Contents

1 Basic Test Results

```
ex5/
ex5/README.md
ex5/answer_q1.txt
ex5/answer_q2.txt
ex5/depth_plot_deblur.png
ex5/depth_plot_denoise.png
ex5/sol5.py
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 ===

README file:

tal.porezky
sol5.py
sol5_utils.py
answer_q1.txt
answer_q2.txt


=== Answers to questions ===

Answer to Q1:
Denoising:
One could notice a steady decline in the error for each run with increasing
the residual blocks however the runtime increased as well since
there were overall more variables for the network to compute and to deal with.
As for the quality there was a noticeable difference between every run since
the picture became a bit cleaner with each run of the network.
There is one point where the error is higher than the previous. I think it
might indicate a start of "overfitting" in the network.

Deblurring:
Similar results with the quantative measure as the plot curved downwards.
Obviously the runtime increased with the amount of residual blocks however the deblurring seemed to be increasingly better w


Answer to Q2:
We can train a model with 'corruption' function that her porpuse is to
reduce the resulotion of the image. It will  reduce the size of a patch
by sampling every 2 pixels for few iteration and then expend to the real size
again. The train will be with the corrupt image and the real patch.

=== Plots for Q1 ===
Please verify that your plot image file depth_plot_denoise.png includes the right plot.
Please verify that your plot image file depth_plot_deblur.png includes the right plot.
=== Load Student Library ===

Loading...

=== Section 3 ===
```

```
Trying to load basic datasets...
    Passed!

=== Section 4 ===

Trying to build a model...
WARNING: Logging before flag parsing goes to stderr.
W0204 20:19:29.692249 140150177003328 deprecation.py:506] From /usr/local/tensorflow/avx-avx2-cpu/1.14.0/python3.7/site-pack
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the constructor
    Passed!

=== Section 5 ===

Trying to train a model...
Epoch 1/2
2020-02-04 20:19:30.403830: I tensorflow/core/platform/profile_utils/cpu_utils.cc:94] CPU Frequency: 2095074999 Hz
2020-02-04 20:19:30.406025: I tensorflow/compiler/xla/service/service.cc:168] XLA service 0x1d2fb10 executing computations o
2020-02-04 20:19:30.406064: I tensorflow/compiler/xla/service/service.cc:175]   StreamExecutor device (0): <undefined>, <und
2020-02-04 20:19:30.603633: W tensorflow/compiler/jit/mark_for_compilation_pass.cc:1412] (One-time warning): Not using XLA:C

1/3 [=========>.....................] - ETA: 1s - loss: 0.0095^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^H^
3/3 [==============================] - 1s 373ms/step - loss: 0.0053 - val_loss: 0.0026
Epoch 2/2

1/3 [=========>.....................] - ETA: 0s - loss: 0.0020^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^H^
3/3 [==============================] - 0s 81ms/step - loss: 0.0012 - val_loss: 2.0980e-04
    Passed!
Trying to train a model...
Epoch 1/2

1/3 [=========>.....................] - ETA: 1s - loss: 0.0036^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^H^
3/3 [==============================] - 1s 342ms/step - loss: 0.0119 - val_loss: 0.0019
Epoch 2/2

1/3 [=========>.....................] - ETA: 0s - loss: 0.0021^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^H^
3/3 [==============================] - 0s 67ms/step - loss: 0.0053 - val_loss: 0.0044
    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
ex5_presubmit.py:9: DeprecationWarning: `imread` is deprecated!
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
  im = imread(filename)

1/3 [=========>.....................] - ETA: 1s - loss: 0.0130^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^H^
3/3 [==============================] - 1s 423ms/step - loss: 0.0315 - val_loss: 0.0133
Epoch 2/2

1/3 [=========>.....................] - ETA: 0s - loss: 0.0104^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^H^
3/3 [==============================] - 0s 127ms/step - loss: 0.0158 - val_loss: 0.0138
    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

1/3 [=========>....................] - ETA: 2s - loss: 0.0238^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^H^
3/3 [==============================] - 1s 434ms/step - loss: 0.0257 - val_loss: 0.0282
Epoch 2/2

1/3 [=========>....................] - ETA: 0s - loss: 0.0287^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^H^
3/3 [==============================] - 0s 79ms/step - loss: 0.0232 - val_loss: 0.0271
    Passed!
Trying to use learned model for deblurring... (not checking for quality!)
    Passed!


=== Presubmission Completed Successfully ===


    Please go over the output and verify that there were 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 ex5/README.md

