Traffic_Sign_Classifier

October 2, 2018

1 Self-Driving Car Engineer Nanodegree

1.1 Deep Learning

1.2 Project: Build a Traffic Sign Recognition Classifier

1.3 Author: Felipe Pamboukian

In this notebook, a traffic sign classifier is implemented using Convolucional Neural Network based on LeNet architecture, German Traffic Sign Dataset is used to train the model.

1.4 Step 0: Load The Data

In [1]:
```
import  cv2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from sklearn.utils import shuffle
import tensorflow as tf
from tensorflow.contrib.layers import flatten
```

In [2]:
```
# Load pickled data
import pickle

training_file = '../data/train.p'
validation_file= '../data/valid.p'
testing_file = '../data/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']
```

1.5 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**

1.5.1 Basic Summary of the Dataset

```
In [3]: n_train = len(X_train)
        n_validation = len(X_valid)
        n_test = len(X_test)

        image_shape = X_train[0].shape

        classes = pd.DataFrame(y_train)[0].unique()
        n_classes = len(classes)

        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
```

1.5.2 Visualization of the dataset

Showing a sample for each class

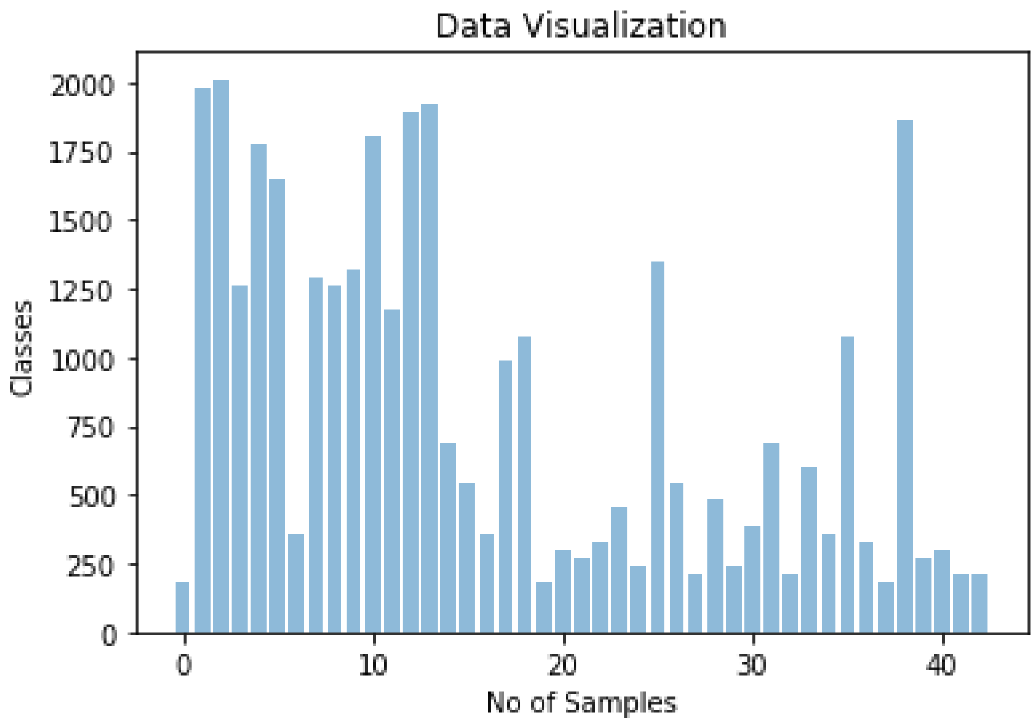
```
In [4]: ### Data exploration visualization code goes here.
        ### Feel free to use as many code cells as needed.
```

```
import matplotlib.cm as cm
# Visualizations will be shown in the notebook.
%matplotlib inline

def plot_each_class(x, y, classes):
    fig = plt.figure(figsize=(12,19))
    for j in range(len(classes)-1):
        ax = fig.add_subplot(15, 11, j+1)
        index = np.random.choice(np.where(y==j)[0])
        if x.shape[3] == 1:
            ax.imshow(x[index], cmap='gray')
        else:
            ax.imshow(x[index])
        plt.xticks(np.array([]))
        plt.yticks(np.array([]))
        plt.title('Class:'+str(j))
        plt.tight_layout()

plot_each_class(X_train, y_train, classes)
```

Showing number of samples for each class

In [5]:
```
class_arr= []
samples_arr=[]

for class_n in range(n_classes):
    class_indices = np.where(y_train == class_n)
    n_samples = len(class_indices[0])
    class_arr.append(class_n)
    samples_arr.append(n_samples)
```

```
#plt.hist(y_train,bins=43)
plt.bar( class_arr, samples_arr,align='center', alpha=0.5)
plt.ylabel('Classes')
plt.xlabel('No of Samples')
plt.title('Data Visualization')
plt.show()
```



1.6 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.

