CSCI 677 Homework 6

Jingyun Yang

Student ID: 8357011225

1 Introduction

In this assignment, we train and test a convolutional neural network for performing image segmentation on the KITTI dataset. In section 2, we provide a summary of the program we write, including details of the program, the directory structure, and all the sources used in our program; in section 3, we show the loss curves produced by our program; in section 4, we present and provide insights to our results, including evaluation of our trained model, analysis of influences of parameter choices on the model, and visualizations of the results.

2 Summary of Program

2.1 Detailed Information of Program

The detailed information of our program is listed in the following bulleted list:

- **Architecture:** we implemented FCN-32s as our model for semantic segmentation.
- **Weight Initialization:** we used Xavier initializer for all kernel initializations.
- **Loss:** we used sigmoid cross entropy loss for our training, validation, and testing losses.
- **Evaluation Metric:** we used pixel-level IOU to evaluate the final performance of our model.
- **Preprocessing:** we normalized all images to pixel value range of $[0, 1]$.
- **Training Parameters:** we used learning rate of 0.002, batch size of 1, and momentum of 0.99 for our training process.

2.2 Directory Structure

Under the program directory, `main.py` is the main python program that reads data from `data` using functions in `data_util.py`, trains a CNN defined in `model.py` using configurations provided by `configs.py`, and outputs results to directory `train_dir`; `test.py` is the program we use to evaluate our program; `ops.py` defines building blocks of the CNN model used by `model.py`; `plot.ipynb` takes raw test results produces plots from the results; `data` is the directory containing the KITTI road semantic segmentation dataset; `train_dir` is the directory with all checkpoints, Tensorboard summaries, and logs of all experiments. The directory tree, with `hw6` as the base directory name, is shown below.

```
hw6/
|-- main.py
|-- test.py
|-- ops.py
|-- configs.py
|-- model.py
|-- data_util.py
|-- plot.ipynb
```

```
|-- data/
|-- train_dir/
```

2.3 Executing the Program

To execute the program, run `python3 main.py` in the base directory, possibly with custom command line arguments. Outputs will be saved to diretory `train_dir`. For more details of what command line arguments are available, inspect `configs.py`.

To rerun the experiment we executed, run `python3 main.py --learning_rate 0.002 --max_steps 28000`.

2.4 Source Listing

Since the source code is very long, we include them in the appendix of this document for this time.

3 Loss Curves

Here we present the loss curves produced by our program. We The curves are shown in Figure 1. The blue curve corresponds to the loss curve for training set; while the orange curve corresponds to the loss curve for validation set. Note that in order to make visual inspections easier, we smoothed both the training and the validation loss curves.

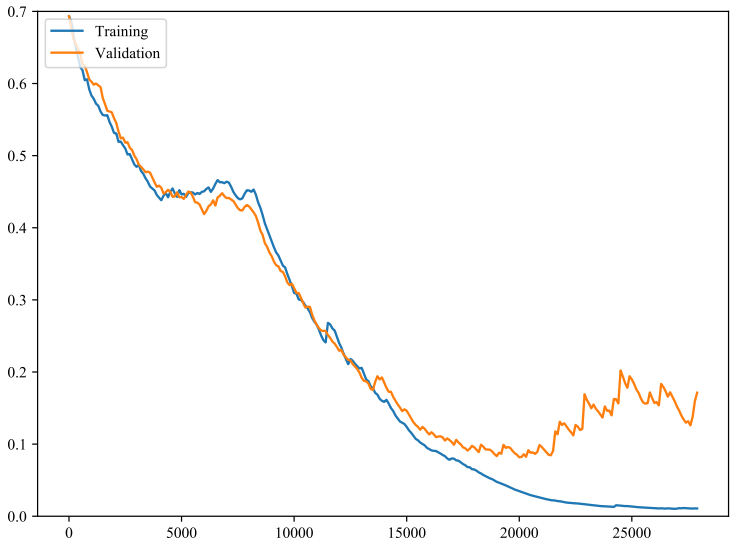


Figure 1: Loss Curve

4 Discussion and Analysis

Here, we discuss the results of our experiments.

4.1 Quantitative Results

Our model achieves **pixel-level IOU value of 85.1%** and **sigmoid cross-entropy loss of 0.238** on the testing set of KITTI Road Semantic Segmentation Dataset. The evaluation results show that our model does a good job in distinguishing "road" pixels from "non-road" pixels. We are confident that if more complicated models such as FCN-16s and FCN-8s are used, the evaluation results of the segmentation will be even better.

Detailed evaluation results are shown below in Table 1.

