Korea Advanced Institute of Science and Technology

School of Electrical Engineering

EE838B Special Topic on Image Engineering

Advanced Image Restoration and Quality Enhancement

Student's Name: Dinh Vu

Student's ID: 20184187

**Homework 3**

## 1. Dataset

The given dataset is the GOPRO dataset which is split into 2103 pairs for training and 1111 pairs for validation. Each pair consists 1 sharp image and 1 blurred image. The images is categorized into various scenes.

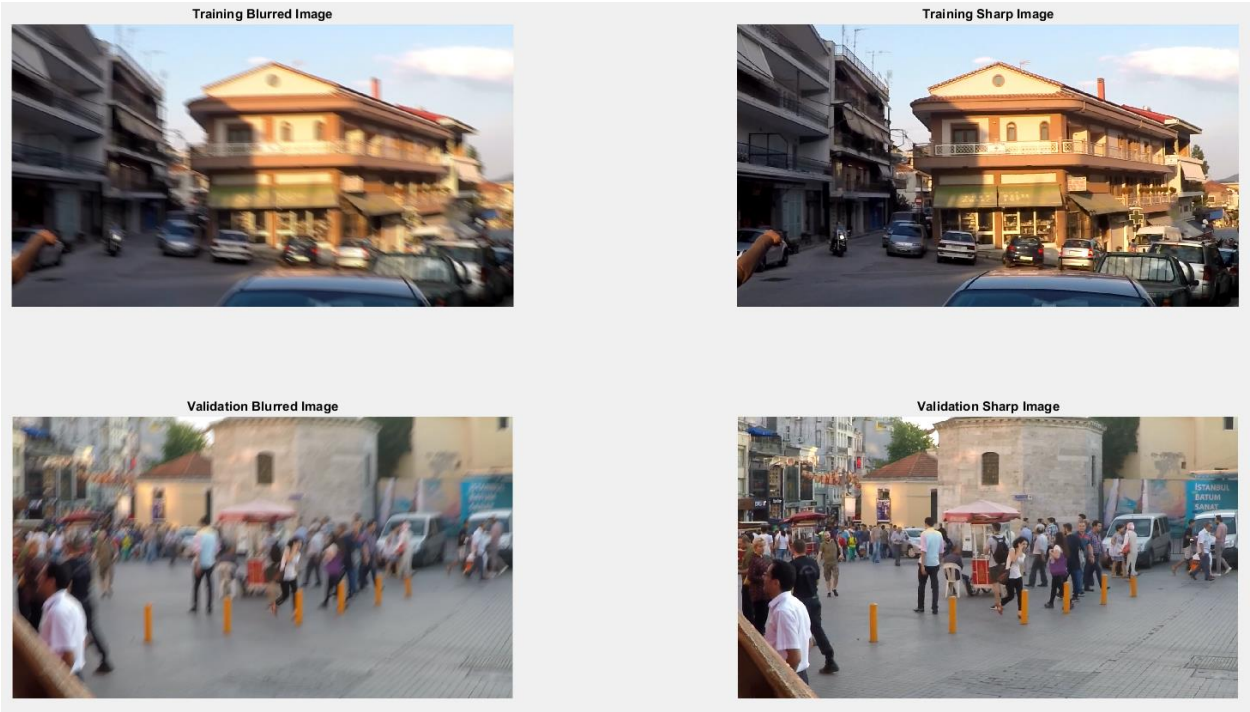


**Figure 1.1.** The examples of GOPRO dataset

## 2. Model

The structure of the given deblurred multiscale neural network is well defined in the Figure 2.1 below. There are 3 levels in the network and each level includes 9 Residual Blocks.

