In [8]:

```
%load_ext watermark
%watermark -p tensorflow,skimage,matplotlib,numpy,random
```

```
The watermark extension is already loaded. To reload it, use:
  %reload_ext watermark
tensorflow 1.1.0
skimage 0.12.3
matplotlib 2.0.0
numpy 1.12.1
random n□
```

In [9]:

```
import tensorflow as tf
from skimage import transform
from skimage import data
import matplotlib.pyplot as plt
import os
import numpy as np
from skimage.color import rgb2gray
import random
```

TensorFlow Basics

In [10]:

```
# Import `tensorflow`
import tensorflow as tf

# Initialize two constants
x1 = tf.constant([1,2,3,4])
x2 = tf.constant([5,6,7,8])

# Multiply
result = tf.multiply(x1, x2)

# Print the result
print(result)
```

```
Tensor("Mul_3:0", shape=(4,), dtype=int32)
```

In [11]:

```
# Import `tensorflow`
import tensorflow as tf

# Initialize two constants
x1 = tf.constant([1,2,3,4])
x2 = tf.constant([5,6,7,8])

# Multiply
result = tf.multiply(x1, x2)

# Intialize the Session
sess = tf.Session()

# Print the result
print(sess.run(result))

# Close the session
sess.close()
```

```
[ 5 12 21 32]
```

In [12]:

```
# Import `tensorflow`
import tensorflow as tf

# Initialize two constants
x1 = tf.constant([1,2,3,4])
x2 = tf.constant([5,6,7,8])

# Multiply
result = tf.multiply(x1, x2)

# Initialize Session and run `result`
with tf.Session() as sess:
    output = sess.run(result)
    print(output)
```

```
[ 5 12 21 32]
```

Loading And Exploring The Data

In [13]:

```
def load_data(data_dir):
    # Get all subdirectories of data_dir. Each represents a label.
    directories = [d for d in os.listdir(data_dir)
                   if os.path.isdir(os.path.join(data_dir, d))]
    # Loop through the label directories and collect the data in
    # two lists, labels and images.
    labels = []
    images = []
    for d in directories:
        label_dir = os.path.join(data_dir, d)
        file_names = [os.path.join(label_dir, f)
                      for f in os.listdir(label_dir)
                      if f.endswith(".ppm")]
        for f in file_names:
            images.append(data.imread(f))
            labels.append(int(d))
    return images, labels

ROOT_PATH = "/Users/karlijnwillems/Downloads/"
train_data_dir = os.path.join(ROOT_PATH, "TrafficSigns/Training")
test_data_dir = os.path.join(ROOT_PATH, "TrafficSigns/Testing")

images, labels = load_data(train_data_dir)
```

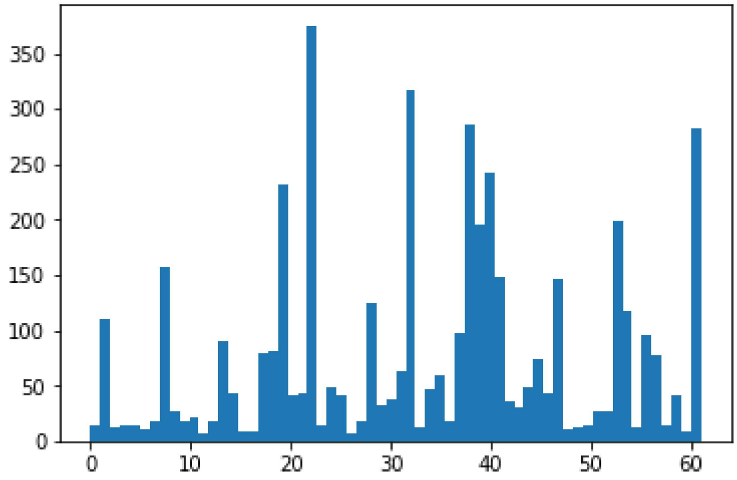
In [14]:

```
images_array = np.array(images)
labels_array = np.array(labels)

# Print the `images` dimensions
print(images_array.ndim)

# Print the number of `images`'s elements
print(images_array.size)

# Print the first instance of `images`
images_array[0]

# Print the `labels` dimensions
print(labels_array.ndim)

# Print the number of `labels`'s elements
print(labels_array.size)

# Count the number of labels
print(len(set(labels_array)))
```

```
1
4575
1
4575
62
```