ID	Loss	Pixel-level IOU	TP	FP	FN
um_000000	0.09766	0.81856	49694	10868	147
um_000001	0.03833	0.92021	55946	3932	919
um_000002	0.01658	0.95019	51580	1864	840
um_000003	0.02439	0.93414	54008	2896	912
um_000004	0.06576	0.91333	74043	3063	3963
um_000005	0.09647	0.90868	86623	4275	4430
um_000006	0.04532	0.92229	57540	1847	3001
um_000007	0.62205	0.83445	64775	906	11945
um_000008	0.02223	0.94262	58895	2609	976
um_000009	0.02077	0.95580	57996	1711	971
um_000010	0.03691	0.94033	59205	887	2870
um_000011	0.47598	0.68558	54778	18400	6722
um_000012	0.97590	0.75775	54849	2415	15120
um_000013	0.02987	0.93325	57144	2637	1450
um_000014	0.01615	0.96602	62693	449	1756
umm_000000	0.11567	0.89711	75897	706	7999
umm_000001	0.41592	0.78199	68781	1115	18060
umm_000002	0.24416	0.82573	71487	91	14996
umm_000003	0.52204	0.84656	86386	70	15587
umm_000004	0.25910	0.84287	88102	1337	15087
umm_000005	0.85548	0.79386	71675	201	18411
umm_000006	0.18117	0.79833	46784	3178	8640
umm_000007	0.11625	0.83305	44769	2110	6862
umm_000008	0.51290	0.77335	46028	601	12889
umm_000009	0.57455	0.76949	74098	216	21981
umm_000010	0.10409	0.86483	63259	1018	8869
umm_000011	0.06149	0.88735	64734	4494	3724
umm_000012	0.11944	0.84335	82669	14779	576
umm_000013	0.96600	0.77356	85724	1389	23704
umm_000014	0.14650	0.86768	78302	2257	9684
uu_000000	0.36918	0.62658	55082	31760	1067
uu_000001	0.02110	0.94064	57082	2394	1208
uu_000002	0.12420	0.77943	51779	12967	1686
uu_000003	0.05925	0.87779	55032	4081	3581
uu_000004	0.03141	0.93958	55143	1533	2013
uu_000005	0.01857	0.94947	57331	2172	879
uu_000006	0.35181	0.79812	66573	7717	9122
uu_000007	0.35651	0.83144	68660	4847	9073
uu_000008	0.10322	0.88318	76155	3866	6207
uu_000009	0.13424	0.85981	68418	4022	7133
uu_000010	0.66602	0.63049	48565	2435	26028
uu_000011	0.66567	0.85880	55023	1183	7864
uu_000012	0.01027	0.96941	55147	660	1080
uu_000013	0.01186	0.96398	54495	1476	560
uu_000014	0.00937	0.97032	58264	1370	412

Table 1: Detailed Evaluation Results

4.2 Effects of Parameter Choices

We found during the experiments that it was very hard for the training to converge. For example, in one run, when we were running with learning rate of 0.001, the network still didn't converge after 28,000 steps (or approximately 140 epochs). The loss curve for this run is shown below in Figure 2. To make the network converge faster, we increased the learning rate of the network to 0.002. The network appeared to converge faster than before when the learning rate was 0.001.

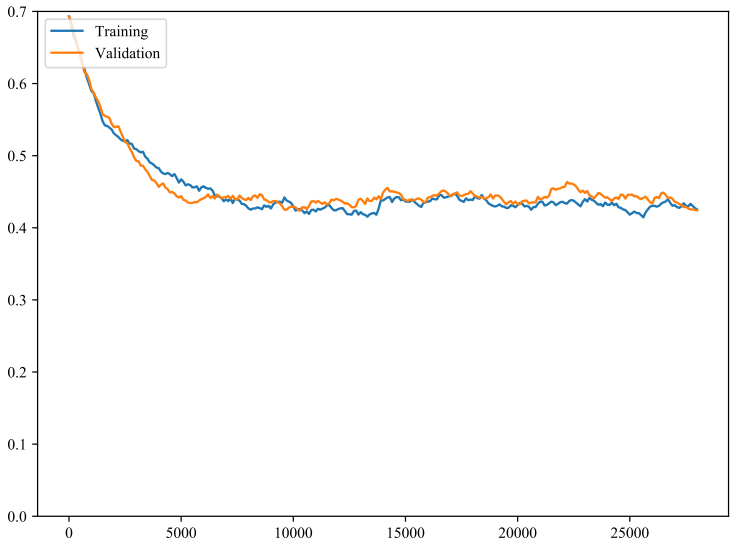


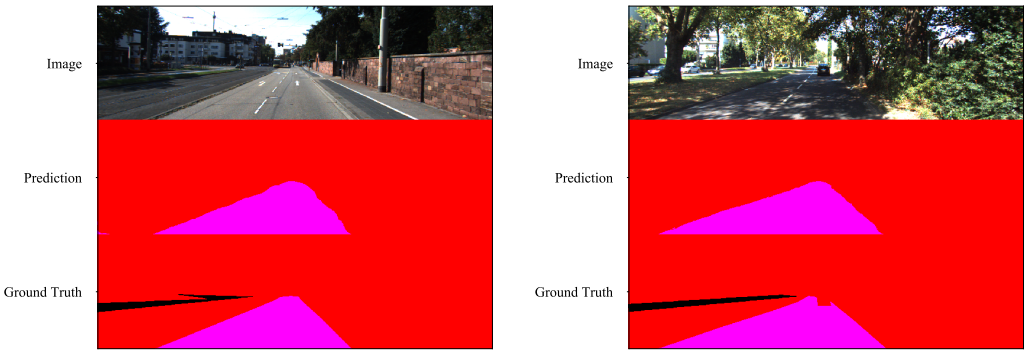
Figure 2: Loss Curve for Failed Experiment in which Network Didn't Converge

