TensorFlow Tutorial For Beginners

This notebook was made in preparation for the DataCamp tutorial "TensorFlow Tutorial for Beginners"; If you want more explanations on the code or on using TensorFlow, go to the full tutorial [here (https://www.datacamp.com/community/tutorials/tensorflow-tutorial)](https://www.datacamp.com/community/tutorials/tensorflow-tutorial).

The full tutorial covers the following topics:

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)
```

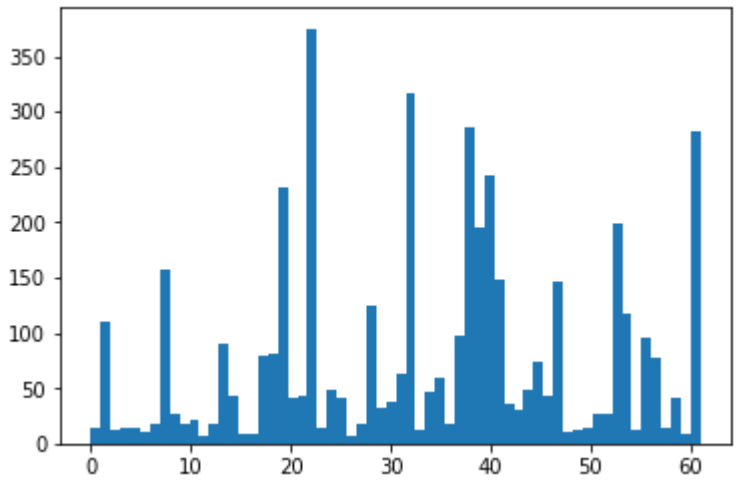
In [14]:

```
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 values
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]].shape,
                                                  images[traffic_signs[i]].min(),
                                                  images[traffic_signs[i]].max()))
```