In [15]:

```
# Import the `pyplot` module
import matplotlib.pyplot as plt

# Make a histogram with 62 bins of the `labels` data
plt.hist(labels, 62)

# Show the plot
plt.show()
```



In [16]:

```
# Import the `pyplot` module of `matplotlib`
import matplotlib.pyplot as plt

# Determine the (random) indexes of the images that you want to see
traffic_signs = [300, 2250, 3650, 4000]

# Fill out the subplots with the random images that you defined
for i in range(len(traffic_signs)):
    plt.subplot(1, 4, i+1)
    plt.axis('off')
    plt.imshow(images[traffic_signs[i]])
    plt.subplots_adjust(wspace=0.5)

plt.show()
```

In [17]:

```
# Import `matplotlib`
import matplotlib.pyplot as plt

# Determine the (random) indexes of the images
traffic_signs = [300, 2250, 3650, 4000]

# Fill out the subplots with the random images and add shape, min and max val
for i in range(len(traffic_signs)):
    plt.subplot(1, 4, i+1)
    plt.axis('off')
    plt.imshow(images[traffic_signs[i]])
    plt.subplots_adjust(wspace=0.5)
    plt.show()
    print("shape: {0}, min: {1}, max: {2}".format(images[traffic_signs[i]].sh
                                                  images[traffic_signs[i]].mi
                                                  images[traffic_signs[i]].ma
```

```
shape: (62, 61, 3), min: 3, max: 160
```

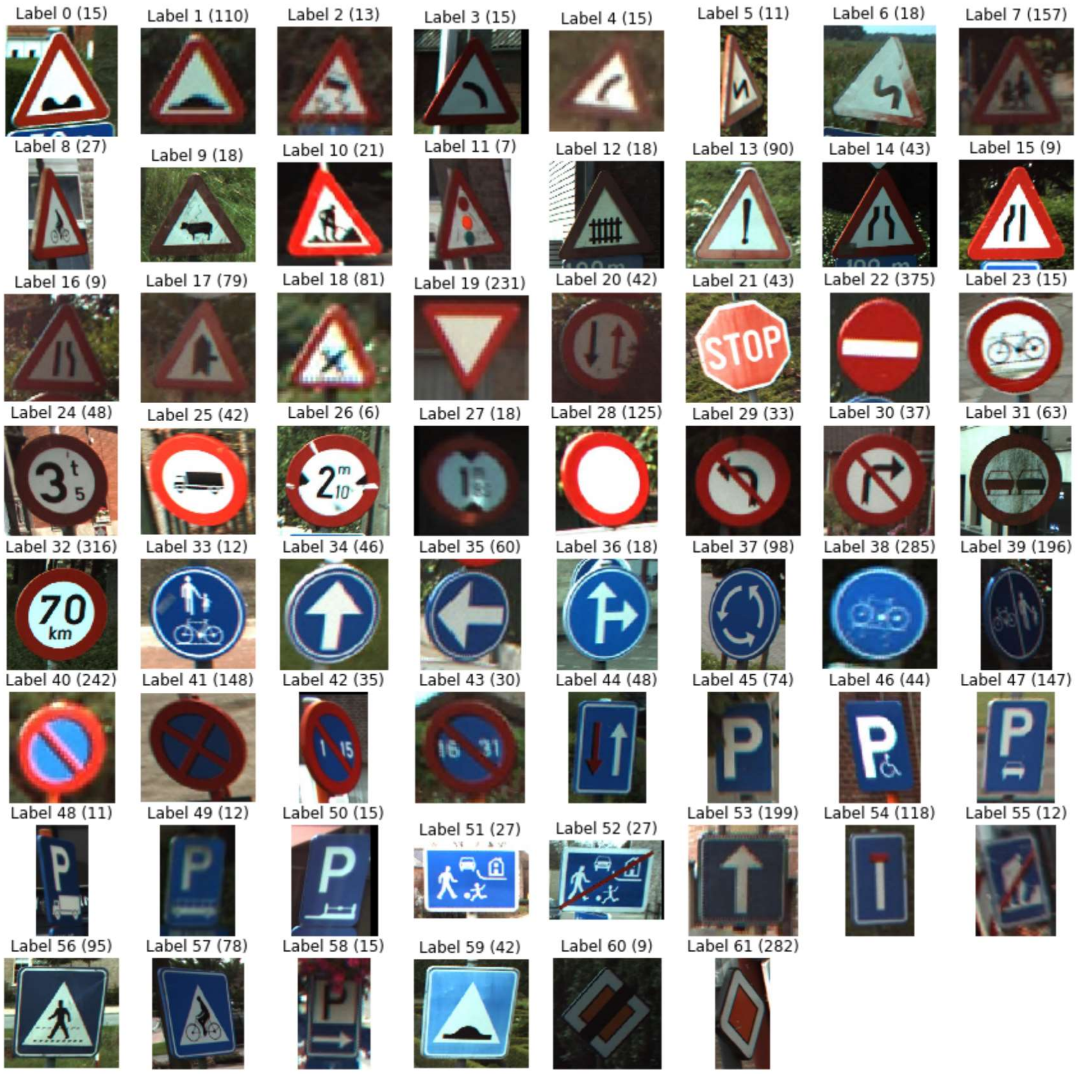
```
shape: (110, 96, 3), min: 3, max: 255
```

```
shape: (379, 153, 3), min: 0, max: 255
```

```
shape: (100, 68, 3), min: 17, max: 255
```

