Autonomous Vehicle AI Analysis

Ankit Patel, Cristian Vives, Dr. Joseph Ernst, Mark Rogers

Hume Center Colloquium, April 18th 2018

Project Background:

Using Machine Learning and Artificial Intelligence to classify a large traffic sign dataset using a convolutional neural network (CNN)

Adversarial: manipulation of images within dataset to test vulnerabilities of machine learning and our neural network model

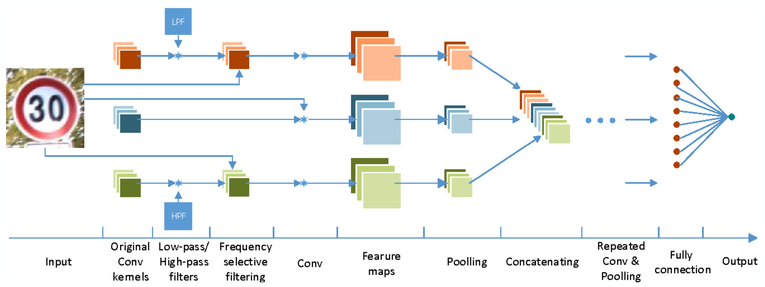

Figure 1. Simple Image showing "under the hood" structure of CNN

Sample Images from the dataset:

(Graphs below showing the varied class set)
Number of different classes: 47

Data Processing:

Over 7000 raw images varying in size

Grayscaled to reduce channels

Images were upscaled by padding or downscaled

Creating the Neural Network:

```
def model_fn(features, labels, mode, params):
```

5 layers (Convolutional, Flattening, Pooling, Fully Connected)

Weights, biases, and filters to create the feature maps

```
# First convolutional layer.
net = tf.layers.conv2d(inputs=x, name='layer_conv1',
                       filters=32, kernel_size=3,
                       padding='same', activation=tf.nn.relu)
net = tf.layers.max_pooling2d(inputs=net, pool_size=2, strides=2)

# Second convolutional layer.
net = tf.layers.conv2d(inputs=net, name='layer_conv2',
                       filters=32, kernel_size=3,
                       padding='same', activation=tf.nn.relu)
net = tf.layers.max_pooling2d(inputs=net, pool_size=2, strides=2)

# First fully-connected / dense layer.
# This uses the ReLU activation function.
net = tf.layers.dense(inputs=net, name='layer_fc1',
                      units=128, activation=tf.nn.relu)

# Second fully-connected / dense layer.
# This is the last layer so it does not use an activation function.
net = tf.layers.dense(inputs=net, name='layer_fc_2',
                      units=2)
```

```
# Flatten to a 2-rank tensor.
net = tf.contrib.layers.flatten(net)
# Eventually this should be replaced with:
# net = tf.layers.flatten(net)
```

Key features of CNN:

Dropout: Randomly ignoring neuron nodes during the training phase to prevent overfitting in the fully connected layer

Training and Testing Neural Network:

# of Images in Training: ____

# of Images in Testing: _____

(Place output code below showing accuracy)

Adversarial Phase:

```
def add_noise(X_image)
        #create copy as to not modify the actual image
        X_image_copy = X_image.copy()
        height,width,channel = X_image_copy.shape #gather dimensions and channel number for shape
        mean = 0
        stdev = 0.4
        sigma = stdev**0.5
        generated_noise = np.random.normal(mean, sigma, (height,width,channel))
        #gaussian noise is a statistical method used to generate noise (distribution curve)
        gaussian_noise = generated_noise.reshape(height,width,channel)
        image_with_noise = X_image_copy + gaussian_noise

return image_with_noise
```

Gaussian Noise:

Probability density function to generate the noise within the images

(Sample picture(s) with noise added)

Results:

(show result when fed back into CNN for testing)

(Another histogram plotting the results)