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

I got better results using color images, so now I just normalize.

```
In [6]: def preprocessing(imageset):
            new_shape = image_shape[0:2] + (3,)
            new_imageset = np.empty(shape=(len(imageset),) + new_shape, dtype=int)

            for i in range(0, len(imageset)):
                #gray_img = cv2.cvtColor(imageset[i], cv2.COLOR_RGB2GRAY)
```

```
            #norm_img = cv2.normalize(gray_img, np.zeros(image_shape[0:2]), 0, 255, cv2.NORM
            norm_img = cv2.normalize(imageset[i], np.zeros(image_shape[0:2]), 0, 255, cv2.NO
            new_imageset[i] = np.reshape(norm_img, new_shape)

        return new_imageset

    #X_train = preprocessing(X_train)
    #X_valid = preprocessing(X_valid)
    #X_test = preprocessing(X_test)
```

1.6.2 Shuffle training dataset

```
In [7]: X_train, y_train = shuffle(X_train, y_train)
```

1.6.3 Model Architecture

The model architecture is almost the same, I just modified to use color images and output to 43 classes.

```
In [8]: def LeNet(x, keep_prob):
         # Arguments used for tf.truncated_normal, randomly defines variables for the weights an
            mu = 0
            sigma = 0.1

            # Layer 1: Convolutional. Input = 32x32x3. Output = 28x28x6.
            conv1_W = tf.Variable(tf.truncated_normal(shape=(5, 5, 3, 6), mean = mu, stddev = si
            conv1_b = tf.Variable(tf.zeros(6))
            conv1   = tf.nn.conv2d(x, conv1_W, strides=[1, 1, 1, 1], padding='VALID') + conv1_b
            # Activation.
            conv1 = tf.nn.relu(conv1)
            # Pooling. Input = 28x28x6. Output = 14x14x6.
            conv1 = tf.nn.max_pool(conv1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='VAL

            # Layer 2: Convolutional. Output = 10x10x16.
            conv2_W = tf.Variable(tf.truncated_normal(shape=(5, 5, 6, 16), mean = mu, stddev = s
            conv2_b = tf.Variable(tf.zeros(16))
            conv2   = tf.nn.conv2d(conv1, conv2_W, strides=[1, 1, 1, 1], padding='VALID') + conv
            # Activation.
            conv2 = tf.nn.relu(conv2)
            # Pooling. Input = 10x10x16. Output = 5x5x16.
            conv2 = tf.nn.max_pool(conv2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='VAL
            # Flatten. Input = 5x5x16. Output = 400.
            fc0   = flatten(conv2)
            fc0   = tf.nn.dropout(fc0, keep_prob=keep_prob)

            # 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
    # Activation.
    fc1    = tf.nn.relu(fc1)

    # 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
    # Activation.
    fc2    = tf.nn.relu(fc2)

    # 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
```

1.6.4 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 [9]:
```
### 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 = tf.placeholder(tf.float32, (None, 32, 32, 3))
y = tf.placeholder(tf.int32, (None))
one_hot_y = tf.one_hot(y, 43)

EPOCHS = 50
BATCH_SIZE = 128
rate = 0.0009
KEEP_PROB = 0.5

keep_prob = tf.placeholder(tf.float32)

logits = LeNet(x, keep_prob)
cross_entropy = tf.nn.softmax_cross_entropy_with_logits(labels=one_hot_y, logits=logits)
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))
```

```python
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_
        accuracy = sess.run(accuracy_operation, feed_dict={x: batch_x, y: batch_y, keep_
        total_accuracy += (accuracy * len(batch_x))
    return total_accuracy / num_examples

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    num_examples = len(X_train)

    print("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: K

        validation_accuracy = evaluate(X_valid, y_valid)
        print("EPOCH {} ...".format(i+1))
        print("Validation Accuracy = {:.3f}".format(validation_accuracy))
        print()

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

```
Training...

