jupyter Traffic Sign Classifier Last Checkpoint: Last Monday at 11:20 PM (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Preliminary System Preparation

> Import OpenCV and OS
> Check working directory
>
> Ensure NVIDIA GPU is active

In [1]:
```
import cv2
import os
os.getcwd()
```

Out[1]:
```
'/home/ubuntu/CARND1/CarND-Traffic-Sign-Classifier-Project-1'
```

In [2]:
```
!nvidia-smi
```

```
Mon Oct  2 13:06:56 2017
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 375.88                 Driver Version: 375.88                    |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla K80           Off  | 0000:00:1E.0     Off |                    0 |
| N/A   56C    P0    70W / 149W |      0MiB / 11439MiB |     99%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID  Type  Process name                               Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Traffic Sign Classifier Project

This Jupyter Notebook contains information and Python code using the TensorFlow platform to architect and deploy a basic Convolutional Neural Network (CNN) model for the task of classifying German traffic signs.

The training, validation, and test data sets will be drawn from the German Traffic Sign Dataset.

Load the required data sets and split into train/test

In [3]:
```
# Load pickled data
import pickle

training_file = '/home/ubuntu/CARND1/CarND-Traffic-Sign-Classifier-Project/train.p'
testing_file = '/home/ubuntu/CARND1/CarND-Traffic-Sign-Classifier-Project/test.p'

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

X_train_non_normalized, y_train_non_normalized = train['features'], train['labels']
X_test, y_test = test['features'], test['labels']

print("No of training examples: {}".format(len(X_train_non_normalized)))
print("No of test examples: {}".format(len(X_test)))
```

```
No of training examples: 34799
No of test examples: 12630
```

In [4]:
```
from sklearn.model_selection import train_test_split

X_train_non_normalized, X_validation, y_train_non_normalized, y_validation = train_test_split(X_train_non_normalized, y_train_non

print("Amended Image Shape: {}".format(X_train_non_normalized[0].shape))
```

```
Amended Image Shape: (32, 32, 3)
```

Normalize all images with min/max

In [5]:
```
import cv2
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
matplotlib.rcParams['figure.figsize'] = (20.0, 10.0)

X_train = []
y_train = []
all_images = dict()
images = dict()
for i, (image, label) in enumerate(zip(X_train_non_normalized, y_train_non_normalized)):
    if label not in all_images:
        all_images[label] = []
    #zeros = np.zeros((32,32,3))
    zeros = np.zeros((X_train_non_normalized[0].shape))
    norm_image = cv2.normalize(image, zeros, alpha=0, beta=1, norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_32F)
    X_train.append(norm_image)
    y_train.append(label)
    images[label] = norm_image
    all_images[label].append(norm_image)
```

Summarize data sets

> High-level aggregate view

The data sets are available at German Traffic Sign Dataset. It consists of 43 categories of real traffic signs in use in Germany.

In this section, we will review the types of traffic signs that will form the basis of our predictive modelling using CNNs.

Visualize the data sets to identify patterns and outliers

> The visualization is tentative and exploratory by nature.

The images need to be visually diplayed to provide an idea of its type, content, and format. The training data set will be used to do achieve this.

1. Create a dictionary of labels and one of normalized images allowing for the visualization of labels.
2. Create another dictionary of labels and an array of normalized images for a sample to be visualized and its key differences.

The freqquency with which the each label appears in the data set is important to the data preprocessing and modelling steps. The following table lists the labels in order of frequebcy.