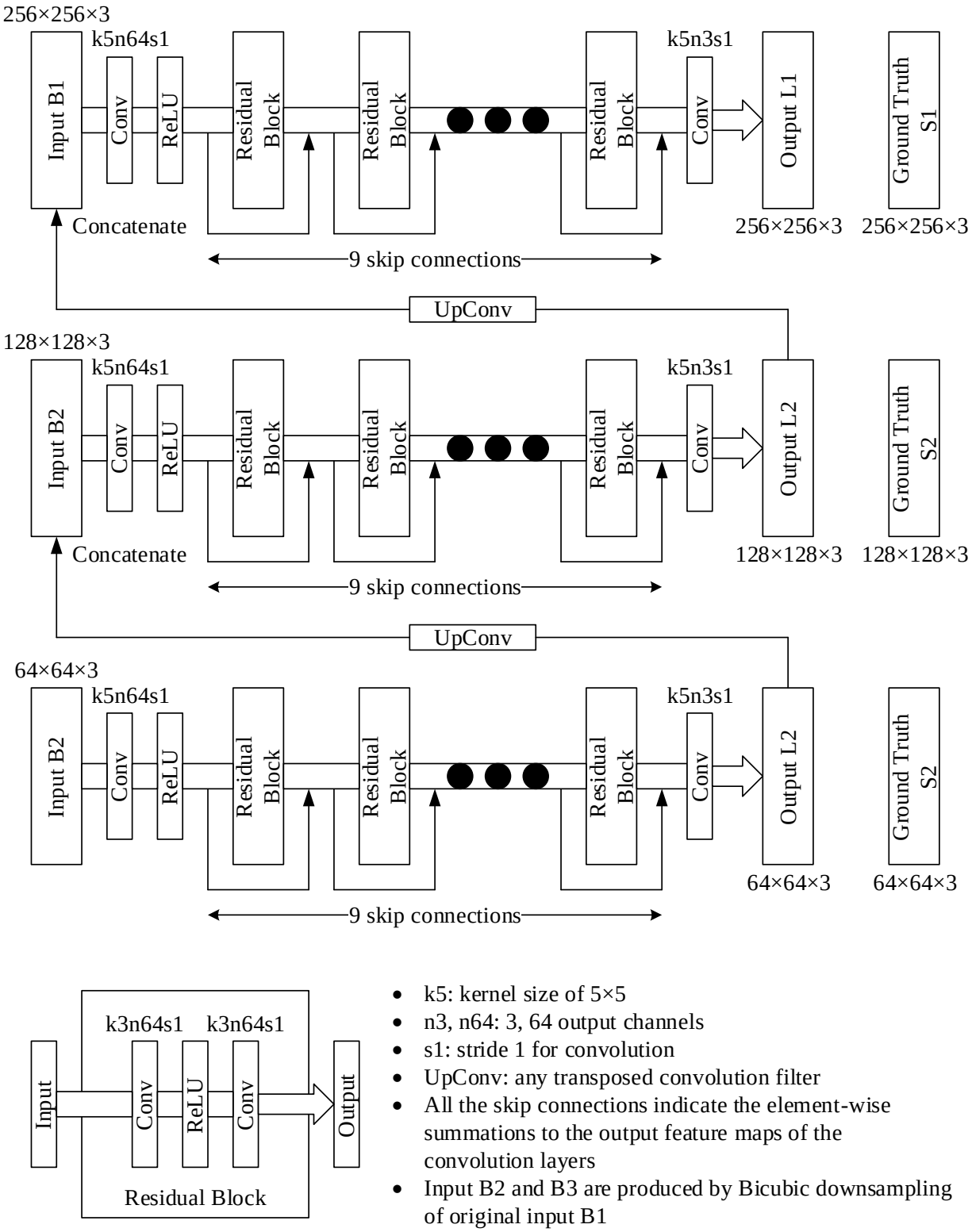


**Figure 2.1.** The structure of the multiscale convolution neural network

Although, there is still one confused component which is up-convolution. In the paper [2], the authors mention that the transposed convolution is used, but in the reference code, released by the first author Seungjun Nah, the pixel shuffle is applied. For the sake of comparison, both of up-convolution methods are implemented and experimented.

```
18  def upsampling(x, scale, reuse, name):
19      #x = tf.layers.conv2d_transpose(x, 3, 3, 2, 'same', name=name, reuse=reuse)
20      x = tf.layers.conv2d(x, 12, 5, 1, 'same', name='%s/conv'%name, reuse=reuse)
21      x = tf.depth_to_space(x, scale)
22      return x
```

**Figure 2.2.** The pixel shuffle

```
18  def upsampling(x, scale, reuse, name):
19      x = tf.layers.conv2d_transpose(x, 3, 3, 2, 'same', name=name, reuse=reuse)
20      #x = tf.layers.conv2d(x, 12, 5, 1, 'same', name='%s/conv'%name, reuse=reuse)
21      #x = tf.depth_to_space(x, scale)
22      return x
```