In [18]:

```
# Import the `pyplot` module as `plt`
import matplotlib.pyplot as plt

# Get the unique labels
unique_labels = set(labels)

# Initialize the figure
plt.figure(figsize=(15, 15))

# Set a counter
i = 1

# For each unique label,
for label in unique_labels:
    # You pick the first image for each label
    image = images[labels.index(label)]
    # Define 64 subplots
    plt.subplot(8, 8, i)
    # Don't include axes
    plt.axis('off')
    # Add a title to each subplot
    plt.title("Label {0} ({1})".format(label, labels.count(label)))
    # Add 1 to the counter
    i += 1
    # And you plot this first image
    plt.imshow(image)

# Show the plot
plt.show()
```



Feature Extraction

Rescaling Images

In [19]:

```
# Resize images
images32 = [transform.resize(image, (28, 28)) for image in images]
images32 = np.array(images32)
```

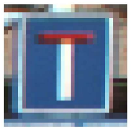
In [20]:

```
# Import `matplotlib`
import matplotlib.pyplot as plt

# Determine the (random) indexes of the images
traffic_signs = [300, 2250, 3650, 4000]

# Fill out the subplots with the random images and add shape, min and max val
for i in range(len(traffic_signs)):
    plt.subplot(1, 4, i+1)
    plt.axis('off')
    plt.imshow(images32[traffic_signs[i]])
    plt.subplots_adjust(wspace=0.5)
    plt.show()
    print("shape: {0}, min: {1}, max: {2}".format(images32[traffic_signs[i]].
                                                   images32[traffic_signs[i]].
                                                   images32[traffic_signs[i]].
```

```
shape: (28, 28, 3), min: 0.061764705882353076, max: 0.6161764705882353
```

```
shape: (28, 28, 3), min: 0.07634053621448501, max: 1.0
```

```
shape: (28, 28, 3), min: 0.08464760904361845, max: 1.0
```



```
shape: (28, 28, 3), min: 0.08907563025210051, max: 1.0
```

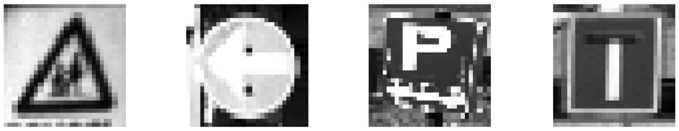
Image Conversion to Grayscale

In [21]:

```
images32 = rgb2gray(np.array(images32))
```

In [22]:

```
for i in range(len(traffic_signs)):
    plt.subplot(1, 4, i+1)
    plt.axis('off')
    plt.imshow(images32[traffic_signs[i]], cmap="gray")
    plt.subplots_adjust(wspace=0.5)

plt.show()

print(images32.shape)
```



```
(4575, 28, 28)
```

