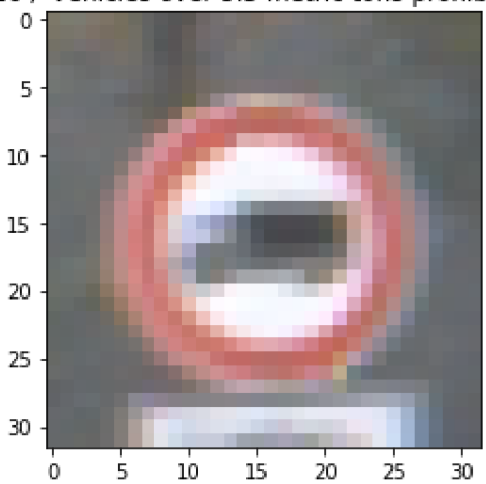
['14', 'Stop']



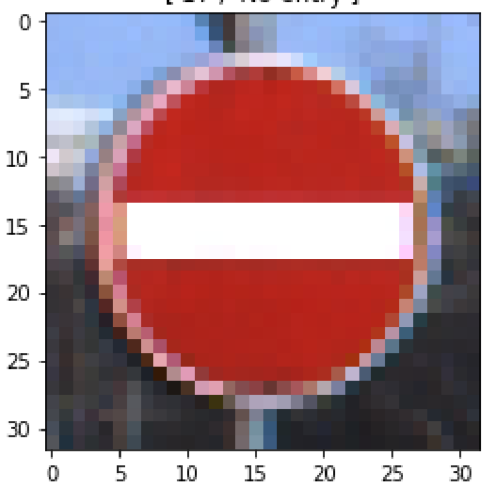
['15', 'No vehicles']



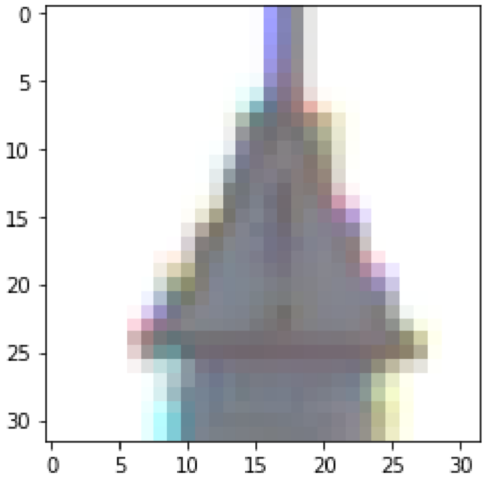
['16', 'Vehicles over 3.5 metric tons prohibited']



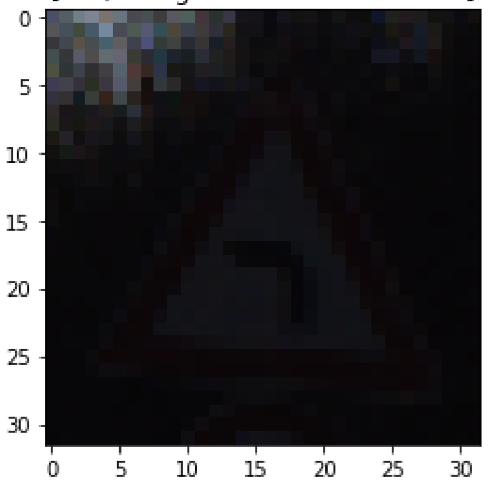
['17', 'No entry']



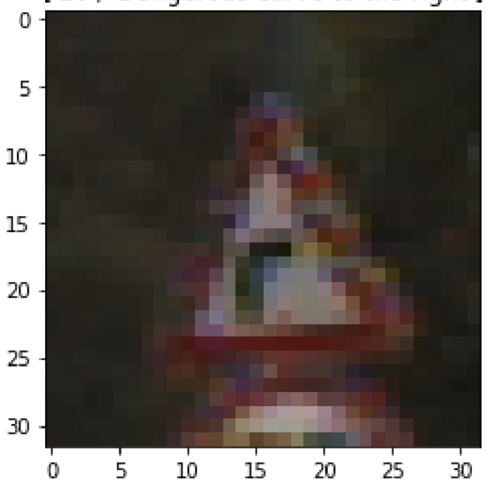
['18', 'General caution']