```
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 values
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]].shape,
                                                  images32[traffic_signs[i]].min(),
                                                  images32[traffic_signs[i]].max()))
```

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 = y, logits
= logits))
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: images32, y: lab
els})
        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
EPOCH 9
DONE WITH EPOCH
EPOCH 10
Loss:  Tensor("Mean:0", shape=(), dtype=float32)
DONE WITH EPOCH
EPOCH 11
DONE WITH EPOCH
EPOCH 12
DONE WITH EPOCH
EPOCH 13
DONE WITH EPOCH
EPOCH 14
DONE WITH EPOCH
EPOCH 15
DONE WITH EPOCH
EPOCH 16
DONE WITH EPOCH
EPOCH 17
DONE WITH EPOCH
EPOCH 18
DONE WITH EPOCH
EPOCH 19
DONE WITH EPOCH
EPOCH 20
Loss:  Tensor("Mean:0", shape=(), dtype=float32)
DONE WITH EPOCH
EPOCH 21
DONE WITH EPOCH
EPOCH 22
DONE WITH EPOCH
EPOCH 23
DONE WITH EPOCH
EPOCH 24
DONE WITH EPOCH
EPOCH 25
DONE WITH EPOCH
EPOCH 26
DONE WITH EPOCH
EPOCH 27
DONE WITH EPOCH
EPOCH 28
DONE WITH EPOCH
```

```
EPOCH 29
DONE WITH EPOCH
EPOCH 30
Loss:  Tensor("Mean:0", shape=(), dtype=float32)
DONE WITH EPOCH
EPOCH 31
DONE WITH EPOCH
EPOCH 32
DONE WITH EPOCH
EPOCH 33
DONE WITH EPOCH
EPOCH 34
DONE WITH EPOCH
EPOCH 35
DONE WITH EPOCH
EPOCH 36
DONE WITH EPOCH
EPOCH 37
DONE WITH EPOCH
EPOCH 38
DONE WITH EPOCH
EPOCH 39
DONE WITH EPOCH
EPOCH 40
Loss:  Tensor("Mean:0", shape=(), dtype=float32)
DONE WITH EPOCH
EPOCH 41
DONE WITH EPOCH
EPOCH 42
DONE WITH EPOCH
EPOCH 43
DONE WITH EPOCH
EPOCH 44
DONE WITH EPOCH
EPOCH 45
DONE WITH EPOCH
EPOCH 46
DONE WITH EPOCH
EPOCH 47
DONE WITH EPOCH
EPOCH 48
DONE WITH EPOCH
EPOCH 49
DONE WITH EPOCH
EPOCH 50
Loss:  Tensor("Mean:0", shape=(), dtype=float32)
DONE WITH EPOCH
EPOCH 51
DONE WITH EPOCH
EPOCH 52
DONE WITH EPOCH
EPOCH 53
DONE WITH EPOCH
EPOCH 54
DONE WITH EPOCH
EPOCH 55
DONE WITH EPOCH
EPOCH 56
DONE WITH EPOCH
EPOCH 57
DONE WITH EPOCH
```

```
EPOCH 58
DONE WITH EPOCH
EPOCH 59
DONE WITH EPOCH
EPOCH 60
Loss:  Tensor("Mean:0", shape=(), dtype=float32)
DONE WITH EPOCH
EPOCH 61
DONE WITH EPOCH
EPOCH 62
DONE WITH EPOCH
EPOCH 63
DONE WITH EPOCH
EPOCH 64
DONE WITH EPOCH
EPOCH 65
DONE WITH EPOCH
EPOCH 66
DONE WITH EPOCH
EPOCH 67
DONE WITH EPOCH
EPOCH 68
DONE WITH EPOCH
EPOCH 69
DONE WITH EPOCH
EPOCH 70
Loss:  Tensor("Mean:0", shape=(), dtype=float32)
DONE WITH EPOCH
EPOCH 71
DONE WITH EPOCH
EPOCH 72
DONE WITH EPOCH
EPOCH 73
DONE WITH EPOCH
EPOCH 74
DONE WITH EPOCH
EPOCH 75
DONE WITH EPOCH
EPOCH 76
DONE WITH EPOCH
EPOCH 77
DONE WITH EPOCH
EPOCH 78
DONE WITH EPOCH
EPOCH 79
DONE WITH EPOCH
EPOCH 80
Loss:  Tensor("Mean:0", shape=(), dtype=float32)
DONE WITH EPOCH
EPOCH 81
DONE WITH EPOCH
EPOCH 82
DONE WITH EPOCH
EPOCH 83
DONE WITH EPOCH
EPOCH 84
DONE WITH EPOCH
EPOCH 85
DONE WITH EPOCH
EPOCH 86
DONE WITH EPOCH
```

```
EPOCH 87
DONE WITH EPOCH
EPOCH 88
DONE WITH EPOCH
EPOCH 89
DONE WITH EPOCH
EPOCH 90
Loss:  Tensor("Mean:0", shape=(), dtype=float32)
DONE WITH EPOCH
EPOCH 91
DONE WITH EPOCH
EPOCH 92
DONE WITH EPOCH
EPOCH 93
DONE WITH EPOCH
EPOCH 94
DONE WITH EPOCH
EPOCH 95
DONE WITH EPOCH
EPOCH 96
DONE WITH EPOCH
EPOCH 97
DONE WITH EPOCH
EPOCH 98
DONE WITH EPOCH