```
1  tal.porezky
2  sol5.py
3  sol5_utils.py
4  answer_q1.txt
5  answer_q2.txt
```

3 ex5/answer q1.txt

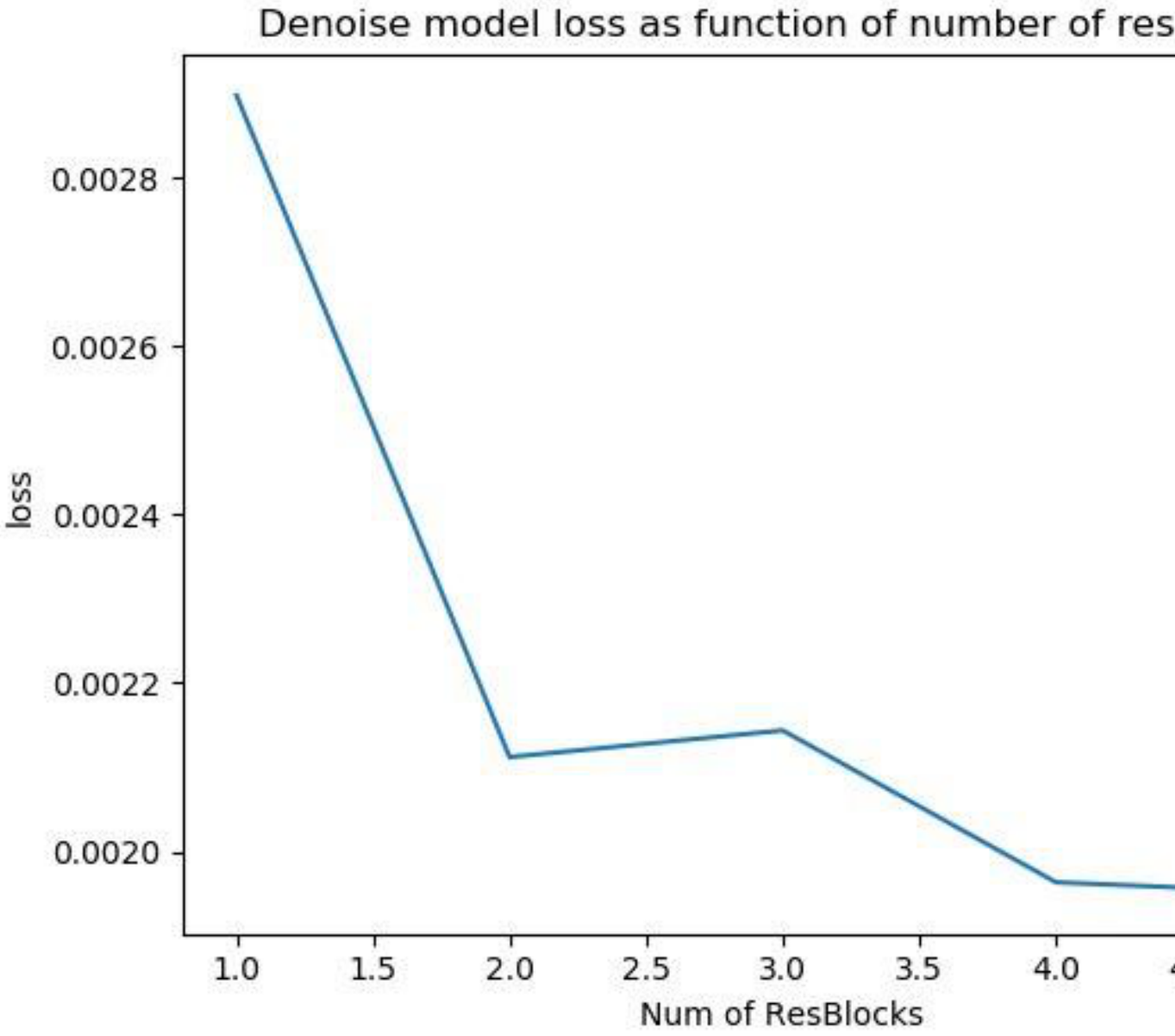
```
Denoising:
One could notice a steady decline in the error for each run with increasing
the residual blocks however the runtime increased as well since
there were overall more variables for the network to compute and to deal with.
As for the quality there was a noticeable difference between every run since
the picture became a bit cleaner with each run of the network.
There is one point where the error is higher than the previous. I think it
might indicate a start of "overfitting" in the network.