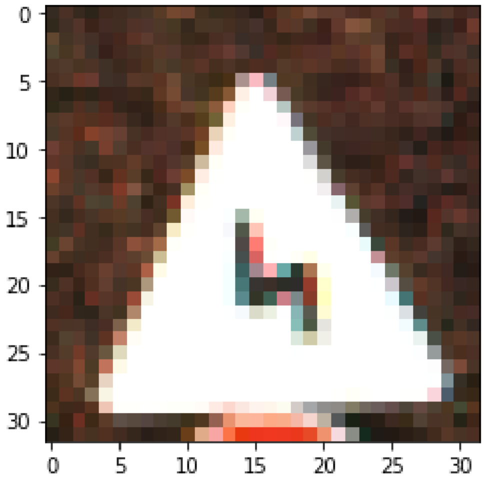
['19', 'Dangerous curve to the left']



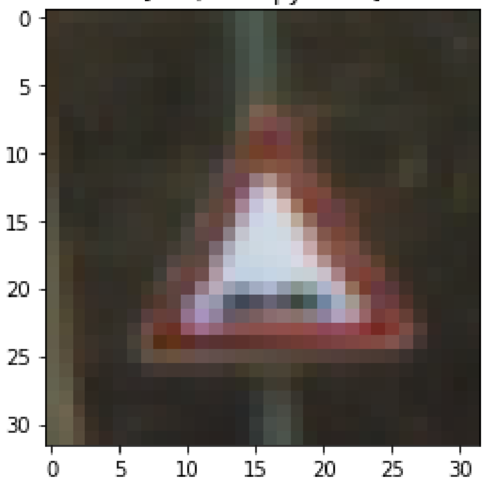
['20', 'Dangerous curve to the right']



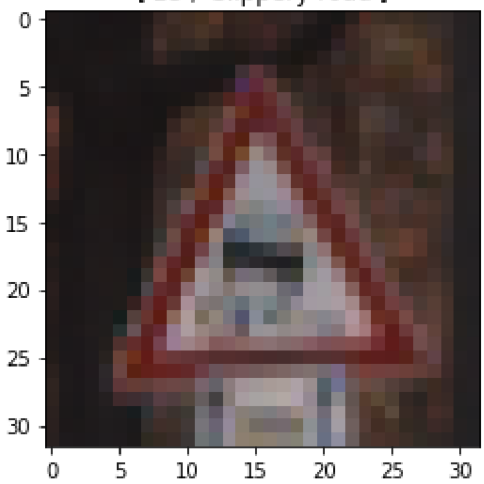
['21', 'Double curve']



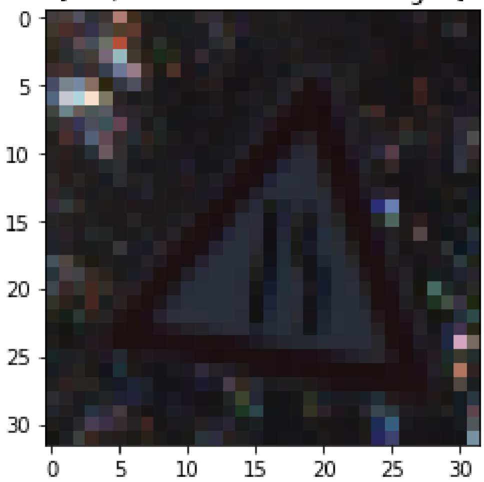
['22', 'Bumpy road']



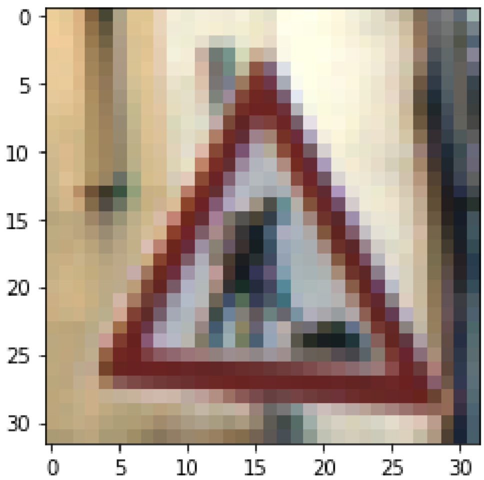
['23', 'Slippery road']