```
[(37, 161), (0, 169), (19, 176), (41, 186), (27, 193), (42, 195), (32, 201), (24, 203), (29, 212), (39, 235), (21, 261),
 (20, 286), (40, 289), (22, 313), (36, 317), (16, 329), (6, 332), (34, 348), (30, 349), (23, 411), (28, 415), (26, 467),
 (15, 513), (33, 576), (14, 614), (31, 623), (17, 884), (18, 982), (35, 983), (11, 1040), (3, 1134), (7, 1143), (8, 114
7), (25, 1181), (9, 1189), (5, 1485), (4, 1578), (10, 1619), (38, 1627), (12, 1691), (13, 1730), (1, 1781), (2, 1799)]
```

Thus it is apparent that label no. 2 is the most common and label no. 37 the least common. The data set have multiple occurrences of the same images with different intensity of brightness, zoom, and blur.

As part of the preprocessing, the images are shuffled to remove any unintended bias inherent in the order in which the images are listed.

Preprocess the data sets

> Details of the preprocessing techniques employed are shown below. Rather simple techniques such as normalization and shuffling have been used and proved sufficient to achieve near-optimal results.

20% of the dataset has been put aside for validation and will be seen by the model only after training and testing has been finalised.

Normalize to zero mean

Normalization to zero mean and a small standard deviation will assist the model to do its job more efficiently. This is achieved by deducting 128 and then dividing the result by 128 for each channel of the image using OpenCV.

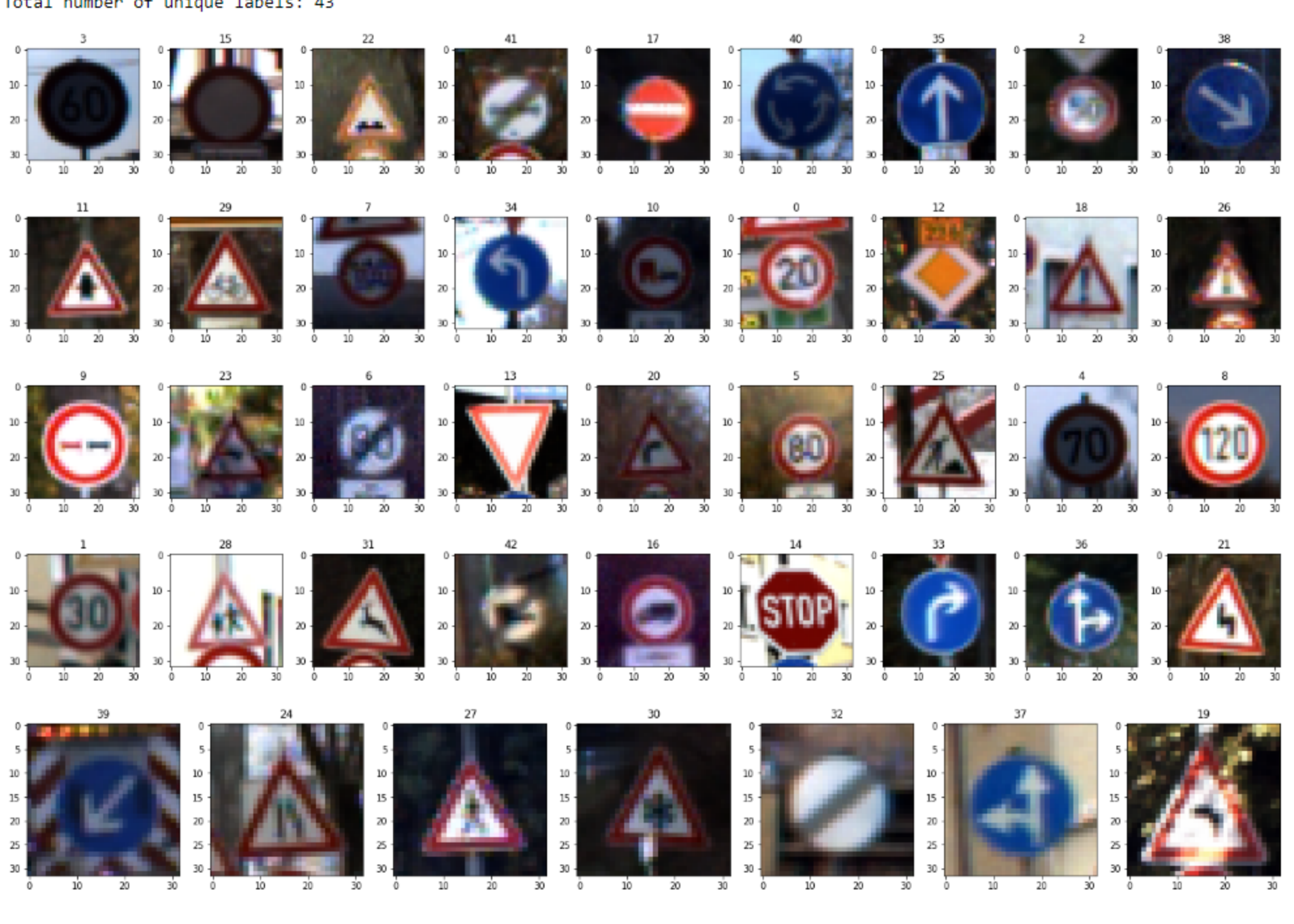
`cv2.normalize(image, zeros, alpha=0, beta=1, norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_32F)`

This normalizes the images for the optimizer.

Display the images

In [6]:
```
mini_batch_size = 9
images_labels = list(images.keys())
print("Total number of unique labels: {}".format(len(images_labels)))
for start in range(0, len(images_labels), mini_batch_size):
    end = start+mini_batch_size
    end = end if end < len(images_labels) else len(images_labels)
    mini_batch_size = end - start
    fig, axes = plt.subplots(1,mini_batch_size)
    for i, (label, ax) in enumerate(zip(images_labels[start:end], axes)):
        ax.set_title(label)
        ax.imshow(images[label].squeeze())