EPOCH 1 ...
Validation Accuracy = 0.375

EPOCH 2 ...
Validation Accuracy = 0.598

EPOCH 3 ...
Validation Accuracy = 0.722

EPOCH 4 ...
Validation Accuracy = 0.806
```

```
EPOCH 5 ...
Validation Accuracy = 0.837

EPOCH 6 ...
Validation Accuracy = 0.861

EPOCH 7 ...
Validation Accuracy = 0.870

EPOCH 8 ...
Validation Accuracy = 0.875

EPOCH 9 ...
Validation Accuracy = 0.897

EPOCH 10 ...
Validation Accuracy = 0.904

EPOCH 11 ...
Validation Accuracy = 0.890

EPOCH 12 ...
Validation Accuracy = 0.900

EPOCH 13 ...
Validation Accuracy = 0.912

EPOCH 14 ...
Validation Accuracy = 0.911

EPOCH 15 ...
Validation Accuracy = 0.921

EPOCH 16 ...
Validation Accuracy = 0.920

EPOCH 17 ...
Validation Accuracy = 0.930

EPOCH 18 ...
Validation Accuracy = 0.913

EPOCH 19 ...
Validation Accuracy = 0.916

EPOCH 20 ...
Validation Accuracy = 0.926
```

```
EPOCH 21 ...
Validation Accuracy = 0.931

EPOCH 22 ...
Validation Accuracy = 0.927

EPOCH 23 ...
Validation Accuracy = 0.931

EPOCH 24 ...
Validation Accuracy = 0.936

EPOCH 25 ...
Validation Accuracy = 0.936

EPOCH 26 ...
Validation Accuracy = 0.933

EPOCH 27 ...
Validation Accuracy = 0.926

EPOCH 28 ...
Validation Accuracy = 0.937

EPOCH 29 ...
Validation Accuracy = 0.935

EPOCH 30 ...
Validation Accuracy = 0.942

EPOCH 31 ...
Validation Accuracy = 0.933

EPOCH 32 ...
Validation Accuracy = 0.938

EPOCH 33 ...
Validation Accuracy = 0.929

EPOCH 34 ...
Validation Accuracy = 0.936

EPOCH 35 ...
Validation Accuracy = 0.937

EPOCH 36 ...
Validation Accuracy = 0.941
```

```
EPOCH 37 ...
Validation Accuracy = 0.942

EPOCH 38 ...
Validation Accuracy = 0.932

EPOCH 39 ...
Validation Accuracy = 0.949

EPOCH 40 ...
Validation Accuracy = 0.941

EPOCH 41 ...
Validation Accuracy = 0.944

EPOCH 42 ...
Validation Accuracy = 0.941

EPOCH 43 ...
Validation Accuracy = 0.944

EPOCH 44 ...
Validation Accuracy = 0.943

EPOCH 45 ...
Validation Accuracy = 0.944

EPOCH 46 ...
Validation Accuracy = 0.941

EPOCH 47 ...
Validation Accuracy = 0.944

EPOCH 48 ...
Validation Accuracy = 0.946

EPOCH 49 ...
Validation Accuracy = 0.939

EPOCH 50 ...
Validation Accuracy = 0.942

Model saved
```

1.6.5 Evaluate trained model using testset

```
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))
```

```
INFO:tensorflow:Restoring parameters from ./lenet
Test Accuracy = 0.942
```

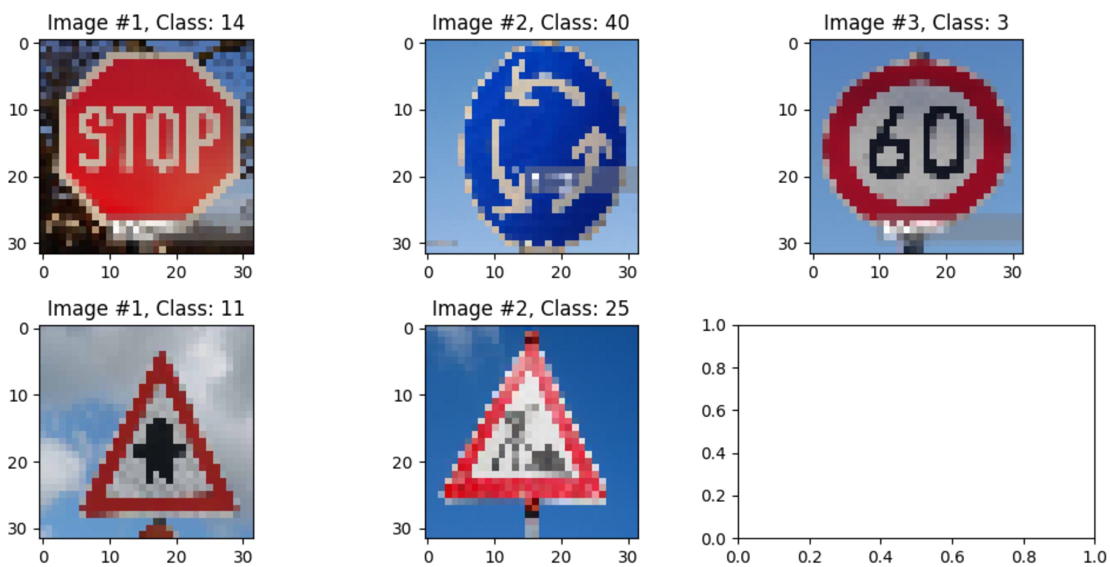
1.7 Step 3: Test a Model on New Images

Now I applied the trained model to the German trafic sign images that were obtained from the Internet.

1.7.1 Load and Output the Images

