Online Assessment for Coursework 1

Gargeya Sharma 220278025 MSc Artificial Intelligence

Question (Q3.4): *What is the typical numerical measure metric for quantitative analysis?*

Answer 3.4: There are various numerical measure metrics that can be used to perform quantitative analysis (in our case) for image super-resolution. In the coursework and the original paper, we are asked to implement and understand the Peak Signal-to-Noise ratio (PSNR). There are many more that are sometimes used in image super-resolution tasks such as Structural similarity index measure (SSIM), Mean squared error (MSE), root mean squared error (RMSE) and mean absolute error (MAE), however, the most commonly used ones are PSNR and SSIM. A brief introduction to both of them is mentioned below:

PSNR:

Pixel values of the reconstructed image as compared to the original image are measured by PSNR. It is defined as the ratio of the mean square error (MSE) of the reconstructed and ground truth images to the signal's highest possible value (i.e., the pixel intensity range).

SSIM:

In contrast, SSIM is a perception-based metric that assesses how structurally similar the reconstructed and original pictures are. It considers how similar the images' structural elements, contrast, and brightness are. A score of 1 indicates a perfect match between the two images. SSIM values range from -1 to 1.

Exercise (E3.6): *To get the high-resolution image by interpolation algorithm (baseline result)*

Answer E3.6: The code to perform the task:

```
# Necessary libraries for the task

import matplotlib.pyplot as plt
import numpy as np
import imageio as iio
from scipy import ndimage

def imread(path, is_grayscale=True):
    """
    Read image from the giving path.
    Default value is gray-scale, and image is read by YCbCr format as the paper.
    """
    if is_grayscale:
        return iio.imread(path, as_gray=True, pilmode='YCbCr').astype(np.float32)
    else:
        return iio.imread(path, pilmode='YCbCr').astype(np.float32)


def modcrop(image, scale=3):
    """
    To scale down and up the original image, first thing to do is to have no remainder while
scaling operation.

    We need to find modulo of height (and width) and scale factor.
    Then, subtract the modulo from height (and width) of original image size.
    There would be no remainder even after scaling operation.
    """
```

```
    if len(image.shape) == 3:
        h, w, _ = image.shape
        h = h - np.mod(h, scale)
        w = w - np.mod(w, scale)
        image = image[0:h, 0:w, :]
    else:
        h, w = image.shape
        h = h - np.mod(h, scale)
        w = w - np.mod(w, scale)
        image = image[0:h, 0:w]
    return image


def preprocess(path, scale=3):
    """
    Preprocess single image file
      (1) Read original image as YCbCr format (and grayscale as default)
      (2) Normalize
      (3) Apply image file with interpolation
    Args:
      path: file path of desired file
      input_: image applied interpolation (low-resolution)
      label_: image with original resolution (high-resolution), ground truth
    """

    image = imread(path, is_grayscale=True)
    label_ = modcrop(image, scale)



    # Must be normalized
    label_ = label_ / 255.

## Creating the baseline result by first zooming in on the image with the provided scale and
## then zooming out the result with that same scale
    input_ = ndimage.interpolation.zoom(label_, (1. / scale), prefilter=False)
    input_ = ndimage.interpolation.zoom(input_, (scale / 1.), prefilter=False)

    return input_, label_

LR_image, HR_image = preprocess('./image/butterfly_GT.bmp') ## Please keep the image on the same
location as mentioned in the coursework before running the code.
iio.imsave("LR.jpg", HR_image.squeeze())
print("Baseline Result with Interpolation:")
plt.imshow(LR_image.squeeze(), cmap='gray')
```

Output:

```
Baseline Result with Interpolation:
[8]: <matplotlib.image.AxesImage at 0x7f2e1436b160>
```