4.3 Visualization of Success and Failure Examples

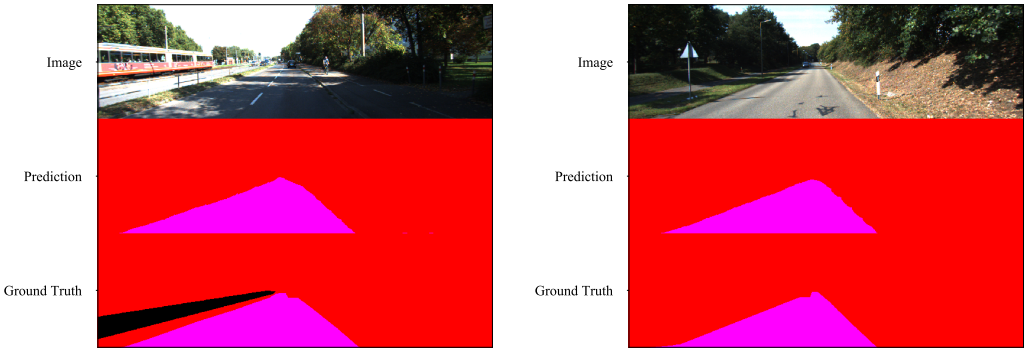
Now we present some success and failure examples of our model. In Figure 3, we can see that our model performs extremely well in some cases, producing almost the exact same segmentation prediction as the ground truth segmentations.

There are, however, some cases in which the model is not performing as well. In Figure 4, we can see that our model makes errors in predicting road and non-road pixels in some cases. From the two images on the left side of the figure, we can see that some of these errors come from the samples in which there are side walks or parking lots whose grounds resemble those of the roads. In these circumstances, it makes sense that the model might make mistakes by classifying the road-like pixels as road. From the image on the top-right side of the figure, we can see that the model is disturbed by the railroad that cuts the road in the middle. In this situation, the model is confused whether the railroad part of the image should be classified as road or not, resulting in low IOU. For the image on the lower-right side of the figure, the network failed to see that at the bottom-right corner of the image another road is merging with the road at the center of the image.

In summary, the network is not able to figure out some extremely tricky cases, but generally it performs reasonably well in this road semantic segmentation task.



um_000002 (IOU: 95.02%) um_000009 (IOU: 95.58%)



um_000014 (IOU: 96.60%) uu_000012 (IOU: 96.94%)