    plt.tight_layout()
    plt.subplots_adjust(top=0.85)
    plt.show()

images.clear()
```

```
Total number of unique labels: 43
```



Sort images in descending order of frequency of appearance in data set

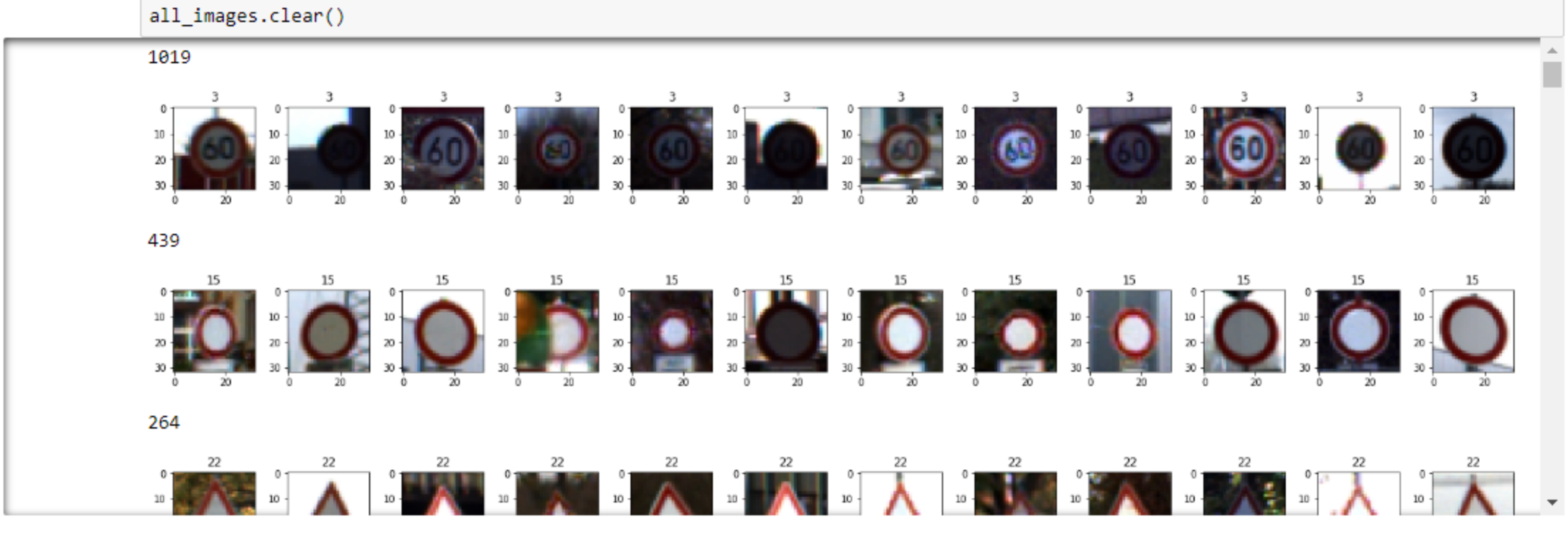
In [7]:
```
images_labels = list(all_images.keys())
images_labels = [(key, len(all_images[key])) for key in images_labels]
images_labels = sorted(images_labels, key=lambda x: x[1])
print(images_labels)
```