Deblurring:
Similar results with the quantative measure as the plot curved downwards.
Obviously the runtime increased with the amount of residual blocks however the deblurring seemed to be increasingly better w
```

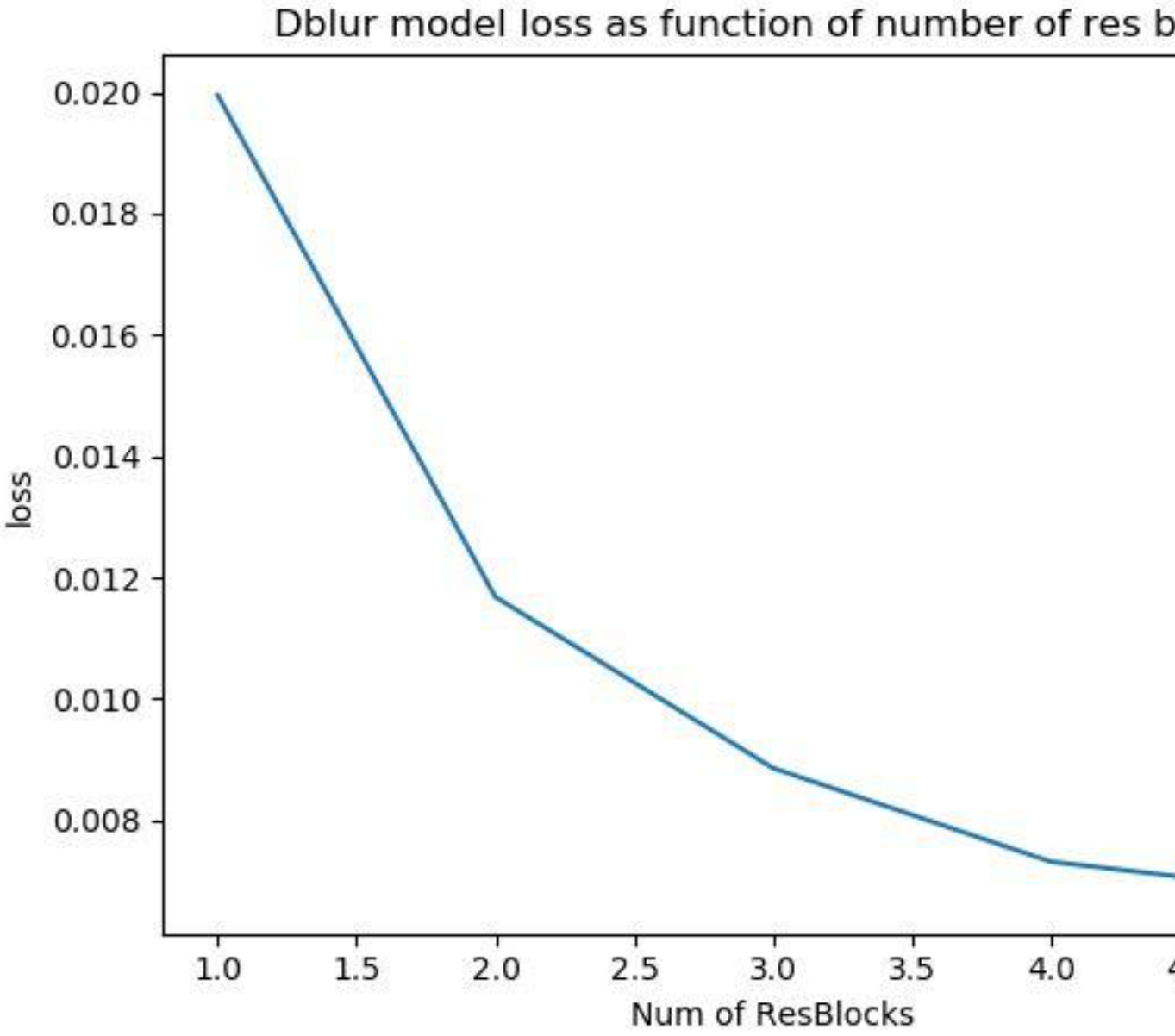
4 ex5/answer q2.txt

```
1  We can train a model with 'corruption' function that her porpuse is to
2  reduce the resulotion of the image. It will  reduce the size of a patch
3  by sampling every 2 pixels for few iteration and then expend to the real size
4  again. The train will be with the corrupt image and the real patch.
```

5 ex5/depth plot deblur.png



6 ex5/depth plot denoise.png



7 ex5/sol5.py

```python
# ----------- Imports -----------

import numpy as np
from imageio import imwrite, imread
import scipy.ndimage  # todo ?
from tensorflow.keras.layers import Conv2D, Activation, Add, Input
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from skimage.color import rgb2gray
import sol5_utils  # todo ?


# ----------- Helper functions -----------


def read_image(filename, representation):
    """
    function which reads an image ~Lle and converts it into a given
    representation.
    :param filename: the filename of an image on disk (could be grayscale or
    RGB).
    :param representation: representation code, either 1 or 2 defining
    whether the output should be a grayscaleimage (1) or an RGB image (2).
    If the input image is grayscale, we won't call it with representation = 2.
    :return: rgb or grayscale img
    """
    image = imread(filename)
    new_image = image.astype(np.float64)
    new_image /= 255
    if representation == 1:
        new_image = rgb2gray(new_image)
    return new_image


# ----------- 3: Dataset handling  -----------

def _read_images_and_put_in_dict(filenames, cache, batch_size):
    """
    reads images and puts them in the dict, if needed.
    :param filenames: list of filenames of images
    :param cache: dict with keys as filenames and images as values
    :param batch_size:  The size of the batch of images for each iteration
    of Stochasic Gradient Descent.
    :return: list of the chosen filenames
    """
    chosen_filenames = np.random.choice(filenames, size=batch_size)
    for chosen_filename in chosen_filenames:
        if chosen_filename not in cache.keys():
            im = read_image(chosen_filename, 1)
            cache[chosen_filename] = im
    return chosen_filenames


def _get_random_patch_of_the_image(im, crop_size):
    """
    crops the image to size crop_size and returns it
    :param im: image to crop
    :param crop_size: A tuple (height, width) specifying the crop size of
    the patches to extract.
```

```
    :return: same image but cropped to size crop_size in random location.
    """
    y, x = im.shape
    rand_x = np.random.randint(x - crop_size[1])
    rand_y = np.random.randint(y - crop_size[0])
    patch = im[rand_y:rand_y + crop_size[0],
            rand_x:rand_x + crop_size[1]]
    return patch