Figure 3: Success Examples

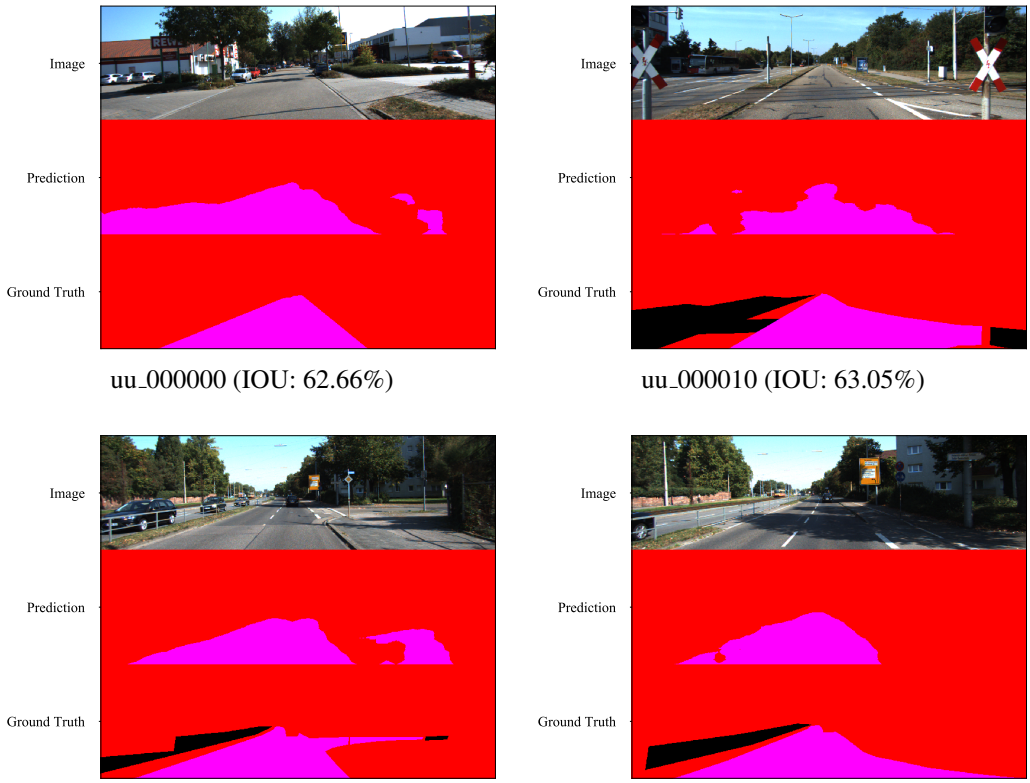


Figure 4: Failure Examples

A Appendix: Source Listing

A.1 `main.py`

```
import tensorflow as tf
import os
import os.path as osp
import time
import pickle

from data_util import *
from model import Model
from configs import config

def main():
    # limit GPU's visible by the program
    os.environ["CUDA_DEVICE_ORDER"] = "PCI_BUS_ID"
    os.environ["CUDA_VISIBLE_DEVICES"] = str(config.gpu)
    print("Using GPU {}.".format(config.gpu))

    # load data
    print("Loading datasets...")
    dataset_train, dataset_val, dataset_test = load_datasets('data')
    config.data_info = list(dataset_train['image'].shape[1:])
    print("Dataset loading completes.")

    # create log directory
    hyper_parameter_str = 'bs_{}_lr_{}'.format(
        config.batch_size,
        config.learning_rate,
    )

    train_dir = './train_dir/{}_{}_{}_{}'.format(
        'kitti',
        config.prefix,
        hyper_parameter_str,
        time.strftime("%Y%m%d-%H%M%S")
    )

    if not osp.exists(train_dir): os.makedirs(train_dir)
    print("Train Dir: {}".format(train_dir))

    # reset default graph
    tf.reset_default_graph()

    # create model
    model = Model(config)

    # training setups
    saver = tf.train.Saver(max_to_keep=100)
    summary_writer = tf.summary.FileWriter(train_dir)
    session_config = tf.ConfigProto(
        gpu_options=tf.GPUOptions(allow_growth=True),
        device_count={'GPU': 1}
    )

    with tf.Session(config=session_config) as sess:
        sess.run(tf.global_variables_initializer())
        sess.run(tf.local_variables_initializer())

        # buffers for train and val losses
        train_losses = []
        val_losses = []

        # train model
        for step in range(config.max_steps):
            # run validation step
```

```
            if step % config.val_step == 0:
                val_batch = sample_batch(dataset_val, config.batch_size)
                val_stats = run_single_step(sess, model, val_batch,
                                            summary_writer,
                                            mode='val')
                val_losses.append([val_stats['step'], val_stats['loss']])

            # run train step
            train_batch = sample_batch(dataset_train, config.batch_size)
            train_stats = run_single_step(sess, model, train_batch,
                                          summary_writer,
                                          mode='train',
                                          log=step % config.log_step == 0)
            train_losses.append([train_stats['step'], train_stats['loss']])

            # save checkpoint
            if step % config.save_checkpoint_step == 0:
                print("Saved checkpoint at step {}".format(step))
                saver.save(sess, os.path.join(train_dir, 'model'), global_step=step)

        # test model
        test_logfile = os.path.join(train_dir, 'test_result.txt')
        n_test = dataset_test['image'].shape[0]
        ind_exp_results = {
            "loss": [],
            "tp": [],
            "fp": [],
            "fn": [],
            "pixel_level_iou": [],
            "pred": [],
            "probs": []
        }
        for i in range(n_test):
            test_batch = {key: dataset_test[key][[i]] for key in dataset_test}
            exp_results = run_single_step(sess, model, test_batch,
                                          summary_writer,
                                          mode='test',
                                          test_logfile=test_logfile)
            for key in exp_results:
                ind_exp_results[key].append(exp_results[key])
        exp_results = {}
        for key in ind_exp_results:
            exp_results[key] = np.stack(ind_exp_results[key])
        exp_results["loss"] = np.mean(exp_results["loss"])
        exp_results["tp"] = np.sum(exp_results["tp"])
        exp_results["fp"] = np.sum(exp_results["fp"])
        exp_results["fn"] = np.sum(exp_results["fn"])
        exp_results["pixel_level_iou"] = exp_results["tp"] / (
                exp_results["tp"] + exp_results["fp"] + exp_results["fn"])
        print("Test Average Loss: {:3f}; Test IOU: {:3f}".format(
                exp_results["loss"], exp_results["pixel_level_iou"]))

        # add loss curves to experiment results
        exp_results['train_losses'] = train_losses
        exp_results['val_losses'] = val_losses

        # log test results
        with open(os.path.join(train_dir, 'test_result.p'), 'wb') as f:
            pickle.dump(exp_results, f)
            print("Logged experiment results to {}".format(f.name))

        # flush Tensorboard summaries
        summary_writer.flush()

def run_single_step(session, model, batch, summary_writer,
                    mode='train', log=True, test_logfile=None):
    # construct feed dict
    feed_dict = {
```

```
        model.images: batch['image'],
        model.labels: batch['gt'],
        model.is_training: mode == 'train'
    }

    # select proper summary op
    if mode == 'train':
        summary_op = model.train_summary_op
    elif mode == 'val':
        summary_op = model.val_summary_op
    else:
        summary_op = model.test_summary_op

    # construct fetch list
    fetch_list = [model.global_step, summary_op,
                  model.loss, model.pixel_level_iou,
                  model.tp, model.fp, model.fn]
    if mode == 'train':
        fetch_list.append(model.optimizer_op)

    # run single step
    _start_time = time.time()
    _step, _summary, _loss, _iou, _tp, _fp, _fn = session.run(fetch_list, feed_dict=feed_dict)[:7
                                                                                               ]
    _end_time = time.time()

    # collect step statistics
    step_time = _end_time - _start_time
    batch_size = batch['image'].shape[0]

    # log in console
    if log:
        print(('[{:5s} step {:4d}] loss: {:.5f}; iou: {:.5f}; tp: {:5d}; fp: {:5d}; fn: {:5d} ' +
               '({:.3f} sec/batch; {:.3f} instances/sec)'
               ).format(mode, _step, _loss, _iou, _tp, _fp, _fn,
                        step_time, batch_size / step_time))

        # log in Tensorboard
        summary_writer.add_summary(_summary, global_step=_step)

    # log results to file and return statistics
    if mode == 'test':
        test_fetch_list = [model.loss, model.pixel_level_iou, model.pred, model.probs]
        _loss, _iou, _pred, _probs = \
                session.run(test_fetch_list, feed_dict=feed_dict)

        # Log detailed test results to file
        with open(os.path.join(test_logfile), 'w+') as f:
            f.write("Loss: {:.3f} \n".format(_loss))
            f.write("Pixel-level IOU: {:.3f} \n".format(_iou))
            print("Logged test results to {}".format(f.name))

        # Log detailed test results in pickle format
        stats = {
            "loss": _loss,
            "tp": _tp,
            "fp": _fp,
            "fn": _fn,
            "pixel_level_iou": _iou,
            "pred": _pred,
            "probs": _probs
        }
    else:
        stats = {
            "step": _step,
            "loss": _loss,
            "pixel_level_iou": _iou
        }
```

```
    return stats

def sample_batch(dataset, batch_size):
    N = dataset['image'].shape[0]
    # indices = list(range(batch_size))
    indices = np.random.randint(N, size=batch_size)
    return {key: dataset[key][indices] for key in dataset}

if __name__ == '__main__':
    main()
```