```
[(19, 141), (0, 142), (37, 143), (42, 160), (27, 162), (41, 174), (32, 175), (24, 194), (29, 197), (21, 210), (39, 215), (20, 2
31), (40, 238), (36, 262), (22, 264), (16, 282), (34, 304), (6, 307), (30, 323), (23, 352), (28, 382), (26, 435), (15, 439), (3
3, 483), (14, 544), (31, 556), (17, 799), (18, 861), (35, 869), (11, 948), (8, 996), (3, 1019), (7, 1026), (25, 1059), (9, 107
2), (5, 1323), (4, 1377), (10, 1456), (38, 1496), (12, 1507), (13, 1515), (1, 1594), (2, 1607)]
```

Display images after sorting by descending frequency

In [8]:
```
images_labels = list(all_images.keys())
for image_key in images_labels:
    subplots = 12
    fig, axes = plt.subplots(1,subplots)
    images_total = len(all_images[image_key])
    print(images_total)
    offset = images_total // subplots
    images = [all_images[image_key][j*offset] for j in range(subplots)]
    for i, (image, ax) in enumerate(zip(images, axes)):
        ax.set_title(image_key)
        ax.imshow(image.squeeze())
    plt.tight_layout()
    plt.subplots_adjust(top=0.85)
    plt.show()

all_images.clear()
```

```
1019
```



Shuffle images to remove inherent bias

In [9]:
```
from sklearn.utils import shuffle

X_train, y_train = shuffle(X_train, y_train)
```

Model Architecture

> Details of the characteristics and qualities of the architecture, such as the type of model used, the number of layers, the size of each layer are provided below.

The LeNet framework appears to deliver surprising good results on this data set, achieving an accuracy of > 98% on the validation set after only 100 epochs.

Perhaps the LeNet's good fit for the task of classifying images arises from the fact that the framework was first used in classifying handwritten digits also contained in images with 32 by 32 pixels. However, in this case no normalization to grayscale has been undertaken.

LeNet-5 architecture

The LeNet architecture contains 5 layers as detailed below. ReLUs are used for activations and max pooling is employed. Drop-outs are added to limit overfitting.

1. Convolution layer (Using 5x5 patch and the 3 filters)
2. Subsample layer (Activation and max pool with a stride of 2x2)
3. Convolution layer (Using a 5x5 patch with 6 filters)
4. Subsample layer (Another activation using a 2x2 stride to reduce the dimensions)
5. Fully connected layer to flatten the features and activation
6. 4th layer fully connected
7. Activation and relu
8. 5th layer, we extract our features and apply softmax to convert them to probabilities that add to one.

Potential enhancements to LeNet framework

LeNet's advantage is its speed of computation and training, but the downside is its potential to overfitting.

The LeNet framework may be improved by using inception techniques to improve the CNN by using multiple patch sizes, 1x1 convolutions to increase the depth of the image, as well as pooling techniques. Running such a model on the traffic signs data sets may need deep GPU computaional resources.

Dataset augmentation and training

> Data augmentation considered but deemed unecessary.

The generation of new data by rotating the images and multiple brightness levels did not help and was discarded.

Selected solution

> Why the LeNet framework was chosen for the task at hand.

The LeNet framework appeared to be fit-for-purpose for the problem at hand of traffic signs classification.and provided surprisingly good results out of the box with little or no additional tweaks required.

The framework led to optimization within only 50 epochs and the model trained with computational efficiency.

Build the pipeline

In [10]:
```
import tensorflow as tf
from tensorflow.contrib.layers import flatten

def LeNet(x):
    # Hyperparameters
    mu = 0
    sigma = 0.1

    # SOLUTION: Layer 1: Convolutional. Input = 32x32x1. Output = 28x28x6.
    conv1_W = tf.Variable(tf.truncated_normal(shape=(5, 5, 3, 6), mean = mu, stddev = sigma))
    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.
    conv1 = tf.nn.relu(conv1)