def load_dataset(filenames, batch_size, corruption_func, crop_size):
    """
    Creates a generator of corrupt and normal image.
    :param filenames: A list of filenames of clean images.
    :param batch_size: The size of the batch of images for each iteration
    of Stochasic Gradient Descent.
    :param corruption_func: A function receiving a numpy's array
    representation of an image as a single argument, and returns a randomly
    corrupted version of the input image.
    :param crop_size: A tuple (height, width) specifying the crop size of
    the patches to extract.
    :return: Pythons generator which outputs random tuples of the form (
    sorce_batch, target_batch), where each output variable is an array of
    shape (batch_size, height, width, 1)
    """
    cache = dict()
    while True:
        source_batch = np.zeros((batch_size, crop_size[0], crop_size[1], 1))
        target_batch = np.zeros((batch_size, crop_size[0], crop_size[1], 1))
        chosen_filenames = _read_images_and_put_in_dict(filenames,
                                                        cache,
                                                        batch_size)
        for filename_idx in range(len(chosen_filenames)):
            filename = chosen_filenames[filename_idx]
            im = cache[filename]
            patch = _get_random_patch_of_the_image(im, crop_size)
            corrupted_im = corruption_func(patch)
            source_batch[filename_idx, :, :, 0] = corrupted_im - 0.5
            target_batch[filename_idx, :, :, 0] = patch - 0.5
        yield source_batch, target_batch


# ----------- 4: Neural Network Model  -----------


def resblock(input_tensor, num_channels):
    """
    Takes as input symbolic input tensor and the number of channels for
    each of its convolutional layers, and returns the symbolic output
    tensor of the layer configuration described in the PDF.
    :param input_tensor: input tensor...
    :param num_channels: the number of channels for
    each of its convolutional layers
    :return: output tensor of the layer configuration described in the PDF.
    """
    tensor = Conv2D(filters=num_channels, kernel_size=3, padding='same')(
        input_tensor)
    tensor = Activation('relu')(tensor)
    tensor = Conv2D(filters=num_channels, kernel_size=3, padding='same')(
        tensor)
    tensor = Add()([tensor, input_tensor])
    output_tensor = Activation('relu')(tensor)
    return output_tensor


def build_nn_model(height, width, num_channels, num_res_blocks):
    """
    Returns output of the whole neural network
```

```
    :param height: number of pixels in the height
    :param width: number of pixels in the width
    :param num_channels: number of res block in the network
    :param num_res_blocks  number of res block in the network
    :return: output
    """
    input_network = Input(shape=(height, width, 1))
    tensor = Conv2D(num_channels, (3, 3), padding='same')(input_network)
    tensor = Activation('relu')(tensor)
    for _ in range(num_res_blocks):
        tensor = resblock(tensor, num_channels)
    tensor = Conv2D(1, (3, 3), padding='same')(tensor)
    tensor = Add()([tensor, input_network])
    model = Model(inputs=input_network, outputs=tensor)
    return model


# ----------- 5: Training Networks for Image Restoration  -----------


def train_model(model, images, corruption_func, batch_size,
                steps_per_epoch, num_epochs, num_valid_samples):
    """
    trains the model.
    :param model: a general neural network model for image restoration.
    :param images: a list of ~lle paths pointing to image ~lles. You should
    assume these paths are complete, and should append anything to them.
    :param corruption_func: same as described in section 3.
    :param batch_size: the size of the batch of examples for each iteration of
    SGD.
    :param steps_per_epoch: The number of update steps in each epoch.
    :param num_epochs: The number of epochs for which the optimization will run.
    :param num_valid_samples: The number of samples in the validation set to
    test on after every epoch.
    :return: none
    """
    train_set_size = np.int(len(images) * 0.8)
    train_images = images[: train_set_size]
    validation_images = images[train_set_size :]
    train_set = load_dataset(train_images,
                             batch_size,
                             corruption_func,
                             model.input_shape[1: 3])
    validation_set = load_dataset(validation_images,
                                  batch_size,
                                  corruption_func,
                                  model.input_shape[1: 3])
    model.compile(loss='mean_squared_error', optimizer=Adam(beta_2=0.9))
    model.fit_generator(train_set, steps_per_epoch=steps_per_epoch,
                        epochs=num_epochs, validation_data=validation_set,
                        validation_steps=(num_valid_samples // batch_size),
                        use_multiprocessing=True)
    # todo


# ----------- 6: Image Restoration of Complete Images  -----------


def restore_image(corrupted_image, base_model):
    """
    using our model to restore big size images.
    :param corrupted_image: a grayscale image of shape (height, width) and
    with values in the [0; 1] range of type float64.
    :param base_model: a neural network trained to restore small patches.
    :return:
    """
    a = Input(shape=(corrupted_image.shape[0], corrupted_image.shape[1], 1))
    b = base_model(a)
```