A.2 test.py

```
import tensorflow as tf
import os
import os.path as osp
import time
import pickle

from data_util import *
from model import Model
from configs import config
from main import run_single_step

def main():
  # limit GPU's visible by the program
  os.environ["CUDA_DEVICE_ORDER"] = "PCI_BUS_ID"
  os.environ["CUDA_VISIBLE_DEVICES"] = str(config.gpu)
  print("Using GPU {}.".format(config.gpu))

  # load data
  print("Loading datasets")
  _, _, dataset_test = load_datasets('data')
  config.data_info = list(dataset_test['image'].shape[1:])
  print("Dataset loading completes.")

  tf.reset_default_graph()

  model = Model(config)

  session_config = tf.ConfigProto(
    gpu_options=tf.GPUOptions(allow_growth=True),
    device_count={'GPU': 1}
  )

  all_vars = tf.trainable_variables()

  ckpt_path = config.checkpoint
  ckpt_dir = osp.dirname(ckpt_path)
  summary_writer = tf.summary.FileWriter(ckpt_dir)

  with tf.Session(config=session_config) as sess:
    sess.run(tf.global_variables_initializer())
    sess.run(tf.local_variables_initializer())

    # restore checkpoint
    if ckpt_dir is not None:
      print("Checkpoint path: %s", ckpt_path)
      tf.train.Saver(var_list=all_vars).restore(
        sess, ckpt_path)
      print("Loaded the parameters from the provided checkpoint path")
    else:
      print("Please provide checkpoint path.")
      exit(0)
```

```
    test_logfile = os.path.join(ckpt_dir, 'test_result.txt')
    n_test = dataset_test['image'].shape[0]
    ind_exp_results = {
      "loss": [],
      "tp": [],
      "fp": [],
      "fn": [],
      "pixel_level_iou": [],
      "pred": [],
      "probs": []
    }
    for i in range(n_test):
      test_batch = {key: dataset_test[key][[i]] for key in dataset_test}
      exp_results = run_single_step(sess, model, test_batch,
                      summary_writer,
                      mode='test',
                      test_logfile=test_logfile)
      for key in exp_results:
        ind_exp_results[key].append(exp_results[key])
    exp_results = {}
    for key in ind_exp_results:
      exp_results[key] = np.stack(ind_exp_results[key])
    exp_results["loss"] = np.mean(exp_results["loss"])
    exp_results["tp"] = np.sum(exp_results["tp"])
    exp_results["fp"] = np.sum(exp_results["fp"])
    exp_results["fn"] = np.sum(exp_results["fn"])
    exp_results["pixel_level_iou"] = exp_results["tp"] / (
        exp_results["tp"] + exp_results["fp"] + exp_results["fn"])
    print("Test Average Loss: {:3f}; Test IOU: {:3f}".format(
        exp_results["loss"], exp_results["pixel_level_iou"]))

    # log test results
    with open(os.path.join(ckpt_dir, 'test_result.p'), 'wb') as f:
      pickle.dump(exp_results, f)
      print("Logged experiment results to {}".format(f.name))

    # flush Tensorboard summaries
    summary_writer.flush()


if __name__ == '__main__':
  main()
```