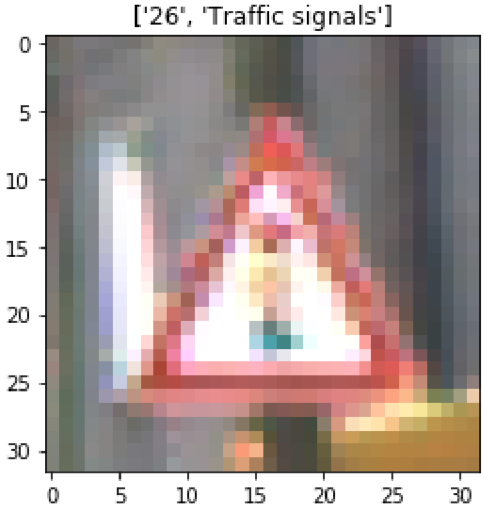
['24', 'Road narrows on the right']



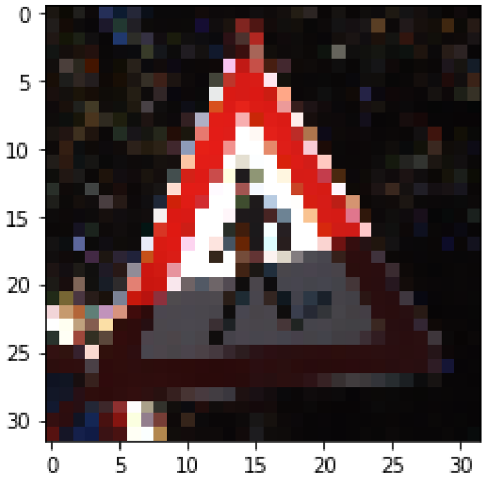
['25', 'Road work']



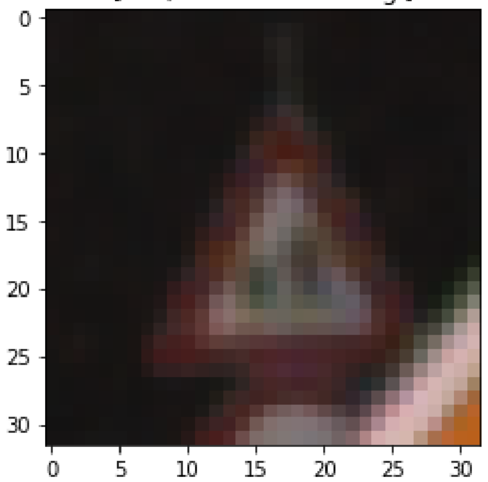
['26', 'Traffic signals']



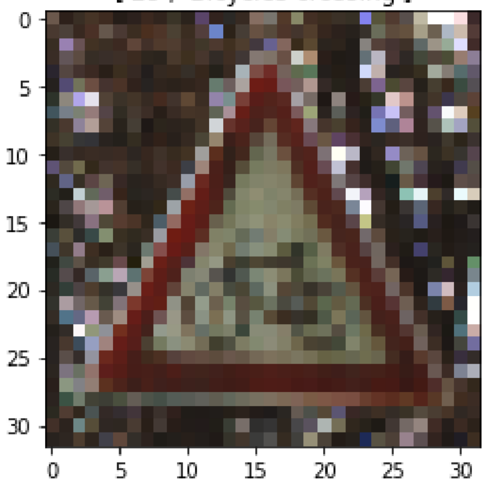
['27', 'Pedestrians']



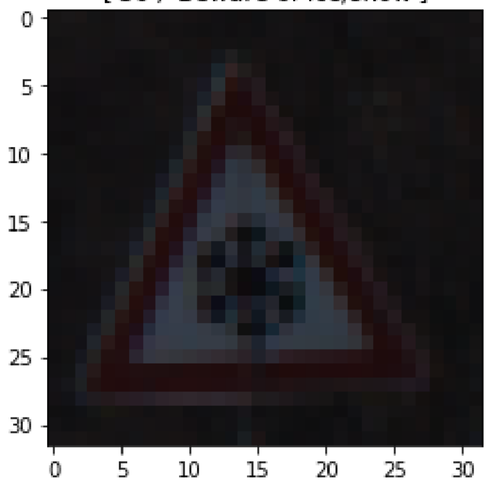
['28', 'Children crossing']



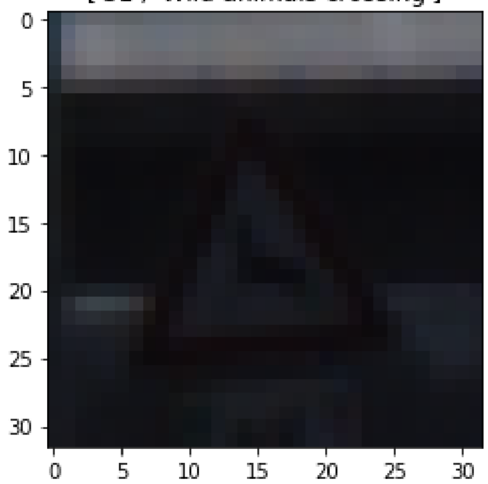
['29', 'Bicycles crossing']



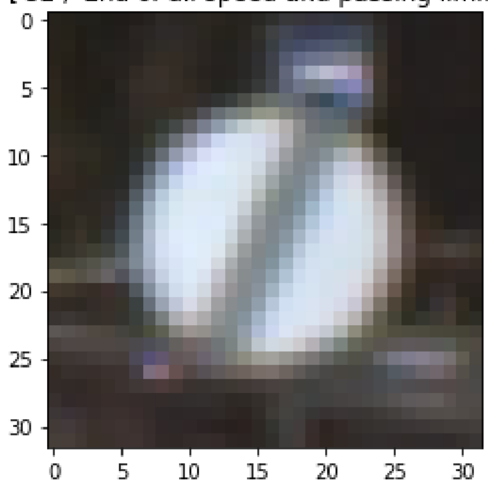
['30', 'Beware of ice/snow']



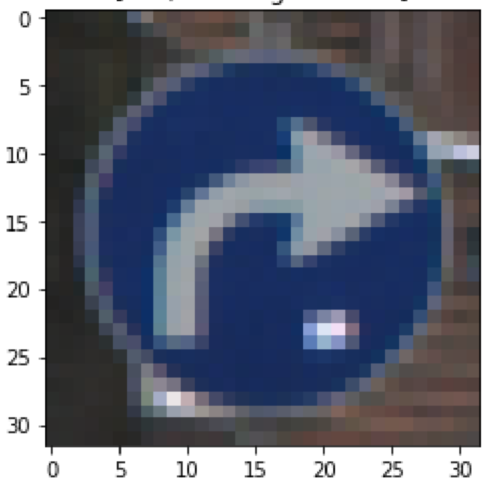
['31', 'Wild animals crossing']



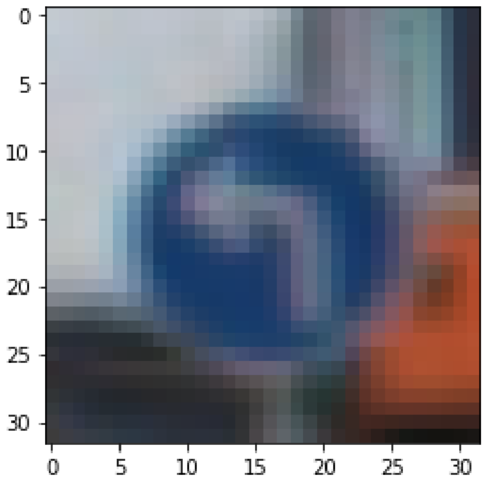
['32', 'End of all speed and passing limits']



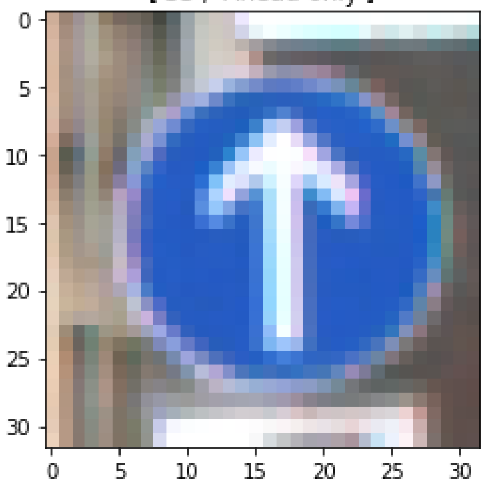
['33', 'Turn right ahead']



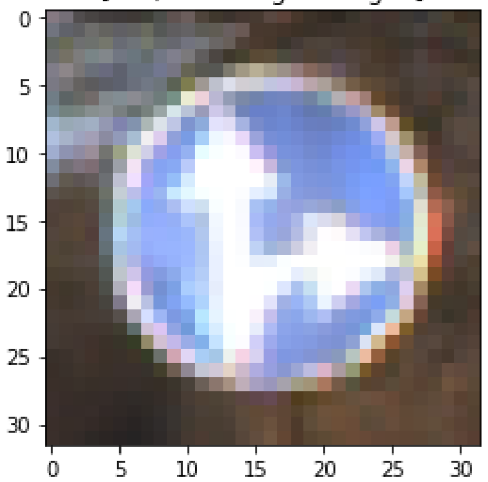
['34', 'Turn left ahead']



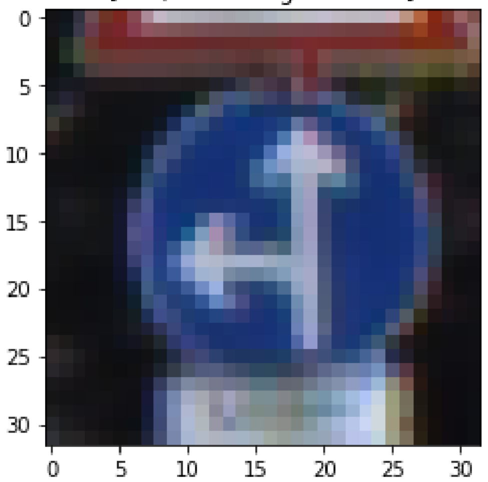
['35', 'Ahead only']



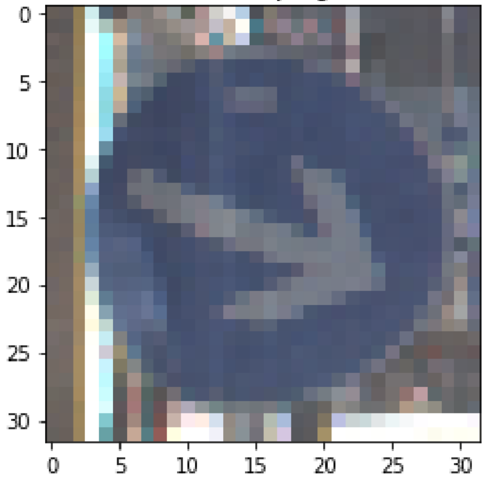
['36', 'Go straight or right']

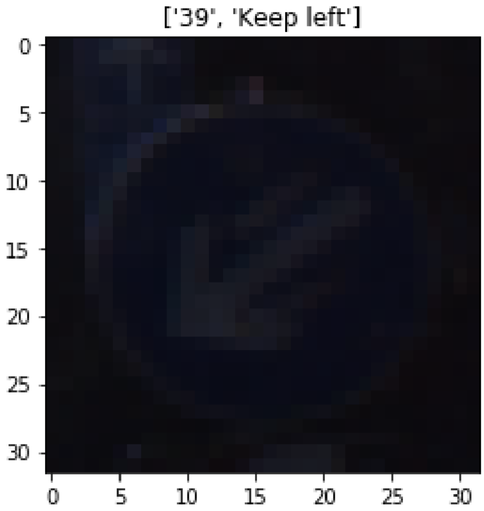


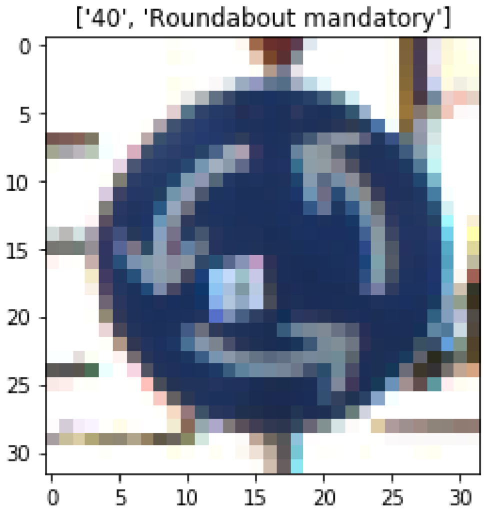
['37', 'Go straight or left']

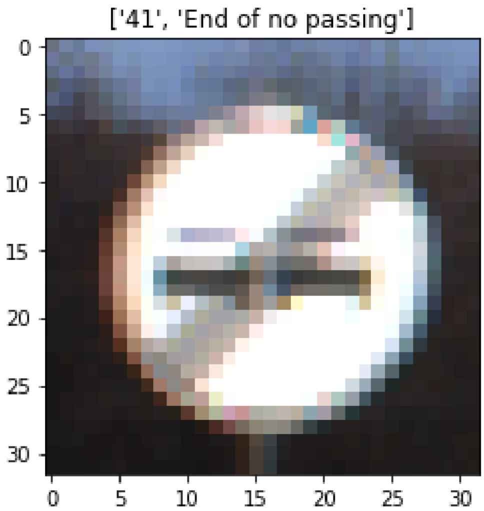


['38', 'Keep right']