EPOCH 99
DONE WITH EPOCH
EPOCH 100
Loss:  Tensor("Mean:0", shape=(), dtype=float32)
DONE WITH EPOCH
EPOCH 101
DONE WITH EPOCH
EPOCH 102
DONE WITH EPOCH
EPOCH 103
DONE WITH EPOCH
EPOCH 104
DONE WITH EPOCH
EPOCH 105
DONE WITH EPOCH
EPOCH 106
DONE WITH EPOCH
EPOCH 107
DONE WITH EPOCH
EPOCH 108
DONE WITH EPOCH
EPOCH 109
DONE WITH EPOCH
EPOCH 110
Loss:  Tensor("Mean:0", shape=(), dtype=float32)
DONE WITH EPOCH
EPOCH 111
DONE WITH EPOCH
EPOCH 112
DONE WITH EPOCH
EPOCH 113
DONE WITH EPOCH
EPOCH 114
DONE WITH EPOCH
EPOCH 115
DONE WITH EPOCH
```

```
EPOCH 116
DONE WITH EPOCH
EPOCH 117
DONE WITH EPOCH
EPOCH 118
DONE WITH EPOCH
EPOCH 119
DONE WITH EPOCH
EPOCH 120
Loss:  Tensor("Mean:0", shape=(), dtype=float32)
DONE WITH EPOCH
EPOCH 121
DONE WITH EPOCH
EPOCH 122
DONE WITH EPOCH
EPOCH 123
DONE WITH EPOCH
EPOCH 124
DONE WITH EPOCH
EPOCH 125
DONE WITH EPOCH
EPOCH 126
DONE WITH EPOCH
EPOCH 127
DONE WITH EPOCH
EPOCH 128
DONE WITH EPOCH
EPOCH 129
DONE WITH EPOCH
EPOCH 130
Loss:  Tensor("Mean:0", shape=(), dtype=float32)
DONE WITH EPOCH
EPOCH 131
DONE WITH EPOCH
EPOCH 132
DONE WITH EPOCH
EPOCH 133
DONE WITH EPOCH
EPOCH 134
DONE WITH EPOCH
EPOCH 135
DONE WITH EPOCH
EPOCH 136
DONE WITH EPOCH
EPOCH 137
DONE WITH EPOCH
EPOCH 138
DONE WITH EPOCH
EPOCH 139
DONE WITH EPOCH
EPOCH 140
Loss:  Tensor("Mean:0", shape=(), dtype=float32)
DONE WITH EPOCH
EPOCH 141
DONE WITH EPOCH
EPOCH 142
DONE WITH EPOCH
EPOCH 143
DONE WITH EPOCH
EPOCH 144
DONE WITH EPOCH
```

```
EPOCH 145
DONE WITH EPOCH
EPOCH 146
DONE WITH EPOCH
EPOCH 147
DONE WITH EPOCH
EPOCH 148
DONE WITH EPOCH
EPOCH 149
DONE WITH EPOCH
EPOCH 150
Loss:  Tensor("Mean:0", shape=(), dtype=float32)
DONE WITH EPOCH
EPOCH 151
DONE WITH EPOCH
EPOCH 152
DONE WITH EPOCH
EPOCH 153
DONE WITH EPOCH
EPOCH 154
DONE WITH EPOCH
EPOCH 155
DONE WITH EPOCH
EPOCH 156
DONE WITH EPOCH
EPOCH 157
DONE WITH EPOCH
EPOCH 158
DONE WITH EPOCH
EPOCH 159
DONE WITH EPOCH
EPOCH 160
Loss:  Tensor("Mean:0", shape=(), dtype=float32)
DONE WITH EPOCH
EPOCH 161
DONE WITH EPOCH
EPOCH 162
DONE WITH EPOCH
EPOCH 163
DONE WITH EPOCH
EPOCH 164
DONE WITH EPOCH
EPOCH 165
DONE WITH EPOCH
EPOCH 166
DONE WITH EPOCH
EPOCH 167
DONE WITH EPOCH
EPOCH 168
DONE WITH EPOCH
EPOCH 169
DONE WITH EPOCH
EPOCH 170
Loss:  Tensor("Mean:0", shape=(), dtype=float32)
DONE WITH EPOCH
EPOCH 171
DONE WITH EPOCH
EPOCH 172
DONE WITH EPOCH
EPOCH 173
DONE WITH EPOCH
```

```
EPOCH 174
DONE WITH EPOCH
EPOCH 175
DONE WITH EPOCH
EPOCH 176
DONE WITH EPOCH
EPOCH 177
DONE WITH EPOCH
EPOCH 178
DONE WITH EPOCH
EPOCH 179
DONE WITH EPOCH
EPOCH 180
Loss:  Tensor("Mean:0", shape=(), dtype=float32)
DONE WITH EPOCH
EPOCH 181
DONE WITH EPOCH
EPOCH 182
DONE WITH EPOCH
EPOCH 183
DONE WITH EPOCH
EPOCH 184
DONE WITH EPOCH
EPOCH 185
DONE WITH EPOCH
EPOCH 186
DONE WITH EPOCH
EPOCH 187
DONE WITH EPOCH
EPOCH 188
DONE WITH EPOCH
EPOCH 189
DONE WITH EPOCH
EPOCH 190
Loss:  Tensor("Mean:0", shape=(), dtype=float32)
DONE WITH EPOCH
EPOCH 191
DONE WITH EPOCH
EPOCH 192
DONE WITH EPOCH
EPOCH 193
DONE WITH EPOCH
EPOCH 194
DONE WITH EPOCH
EPOCH 195
DONE WITH EPOCH
EPOCH 196
DONE WITH EPOCH
EPOCH 197
DONE WITH EPOCH
EPOCH 198
DONE WITH EPOCH
EPOCH 199
DONE WITH EPOCH
EPOCH 200
Loss:  Tensor("Mean:0", shape=(), dtype=float32)
DONE WITH EPOCH
```