    # SOLUTION: Pooling. Input = 28x28x6. Output = 14x14x6.
    conv1 = tf.nn.max_pool(conv1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='VALID')

    # SOLUTION: Layer 2: Convolutional. Output = 10x10x16.
    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)

    # 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.
    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 and dropout.
    fc1    = tf.nn.relu(fc1)
    fc1  = tf.nn.dropout(fc1, keep_prob)

    # 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 and dropout
    fc2    = tf.nn.relu(fc2)
    fc2    = tf.nn.dropout(fc2, keep_prob)

    # SOLUTION: Layer 5: Fully Connected. Input = 84. Output = 43.
    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 [11]:
```
x = tf.placeholder(tf.float32, (None, 32, 32, 3))
y = tf.placeholder(tf.int32, (None))
keep_prob = tf.placeholder(tf.float32)
one_hot_y = tf.one_hot(y, 43)

rate = 0.001
EPOCHS = 100
BATCH_SIZE = 128

logits = LeNet(x)
cross_entropy = tf.nn.softmax_cross_entropy_with_logits(logits=logits, labels=one_hot_y)
loss_operation = tf.reduce_mean(cross_entropy)
optimizer = tf.train.AdamOptimizer(learning_rate = rate)
training_operation = optimizer.minimize(loss_operation)

correct_prediction = tf.equal(tf.argmax(logits, 1), tf.argmax(one_hot_y, 1))
accuracy_operation = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
saver = tf.train.Saver()

def evaluate(X_data, y_data):
    num_examples = len(X_data)
    total_accuracy = 0
    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, keep_prob: 1.0})
        total_accuracy += (accuracy * len(batch_x))
    return total_accuracy / num_examples
```

In [12]:
```
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    num_examples = len(X_train)

    print("Model training...")
    print()
    for i in range(EPOCHS):
        X_train, y_train = shuffle(X_train, y_train)
        for offset in range(0, num_examples, BATCH_SIZE):
            end = offset + BATCH_SIZE
            batch_x, batch_y = X_train[offset:end], y_train[offset:end]
            sess.run(training_operation, feed_dict={x: batch_x, y: batch_y, keep_prob: 0.5})

        validation_accuracy = evaluate(X_validation, y_validation)
        if i%10 == 0:
            print("EPOCH No. {} ...".format(i+1))
            print("Validation Accuracy = {:.3f}".format(validation_accuracy))
            print()

    saver.save(sess, 'model')
    print("Model saved")
```

```
Model training...

EPOCH No. 1 ...
Validation Accuracy = 0.613

EPOCH No. 11 ...
Validation Accuracy = 0.966

EPOCH No. 21 ...
Validation Accuracy = 0.965

EPOCH No. 31 ...
Validation Accuracy = 0.979

EPOCH No. 41 ...
Validation Accuracy = 0.986

EPOCH No. 51 ...
Validation Accuracy = 0.986

EPOCH No. 61 ...
Validation Accuracy = 0.984

EPOCH No. 71 ...
Validation Accuracy = 0.982

EPOCH No. 81 ...
Validation Accuracy = 0.986

EPOCH No. 91 ...
Validation Accuracy = 0.983

Model saved
```

In [13]:
```
def test():
    with tf.Session() as sess:
        saver.restore(sess, tf.train.latest_checkpoint('.'))

        test_accuracy = evaluate(X_test, y_test)
        print("Test Accuracy = {:.3f}".format(test_accuracy))
```

In [14]:
```
test()
```

```
INFO:tensorflow:Restoring parameters from ./model
Test Accuracy = 0.948
```

Acquire new images from the web

> Five more images depicting German traffic signs were selected from the internet.

The images were selected based on their variety in terms of clarity, brightness, and focus.

Performance on new images

> The model performed surprisingly well on the new images, scoring a 100% success rate at correct identification.

The probalibities returned by the model on the new images are 100% and this may call for further investigation tot understand if the results are biased.