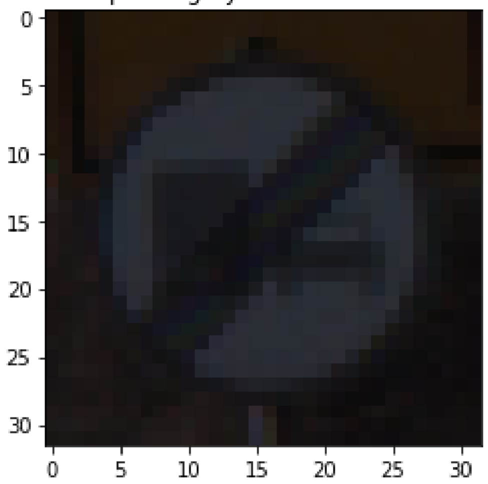



['39', 'Keep left']


['40', 'Roundabout mandatory']


['41', 'End of no passing']

['42', 'End of no passing by vehicles over 3.5 metric tons']



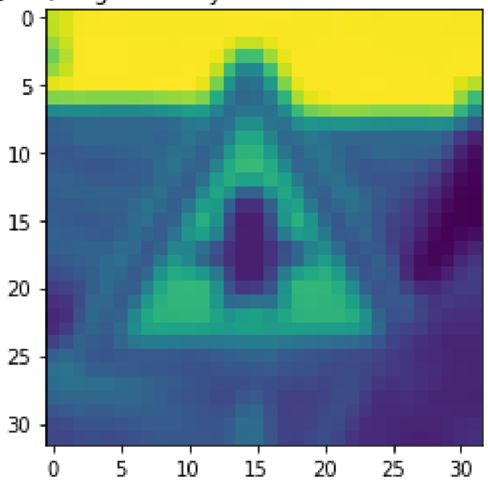
Predict the Sign Type for Each Image

In [30]:

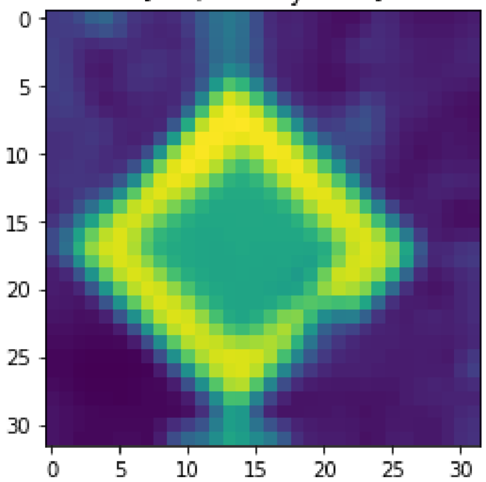
```
########## Function (19) Load my  image to make sign data-set ##########

### Load the images and plot them here.
### Feel free to use as many code cells as needed.
import numpy as np
import cv2
from PIL import Image

img =[]
sign=[]
#161x161.png:(11) Right-of-way at the next intersection
img.append(cv2.imread('./161x161.png', cv2.IMREAD_COLOR))
sign.append(11)

#186x186.png:(12) Priority road
img.append(cv2.imread('./186x186.png', cv2.IMREAD_COLOR))
sign.append(12)

#110x110.png: (2) Speed limit (50km/h)
img.append(cv2.imread('./110x110.png', cv2.IMREAD_COLOR))
sign.append(2)


#80x80.png: (1)Speed limit (30km/h)
img.append(cv2.imread('./80x80.png', cv2.IMREAD_COLOR))
sign.append(1)

#150x150.png: (14)Stop
img.append(cv2.imread('./150x150.png', cv2.IMREAD_COLOR))
sign.append(14)


#348x348.png: (40) Roundabout mandatory
img.append(cv2.imread('./348x348.png', cv2.IMREAD_COLOR))
sign.append(40)


test_image =[]
test_data=[]

for i in range(len(img)):
    test_image.append(cv2.resize(img[i], None, fx = 32/img[i].shape[0], fy = 32/img[i].shape[1
]))
    test_data.append(MyConvertRGB2NRMGRAY(test_image[i],1+2+4))


X_My_data=np.asarray(test_data)
print(type(X_try))
y_My_data=np.asarray(sign)

for i in range(X_My_data.shape[0]):
    plt.figure()
    plt.imshow(X_My_data[i].squeeze())
    plt.title(xlist[y_My_data[i]])
```

```
<class 'numpy.ndarray'>
```