In [25]:

```
# Alternatively, you can also run the following lines of code instead of the code chunk
above:
#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: images32, y: la
bels})
#        if i % 10 == 0:
#            print("Loss: ", loss)
#        print('DONE WITH EPOCH')
```

Evaluating The Neural Network

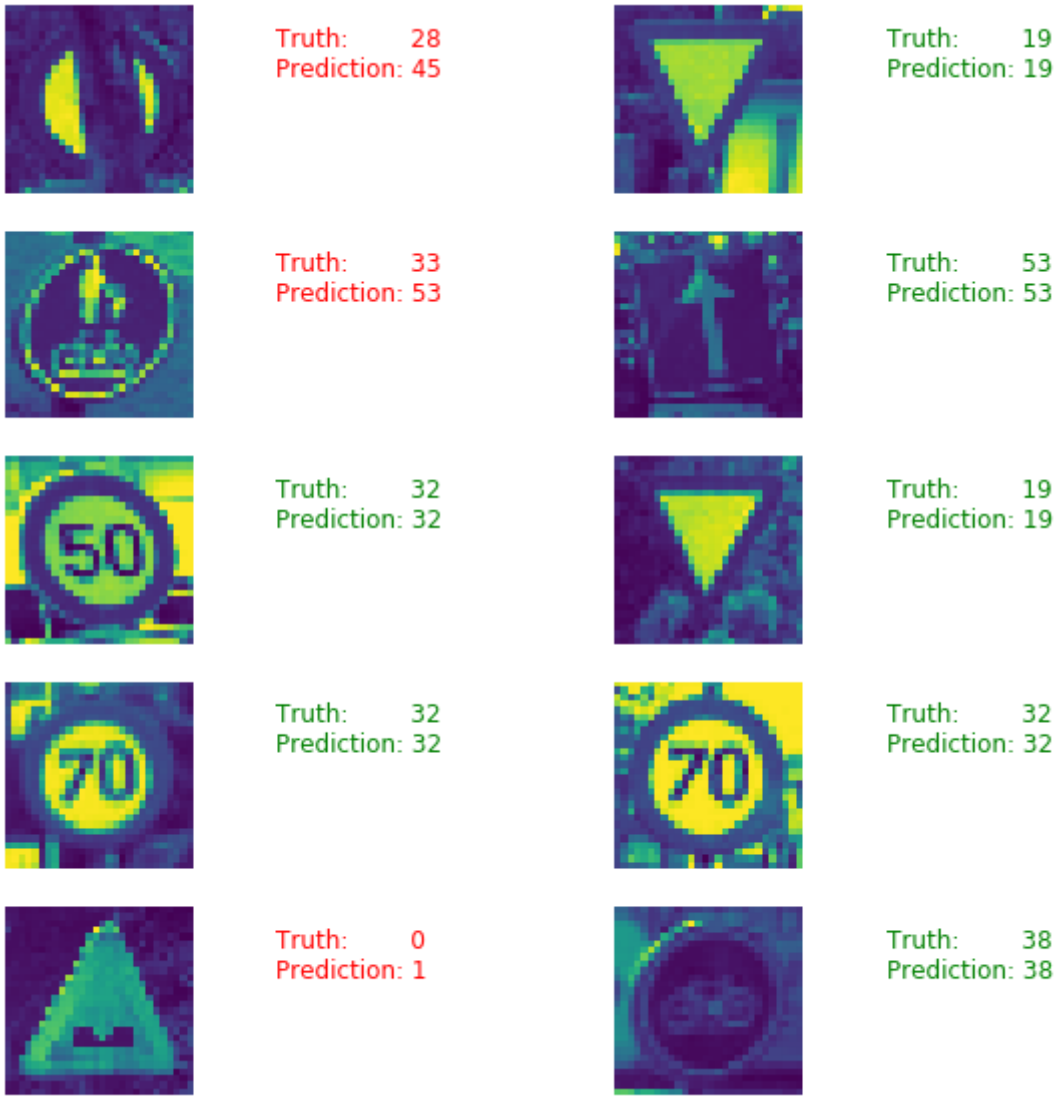
In [26]:

```
# 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, prediction),
             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()
```