Visualize model softmax probabilities

> The softmax probabilities of the predictions on the captured images are visualized.

The model appears to be very confident of its predictions.

Counted 0 bad predictions

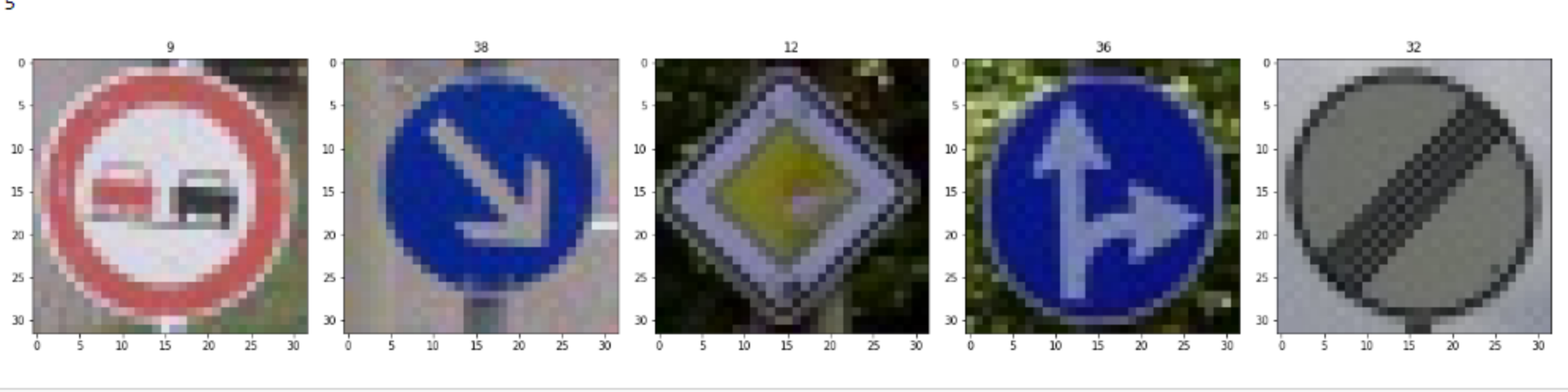
- For Label 9 it predicted 9
- For Label 39 it predicted 39
- For Label 12 it predicted 12
- For Label 36 it predicted 36
- For Label 32 it predicted 32

Conclusion

It would appear that the LeNet framework was very well suited to the problem that was presented. It delivers above-average results in a short time. However, the jury is still out on whether the model will work equally well on other large data sets, in particular, whether overfitting will be a serious issue. If that indeed transpires to be the case, additional tweaks such as data augmentation and the use of inception models may need to be considered.

In [15]:
```
import os

dir = './WebImages'
local_files = os.listdir(dir)

local_labels = [i.partition("-")[0] for i in local_files]
files = [dir + '/' + f for f in local_files]
local_images = [cv2.imread(f, 3) for f in files]
for i in range(len(local_images)):
    b,g,r = cv2.split(local_images[i])
    local_images[i] = cv2.merge([r,g,b])

print(local_images[0].shape)
print(len(local_images))

mini_batch_size = 9
for start in range(0, len(local_images), mini_batch_size):
    end = start+mini_batch_size
    end = end if end < len(local_images) else len(local_images)
    mini_batch_size = end - start
    fig, axes = plt.subplots(1,mini_batch_size)
    end = start+mini_batch_size
    for i, ((image, ax), label) in enumerate(zip(zip(local_images[start:end], axes), local_labels[start:end])):
        ax.set_title(label)
        ax.imshow(image.squeeze())

    plt.tight_layout()
    plt.subplots_adjust(top=0.85)
    plt.show()
```

```
(32, 32, 3)
5
```