Deep Learning with Tensorflow

Modeling The Neural Network

In [23]:

```
x = tf.placeholder(dtype = tf.float32, shape = [None, 28, 28])
y = tf.placeholder(dtype = tf.int32, shape = [None])
images_flat = tf.contrib.layers.flatten(x)
logits = tf.contrib.layers.fully_connected(images_flat, 62, tf.nn.relu)
loss = tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(labels =
train_op = tf.train.AdamOptimizer(learning_rate=0.001).minimize(loss)
correct_pred = tf.argmax(logits, 1)
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))

print("images_flat: ", images_flat)
print("logits: ", logits)
print("loss: ", loss)
print("predicted_labels: ", correct_pred)
```

```
images_flat:  Tensor("Flatten/Reshape:0", shape=(?, 784), dtype=float32)
logits:  Tensor("fully_connected/Relu:0", shape=(?, 62), dtype=float32)
loss:  Tensor("Mean:0", shape=(), dtype=float32)
predicted_labels:  Tensor("ArgMax:0", shape=(?,), dtype=int64)
```

Running The Neural Network

In [24]:

```
sess = tf.Session()

sess.run(tf.global_variables_initializer())

for i in range(201):
        print('EPOCH', i)
        _, accuracy_val = sess.run([train_op, accuracy], feed_dict={x: images
        if i % 10 == 0:
            print("Loss: ", loss)
        print('DONE WITH EPOCH')
```

```
EPOCH 0
Loss:  Tensor("Mean:0", shape=(), dtype=float32)
DONE WITH EPOCH
EPOCH 1
DONE WITH EPOCH
EPOCH 2
DONE WITH EPOCH
EPOCH 3
DONE WITH EPOCH
EPOCH 4
DONE WITH EPOCH
EPOCH 5
DONE WITH EPOCH
EPOCH 6
DONE WITH EPOCH
EPOCH 7
DONE WITH EPOCH
EPOCH 8
DONE WITH EPOCH
```

In [25]:

```
# Alternatively, you can also run the following lines of code instead of the
#with tf.Session() as sess:
#    sess.run(tf.global_variables_initializer())
#    for i in range(201):
#        print('EPOCH', i)
#        _, accuracy_val = sess.run([train_op, accuracy], feed_dict={x: image
#        if i % 10 == 0:
#            print("Loss: ", loss)
#        print('DONE WITH EPOCH')
```