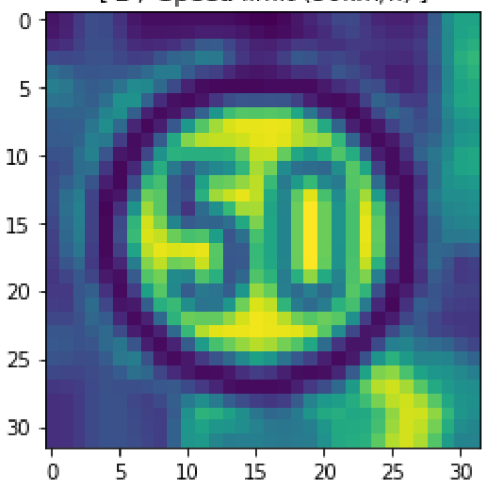
['11', 'Right-of-way at the next intersection']



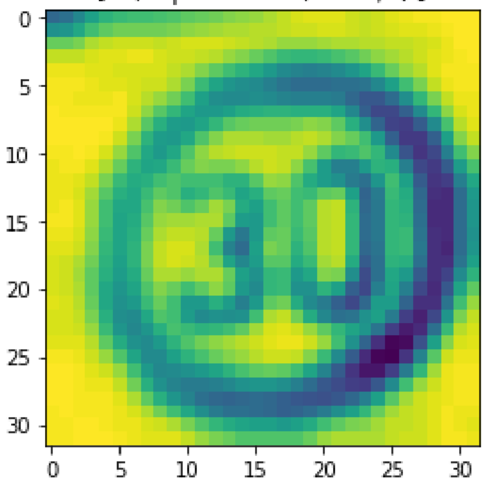
['12', 'Priority road']



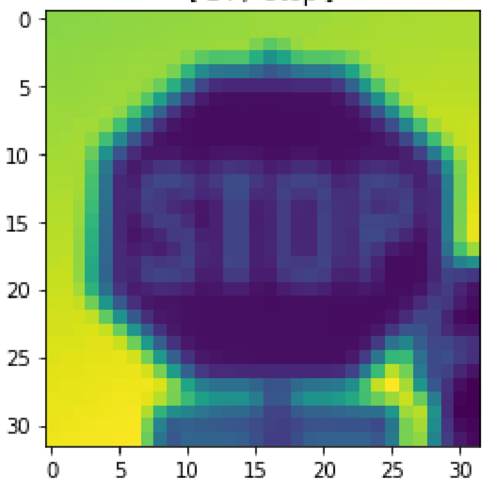
['2', 'Speed limit (50km/h)']



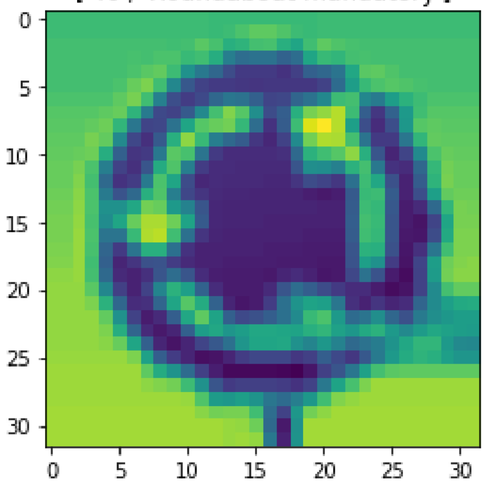
['1', 'Speed limit (30km/h)']



['14', 'Stop']



['40', 'Roundabout mandatory']



