ECE471, Selected Topics in Machine Learning – Midterm
Learning to See in the Dark
Nikola Janjušević
October 25, 2018

Goal

This assignment was an attempt to replicate the results of the paper: Learning to See in the Dark [1], in which the group trains a fully convolutional neural network to infer a properly exposed RGB image from an input, severely underexposed, RAW image. The Seeing in the Dark (SID) group's code can be seen at `https://github.com/cchen156/Learning-to-See-in-the-Dark`

Remarks

RAW files are large. I decided to decimate my input RAW files and ground truth PNGs by a factor of 16 so that all the training images could be easily loaded into memory. It is possible that this decimation ruins the set up that makes the SID group's results possible, namely, the spatial context from the original RAW file may be lost.

Sad (approaching not sad) Results

After 100+ epochs, my network stagnated, producing only brown squares. I then changed some hyper parameters around, including input patch-size, activation function, kernel initialization, and annealed learning rates. I was then able to train to a much lower average cost, but the model still stagnates at a relatively high value. Further investigation is needed to train further. My next step, if time was permitting, would be to apply further data-augmentation (as performed in the paper). **Note**: input images are not shown as they are essentially black to the human eye.

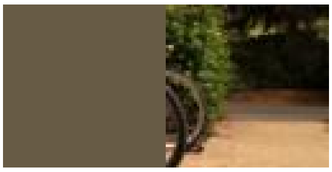

Figure 1: 20 epochs, activation fcn: relu6, patch-size: 32

Figure 2: 40 epochs, activation fcn: relu, xavier init, patch-size: 64

Figure 3: 100 epochs, activation fcn: relu, xavier init, patch-size: 64

References

[1] Chen Chen, Qifeng Chen, Jia Xu, and Vladlen Koltun, ”Learning to See in the Dark”, in CVPR, 2018. `https://arxiv.org/pdf/1805.01934.pdf`