A.3 configs.py

```
import tensorflow as tf

flags = tf.app.flags

# experiment params
flags.DEFINE_string("prefix", "default", "Nickname for the experiment [default]")

# model params
flags.DEFINE_boolean("use_bilinear_deconv2d", False, "Whether to use bilinear deconv2d [False]")

# training params
flags.DEFINE_integer("max_steps", 50000, "Number of steps to train. [50000]")
flags.DEFINE_float("learning_rate", 1e-3, "Learning rate for the model. [1e-3]")
flags.DEFINE_float("momentum", 0.99, "Momentum for the optimizer. [0.99]")
flags.DEFINE_integer("batch_size", 1, "Number of images in batch [1]")

# logging params
flags.DEFINE_integer("log_step", 100, "Interval for console logging [100]")
flags.DEFINE_integer("val_step", 100, "Interval for validation [100]")
flags.DEFINE_integer("save_checkpoint_step", 500, "Interval for checkpoint saving [500]")
```

```
# checkpoint for testing
flags.DEFINE_string("checkpoint", None, "Checkpoint path for testing [None]")

# gpu
flags.DEFINE_integer("gpu", 0, "GPU to use [0]")

config = flags.FLAGS
```

A.4 model.py

```
import tensorflow as tf
from ops import *

class Model(object):
    def __init__(self, config, is_train=True):
        self.config = config
        self.input_height = self.config.data_info[0]
        self.input_width = self.config.data_info[1]
        self.c_dim = self.config.data_info[2]
        self.learning_rate = self.config.learning_rate
        self.momentum = self.config.momentum
        self.use_bilinear_deconv2d = self.config.use_bilinear_deconv2d

        self.images = tf.placeholder(name='images', dtype=tf.float32,
                                     shape=[None] + self.config.data_info)
        self.labels = tf.placeholder(name='labels', dtype=tf.int32,
                                     shape=[None] + self.config.data_info[:2])
        self.is_training = tf.placeholder(name='is_training', dtype=tf.bool,
                                          shape=[])

        self.build_model()

    def build_model(self):
        print('[Input] {}'.format(shape(self.images)))

        # logits & predictions
        logits = self.fcn_32(self.images)
        self.probs = tf.nn.sigmoid(logits)
        self.pred = tf.cast(tf.round(self.probs), tf.int32)

        print('[Logits] {}'.format(shape(logits)))
        print('[Probs] {}'.format(shape(self.probs)))
        print('[Pred] {}'.format(shape(self.pred)))

        # loss
        self.loss = tf.reduce_mean(tf.boolean_mask(
                tf.nn.sigmoid_cross_entropy_with_logits(
                    labels=tf.to_float(self.labels), logits=logits),
                tf.not_equal(self.labels, -1)
            ))

        # pixel-level iou
        sum_logical_and = lambda a, b: tf.count_nonzero(tf.logical_and(a, b))
        tp = sum_logical_and(tf.equal(self.labels, 1), tf.equal(self.pred, 1))
        fp = sum_logical_and(tf.equal(self.labels, 0), tf.equal(self.pred, 1))
        fn = sum_logical_and(tf.equal(self.labels, 1), tf.equal(self.pred, 0))
        self.tp, self.fp, self.fn = tp, fp, fn
        self.pixel_level_iou = tp / (tp + fp + fn)

        # optimization
        optimizer = tf.train.MomentumOptimizer(self.learning_rate, self.momentum)
        self.global_step = tf.train.get_or_create_global_step(graph=None)
        self.optimizer_op = optimizer.minimize(self.loss,
                                               global_step=self.global_step)

        # scalar summaries
```

```
        self.add_summary("global_step", self.global_step)
        self.add_summary("loss", self.loss)
        self.add_summary("pixel_level_iou", self.pixel_level_iou)

        # image summaries
        mask = lambda x: tf.to_float(x) * tf.to_float(self.labels >= 0)
        vis = tf.stack([
            self.images,
            self.visualize_labels(mask(self.labels), self.images),
            self.visualize_labels(self.probs, self.images),
            self.visualize_labels(self.pred, self.images)
        ]) # (4, B, H, W, C)
        vis = tf.reshape(
            tf.transpose(vis, (1, 0, 2, 3, 4)),
            (-1, 4 * self.input_height, self.input_width, self.c_dim)
        ) # (B, 4 * H, W, C)
        self.add_summary("image", vis, summary_type='image',
                         collections=['test'])
        vis = vis[:, ::3, ::3, :] # resize image
        self.add_summary("image", vis, summary_type='image',
                         collections=['train', 'val'])

        # summary ops
        self.train_summary_op = tf.summary.merge_all(key='train')
        self.val_summary_op = tf.summary.merge_all(key='val')
        self.test_summary_op = tf.summary.merge_all(key='test')

    def fcn_32(self, images, scope='fcn'):
        with tf.variable_scope(scope):
            _ = images

            blocks = [(64, 2), (128, 2), (256, 3), (512, 3), (512, 3)]
            for ix, block in enumerate(blocks):
                num_outputs, num_conv_layers = block
                for conv_layer_ix in range(num_conv_layers):
                    _ = conv2d(_, num_outputs,
                               name='conv{}_{}'.format(ix + 1, conv_layer_ix + 1))
                _ = maxpool2d(_, name='pool{}'.format(ix + 1))

            _ = conv2d(_, 4096, (7, 7), name='fc6')
            _ = conv2d(_, 4096, (1, 1), name='fc7')
            _ = conv2d(_, 1, (1, 1), name='fc8')

            if self.use_bilinear_deconv2d:
                logits = bilinear_deconv2d(_, 1, (64, 64), 32,
                                           activation=None,
                                           name='deconv9')
            else:
                logits = deconv2d(_, 1, (64, 64), 32,
                                  activation=None,
                                  name='deconv9')

        return tf.squeeze(logits, -1)

    def add_summary(self, name, value,
                    summary_type='scalar',
                    collections=['train', 'val', 'test']):
        if summary_type == 'scalar':
            for collection in collections:
                tf.summary.scalar('{}_{}'.format(collection, name),
                                  value,
                                  collections=[collection])
        elif summary_type == 'image':
            for collection in collections:
                tf.summary.image('{}_{}'.format(collection, name),
                                 value,
                                 collections=[collection])
```

```
    def visualize_labels(self, labels, images, production=True):
        rgb_labels = tf.to_float(tf.concat([tf.expand_dims(labels, -1)] * 3, -1))
        if production:
            return rgb_labels * tf.constant([0., 0., 1.]) + tf.constant([1., 0., 0.])
        else:
            return rgb_labels * images + (1.0 - rgb_labels) * images * 0.1
```

A.5 ops.py

```
import tensorflow as tf

def shape(tensor):
    return tensor.get_shape().as_list()

def conv2d(inputs, num_outputs,
           kernel_size=(3,3),
           stride=1,
           padding='same',
           activation=tf.nn.relu,
           initializer=tf.contrib.layers.xavier_initializer(),
           name=None):
    output = tf.layers.conv2d(inputs, num_outputs, kernel_size, stride,
                              padding=padding,
                              activation=activation,
                              kernel_initializer=initializer,
                              name=name)
    print('[Layer: conv2d] {} {}'.format(name, shape(output)))
    return output

def maxpool2d(inputs,
              kernel_size=(2,2),
              stride=2,
              padding='same',
              name=None):
    output = tf.layers.max_pooling2d(inputs,
                                     kernel_size,
                                     stride,
                                     padding,
                                     name=name)
    print('[Layer: maxpool2d] {} {}'.format(name, shape(output)))
    return output

def deconv2d(inputs, num_outputs,
             kernel_size=(2,2),
             stride=2,
             padding='same',
             activation=tf.nn.relu,
             initializer=tf.contrib.layers.xavier_initializer(),
             name=None):
    output = tf.layers.conv2d_transpose(inputs, num_outputs, kernel_size, stride,
                                        padding=padding,
                                        activation=activation,
                                        kernel_initializer=initializer,
                                        name=name)
    print('[layer: deconv2d] {} {}'.format(name, shape(output)))
    return output

def bilinear_deconv2d(inputs, num_outputs,
                      kernel_size=(2,2),
                      stride=2,
                      padding='same',
                      activation=tf.nn.relu,
                      initializer=tf.contrib.layers.xavier_initializer(),
                      name=None):
    h = int(inputs.get_shape()[1]) * stride
    w = int(inputs.get_shape()[2]) * stride
```

```
    with tf.variable_scope(name):
        output = tf.image.resize_bilinear(inputs, [h, w])
        output = conv2d(output, num_outputs, kernel_size, padding=padding,
                        activation=activation, initializer=initializer)
    print('[layer: bilinear-deconv2d] {} {}'.format(name, shape(output)))
    return output
```

A.6 data_util.py

```
import os.path as osp
import numpy as np
import glob
import cv2
import pickle

def load_datasets(dir):
    train_val_data = load_dataset('data', 'train')
    train_data, val_data = split_train_val(train_val_data, factor=199/244)
    test_data = load_dataset('data', 'test')

    return train_data, val_data, test_data

def load_dataset(dir, mode):
    image_files = glob.glob(osp.join(dir, "image/{}/*.png".format(mode)))
    data = {
        "id": [],
        "image": [],
        "gt": []
    }

    for image_file in image_files:
        sample_img = cv2.cvtColor(cv2.imread(image_file), cv2.COLOR_BGR2RGB)
        sample_id = image_file.split('/')[-1].split('.')[0]

        gt_filename = sample_id.split('_')[0] + '_road_' + sample_id.split('_')[1]
        gt_file = osp.join(dir, "label/{}/{}.label".format(mode, gt_filename))
        with open(gt_file, 'rb') as f:
            sample_gt = pickle.load(f, encoding='latin1')

        data["id"].append(sample_id)
        data["image"].append(sample_img)
        data["gt"].append(sample_gt)

    data = {key: np.array(data[key]) for key in data.keys()}
    data["image"] = data["image"].astype(np.float32) / 255.0

    return data

def split_train_val(data, factor=0.8):
    N = len(data['id'])
    perm = np.random.permutation(N)
    train_ix, val_ix = perm[:int(N * factor)], perm[int(N * factor):]
    train_data = {key: data[key][train_ix] for key in data}
    val_data = {key: data[key][val_ix] for key in data}
    return train_data, val_data
```

A.7 plot.ipynb

Preparation

In [1]:
```
%matplotlib inline
%load_ext autoreload
%autoreload 2
```

In [2]:
```
import pandas as pd
from matplotlib import pyplot as plt
import numpy as np
import os
import os.path as osp
import pickle
import cv2
plt.rcParams["font.family"] = "Times New Roman"
```

Generate Loss Curve

In [3]:
```
def vis_loss_curve(filename, data):
    plt.figure(figsize=(8, 6))

    def smooth_curve(scalars, weight):
        last = scalars[0]
        smoothed = list()
        for point in scalars:
            smoothed_val = last * weight + (1 - weight) * point
            smoothed.append(smoothed_val)
            last = smoothed_val
        return smoothed

    train_losses = np.transpose(data[:,[0,1]])
    plt.plot(train_losses[0], smooth_curve(train_losses[1], 0.95))

    val_losses = np.transpose(data[:,[0,2]])
    plt.plot(val_losses[0], smooth_curve(val_losses[1], 0.95))

    plt.legend(['Training', 'Validation'], loc='upper left')

    plt.savefig(filename, bbox_inches='tight')
```

In [4]:
```
df = pd.read_csv('loss_curve.csv')
data = df.as_matrix()
vis_loss_curve('loss_curve.pdf', data)
```

Get Success and Failure Examples

In [5]:
```
train_dir = 'train_dir/kitti_long_bs_1_lr_0.002_20181126-131308'
```

In [6]:
```
test_logfile = osp.join(train_dir, 'test_result.p')
with open(test_logfile, 'rb') as f:
    res = pickle.load(f)
```

In [7]:
```
print(res.keys())
```

```
dict_keys(['pred', 'loss', 'tp', 'probs', 'pixel_level_iou', 'fp', 'f
n'])
```

In [8]:
```
print('Testing Loss: {:3f}'.format(res['loss']))
print('Testing Pixel-level IOU: {:3f}'.format(
        res['pixel_level_iou']))
```

```
Testing Loss: 0.238040
Testing Pixel-level IOU: 0.850975
```

In [9]:
```
from data_util import load_datasets
_, _, dataset_test = load_datasets('data')

pred = np.squeeze(res['pred'])[:, :, :]
pred = np.stack((pred,)*3, axis=-1) * np.array([0., 0., 1.]) \
        + np.array([1., 0., 0.])

images = dataset_test["image"]
```

In [10]:
```
gt_images = []
for image_id in dataset_test['id']:
    gt_filename = image_id.split('_')[0] + '_road_' \
        + image_id.split('_')[1]
    img = cv2.imread(osp.join('data/gt/test/',
                              gt_filename + '.png'))
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    gt_images.append(img)
gt_images = np.stack(gt_images) * 255
```

In [11]:
```
for ix in range(gt_images.shape[0]):
    vis = np.concatenate((images[ix], pred[ix], gt_images[ix]),
                         axis=0)

    # setup figure
    fig, ax = plt.subplots(figsize=(12, 12))
    im = ax.imshow(vis)

    # configure axis ticks
    ax.set_yticks((np.arange(3) + 0.5) * gt_images.shape[1])
    ax.set_yticklabels(['Image', 'Prediction', 'Ground Truth'],
                       fontsize=24, position=(-0.03, 0))
    ax.get_xaxis().set_visible(False)

    fname = osp.join('test_images', 'test_image_' + str(ix) + '.pdf')
    plt.savefig(fname, bbox_inches='tight')
    plt.close(fig)
```