In [31]:

```
########## Function (20) Output  predictions  ##########

### Run the predictions here and use the model to output the prediction for each image.
### Make sure to pre-process the images with the same pre-processing pipeline used earlier.
### Feel free to use as many code cells as needed.
def TestPredictions(X_data,Y_data):
    import tensorflow as tf
    saver = tf.train.Saver()

    global nDROP

    with tf.Session() as sess:
    #先に記録した定義モデルをrestoreでファイルからリロードする！
        sess.run(tf.global_variables_initializer())
        saver.restore(sess, './lenet')

        nDROP[0]=0
        for i in range(X_data.shape[0]):
            X_try = np.array( [X_data[i]])
            y_try = np.array( [Y_data[i]])

            n=( evaluatecustom(X_try, y_try))
            print('predictions(',i ,')',n,'¥t',xlist[y_try[0]] )
#################

TestPredictions(X_My_data,y_My_data)
```

```
INFO:tensorflow:Restoring parameters from ./lenet
predictions( 0 ) [array([ True], dtype=bool), 11, 11]    ['11', 'Right-of-way at t
he next intersection']
predictions( 1 ) [array([ True], dtype=bool), 12, 12]    ['12', 'Priority road']
predictions( 2 ) [array([ True], dtype=bool), 2, 2]      ['2', 'Speed limit (50km/
h)']
predictions( 3 ) [array([ True], dtype=bool), 1, 1]      ['1', 'Speed limit (30km/
h)']
predictions( 4 ) [array([ True], dtype=bool), 14, 14]    ['14', 'Stop']
predictions( 5 ) [array([ True], dtype=bool), 40, 40]    ['40', 'Roundabout mandat
ory']
```

Analyze Performance

In [32]:

```
########## Function (21) Output the result of Analyze Performance ##########

### Calculate the accuracy for these 5 new images.
### For example, if the model predicted 1 out of 5 signs correctly, it's 20% accurate on these n
ew images.

def TestSumPredictions(X_data,Y_data):
    import tensorflow as tf
    saver = tf.train.Saver()

    global nDROP

    with tf.Session() as sess:
    #先に記録した定義モデルをrestoreでファイルからリロードする！
        sess.run(tf.global_variables_initializer())
        saver.restore(sess, './lenet')

        nDROP[0]=0
        n= evaluate(X_data, Y_data)
        print('Analyze Performance' ,n)
###############################

TestSumPredictions(X_My_data, y_My_data)
```

```
INFO:tensorflow:Restoring parameters from ./lenet
Analyze Performance 1.0
```

Output Top 5 Softmax Probabilities For Each Image Found on the Web

For each of the new images, print out the model's softmax probabilities to show the **certainty** of the model's predictions (limit the output to the top 5 probabilities for each image). [`tf.nn.top_k`](https://www.tensorflow.org/versions/r0.12/api_docs/python/nn.html#top_k) could prove helpful here.

The example below demonstrates how tf.nn.top_k can be used to find the top k predictions for each image.

`tf.nn.top_k` will return the values and indices (class ids) of the top k predictions. So if k=3, for each sign, it'll return the 3 largest probabilities (out of a possible 43) and the correspoding class ids.

Take this numpy array as an example. The values in the array represent predictions. The array contains softmax probabilities for five candidate images with six possible classes. `tf.nn.top_k` is used to choose the three classes with the highest probability:

```
# (5, 6) array
a = np.array([[ 0.24879643,  0.07032244,  0.12641572,  0.34763842,  0.07893497,
         0.12789202],
       [ 0.28086119,  0.27569815,  0.08594638,  0.0178669 ,  0.18063401,
         0.15899337],
       [ 0.26076848,  0.23664738,  0.08020603,  0.07001922,  0.1134371 ,
         0.23892179],
       [ 0.11943333,  0.29198961,  0.02605103,  0.26234032,  0.1351348 ,
         0.16505091],
       [ 0.09561176,  0.34396535,  0.0643941 ,  0.16240774,  0.24206137,
         0.09155967]])
```

Running it through `sess.run(tf.nn.top_k(tf.constant(a), k=3))` produces:

```
TopKV2(values=array([[ 0.34763842,  0.24879643,  0.12789202],
       [ 0.28086119,  0.27569815,  0.18063401],
       [ 0.26076848,  0.23892179,  0.23664738],
       [ 0.29198961,  0.26234032,  0.16505091],
       [ 0.34396535,  0.24206137,  0.16240774]]), indices=array([[3, 0, 5],
       [0, 1, 4],
       [0, 5, 1],
       [1, 3, 5],
       [1, 4, 3]], dtype=int32))
```

Looking just at the first row we get `[ 0.34763842,  0.24879643,  0.12789202]`, you can confirm these are the 3 largest probabilities in `a`. You'll also notice `[3, 0, 5]` are the corresponding indices.