Program: sid2.py

```
#!/usr/bin/python3
import time
import scipy as sp
import scipy.misc
import matplotlib.pyplot as plt
import pandas
import imageio
import numpy as np
import tensorflow as tf
from tqdm import tqdm
import rawpy
import os, glob
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

SIZE_REDUCTION = 16
logs_dir = './logs/'
tflog = logs_dir + 'tflog'
logfile = logs_dir + 'log.txt'
check_point_dir = './ckpnt/'
in_dir   = 'DWN%d/short/'%SIZE_REDUCTION
gt_dir   = 'DWN%d/long/'%SIZE_REDUCTION

# hyper-parameters
BATCH_SIZE = 1
NUM_EPOCHS = 4000
SAVE_RATE = 20
DISP_RATE = 1
LR_DECAY_RATE = 40
LEARNING_RATE = 1e-3
PATCH_SIZE = 64

# ----------------------------------------------
# ------------------- NETWORK ------------------
# ----------------------------------------------

# activation fcn used in architecture
def act(x):
    return tf.nn.relu(x)

def conv(x,chan,ksz):
    return tf.layers.conv2d(x,chan,ksz,padding='SAME',activation=act,
        kernel_initializer=tf.contrib.layers.xavier_initializer())

# contraction module of UNET
def contract_mod(x,chans=None):
    num_fchans = 2*int(x.shape[-1]) if chans==None else chans
    c = conv(x,num_fchans,3)
    c = conv(c,num_fchans,3)
    p = tf.layers.max_pooling2d(c,2,2,padding='SAME')
    return c,p

# upsample in1, concatenate both inputs
# output 2*c feature channels
def upsample_cat(in1,in2,c):
    ups = tf.layers.conv2d_transpose(in1,c,2,2,padding='SAME')
    x = tf.concat([ups,in2],3)
    return x

# expansion module of UNET
# input1 upsample connection, input2 concat connection
def expand_mod(input1,input2):
    out_chans= input2.get_shape()[-1]
    x = upsample_cat(input1,input2,out_chans)
    x = conv(x,out_chans,3)
```

```
    x = conv(x,out_chans,3)
    return x

# UNET ARCHITECTURE
def f(x):
    # contract
    c1,p1 = contract_mod(x,chans=32)
    c2,p2 = contract_mod(p1)
    c3,p3 = contract_mod(p2)
    c4,p4 = contract_mod(p3)
    c5, _ = contract_mod(p4)
    # expand
    e1 = expand_mod(c5,c4)
    e2 = expand_mod(e1,c3)
    e3 = expand_mod(e2,c2)
    e4 = expand_mod(e3,c1)
    # out of UNET
    sub = tf.layers.conv2d(e4,12,1,padding='SAME')
    # depth2space == subpixel reconstruction
    # out chans =  12/(2*2) = 3 <-- perfect!
    out = tf.depth_to_space(sub,2)
    return out

# learning rate
lr = tf.placeholder(tf.float32)
# input, truth, est.
x = tf.placeholder(tf.float32, shape=[None,None,None,4])
y = tf.placeholder(tf.float32, shape=[None,None,None,3])
y_hat = f(x)

# ---------------------------------------------
# ------------------- FILES -------------------
# ---------------------------------------------

gt_fns = glob.glob(gt_dir+'*')
in_fns = glob.glob(in_dir+'*')
train_ids = [int(os.path.basename(fn)[0:5]) for fn in gt_fns]
[ x.sort() for x in [gt_fns, in_fns, train_ids] ]
# exposure ratios
ratios = np.empty((len(train_ids)))
for i in range(len(train_ids)):
    ratios[i] = float(os.path.basename(gt_fns[i])[9:-5]) / \
        float(os.path.basename(in_fns[i])[9:-5])
print(ratios[0])
print('Loading data...')
start_time = time.time()

gt_imgs = np.stack( [scipy.misc.imread(x) for x in gt_fns] )
in_imgs = np.stack( [np.load(x) for x in in_fns] )
# subtracting black level and normalizing to [0,1] scale
in_imgs = np.maximum(in_imgs-512,0) / (16383-512)

time_elapsed = time.time() - start_time
print('%3fs to load data'%time_elapsed)

# demo of batches
# yb = gt_imgs[0,:,:,:]
# xb = in_imgs[0,:,:,0]
# print(xb.shape,yb.shape)
# figure, (ax1,ax2) = plt.subplots(1,2)
# ax1.imshow(np.squeeze(xb))
# ax2.imshow(np.squeeze(yb))
# plt.show()

# ---------------------------------------------
# ------------------- DATA -------------------
```

```
# -------------------------------------------

# pk: input image in pack form
# gt: ground truth image (in sRGB)
# ps: patch size (square)
# a patch of the packed bayers corresponds to a patch
# of the full size image (gt) that is twice as large
def to_patch(pk,gt,ps=PATCH_SIZE):
    H = pk.shape[0]
    W = pk.shape[1]
    h = np.random.randint(H-ps)
    w = np.random.randint(W-ps)
    pk_patch = pk[h:(h+ps),     w:(w+ps),    :]
    gt_patch = gt[2*h:2*(h+ps), 2*w:2*(w+ps),:]
    return pk_patch, gt_patch

def batch_to_patch(xb,yb,ps=PATCH_SIZE):
    xp = []
    yp = []
    H = xb.shape[1]
    W = xb.shape[2]
    for i in range(xb.shape[0]):
        h = np.random.randint(H-ps)
        w = np.random.randint(W-ps)
        xp.append(xb[i, h:(h+ps),     w:(w+ps),    :])
        yp.append(yb[i, 2*h:2*(h+ps), 2*w:2*(w+ps),:])
    return np.stack(xp), np.stack(yp)

def get_batch():
    choices = np.random.choice(len(train_ids),size=BATCH_SIZE)
    rb = ratios[choices] # batch of ratios
    xb, yb = batch_to_patch(in_imgs[choices,:,:,:], gt_imgs[choices,:,:,:])
    for i in range(BATCH_SIZE):
        xb[i,:,:,:] = xb[i,:,:,:]*rb[i]
    return xb, yb

# demo of batches
# xb,yb = get_batch()
# print(xb.shape,yb.shape)
# figure, (ax1,ax2) = plt.subplots(1,2)
# ax1.imshow(np.squeeze(xb[0,:,:,0]))
# ax2.imshow(np.squeeze(yb[0,:,:,:]))
# plt.show()

# -----------------------------------------------
# ------------------- TRAINING ------------------
# -----------------------------------------------


# L1 LOSS ONLY
loss = tf.reduce_mean( tf.abs(y_hat - y) )
optim = tf.train.AdamOptimizer(learning_rate=lr).minimize(loss)
init  = tf.global_variables_initializer()

# Create a summary to monitor cost tensor
tf.summary.scalar("loss", loss)
# Create a summary to monitor accuracy tensor
# tf.summary.scalar("accuracy", accuracy)
# Merge all summaries into a single op
merged_summary_op = tf.summary.merge_all()

with tf.Session() as sess:
    sess.run(init)

    # op to write logs to Tensorboard
    summary_writer = tf.summary.FileWriter(tflog,
```

```
        graph=tf.get_default_graph())
    learning_rate = LEARNING_RATE
    # training
    st = time.time()
    for epoch in range(NUM_EPOCHS):
        avg_cost = 0.
        num_batches = int( np.ceil( len(train_ids) / BATCH_SIZE ) )

        if (epoch+1)%LR_DECAY_RATE == 0:
            learning_rate = learning_rate*.1

        for i in tqdm(range(num_batches)):
            xb, yb = get_batch()
            fd = {x: xb, y: yb, lr: learning_rate}
            output, loss_np, _, summary \
                = sess.run([y_hat, loss, optim, merged_summary_op], feed_dict=fd)
            # logs every batch
            summary_writer.add_summary(summary, epoch * num_batches + i)
            avg_cost  += loss_np/num_batches

        # save output of part of batch
        if (epoch+1) % SAVE_RATE == 0:
            cat = np.concatenate((output[0,:,:,:],yb[0,:,:,:]),axis=1)
            scipy.misc.imsave(logs_dir + '%04d.jpg'%(epoch+1), cat)

        # log to stdout at each epoch
        if (epoch+1) % DISP_RATE == 0:
            print('Time: %06f, Epoch: %03d, Cost: %5ld'%(time.time()-st,epoch+1, avg_cost))

        # print('Validation Set Accuracy:',
            # accuracy.eval({x: data.x_val, y: data.y_val, phase: False}))

    print("Run the command line:\n--> tensorboard --logdir=./tf_logs ")
```