Contents

1 Basic Test Results 2

2 README 5

3 answer q1.txt 6

4 sol5.py 7

1 Basic Test Results

```
Archive:  /tmp/bodek.OJxsJF/impr/ex5/liavst2/presubmission/submission
  inflating: current/sol5.py
  inflating: current/README
  inflating: current/answer_q1.txt
ex5 presubmission script

    Disclaimer
    ----------
    The purpose of this script is to make sure that your code is compliant
    with the exercise API and some of the requirements
    The script does not test the quality of your results.
    Don't assume that passing this script will guarantee that you will get
    a high grade in the exercise

=== Check Submission ===

login:  liavst2

submitted files:

==========================================================
= README for ex5: Neural Networks For Image Restoration =
==========================================================


===========================
= List Of Submitted Files =
===========================

- README - this file.
- answer_q1.txt - Answer to Q1 in the pdf.
- sol5.py - contains the implementation of all function.

=== Answers to questions ===

Answer to Q1:

Answer to Q1:
============

The idea is as follows.
Our corruption function will, randomly, choose an image size
from a given list of image sizes and resize the current-iteration
image to that size. We slightly modify this function so it
returns not only the corrupted image, but the size it chose
(Lets call this size corrupted_size). Next, instead of extracting
corresponding patches of the same size in each iteration,
we will extract crop_size sized patches from the clean images,
and "adjusted" crop_size sized patches from the corrupted images,
where the adjustment is calculated from the returned corrupted_size.
Then, for each "adjusted" crop_size sized patch we will use linear
interpolation to resize it to crop_size, and then add it to the
"source" batch.
In addition to that, we add a wrapper function to the restore_image
function, which takes three arguments - corrupted_image, clean_size
and num_channels. clean-size is the size of one of the clean images
in the in the repository. This wrapper function will resize
corrupted_image to the size of clean_size, and then calls restore_image
with the new cropped image and num_channels, and returns the prediction
```

```
restore_image has returned.

=== Section 3 ===

Trying to load basic datasets...
    Passed!

=== Section 4 ===

Trying to build a model...
    Passed!

=== Section 5 ===

Trying to train a model...
Epoch 1/2

10/20 [==============>...............] - ETA: 1s - loss: 0.0298^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^
20/20 [==============================] - 2s - loss: 0.0933 - val_loss: 0.0550
Epoch 2/2

10/20 [==============>...............] - ETA: 0s - loss: 0.0554^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^
20/20 [==============================] - 1s - loss: 0.0371 - val_loss: 0.0178
    Passed!
Trying to train a model...
Epoch 1/2

10/20 [==============>...............] - ETA: 1s - loss: 0.1171^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^
20/20 [==============================] - 2s - loss: 0.1303 - val_loss: 0.0389
Epoch 2/2

10/20 [==============>...............] - ETA: 0s - loss: 0.0405^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^
20/20 [==============================] - 1s - loss: 0.0448 - val_loss: 0.0517
    Passed!

=== Section 6 ===

Trying to restore image... (not checking for quality!)
    Passed!

=== Section 7 ===

=== Image Denoising ===
Trying to apply random noise on image...
    Passed!
Trying to learn a denoising model...
Epoch 1/2

10/30 [=========>....................] - ETA: 5s - loss: 0.0702^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^
20/30 [===================>..........] - ETA: 2s - loss: 0.0672^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^
30/30 [==============================] - 8s - loss: 0.0632 - val_loss: 0.0305
Epoch 2/2

10/30 [=========>....................] - ETA: 3s - loss: 0.0325^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^
20/30 [===================>..........] - ETA: 1s - loss: 0.0478^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^
30/30 [==============================] - 8s - loss: 0.0370 - val_loss: 0.0083
    Passed!
Trying to use learned model for denoising... (not checking for quality!)
    Passed!
=== Image Deblurring ===
Trying to apply random motion blur on image...
    Passed!
Trying to learn a deblurring model...
Epoch 1/2

10/30 [=========>....................] - ETA: 3s - loss: 0.3785^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^
20/30 [===================>..........] - ETA: 1s - loss: 0.2701^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^
30/30 [==============================] - 4s - loss: 0.2013 - val_loss: 0.0353
```

```
Epoch 2/2

10/30 [=========>....................] - ETA: 1s - loss: 0.0318^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^
20/30 [===================>..........] - ETA: 0s - loss: 0.0378^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^H^
30/30 [==============================] - 2s - loss: 0.0406 - val_loss: 0.0329Using TensorFlow backend.

    Passed!
Trying to use learned model for deblurring... (not checking for quality!)
    Passed!

=== All tests have passed ===
=== Pre-submission script done ===


    Please go over the output and verify that there are no failures/warnings.
    Remember that this script tested only some basic technical aspects of your implementation
    It is your responsibility to make sure your results are actually correct and not only
    technically valid.
```