Evaluating The Neural Network

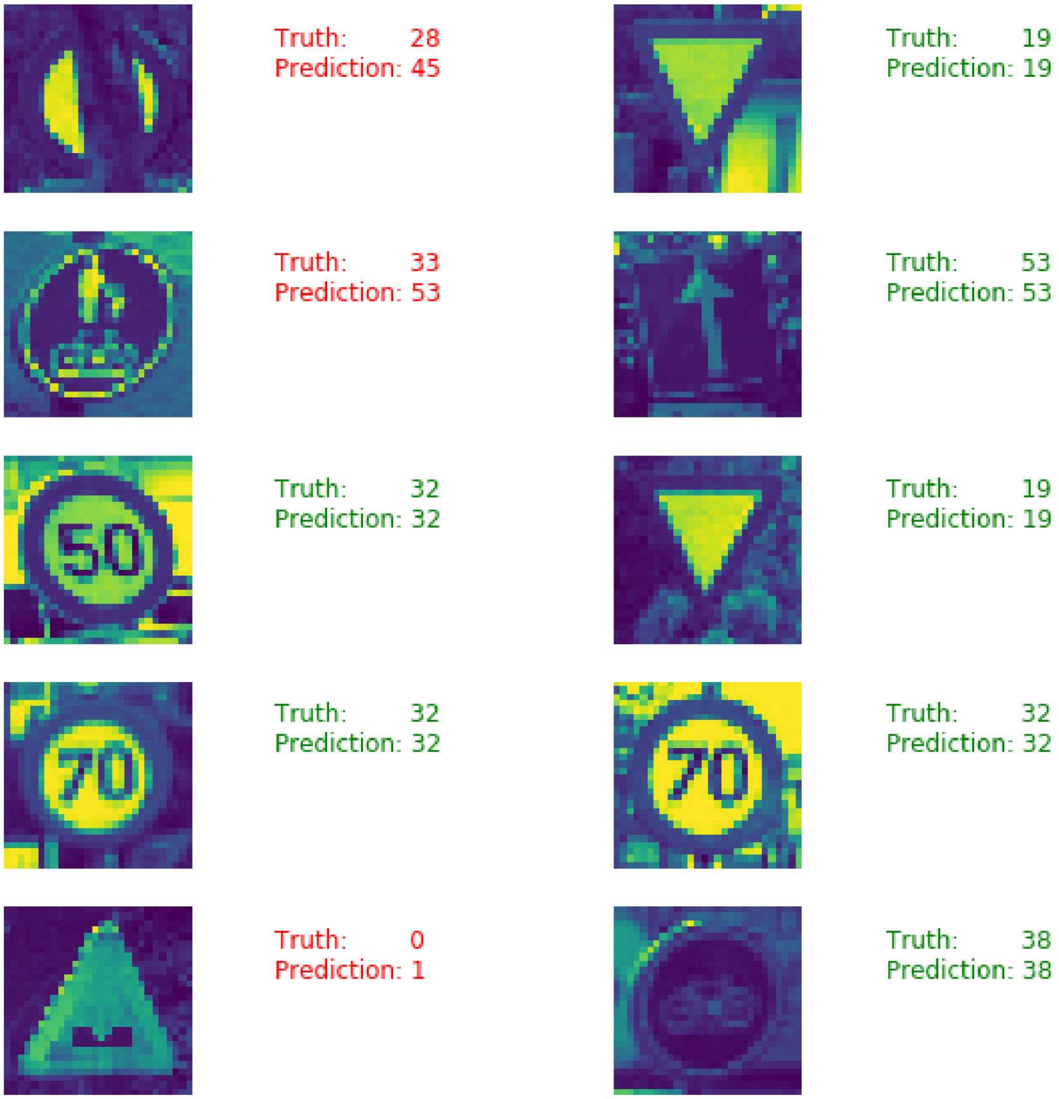
In [26]:

```
# Pick 10 random images
sample_indexes = random.sample(range(len(images32)), 10)
sample_images = [images32[i] for i in sample_indexes]
sample_labels = [labels[i] for i in sample_indexes]

# Run the "predicted_labels" op.
predicted = sess.run([correct_pred], feed_dict={x: sample_images})[0]

# Print the real and predicted labels
print(sample_labels)
print(predicted)
```

```
[28, 19, 33, 53, 32, 19, 32, 32, 0, 38]
[45 19 53 53 32 19 32 32  1 38]
```

In [27]:

```
# Display the predictions and the ground truth visually.
fig = plt.figure(figsize=(10, 10))
for i in range(len(sample_images)):
    truth = sample_labels[i]
    prediction = predicted[i]
    plt.subplot(5, 2,1+i)
    plt.axis('off')
    color='green' if truth == prediction else 'red'
    plt.text(40, 10, "Truth:        {0}\nPrediction: {1}".format(truth, predi
             fontsize=12, color=color)
    plt.imshow(sample_images[i])

plt.show()
```



In [28]:

```
# Load the test data
test_images, test_labels = load_data(test_data_dir)

# Transform the images to 28 by 28 pixels
test_images28 = [transform.resize(image, (28, 28)) for image in test_images]

# Convert to grayscale
from skimage.color import rgb2gray
test_images28 = rgb2gray(np.array(test_images28))

# Run predictions against the full test set.
predicted = sess.run([correct_pred], feed_dict={x: test_images28})[0]

# Calculate correct matches
match_count = sum([int(y == y_) for y, y_ in zip(test_labels, predicted)])

# Calculate the accuracy
accuracy = match_count / len(test_labels)

# Print the accuracy
print("Accuracy: {:.3f}".format(accuracy))
```

```
Accuracy: 0.600
```

In [29]:

```
sess.close()
```