```
In [22]: ### Load the images and plot them here.
         ### Feel free to use as many code cells as needed.
         import glob

         test_signs = glob.glob('./test_images/*')
         X_signs = []

         sign_names = pd.read_csv('signnames.csv')
         y_sign_names = sign_names['SignName']

         #y_signs = [25,40,14,3,11] # CPU
         y_signs = [14,40,3,11,25] # GPU

         for i in range(len(test_signs)):
             img = mpimg.imread(test_signs[i])
             img = cv2.resize(img, (32,32))
             X_signs.append(img)

         def plot_test_signs(X_signs):
             plt.figure(figsize=(10,5))
             plt.subplot(231)
             plt.title('Image #1, Class: '+ str(y_signs[0]))
             plt.imshow(X_signs[0])
             plt.subplot(232)
             plt.title('Image #2, Class: '+ str(y_signs[1]))
             plt.imshow(X_signs[1])
             plt.subplot(233)
             plt.title('Image #3, Class: '+ str(y_signs[2]))
```

```
        plt.imshow(X_signs[2])
        plt.subplot(234)
        plt.title('Image #1, Class: '+ str(y_signs[3]))
        plt.imshow(X_signs[3])
        plt.subplot(235)
        plt.title('Image #2, Class: '+ str(y_signs[4]))
        plt.imshow(X_signs[4])
        plt.subplot(236)
        plt.tight_layout()
        plt.show()
        return None

    plot_test_signs(X_signs)
    X_signs_new = X_signs
    #X_signs_new = preprocessing(X_signs)
```



1.7.2 Predict the Sign Type for Each Image

```
In [23]: ### 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 earl
         ### Feel free to use as many code cells as needed.
         with tf.Session() as sess:
             saver.restore(sess, tf.train.latest_checkpoint('.'))
             prediction = tf.argmax(logits, 1)
             test_prediction = sess.run(
                 prediction,
                 feed_dict={x: X_signs_new, y: y_signs, keep_prob: 1})
```

```
    print (test_prediction)
    print ()
    for i in range(len(test_prediction)):
        print ('Prediction:', y_sign_names[test_prediction[i]], test_prediction[i])
```

```
INFO:tensorflow:Restoring parameters from ./lenet
[14 40  3 11 22]

Prediction: Stop 14
Prediction: Roundabout mandatory 40
Prediction: Speed limit (60km/h) 3
Prediction: Right-of-way at the next intersection 11
Prediction: Bumpy road 22
```

1.7.3 Analyze Performance

```
In [24]: ### Calculate the accuracy for these 5 new images.
         ### For example, if the model predicted 1 out of 5 signs correctly, it's 20% accurate o
         with tf.Session() as sess:
             saver.restore(sess, tf.train.latest_checkpoint('.'))

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

```
INFO:tensorflow:Restoring parameters from ./lenet
Test Accuracy = 0.800
```