```
    new_model = Model(inputs=a, outputs=b)
    updated_corrupted_image = corrupted_image.reshape(
        corrupted_image.shape[0], corrupted_image.shape[1], 1)
    im = new_model.predict(np.expand_dims(updated_corrupted_image - 0.5, axis=0),
                           batch_size=1)[0]
    return np.clip(np.squeeze((im + 0.5), axis=2), 0, 1).astype('float64')


# ----------- 7: Application to Image Denoising and Deblurring  -----------


def add_gaussian_noise(image, min_sigma, max_sigma):
    """
    adds gaussian noise to the image.
    :param image: a grayscale image with values in the [0; 1] range of type
    float64.
    :param min_sigma: a non-negative scalar value representing the minimal
    variance of the gaussian distribution.
    :param max_sigma: a non-negative scalar value larger than or equal to
    min_sigma, representing the maximal variance of the gaussian distribution.
    :return: noised image
    """
    assert(max_sigma >= min_sigma)
    sigma = np.random.uniform(min_sigma, max_sigma)
    noise = np.random.normal(loc=0, scale=sigma, size=image.shape)
    noisy_im = np.floor((image + noise) * 255) / 255
    noisy_im = np.clip(noisy_im, 0, 1)
    return noisy_im


def _gaussian_for_learn_denosing_model(image):
    """
    adds gaussian noise to image with sigma [0, 0.2]
    :param image: image
    :return: function which return image with noise
    """
    return add_gaussian_noise(image, 0, 0.2)


def learn_denoising_model(num_res_blocks=5, quick_mode=False):
    """
    returns trained denoised model
    :param num_res_blocks: number of res block in the network
    :param quick_mode: true or false value
    :return: the model.
    """
    model = build_nn_model(24, 24, 48, num_res_blocks)
    if quick_mode:
        train_model(model, sol5_utils.images_for_denoising(),
                    _gaussian_for_learn_denosing_model, 10, 3, 2, 30)
        return model
    train_model(model, sol5_utils.images_for_denoising(),
                _gaussian_for_learn_denosing_model, 100, 100, 5, 1000)
    return model


def add_motion_blur(image, kernel_size, angle):
    """
    adds motion blur at given angel to the image.
    :param image: image
    :param kernel_size: int
    :param angle: an angle in radians in the range [0, pi).
    :return: blurred image
    """
    kernel = sol5_utils.motion_blur_kernel(kernel_size, angle)
    return scipy.ndimage.filters.convolve(image, kernel)
```

```python
def random_motion_blur(image, list_of_kernel_sizes):
    """

    :param image: a grayscale image with values in the [0; 1] range of type float64.
    :param list_of_kernel_sizes: a list of odd integers.
    :return:
    """
    kernel_size = np.random.choice(list_of_kernel_sizes)
    angle = np.random.uniform(0, np.pi)
    noisy_im = add_motion_blur(image, kernel_size, angle)
    noisy_im = np.floor(noisy_im * 255) / 255
    noisy_im = np.clip(noisy_im, 0, 1)
    return noisy_im


def _motion_blur_for_learn_deblurring_model(image):
    """
    return an specific motion blur to image
    :param image: a grayscale image with values in the [0; 1] range of type float64.
    :return: function which return image with blur
    """
    return random_motion_blur(image, [7])


def learn_deblurring_model(num_res_blocks=5, quick_mode=False):
    """
    creates model which fixes deblurring images.
    :param num_res_blocks: number of res blocks in the network
    :param quick_mode: smaller model that make it run quickly
    :return: the model.
    """
    model = build_nn_model(16, 16, 32, num_res_blocks)
    if quick_mode:
        train_model(model, sol5_utils.images_for_deblurring(),
                    _motion_blur_for_learn_deblurring_model,
                    10, 3, 2, 30)
        return model
    train_model(model, sol5_utils.images_for_deblurring(),
                _motion_blur_for_learn_deblurring_model,
                100, 100, 10, 1000)
    return model
```