2 README

```
liavst2

=========================================================
= README for ex5: Neural Networks For Image Restoration =
=========================================================


===========================
= List Of Submitted Files =
===========================

- README - this file.
- answer_q1.txt - Answer to Q1 in the pdf.
- sol5.py - contains the implementation of all function.
```

3 answer q1.txt

```
Answer to Q1:
============

The idea is as follows.
Our corruption function will, randomly, choose an image size
from a given list of image sizes and resize the current-iteration
image to that size. We slightly modify this function so it
returns not only the corrupted image, but the size it chose
(Lets call this size corrupted_size). Next, instead of extracting
corresponding patches of the same size in each iteration,
we will extract crop_size sized patches from the clean images,
and "adjusted" crop_size sized patches from the corrupted images,
where the adjustment is calculated from the returned corrupted_size.
Then, for each "adjusted" crop_size sized patch we will use linear
interpolation to resize it to crop_size, and then add it to the
"source" batch.
In addition to that, we add a wrapper function to the restore_image
function, which takes three arguments - corrupted_image, clean_size
and num_channels. clean-size is the size of one of the clean images
in the in the repository. This wrapper function will resize
corrupted_image to the size of clean_size, and then calls restore_image
with the new cropped image and num_channels, and returns the prediction
restore_image has returned.
```