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

```
In [26]: # Print out the top five softmax probabilities for the predictions on
         # the German traffic sign images found on the web.
         softmax_logits = tf.nn.softmax(logits)
         top_k = tf.nn.top_k(softmax_logits, k=5)

         with tf.Session() as sess:
             sess.run(tf.global_variables_initializer())
             saver.restore(sess, tf.train.latest_checkpoint('.'))
             my_softmax_logits = sess.run(softmax_logits, feed_dict={x: X_signs_new, keep_prob:
             my_top_k = sess.run(top_k, feed_dict={x: X_signs_new, keep_prob: 1.0})
             print(my_softmax_logits)
             print()
             print(my_top_k)
```

```
INFO:tensorflow:Restoring parameters from ./lenet
[[  8.33419095e-14   2.47560184e-11   4.46718671e-14   8.96737025e-12
    6.62408428e-14   2.11653442e-14   1.42625840e-21   5.31765431e-16
    3.30470624e-15   6.24675225e-13   8.58986710e-13   2.09475297e-12
```

```
  3.11439188e-13 4.95133830e-11 9.99997020e-01 6.76786406e-17
  8.96636384e-19 2.93159883e-06 3.64587804e-11 3.89979329e-17
  6.42612345e-15 1.51237002e-15 6.13543895e-12 2.17136557e-13
  4.27939849e-15 7.07075642e-08 2.31084107e-14 7.61308033e-19
  2.25605897e-13 1.64613687e-11 2.70337797e-11 3.22322963e-12
  4.35938206e-20 2.45900803e-23 1.82881655e-15 7.30913253e-16
  4.95172539e-15 5.70789115e-21 7.33046559e-16 1.86075662e-24
  2.25361791e-19 1.14734980e-17 1.00270483e-16]
[ 8.64663582e-28 6.14643158e-16 2.54786169e-20 9.98630656e-25
  2.08731215e-24 7.23404746e-21 1.29246390e-29 1.24316172e-17
  6.15030596e-22 9.98079944e-22 3.52489997e-25 5.60277793e-23
  1.85616855e-16 1.09113196e-22 2.43387561e-29 3.83000675e-27
  1.91021207e-20 3.73936191e-25 2.01966871e-24 4.15749944e-33
  3.07489953e-22 1.05950905e-27 7.87922751e-34 5.13492489e-25
  3.53437727e-26 5.66590891e-21 5.89116376e-21 3.52302487e-36
  4.93696804e-24 4.13014945e-27 1.18216484e-29 2.50350752e-25
  4.18412455e-22 6.43445783e-15 4.51513136e-16 1.37925490e-10
  2.28020328e-11 3.39953592e-18 3.81641902e-13 3.40208288e-23
  1.00000000e+00 7.94834118e-16 2.97009442e-20]
[ 6.82987795e-22 2.48193386e-16 1.21937174e-11 1.00000000e+00
  4.45892236e-23 1.69297557e-18 4.94503767e-38 6.97144847e-28
  5.61666881e-37 8.30827548e-23 6.21899111e-27 3.57340925e-25
  3.29425906e-36 3.44215257e-27 2.75572611e-27 3.60383574e-28
  2.06479357e-35 2.48102584e-38 3.86937381e-29 4.77271442e-31
  3.43523498e-37 2.14230861e-20 1.69024039e-32 1.52697786e-22
  1.36462585e-27 1.15915675e-24 3.48185146e-34 7.24856425e-32
  2.94186528e-31 1.78283142e-27 5.04649795e-22 8.97375901e-24
  1.06699886e-30 1.26347658e-31 3.31033413e-28 5.87372451e-22
  2.36415676e-35 7.10756747e-35 3.05792248e-32 0.00000000e+00
  3.76908038e-24 3.33743136e-35 2.81900994e-35]
[ 9.41088257e-27 6.76190860e-25 3.37024918e-23 9.97344024e-21
  1.88880859e-30 9.90461094e-24 3.13453427e-38 4.16800970e-23
  7.44358758e-30 5.50617834e-28 4.14711169e-28 9.96304750e-01
  5.26572322e-23 1.16020399e-30 4.85261287e-34 7.50643383e-30
  1.49179799e-27 1.51698119e-32 3.43354477e-11 1.55815430e-16
  7.29498807e-24 7.03804848e-10 4.71729281e-34 4.46725212e-10
  1.96114291e-10 2.51008394e-16 1.46678565e-16 1.13317238e-11
  6.48777690e-13 4.00260225e-17 3.69528192e-03 1.52550584e-17
  5.13945473e-35 8.49671267e-33 1.16259878e-26 1.29697429e-31
  0.00000000e+00 0.00000000e+00 1.34675659e-26 0.00000000e+00
  4.48167510e-25 1.58831045e-31 1.81321555e-31]
[ 1.98316813e-13 3.19795218e-10 2.49086899e-16 1.47970438e-11
  3.38905228e-12 3.29701752e-11 2.42485169e-20 1.49230061e-14
  4.48127592e-13 1.69701251e-16 2.59623634e-09 2.97087945e-11
  2.20081183e-15 7.81289801e-13 4.83160678e-10 6.59537147e-11
  5.94102954e-17 1.19752944e-10 8.56533866e-09 1.75362786e-17
  1.03315810e-14 1.05865788e-10 9.99954462e-01 3.61851933e-14
  6.46400996e-14 4.43562894e-05 1.05863876e-06 9.51114746e-15
```