**Figure 2.3.** The transpose convolution

## 3. Training

Before beginning the training process, all pairs of images must be preprocessed. Each pair of the training set is read from the hard disk then concatenated for randomly cropping. This reading process is presented in `train_parse()` function, shown in Figure 3.1.

```
32  def train_parse(train_blur_dir, train_sharp_dir):
33    blur_string = tf.read_file(train_blur_dir)
34    sharp_string = tf.read_file(train_sharp_dir)
35
36    blur_decoded = tf.image.decode_png(blur_string, channels=3)
37    sharp_decoded = tf.image.decode_png(sharp_string, channels=3)
38    img = tf.concat([blur_decoded, sharp_decoded], axis=2)
39
40    return img
```

**Figure 3.1.** The `train_parse()` function

After reading, at each batch, a 256×256 patch is directly randomly cropped in the blurred and corresponding sharp image. To prevent the overfitting, the patches are randomly flip up-to-down or left-to-right, presented in line 43-45 in Figure 3.2.

As mention in Figure 2.1, the patch B2 and B3 are down-sampling from the original path B1 by using Bicubic algorithm. However, in Tensorflow, the function `tf.image.resize_images()` using Bicubic method gives the output image with very low quality comparing to the function in Matlab. Therefore, the function `imresize()` in Matlab are re-implemented by Python [1]. After using the Python implementation of Matlab `imresize()`, shown in line 49-50 of Figure 3.2, the values of the input and ground truth in each level of the network are convert from `uint8` to `float32` and normalized data range from -0.5 to 0.5 following the paper [2], for more coding detail, see the rest of the `train_preprocess()` in Figure 3.2 above.

```
def train_preprocess(img):
    patches = tf.random_crop(img, [patch_size, patch_size, 6])
    patches = tf.image.random_flip_left_right(patches)
    patches = tf.image.random_flip_up_down(patches)

    blur1, sharp1 = tf.split(patches, 2, axis=2)

    blur2 = tf.py_func(img_resize, [blur1, (ps2,ps2)], [tf.uint8])
    sharp2 = tf.py_func(img_resize, [sharp1, (ps2,ps2)], [tf.uint8])

    blur2 = tf.reshape(blur2, [ps2, ps2, 3])
    sharp2 = tf.reshape(sharp2, [ps2, ps2, 3])

    blur3 = tf.py_func(img_resize, [blur1, (ps3,ps3)], [tf.uint8])
    sharp3 = tf.py_func(img_resize, [sharp1, (ps3,ps3)], [tf.uint8])
    blur3 = tf.reshape(blur3, [ps3, ps3, 3])
    sharp3 = tf.reshape(sharp3, [ps3, ps3, 3])

    blur1 = tf.image.convert_image_dtype(blur1, tf.float32)
    sharp1 = tf.image.convert_image_dtype(sharp1, tf.float32)
    blur2 = tf.image.convert_image_dtype(blur2, tf.float32)
    sharp2 = tf.image.convert_image_dtype(sharp2, tf.float32)
    blur3 = tf.image.convert_image_dtype(blur3, tf.float32)
    sharp3 = tf.image.convert_image_dtype(sharp3, tf.float32)

    blur1 = tf.subtract(blur1, 0.5)
    sharp1 = tf.subtract(sharp1, 0.5)
    blur2 = tf.subtract(blur2, 0.5)
    sharp2 = tf.subtract(sharp2, 0.5)
    blur3 = tf.subtract(blur3, 0.5)
    sharp3 = tf.subtract(sharp3, 0.5)

    return blur1, sharp1, blur2, sharp2, blur3, sharp3
```

**Figure 3.2.** The `train_preprocess()` function

When finishing the reading images and preprocessing, the patches are pushed to the network for training with total 1000 epochs and batch size equals to 2. The Adam optimization is used with default parameters excepting the learning rate. In the first 400 epochs, the initial learning rate is $5\times10^{-5}$, then the learning rate is decreased by half after each 100 epochs. In order to avoid the gradient vanishing and explosion, caused by some unusual patches also called outliers, the gradients are clipped in range [-0.5, 0.5].

The loss function is written in the following equation:

$$L = \sum_{k=1}^{3} \frac{1}{3w_k h_k} \|L_k - S_k\|^2$$

Where:

- $w_k$ is the width of the patch or image

- $h_k$ is the height of the patch or image
- $L_k$ is the pixel value of the prediction at the $k^{th}$ level
- $S_k$ is the pixel value of the ground truth at the $k^{th}$ level

In other word, the loss function is the summation of Mean Squared Errors (MSEs) at three levels of the network. Hence, the loss function could be implemented in Tensorflow as below:

```
loss = tf.losses.mean_squared_error(S1, L1) +
tf.losses.mean_squared_error(S2, L2) +
tf.losses.mean_squared_error(S3, L3)
```

The training process consumed about 53 hours meaning more than 2 days with an Intel(R) Core(TM) i7 CPU @ 4.20 GHz and a GPU NVIDIA GTX GeForce 960Ti. When the convergence happened, the average training loss is about 0.005. By using Tensorboard, the values of the loss function are recorded at every iteration during the training process in the diagram of Figure 3.3.

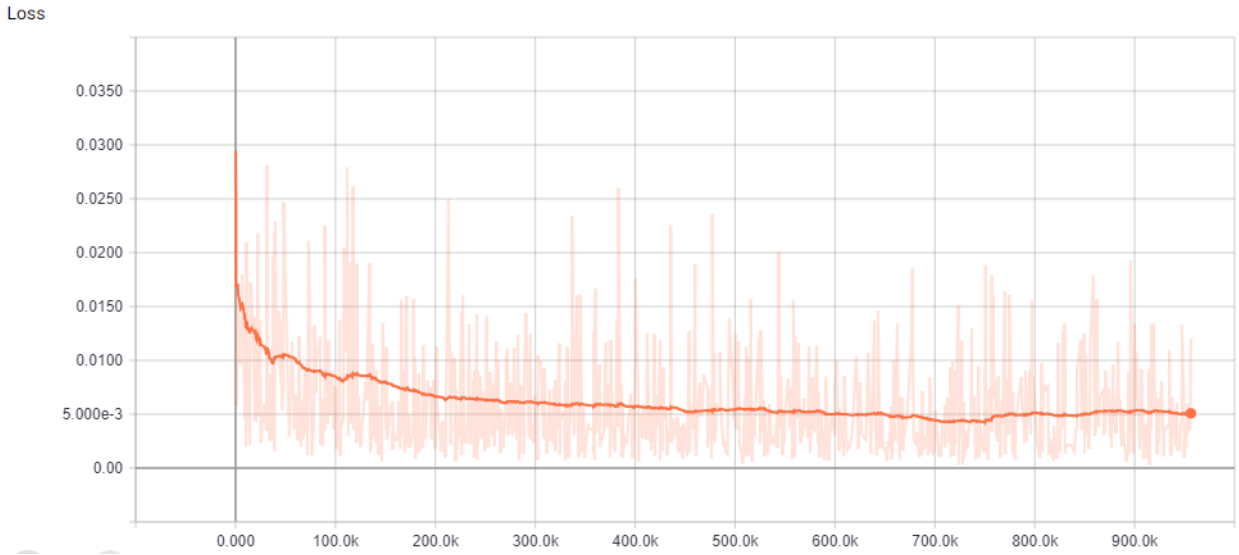


**Figure 3.3.** The loss value at every iteration

## 4. Validation

In the preprocess, the differences between validation and training are that there are not random crop and data augmentation in validation instead entire images are read into the trained model. Total execution time for all validation images is 47 minutes. Hence, one validation image takes averagely 2.5 second. Table 4.1 below shows the performance of the trained model with 3 demanded samples and average of all validation images.

**Table 4.1.** The quality of three required images and average of the validation set with pixel shuffle

| Images | Loss | PSNR (dB) | SSIM | MS-SSIM |
|---|---|---|---|---|
| **GOPR0384_11_00/000001.png** | 0.00473 | 27.77 | 0.8216 | 0.9228 |
| **GOPR0384_11_05/004001.png** | 0.00536 | 27.72 | 0.9017 | 0.9443 |
| **GOPR0385_11_01/003011.png** | 0.00611 | 27.20 | 0.8537 | 0.9070 |
| **Average of validation set** | 0.00542 | 27.99 | 0.8458 | 0.9269 |

It is unsuitable for comparing these results with the paper [2] because the paper [2] does not only consist the multiscale convolution neural network but also the discriminator of GAN and used the adversarial loss. As said above, the up-convolution can be pixel shuffle or transposed convolution. In three required images, the differences in quality are insignificant, but in average, the results of the model using pixel shuffle are better just little bit.

**Table 4.2.** The quality of three required images and average of the validation set with transposed convolution

| Images | Loss | PSNR (dB) | SSIM | MS-SSIM |
|---|---|---|---|---|
| **GOPR0384_11_00/000001.png** | 0.00475 | 27.78 | 0.8222 | 0.9242 |
| **GOPR0384_11_05/004001.png** | 0.00559 | 27.48 | 0.9014 | 0.9436 |
| **GOPR0385_11_01/003011.png** | 0.00604 | 27.31 | 0.8557 | 0.9105 |
| **Average of validation set** | 0.00545 | 27.97 | 0.8452 | 0.9265 |

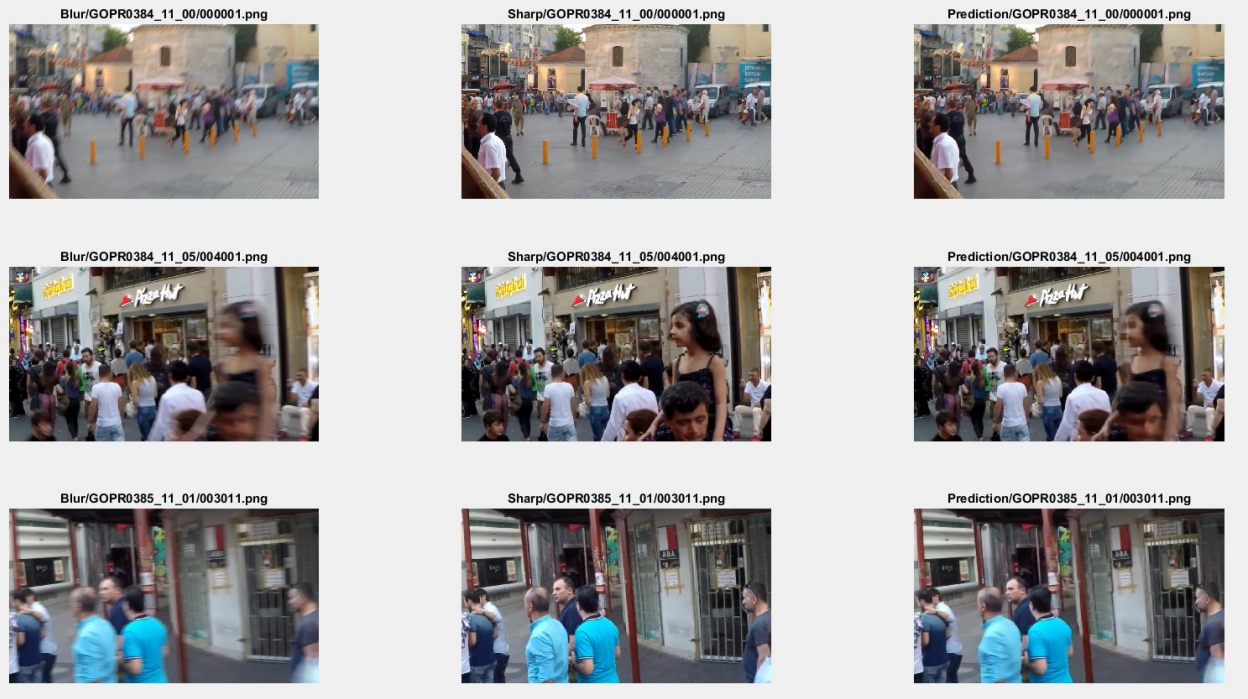


**Figure 4.1.** Three required examples in the validation

Figure 4.1 visualizes the blurred, sharp and prediction images of three required examples. It is noticeable to see that the prediction images are better than the blurred images, but not

sharp as the ground truth. Moreover, there are many details still look blurred in the prediction images such as the legs of the walking woman in the middle of GOPR0384_11_00/000001.png, the girl's face in GOPR0384_11_05/004001.png and the bot-right corner of GOPR0385_11_01/003011.png, which are zoom in Figure 4.2.

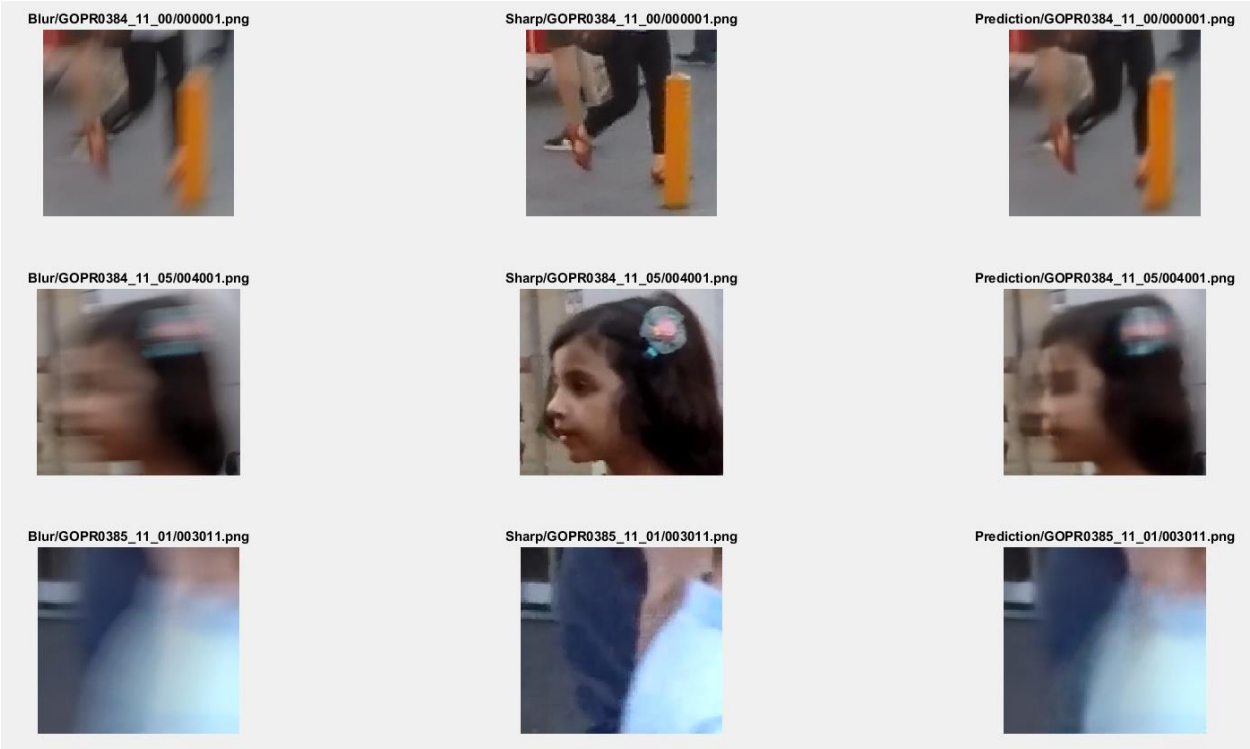


**Figure 4.2.** The unreconstructed details

## 5. Testing

The testing process is similar with the validation process during preprocess data before going to the model. However, in real world, in testing, there is not a sharp image along with the blurred one which mean that the quality of the output images cannot be measure by multi-scale content loss, PSNR, SSIM and MS-SSIM. Therefore, in coding, these measurements are not included.

## Reference

[1] Python implementation of Matlab `imresize()` function:

https://github.com/fatheral/matlab_imresize/blob/master/imresize.py

[2] Seungjun Nah, Tae Hyun Kim, Kyoung Mu Lee, "Deep Multi-scale Convolution Neural Network for Dynamic Scene Deblurring", IEEE Conference on Computer Vision and Pattern Recognition (CVPR), July 2017, USA

[3] Deep Deblur Release: https://github.com/SeungjunNah/DeepDeblur_release