4 sol5.py

```python
##############################################################
# FILE: sol5.py
# WRITER: Liav Steinberg
# EXERCISE : Image Processing ex5
##############################################################

from skimage.color import rgb2gray
from scipy.misc import imread
from keras import layers as klay, models as kmod
from keras.optimizers import Adam
from scipy.ndimage import filters as flt
import sol5_utils as sut
import numpy as np


#-------------------------helpers----------------------------#

def read_image(filename, representation):
    """
    reads an image with the given representation
    """
    image = imread(filename).astype(np.float32) / 255
    return image if representation == 2 else rgb2gray(image)


def learn_X_model(quick_mode, load_func, corr_func, cs, nc, ne):
    """
    to reduce code repetition for section 7.2.1, 7.2.2
    """
    def corruption_func(im):
        """
        wrapper for the corruption function
        """
        return add_gaussian_noise(im, 0, 0.2) if corr_func == "gaussian_noise" \
            else random_motion_blur(im, [7])
    ######################################################
    images = load_func()
    batch_size = 100
    sam_per_epoch = 10000
    num_epochs = ne
    num_valid_sample = 1000
    if quick_mode:
        batch_size = 10
        sam_per_epoch = 30
        num_epochs = 2
        num_valid_sample = 30
    model = build_nn_model(cs, cs, nc)
    train_model(model, images, corruption_func, batch_size,
                sam_per_epoch, num_epochs, num_valid_sample)
    return model, nc

#-------------------------3----------------------------#

def load_dataset(filenames, batch_size, corruption_func, crop_size):
    """
    Generator function for creating dataset of patches
    """
    images = {}
    while True:
```

```
        source, target = np.zeros((batch_size, 1, crop_size[0], crop_size[1]), np.float32), \
                         np.zeros((batch_size, 1, crop_size[0], crop_size[1]), np.float32)
        rand_files = np.random.choice(filenames, batch_size)
        for idx,_file_ in enumerate(rand_files):
            if _file_ not in images:
                images[_file_] = np.array(read_image(_file_, 1))
            image = images[_file_]
            corrupted_image = corruption_func(image)
            rand_X = np.random.randint(image.shape[0]-crop_size[0])
            rand_Y = np.random.randint(image.shape[1]-crop_size[1])
            source[idx,0,:,:] = corrupted_image[rand_X:rand_X + crop_size[0], rand_Y:rand_Y + crop_size[1]]
            target[idx,0,:,:] = image[rand_X:rand_X+crop_size[0], rand_Y:rand_Y+crop_size[1]]
        yield (source-0.5, target-0.5)

#-------------------------4----------------------------#

def resblock(input_tensor, num_channels):
    """
    represent a single block in the network flow
    """
    conv = klay.Convolution2D(num_channels, 3, 3, border_mode="same")(input_tensor)
    relu = klay.Activation("relu")(conv)
    second_conv = klay.Convolution2D(num_channels, 3, 3, border_mode="same")(relu)
    return klay.merge([input_tensor, second_conv], mode="sum")


def build_nn_model(height, width, num_channels):
    """
    builds an untrained Keras model
    """
    def repeat(func, x, y, n):
        """
        high-order function for repeating func on inputs
        x, y as times as n
        """
        for i in range(n):
            x = func(x, y)
        return x
    ########################################################
    input_ = klay.Input(shape=(1, height, width))
    conv = klay.Convolution2D(num_channels, 3, 3, border_mode="same")(input_)
    relu = klay.Activation("relu")(conv)
    after_resblocks = repeat(resblock, relu, num_channels, 5)
    final = klay.merge([relu, after_resblocks], mode="sum")
    output = klay.Convolution2D(1, 3, 3, border_mode="same")(final)
    return kmod.Model(input_, output)

#-------------------------5----------------------------#

def train_model(model, images, corruption_func, batch_size,
                samples_per_epoch, num_epochs, num_valid_sample):
    """
    trains the given model according to a dataset
    """
    training, validation = np.split(images, [int(len(images)*0.8)])
    training_set = load_dataset(training, batch_size, corruption_func, model.input_shape[2:])
    validation_set = load_dataset(validation, batch_size, corruption_func, model.input_shape[2:])
    model.compile(loss="mean_squared_error", optimizer=Adam(beta_2=0.9))
    model.fit_generator(training_set, samples_per_epoch=samples_per_epoch, nb_epoch=num_epochs,
                        validation_data=validation_set, nb_val_samples=num_valid_sample)

#-------------------------6----------------------------#

def restore_image(corrupted_image, base_model, num_channels):
    """
    restores the corrupted image according to the learning model
    """
    height, width = corrupted_image.shape[0], corrupted_image.shape[1]
```

```python
    corrupted_image = np.array(corrupted_image).reshape(1, height, width)-0.5
    model = build_nn_model(height, width, num_channels)
    model.set_weights(base_model.get_weights())
    prediction = model.predict(corrupted_image[np.newaxis,...])[0] + 0.5
    return np.clip(prediction,0,1).reshape(height, width)

#-------------------------7.1.1----------------------------#

def add_gaussian_noise(image, min_sigma, max_sigma):
    """
    adds gaussian noise to a given image
    """
    return np.clip(image + np.random.normal(scale=np.random.uniform(min_sigma, max_sigma),
                                            size=image.shape), 0, 1).astype(np.float32)

#-------------------------7.1.2----------------------------#

def learn_denoising_model(quick_mode=False):
    """
    returns trained model for the denoising set
    """
    return learn_X_model(quick_mode, sut.images_for_denoising, "gaussian_noise", 24, 48, 5)

#-------------------------7.2.1----------------------------#

def add_motion_blur(image, kernel_size, angle):
    """
    adds motion blur to a given image
    """
    return flt.convolve(image, sut.motion_blur_kernel(kernel_size, angle)).astype(np.float32)

def random_motion_blur(image, list_of_kernel_sizes):
    """
    adds random motion blur to a given image
    """
    return add_motion_blur(image, np.random.choice(list_of_kernel_sizes), np.random.uniform(high=np.pi))

#-------------------------7.2.2----------------------------#

def learn_deblurring_model(quick_mode=False):
    """
    returns trained model for the deblurring set
    """
    return learn_X_model(quick_mode, sut.images_for_deblurring, "random_motion_blur", 16, 32, 10)

#-------------------------end----------------------------#
```