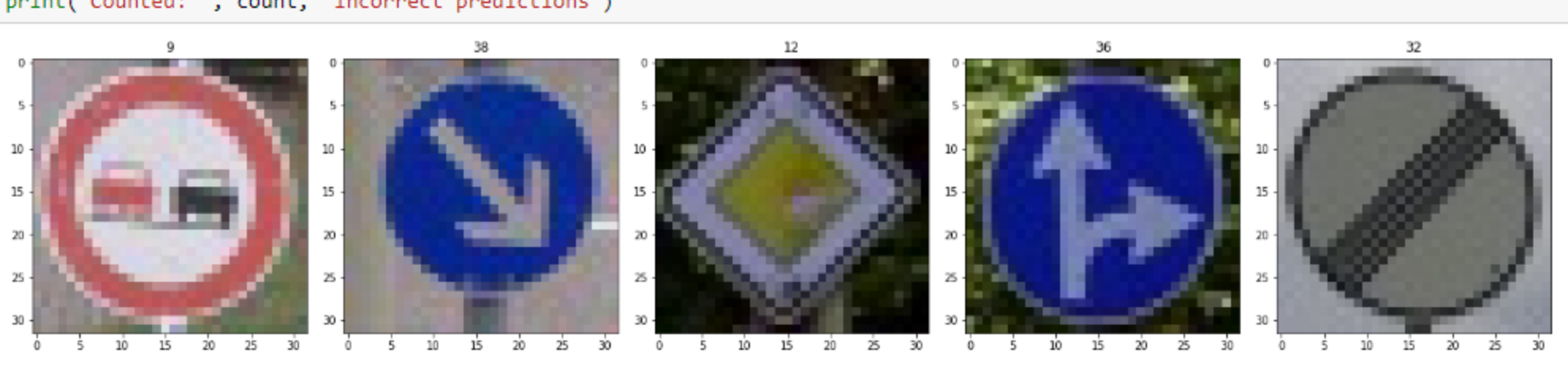
In [16]:
```
with tf.Session() as sess:
    saver.restore(sess, tf.train.latest_checkpoint('.'))

    test_accuracy = evaluate(X_test, y_test)
    print("Test Accuracy = {:.3f}".format(test_accuracy))
    private_images_accuracy = evaluate(local_images, local_labels)
    print("New Web Images - Accuracy = {:.3f}".format(private_images_accuracy))

    softmax = tf.nn.softmax(logits)
    probs = sess.run(softmax, feed_dict={x: local_images, y: local_labels, keep_prob: 1})
```

```
INFO:tensorflow:Restoring parameters from ./model
Test Accuracy = 0.948
New Web Images - Accuracy = 1.000
```

Complete list of unique labels for images

In [17]:
```
labels_text = {
    0:"Speed limit (20km/h)",
    1:"Speed limit (30km/h)",
    2:"Speed limit (50km/h)",
    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",
}
```

In [18]:
```
mini_batch_size = 9
for start in range(0, len(local_images), mini_batch_size):
    end = start+mini_batch_size
    end = end if end < len(local_images) else len(local_images)
    mini_batch_size = end - start
    fig, axes = plt.subplots(1,mini_batch_size)
    end = start+mini_batch_size
    for i, ((image, ax), label) in enumerate(zip(zip(local_images[start:end], axes), local_labels[start:end])):
        ax.set_title(label)
        ax.imshow(image.squeeze())

    plt.tight_layout()
    plt.subplots_adjust(top=0.85)
    plt.show()

count = 0
for image_prob in range(len(probs)):
    probabilities = probs[image_prob]
    for i in range(len(probabilities)):
        if probabilities[i] > 0.5:
            print(image_prob, ':', 'Label=', local_labels[image_prob], ',Predicted=', i, labels_text[i], '=', probabilities[i])
            if int(local_labels[image_prob]) != int(i):
                count += 1

print('Counted: ', count, 'incorrect predictions')
```




```
0 : Label= 9 ,Predicted= 9 No passing = 1.0
1 : Label= 38 ,Predicted= 38 Keep right = 1.0
2 : Label= 12 ,Predicted= 12 Priority road = 1.0
3 : Label= 36 ,Predicted= 36 Go straight or right = 1.0
4 : Label= 32 ,Predicted= 32 End of all speed and passing limits = 1.0
Counted:  0 incorrect predictions
```

Print softmax probabilities

In [19]:
```
print(probs[0])
```

```
[ 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.]
```

In [ ]: