VE581 homework2

Hu Chong 515370910114

July 22, 2019

1 . Data Exploration

a. Dataset Summary

Dataset	train	validation	test
samples	34799	4410	4410

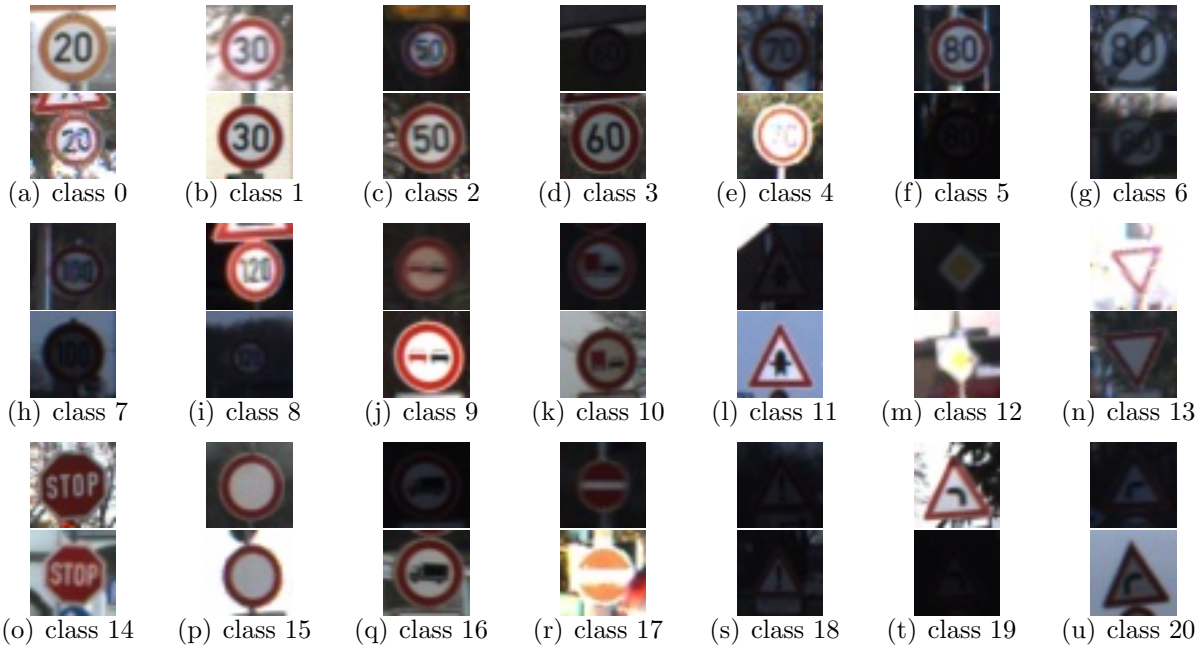
Table 1: Number of samples in each set

i.

ii. Shape of the traffic image: (32, 32, 3)

iii. Number of classes/labels: 43

b. Exploratory Visualization



(a) class 0 (b) class 1 (c) class 2 (d) class 3 (e) class 4 (f) class 5 (g) class 6

(h) class 7 (i) class 8 (j) class 9 (k) class 10 (l) class 11 (m) class 12 (n) class 13

(o) class 14 (p) class 15 (q) class 16 (r) class 17 (s) class 18 (t) class 19 (u) class 20

Figure 1: Sample Image for Each class/label

(a) class 21 (b) class 22 (c) class 23 (d) class 24 (e) class 25 (f) class 26 (g) class 27

(h) class 28 (i) class 29 (j) class 30 (k) class 31 (l) class 32 (m) class 33 (n) class 34

(o) class 35 (p) class 36 (q) class 37 (r) class 38 (s) class 39 (t) class 40 (u) class 41

(v) class 42

Figure 2: Sample Image for Each class/label (continue.)

2 . Design and Test a Classifier (or model architecture)

a. In order to augment the dataset, I write a class named `DataFeeder` to provide different augmentation methods in training, such as noise, brightness, flip and random crop. Notice that only a part of class can be applied with flip method. They are class 9, 11, 12, 13, 15, 17, 18, 22, 26, 29, 30 and 35. Adding noise, flip and crop can increase the robustness of model. Considering the original images have different brightness, I use brightness augmentation to increase the accuracy of model under different brightness.

```
    import cv2
    import numpy as np
    ...
    @staticmethod
    def brightness_augment(img, factor=0.2):
        img_hsv = np.array(cv2.cvtColor(img, cv2.COLOR_RGB2HSV), dtype=np.float64)
        img_hsv[:, :, 2] = img_hsv[:, :, 2] * np.random.uniform(1 - factor, 1 + factor)
        img_hsv[:, :, 2][img_hsv[:, :, 2] > 255] = 255
        img_rgb = cv2.cvtColor(np.array(img_hsv, dtype=np.uint8), cv2.COLOR_HSV2RGB)
        return img_rgb
```

```
    @staticmethod
    def flip_augment(img, factor=0.5):
        if np.random.random() > factor:
            return np.fliplr(img)
        else:
            return img

    @staticmethod
    def add_noise(img, factor=0.15):
        width, height, channel = img.shape
        noise_arr = (np.random.rand(width, height, channel) - 0.5) * factor * 255.0
        img_noise = img + noise_arr
        return np.minimum(np.maximum(img_noise, 0.0), 255.0)

    @staticmethod
    def crop_augment(img, factor=0.5, limit=3):
        if np.random.random() > factor:
            shape_orig = img.shape
            x_left = np.random.randint(0, limit)
            x_right = np.random.randint(shape_orig[0] - limit, shape_orig[0])
            y_top = np.random.randint(0, limit)
            y_bottom = np.random.randint(shape_orig[1] - limit, shape_orig[1])
            img_crop = img[x_left:x_right, y_top:y_bottom, :]
            img_crop_t = cv2.resize(img_crop, dsize=shape_orig[0:2],interpolation=cv2.INTER_LINEAR
            return img_crop_t
        else:
            return img
```

b. The rough network architecture is shown in the following graph, with the corresponding layer type and feature size. The code corresponding the network architecture is appended after.

```
import tensorflow as tf
import numpy as np
class traffic_sign_network():
    def __init__(self, phase="train", num_classes=43):
        self.phase = phase.upper()
        self.num_classes = num_classes

    def is_train(self):
        if self.phase == "TRAIN":
            return True
        elif self.phase == "TEST":
```

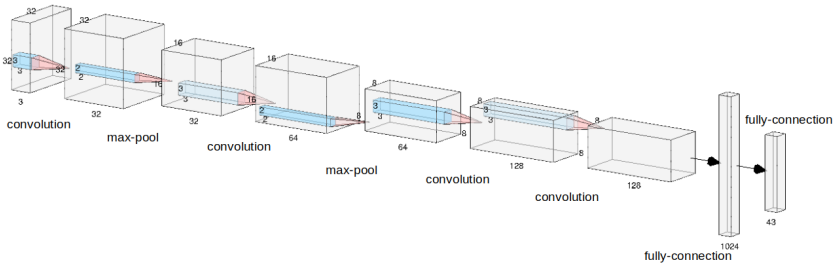


Figure 3: Network Architecture

```
12              return False
13          else:
14              raise ValueError("Not a valid phase")
15
16      def inference(self, input_data):
17          """
18          inference
19          :param input_data:
20          :return: return prediction label and raw output
21          """
22          output_layer = self.forward(input_data)
23          pred = tf.argmax(tf.nn.softmax(output_layer), axis=1, name="pred")
24          return pred, output_layer
25
26      def loss(self, labels, input_data):
27          """
28          compute and loss and do inference
29          :param labels: ground truth lable
30          :param input_data: input data [batch x num_cells]
31          :return: loss and prediction label
32          """
33          with tf.variable_scope(name_or_scope="cnn"):
34              pred, out = self.inference(input_data)
35              loss = tf.reduce_mean(tf.losses.sparse_softmax_cross_entropy(labels, out), name="loss"
36                     tf.losses.get_regularization_loss()
37              return loss, pred
38
39      def conv_layer(self, input_data, out_dims, name):
40          """
41          traditional convolution layer unit
42          :param input_data:
43          :param out_dims:
```

```
        :param name:
        :return:
        """

        with tf.variable_scope(name_or_scope=name):
            [_, _, _, channel_num] = input_data.get_shape().as_list()
            w = tf.get_variable("w", [3, 3, channel_num, out_dims],
                                initializer=tf.contrib.layers.variance_scaling_initializer(),
                                trainable=self.is_train())
            conv = tf.nn.conv2d(input_data, w, [1, 1, 1, 1], "SAME", name="conv")
            bn = tf.contrib.layers.batch_norm(conv, scope="bn", trainable=self.is_train())
            relu = tf.nn.relu(bn, name="relu")
        return relu

    def forward(self, input_data, reg_const=0.001):
        """
        forward process
        :param input_data: input_data [batch x height x width x channel]
        :param reg_const: regularization constant
        :return: output result [batch x num_classes]
        """
        [batch_num, _, _, channel_num] = input_data.get_shape().as_list()

        # conv1
        conv1 = self.conv_layer(input_data, 32, "conv1")
        maxpool1 = tf.nn.max_pool(conv1, [1, 2, 2, 1], [1, 2, 2, 1], "VALID", name="maxpool1")

        # conv2
        conv2 = self.conv_layer(maxpool1, 64, "conv2")
        maxpool2 = tf.nn.max_pool(conv2, [1, 2, 2, 1], [1, 2, 2, 1], "VALID", name="maxpool2")

        # conv3
        conv3 = self.conv_layer(maxpool2, 128, "conv3")

        # conv4
        conv4 = self.conv_layer(conv3, 128, "conv4")

        # fully connection
        shape = conv4.get_shape().as_list()[1:]
        before_fc = tf.reshape(conv4, [-1, int(np.prod(shape))])

        fc1 = tf.layers.dense(before_fc, 1024, kernel_initializer=tf.contrib.layers.variance_scali
                              name="fc1", trainable=self.is_train(),
                              kernel_regularizer=tf.contrib.layers.l2_regularizer(reg_const))
        # fc2 = tf.layers.dense(fc1, 1024, kernel_initializer=tf.contrib.layers.variance_scaling_i
```

```
89          #                           name="fc2", trainable=self.is_train())
90          fc2 = tf.layers.dense(fc1, self.num_classes,
91                                kernel_initializer=tf.contrib.layers.variance_scaling_initializer(),
92                                name="fc2", trainable=self.is_train(),
93                                kernel_regularizer=tf.contrib.layers.l2_regularizer(reg_const))
94          return fc2
```

c. As you can see in the previous code, I choose softmax cross entropy as my loss function. I choose Adam as my optimizer. To evaluate the performance of my model, I simply choose accuracy as the metric.
Code related to optimizer:

```
1       global_step = tf.Variable(0, name="global_step", trainable=False)
2
3       learning_rate = tf.Variable(lr, trainable=False)
4       optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
5
6       grad = optimizer.compute_gradients(loss=loss)
7
8       apply_grad_op = optimizer.apply_gradients(grad, global_step=global_step)
```

Code related to metric:

```
1       def compute_acc(preds, labels):
2           acc = np.size(np.where(preds == labels)) / preds.size
3           return acc
```

The initial learning rate is 0.0005 and batch size is 32. I set max training step is 10000 and save the model weight every 500 steps in case model is overfitting.

d. I plot the training loss in the following figure. For other information during training, such as train accuracy, validation loss and validation accuracy, I only print them in screen.

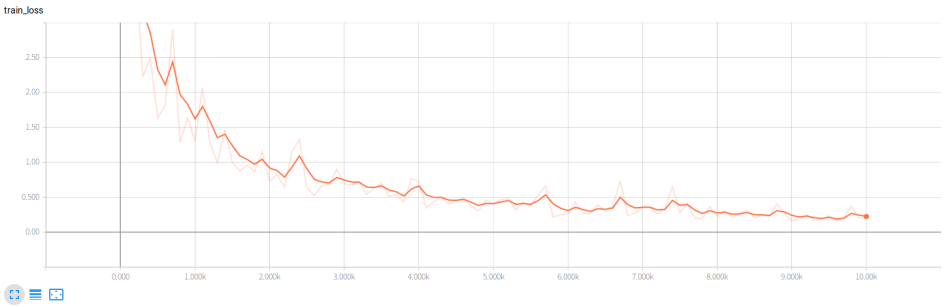


Figure 4: Training loss

```
traffic_sign_train.py:123] epoch: 100  train loss: 6.389  train acc: 37.66%
traffic_sign_train.py:123] epoch: 200  train loss: 2.842  train acc: 64.34%
traffic_sign_train.py:123] epoch: 300  train loss: 2.644  train acc: 70.09%
traffic_sign_train.py:123] epoch: 400  train loss: 1.975  train acc: 79.91%
traffic_sign_train.py:123] epoch: 500  train loss: 2.080  train acc: 80.41%
traffic_sign_train.py:140] epoch: 500  val loss: 2.551  val acc: 76.28%
traffic_sign_train.py:123] epoch: 600  train loss: 1.748  train acc: 85.09%
traffic_sign_train.py:123] epoch: 700  train loss: 1.611  train acc: 85.97%
...
```

e. After several trials, I get this network architecture. In order to improve the validation accuracy, I add the l2 regularization and data augmentation. After those setups, I get 95.37% accuracy in the validation dataset. The model is got after 8000 steps training.

```
epoch: 8000        val loss: 0.451        val acc: 95.37%
```

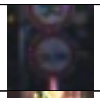
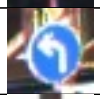
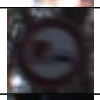
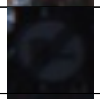
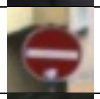
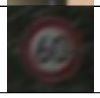
f. On the test dataset, the accuracy of my model is 96.28%.

```
test acc: 96.28%
```

In case you want to try my model on your own computer, please try the following script:

```
python3 train_sign_test.py -w model/traffic_sign_2019-07-20-17-53-30_008000.ckpt
```

For 10 test images, I got the following result.

sample image	label	prediction	top 5 softmax probabilities
	9	9	[9 8 15 22 2]
	34	34	[34 35 9 15 12]
	1	1	[1 0 4 18 32]
	10	10	[10 9 42 23 12]
	42	42	[42 41 6 12 32]
	5	5	[5 3 2 1 7]
	17	17	[17 9 41 37 19]
	3	3	[3 2 5 15 9]

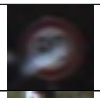
		5	5	[5 2 3 39 14]
		0	0	[0 4 1 14 27]

Table 2: 10 sample test image result.

3 . Test Your Classifier on New Images

a. For this part, I find ten traffic sign image from internet. I simply reshape the picture into 32x32 in the preprocess step. Here is my code.

```
for img_name in images_ls:
    im = cv2.imread(img_name)
    im_t = cv2.cvtColor(cv2.resize(im, (32, 32)), cv2.COLOR_BGR2RGB)
    image_batch.append(im_t)
```

b. Output the top 5 softmax probabilities for each above picture:

sample image	prediction	top 5 softmax probabilities
	13	[13 2 12 9 3]
	14	[14 17 15 3 13]
	17	[17 9 34 41 14]
	9	[9 3 2 19 36]
	1	[1 27 11 6 18]
	34	[34 33 17 9 35]
	0	[0 38 31 37 11]
	39	[39 33 2 31 12]
	13	[13 9 12 17 15]
	18	[18 26 11 25 27]

Table 3: 10 sample demo image result.

Notice that code related to this part is in `traffic_sign_demo.py`.

4 . Visualization

I visualize the some channels of convolution layers in the network. We can clearly see that the bottom level convolution layer only consider some basic features of the image, such as some edges or some colors. But the high level convolution layers will exact very abstract features.

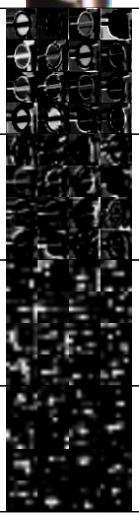
Layer			
Convolution Layer 1			
Convolution Layer 2			
Convolution Layer 3			
Convolution Layer 4			

Table 4: 3 sample image of network visualization result.

Notice that code related to this part is in `traffic_sign_demo.py`.

Appendix

1. `network.py`

```
import tensorflow as tf
import numpy as np


class traffic_sign_network():
    def __init__(self, phase="train", num_classes=43):
        self.phase = phase.upper()
        self.num_classes = num_classes
        self.layers = dict()

    def is_train(self):
```

```
        if self.phase == "TRAIN":
            return True
        elif self.phase == "TEST":
            return False
        else:
            raise ValueError("Not a valid phase")

    def inference(self, input_data):
        """
        inference
        :param input_data:
        :return: return prediction label and raw output
        """
        with tf.variable_scope(name_or_scope="cnn"):
            output_layer = self.forward(input_data)
            pred = tf.argmax(tf.nn.softmax(output_layer), axis=1, name="pred")
            return pred, output_layer

    def loss(self, labels, input_data):
        """
        compute and loss and do inference
        :param labels: ground truth lable
        :param input_data: input data [batch x num_cells]
        :return: loss and prediction label
        """

        pred, out = self.inference(input_data)
        loss = tf.reduce_mean(tf.losses.sparse_softmax_cross_entropy(labels, out), name="loss") +
               tf.losses.get_regularization_loss()
        return loss, pred

    def conv_layer(self, input_data, out_dims, name):
        """
        traditional convolution layer unit
        :param input_data:
        :param out_dims:
        :param name:
        :return:
        """

        with tf.variable_scope(name_or_scope=name):
            [_, _, _, channel_num] = input_data.get_shape().as_list()
            w = tf.get_variable("w", [3, 3, channel_num, out_dims],
                                initializer=tf.contrib.layers.variance_scaling_initializer(),
                                trainable=self.is_train())
```

```
            conv = tf.nn.conv2d(input_data, w, [1, 1, 1, 1], "SAME", name="conv")
            bn = tf.contrib.layers.batch_norm(conv, scope="bn", trainable=self.is_train())
            relu = tf.nn.relu(bn, name="relu")
        return relu

    def forward(self, input_data, reg_const=0.001):
        """
        forward process
        :param input_data: input_data [batch x height x width x channel]
        :param reg_const: regularization constant
        :return: output result [batch x num_classes]
        """
        [batch_num, _, _, channel_num] = input_data.get_shape().as_list()

        # conv1
        conv1 = self.conv_layer(input_data, 32, "conv1")
        self.layers["conv1"] = conv1
        maxpool1 = tf.nn.max_pool(conv1, [1, 2, 2, 1], [1, 2, 2, 1], "VALID", name="maxpool1")

        # conv2
        conv2 = self.conv_layer(maxpool1, 64, "conv2")
        self.layers["conv2"] = conv2
        maxpool2 = tf.nn.max_pool(conv2, [1, 2, 2, 1], [1, 2, 2, 1], "VALID", name="maxpool2")

        # conv3
        conv3 = self.conv_layer(maxpool2, 128, "conv3")
        self.layers["conv3"] = conv3
        # conv4
        conv4 = self.conv_layer(conv3, 128, "conv4")
        self.layers["conv4"] = conv4
        # fully connection
        shape = conv4.get_shape().as_list()[1:]
        before_fc = tf.reshape(conv4, [-1, int(np.prod(shape))])

        fc1 = tf.layers.dense(before_fc, 1024, kernel_initializer=tf.contrib.layers.variance_scali
                              name="fc1", trainable=self.is_train(),
                              kernel_regularizer=tf.contrib.layers.l2_regularizer(reg_const))
        # fc2 = tf.layers.dense(fc1, 1024, kernel_initializer=tf.contrib.layers.variance_scaling_i
        #                       name="fc2", trainable=self.is_train())
        fc2 = tf.layers.dense(fc1, self.num_classes,
                              kernel_initializer=tf.contrib.layers.variance_scaling_initializer(),
                              name="fc2", trainable=self.is_train(),
                              kernel_regularizer=tf.contrib.layers.l2_regularizer(reg_const))
        return fc2
```

2. data_loader.py

```
import os
import pickle

training_file = os.path.join("data", "train.p")
validation_file = os.path.join("data", "valid.p")
testing_file = os.path.join("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_validation, y_validation = valid["features"], valid["labels"]
x_test, y_test = valid["features"], valid["labels"]

if __name__ == '__main__':
    num_train = len(y_train)
    num_test = len(y_test)
    num_validation = len(y_validation)
    print("train samples:\t%d" % num_train)
    print("validation samples:\t%d" % num_validation)
    print("test  samples:\t%d" % num_test)
    print()
    print("shape of traffic sign image:", end="\t")
    print(x_train[0].shape)
    print("number of classes/labels:", end="\t")
    print(max(y_train) - min(y_train) + 1)
```

3. data_feeder.py

```
import cv2
import numpy as np


class DataFeeder:
    def __init__(self, images=None, labels=None, batch_size=1):
        self.images = images
        self.labels = labels
        self.num_classes = np.max(labels) + 1
```

```
        self.batch_size = batch_size

    def set_images(self, images):
        self.images = images

    def set_labels(self, labels):
        self.labels = labels

    def set_num_classes(self, num_classes):
        self.num_classes = num_classes

    def set_batch_size(self, batch_size):
        self.batch_size = batch_size

    @staticmethod
    def brightness_augment(img, factor=0.2):
        img_hsv = np.array(cv2.cvtColor(img, cv2.COLOR_RGB2HSV), dtype=np.float64)
        img_hsv[:, :, 2] = img_hsv[:, :, 2] * np.random.uniform(1 - factor, 1 + factor)
        img_hsv[:, :, 2][img_hsv[:, :, 2] > 255] = 255
        img_rgb = cv2.cvtColor(np.array(img_hsv, dtype=np.uint8), cv2.COLOR_HSV2RGB)
        return img_rgb

    @staticmethod
    def flip_augment(img, factor=0.5):
        if np.random.random() > factor:
            return np.fliplr(img)
        else:
            return img

    @staticmethod
    def add_noise(img, factor=0.15):
        width, height, channel = img.shape
        noise_arr = (np.random.rand(width, height, channel) - 0.5) * factor * 255.0
        img_noise = img + noise_arr
        return np.minimum(np.maximum(img_noise, 0.0), 255.0)

    @staticmethod
    def crop_augment(img, factor=0.5, limit=3):
        if np.random.random() > factor:
            shape_orig = img.shape
            x_left = np.random.randint(0, limit)
            x_right = np.random.randint(shape_orig[0] - limit, shape_orig[0])
            y_top = np.random.randint(0, limit)
            y_bottom = np.random.randint(shape_orig[1] - limit, shape_orig[1])
            img_crop = img[x_left:x_right, y_top:y_bottom, :]
```

```
            img_crop_t = cv2.resize(img_crop, dsize=shape_orig[0:2],interpolation=cv2.INTER_LINEAR
            return img_crop_t
        else:
            return img

    def next_batch(self):
        batch_index = np.random.choice(np.arange(self.labels.size), size=self.batch_size, replace=
        labels_batch = self.labels[batch_index]
        data_batch_orig = self.images[batch_index]
        data_batch_ls = []
        for i in range(self.batch_size):
            img_aug = self.brightness_augment(data_batch_orig[i])
            img_aug = self.add_noise(img_aug)
            if labels_batch[i] in [9, 11, 12, 13, 15, 17, 18, 22, 26, 29, 30, 35]:
                img_aug = self.flip_augment(img_aug)
            img_aug = self.crop_augment(img_aug)
            data_batch_ls.append(img_aug)
        data_batch = np.stack(data_batch_ls)
        return data_batch, labels_batch


if __name__ == '__main__':
    import data_loader

    x_train, y_train = data_loader.x_train, data_loader.y_train
    for i in range(43):
        y_sub = y_train[y_train == i]
        x_sub = x_train[y_train == i]
        sub_index = np.random.choice(np.arange(y_sub.size), size=2, replace=False)
        x_t = x_sub[sub_index]
        for j in range(len(x_t)):
            img_t = np.squeeze(x_t[j])
            img_r = cv2.cvtColor(img_t, cv2.COLOR_RGB2BGR)
            name = "pic/%02d_%02d.jpg" % (i, j)
            cv2.imwrite(name, img_r)

    data_feeder = DataFeeder(x_train, y_train, batch_size=4)

    data_batch, labels_batch = data_feeder.next_batch()
    print()
```

4. traffic_sign_train.py

```
import os
```

```
import tensorflow as tf
from optparse import OptionParser
import time
import numpy as np
import glog as logger
import data_loader
import data_feeder
import network


def compute_acc(preds, labels):
    acc = np.size(np.where(preds == labels)) / preds.size
    return acc


def my_parser():
    """
    parse arguments
    :return: options
    """
    parser = OptionParser()
    parser.add_option("--lr", "--learning_rate", action="store",
                      dest="learning_rate",
                      type="float", default=0.0005,
                      help="set learning_rate")
    parser.add_option("--batch_size", action="store", dest="batch_size",
                      type="int", default=32,
                      help="set batch size")
    parser.add_option("-w", "--weight", action="store", dest="weight_path",
                      type="string",
                      default=None,
                      help="path to pretrain weight or previous weight")
    parser.add_option("--tboard", action="store", dest="tboard",
                      type="string", default="tboard",
                      help="set tensor board log directory")
    parser.add_option("-n", "--steps", action="store", dest="steps",
                      default=10000, type="int", help="set train steps")
    parser.add_option("-s", "--save_path", action="store", dest="save_path",
                      type="string", default="model",
                      help="set model save path")

    options, _ = parser.parse_args()
    return options


```

```
47 def traffic_sign_train(lr, batch_size, weight_path, train_epochs, tboard_dir, save_path):
48     # load data
49     x_train, y_train = data_loader.x_train, data_loader.y_train
50     x_validation, y_validation = data_loader.x_validation, data_loader.y_validation
51     assert len(x_train) == len(y_train)
52     data_feed = data_feeder.DataFeeder(x_train, y_train, batch_size=batch_size)
53     # set configuration
54 
55     # construct network structure
56     input_layer = tf.placeholder(dtype=tf.float32, shape=[batch_size, 32, 32, 3], name="input")
57     labels = tf.placeholder(dtype=tf.int32, shape=[batch_size], name="labels")
58 
59     traffic_sign_net = network.traffic_sign_network(phase="TRAIN", num_classes=43)
60     loss, pred = traffic_sign_net.loss(labels=labels, input_data=input_layer)
61 
62     # set learning rate
63     global_step = tf.Variable(0, name="global_step", trainable=False)
64 
65     learning_rate = tf.Variable(lr, trainable=False)
66     optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
67 
68     grad = optimizer.compute_gradients(loss=loss)
69 
70     apply_grad_op = optimizer.apply_gradients(grad, global_step=global_step)
71 
72     # set tensorflow summary
73     tboard_save_path = tboard_dir
74     os.makedirs(tboard_save_path, exist_ok=True)
75     summary = tf.summary.FileWriter(tboard_save_path)
76 
77     train_loss_scalar = tf.summary.scalar(name="train_loss", tensor=loss)
78     learning_rate_scalar = tf.summary.scalar(name="learning_rate", tensor=learning_rate)
79     train_summary_op_updates = tf.get_collection(tf.GraphKeys.SUMMARIES)
80     # train_merge_summary_op = tf.summary.merge([train_loss_scalar, learning_rate_scalar], train_s
81     train_merge_summary_op = tf.summary.merge_all()
82 
83     # set saver
84     os.makedirs(save_path, exist_ok=True)
85 
86     saver = tf.train.Saver(max_to_keep=10)
87     train_start_time = time.strftime('%Y-%m-%d-%H-%M-%S', time.localtime(time.time()))
88 
89     # set sess config
90     sess_config = tf.ConfigProto(allow_soft_placement=True)
91     sess_config.gpu_options.per_process_gpu_memory_fraction = 0.5
```

```
92
93      sess = tf.Session(config=sess_config)
94      summary.add_graph(sess.graph)
95
96      # start training
97      with sess.as_default():
98          epoch = 0
99          if weight_path is None:
100             logger.info("Training from scratch")
101             init = tf.global_variables_initializer()
102             sess.run(init)
103         else:
104             logger.info("Restore model from {:s}".format(weight_path))
105             saver.restore(sess=sess, save_path=weight_path)
106         train_loss_list = []
107         train_acc_list = []
108         while epoch < train_epochs:
109             epoch += 1
110             data_batch, labels_batch = data_feed.next_batch()
111             _, train_loss, pred_label, train_merge_summary_value = sess.run(
112                 [apply_grad_op, loss, pred, train_merge_summary_op],
113                 feed_dict={input_layer: data_batch,
114                            labels: labels_batch})
115             acc = compute_acc(preds=pred_label, labels=labels_batch) * 100
116             train_loss_list.append(train_loss)
117             train_acc_list.append(acc)
118             if epoch % 100 == 0:
119                 acc = sum(train_acc_list) / len(train_acc_list)
120                 train_loss = sum(train_loss_list) / len(train_loss_list)
121                 train_acc_list = []
122                 train_loss_list = []
123                 logger.info("epoch: {:d}\ttrain loss: {:.3f}\ttrain acc: {:.2f}%%".format(epoch, t
124                 summary.add_summary(summary=train_merge_summary_value, global_step=epoch)
125             if epoch % 500 == 0:
126                 # validation
127                 val_acc_list = []
128                 val_loss_list = []
129                 batch_num = int(np.floor(y_validation.size / batch_size))
130                 for i in range(batch_num):
131                     data_batch = x_validation[i * batch_size:(i + 1) * batch_size]
132                     labels_batch = y_validation[i * batch_size:(i + 1) * batch_size]
133                     val_loss, pred_label = sess.run([loss, pred], feed_dict={input_layer: data_bat
134                                                                              labels: labels_batch}
135                     val_acc = compute_acc(preds=pred_label, labels=labels_batch)
136                     val_acc_list.append(val_acc)
```

```
                    val_loss_list.append(val_loss)
                acc = sum(val_acc_list) / len(val_acc_list) * 100
                val_loss = sum(val_loss_list) / len(val_loss_list)
                logger.info("epoch: {:d}\tval loss: {:.3f}\tval acc: {:.2f}%%".format(epoch, val_l

                model_name = 'traffic_sign_{:s}_{:06d}.ckpt'.format(str(train_start_time), epoch)
                model_save_path = os.path.join(save_path, model_name)
                saver.save(sess=sess, save_path=model_save_path)
        model_name = 'traffic_sign_{:s}_{:06d}.ckpt'.format(str(train_start_time), epoch)
        model_save_path = os.path.join(save_path, model_name)
        saver.save(sess=sess, save_path=model_save_path)


if __name__ == '__main__':
    init_opt = my_parser()
    traffic_sign_train(lr=init_opt.learning_rate, batch_size=init_opt.batch_size, weight_path=init
                       train_epochs=init_opt.steps, tboard_dir=init_opt.tboard, save_path=init_opt

    logger.info("done!")
```

5. traffic_sign_test.py

```
import os
import time
import numpy as np
from optparse import OptionParser
import tensorflow as tf
import glog as logger

import data_loader
import network


def compute_acc(preds, labels):
    acc = np.size(np.where(preds == labels)) / preds.size
    return acc


def my_parser():
    """
    parse arguments
    :return: options
    """
    parser = OptionParser()
```

```
    parser.add_option("--batch_size", action="store", dest="batch_size",
                      type="int", default=1,
                      help="set batch size")
    parser.add_option("-w", "--weight", action="store", dest="weight_path",
                      type="string",
                      default=None,
                      help="path to model weight")

    options, _ = parser.parse_args()
    return options


def traffic_sign_test(batch_size, weight_path):
    # load data
    test_data, test_labels = data_loader.x_test, data_loader.y_test
    assert len(test_data) == len(test_labels)

    # set configuration

    # construct network structure
    input_layer = tf.placeholder(dtype=tf.float32, shape=[batch_size, 32, 32, 3], name="input")
    labels = tf.placeholder(dtype=tf.int32, shape=[batch_size], name="labels")
    traffic_sign_net = network.traffic_sign_network(phase="test", num_classes=43)

    _, pred = traffic_sign_net.loss(labels=labels, input_data=input_layer)

    saver = tf.train.Saver()

    # set sess config
    sess_config = tf.ConfigProto(allow_soft_placement=True)
    sess_config.gpu_options.per_process_gpu_memory_fraction = 0.4

    sess = tf.Session(config=sess_config)

    # start training
    with sess.as_default():
        if weight_path is None:
            raise ValueError("weight path doesn't configured")

        logger.info("Restore model from {:s}".format(weight_path))
        saver.restore(sess=sess, save_path=weight_path)
        test_acc_list = []

        # omit the last few data that less than a batch size
        batch_num = int(np.floor(test_labels.size) / batch_size)
```

```
68
69          for i in range(batch_num):
70              data_batch = test_data[i * batch_size:(i + 1) * batch_size]
71              labels_batch = test_labels[i * batch_size:(i + 1) * batch_size]
72              pred_label = sess.run(pred, feed_dict={input_layer: data_batch,
73                                                     labels: labels_batch})
74              test_acc = compute_acc(preds=pred_label, labels=labels_batch)
75              test_acc_list.append(test_acc)
76          acc = sum(test_acc_list) / len(test_acc_list) * 100
77          logger.info("test acc: {:.2f}%%".format(acc))
78
79
80  if __name__ == '__main__':
81      init_opt = my_parser()
82      traffic_sign_test(batch_size=init_opt.batch_size, weight_path=init_opt.weight_path)
83      logger.info("done!")
```

6. traffic_sign_demo.py

```
1   import os
2   import time
3   import numpy as np
4   from optparse import OptionParser
5   import tensorflow as tf
6   import glog as logger
7   import cv2
8   import data_loader
9   import network
10
11
12  def my_parser():
13      """
14      parse arguments
15      :return: options
16      """
17      parser = OptionParser()
18
19      parser.add_option("-i", "--images", action="store", dest="images",
20                        type="string", default=None,
21                        help="set image path or directory")
22      parser.add_option("-w", "--weight", action="store", dest="weight_path",
23                        type="string",
24                        default=None,
25                        help="path to model weight")
```

```
    parser.add_option("-v", "--visualize", action="store_true", dest="visualize",
                      default=False,
                      help="switch to visualize network")

    options, _ = parser.parse_args()
    return options


def compute_acc(preds, labels):
    acc = np.size(np.where(preds == labels)) / preds.size
    return acc


def visualize_feature(layer_value, layer_num):
    out_dir = "demo/visualization"
    for i in range(len(layer_value)):
        shape = layer_value[i].shape
        # layer_value[i] = layer_value[i] / np.max(layer_value) * 255.0
        features = []
        for j in range(shape[-1]):
            feature = layer_value[i][:, :, j]
            feature = np.expand_dims(feature, -1)
            feature = feature / np.max(feature) * 255.0
            out_im = cv2.resize(feature, (32, 32))
            features.append(out_im)
            out_name = "im_%02d_layer_%01d_%03d.jpg" % (i + 1, layer_num, j)
            # cv2.imwrite(os.path.join(out_dir, out_name), out_im)
        features = np.stack(features[0:16])
        features = np.reshape(features, (4, 4, 32, 32, 1))
        features = np.swapaxes(features, 1, 2)
        features = np.reshape(features, (4 * 32, 4 * 32, 1))
        out_name = "im_%02d_layer_%01d.jpg" % (i + 1, layer_num)
        cv2.imwrite(os.path.join(out_dir, out_name), features)


def traffic_sign_demo_with_test_images(weight_path, visualize=False):
    # prepare output dir
    if os.path.exists("demo/test"):
        os.system("rm -rf demo/test/*")
    else:
        os.mkdir("demo/test")
    if os.path.exists("demo/visualization"):
        os.system("rm -rf demo/visualization/*")
    else:
        os.mkdir("demo/visualization")
```

```
    if visualize:
        out_dir = "demo/visualization"
        if os.path.exists(out_dir):
            os.system("rm -rf %s/*" % out_dir)
        else:
            os.mkdir(out_dir)

    batch_size = 10
    # load data
    test_data, test_labels = data_loader.x_test, data_loader.y_test
    assert len(test_data) == len(test_labels)

    # random pick 10 images
    batch_index = np.random.choice(np.arange(len(test_labels)), size=10, replace=False)
    labels_batch = test_labels[batch_index]
    image_batch = test_data[batch_index]
    # save images
    for i in range(len(image_batch)):
        img_bgr = cv2.cvtColor(image_batch[i], cv2.COLOR_RGB2BGR)
        cv2.imwrite("demo/test/img_%02d_%02d.jpg" % (i, labels_batch[i]), img_bgr)

    # set configuration

    # construct network structure
    input_layer = tf.placeholder(dtype=tf.float32, shape=[batch_size, 32, 32, 3], name="input")
    traffic_sign_net = network.traffic_sign_network(phase="test", num_classes=43)

    pred, raw = traffic_sign_net.inference(input_data=input_layer)
    layers = traffic_sign_net.layers

    saver = tf.train.Saver()

    # set sess config
    sess_config = tf.ConfigProto(allow_soft_placement=True)
    sess_config.gpu_options.per_process_gpu_memory_fraction = 0.4

    sess = tf.Session(config=sess_config)

    # start training
    with sess.as_default():
        if weight_path is None:
            raise ValueError("weight path doesn't configured")

        logger.info("Restore model from {:s}".format(weight_path))
        saver.restore(sess=sess, save_path=weight_path)
```

```
116         if not visualize:
117             pred_label, raw_label = sess.run([pred, raw], feed_dict={input_layer: image_batch})
118         else:
119             out_items = [pred, raw]
120             for name_layer, layer in layers.items():
121                 out_items.append(layer)
122             out_values = sess.run(out_items, feed_dict={input_layer: image_batch})
123             pred_label = out_values[0]
124             raw_label = out_values[1]
125             layers_value = out_values[2:]
126             for layer_num, layer_v in enumerate(layers_value):
127                 visualize_feature(layer_v, layer_num + 1)
128 
129         for i in range(len(pred_label)):
130             max_p_ls = np.argsort(-raw_label[i])[0:5]
131             print("image %02d" % i, end="\t")
132             print("label %02d" % labels_batch[i], end="\t")
133             print("predict %02d" % pred_label[i])
134             print("top 5 softmax probabilities:", end="\t")
135             print(max_p_ls)
136 
137         test_acc = compute_acc(preds=pred_label, labels=labels_batch)
138         acc = test_acc * 100
139         logger.info("test acc: {:.2f}%%".format(acc))
140 
141 
142 def traffic_sign_demo_with_show_images(images, weight_path):
143     # prepare output dir
144 
145     if os.path.exists("demo/visualization"):
146         os.system("rm -rf demo/visualization/*")
147     else:
148         os.mkdir("demo/visualization")
149 
150     # load demo images
151     images_ls = []
152     image_batch = []
153     if os.path.isdir(images):
154         images_list = os.listdir(images)
155         images_list.sort()
156         images_ls = [os.path.join("demo/show/", img_name) for img_name in images_list]
157     else:
158         images_ls.append(images)
159 
160     for img_name in images_ls:
```

```
161             im = cv2.imread(img_name)
162             im_t = cv2.cvtColor(cv2.resize(im, (32, 32)), cv2.COLOR_BGR2RGB)
163             image_batch.append(im_t)
164 
165         # # save images
166         # for i in range(len(image_batch)):
167         #     img_bgr = cv2.cvtColor(image_batch[i], cv2.COLOR_RGB2BGR)
168         #     cv2.imwrite("demo/out/img_show_%02d.jpg" % (i + 1), img_bgr)
169 
170         # set configuration
171         batch_size = len(image_batch)
172         # construct network structure
173         input_layer = tf.placeholder(dtype=tf.float32, shape=[batch_size, 32, 32, 3], name="input")
174         traffic_sign_net = network.traffic_sign_network(phase="test", num_classes=43)
175 
176         pred, raw = traffic_sign_net.inference(input_data=input_layer)
177 
178         saver = tf.train.Saver()
179 
180         # set sess config
181         sess_config = tf.ConfigProto(allow_soft_placement=True)
182         sess_config.gpu_options.per_process_gpu_memory_fraction = 0.4
183 
184         sess = tf.Session(config=sess_config)
185 
186         # start training
187         with sess.as_default():
188             if weight_path is None:
189                 raise ValueError("weight path doesn't configured")
190 
191             logger.info("Restore model from {:s}".format(weight_path))
192             saver.restore(sess=sess, save_path=weight_path)
193 
194             pred_label, raw_label = sess.run([pred, raw], feed_dict={input_layer: image_batch})
195             for i in range(len(pred_label)):
196                 max_p_ls = np.argsort(-raw_label[i])[0:5]
197                 print("image %02d" % (i + 1), end="\t")
198                 print("predict %02d" % pred_label[i])
199                 print("top 5 softmax probabilities:", end="\t")
200                 print(max_p_ls)
201 
202 
203 if __name__ == '__main__':
204     opt_init = my_parser()
205     if opt_init.images is None:
```

```
206             traffic_sign_demo_with_test_images(opt_init.weight_path, opt_init.visualize)
207         else:
208             traffic_sign_demo_with_show_images(opt_init.images, opt_init.weight_path)
209         logger.info("Done!")
```