Introduction:

This is the pytorch-based codes written by "Hamidreza Dastmalchi and Hassan Aghaeinia" corresponding to the relevant article with the title “Super-Resolution of Very Low-Resolution Face Images with a Wavelet Integrated, Identity Preserving, Adversarial Network”, submitted to the “Signal Processing: Image Communication” journal.

A Description about the approach:

The article presents an algorithm called (**WIPA**), standing for **W**avelet-integrated **I**dentity **P**reserving **A**dversarial super-resolution. Most of the super-resolution CNN networks use a pixel-wise MSE function as the loss function to train the network. However, as a remedy to the blurry effect of the pixel-wise MSE loss function, we have proposed to integrate wavelet features to the original super-resolution features as they contain very important high-frequency details. The proposed method uses wavelet prediction blocks to predict wavelet detail coefficients. The estimated wavelet coefficients are concatenated with the original feature maps in different scales to improve the reconstruction of high-frequency components. Furthermore, in order to improve the perceptual quality of the images, the adversarial GAN loss function and also the VGG perceptual loss function have been combined with the final loss function. Additionally, identity of the facial images has been preserved by aggregating an identity loss function.

A description about the code files:

There are mainly 7 folders in the root, each containing some important information. The “pretrained” folder contains the state dictionaries of the pretrained models, while the “checkpoint” folder stores the state dictionaries of the models during training. The “data” folder hold the training and testing images, where the “results” folder contains the resulted super-resolved images during test. The “logs” folder also carry the tensorboard logs to show the loss values and some samplr super-resolved images during training.

The codes are consisted of two main files: the “main.py” file for training the network and the “test.py” file for evaluating the algorithm with different metrics like PSNR, SSIM and verification rate. Before starting to train or test the network, you must put the training images in the corresponding folders:

- Put training images in “.\data\train” directory.
- Put celeba test images in “.\data\test\celeba” , lfw test images in “.\data\test\lfw” and helen test images in “.\data\test\helen”.

To train the network, simply run this code in Anaconda terminal:

```
>> python main.py
```

And for evaluating (testing), kindly run the following code in terminal:

```
>>python test.py
```

We designed different input arguments for controlling the training procedure, listed as below:

```
-h, --help          show this help message and exit
  --epoch_num EPOCH_NUM
                    number of training epochs
  --batch_size BATCH_SIZE
                    batch size of training procedure
  --num_workers NUM_WORKERS
                    number of cpu workers for data loading
```

```
--disable_cuda DISABLE_CUDA
                set True if you want to disable cuda
--lr LR         learning rate for training procedure
--wavelet_integrated WAVELET_INTEGRATED
                set True if you want to integrate wavelet coefficients
--GAN GAN       set True if you want to include GAN adversarial training
--scale SCALE   the upscaling factor
--adv_weight ADV_WEIGHT
                the weight of the adversarial loss function
--id_weight ID_WEIGHT
                the weight of the identity loss function
--mse_weight MSE_WEIGHT
                the weight of the mse loss function
--vgg_weight VGG_WEIGHT
                the weight of the vgg perceptual loss function
--wavelet_weight WAVELET_WEIGHT
                the weight of the wavelet loss function
--beta1 BETA1   the beta1 coefficient for Adam optimizer
--beta2 BETA2   the beta2 coefficient for Adam optimizer
--decay_rate DECAY_RATE
                the decay rate of learning rate
--epochs_todecay EPOCHS_TODECAY
                number of epochs to decay the learning rate
--num_iter_tolog NUM_ITER_TOLOG
                number of iterations to log the performance
--seed SEED     the seed of random generator
--train_root TRAIN_ROOT
--test_root TEST_ROOT
--checkpoints_root CHECKPOINTS_ROOT
--pretrained_folder PRETRAINED_FOLDER
--base_net BASE_NET  the file name of the pre-trained baseline network
--wi_net WI_NET     the file name of the pre-trained wavelet-integrated network
--disc_net DISC_NET  the file name of the pre-trained discriminator network
--sphere_net SPHERE_NET
                the file name of the pre-trained sphere network
--log_dir LOG_DIR
```

It is to be noted that the input arguments are set as default values to match our requirements. However, you can change the argument to the desired values. For example, to train the wavelet-integrated network through GPU with scale factor of 8, without having pre-trained model coefficients, with learning rate of 5e-5, you can simply run the following code in the terminal:

```
>>python main.py –scale 8 –wi_net “” –disc_net “” –wavelet_integrated True –lr 0.00005
```

We have also developed different options as input arguments to control the testing procedure, depicted as below:

```
optional arguments:
  -h, --help            show this help message and exit
  --batch_size BATCH_SIZE
                        batch size of training procedure
  --disable_cuda DISABLE_CUDA
                        set True if you want to disable cuda
  --wavelet_integrated WAVELET_INTEGRATED
                        set True if you want to integrate wavelet coefficients
  --scale SCALE         the upscaling factor
  --test_root TEST_ROOT
  --save_folder SAVE_FOLDER
  --save_flag SAVE_FLAG
                        set True if you want to save the super-resolved images
  --lfw_data LFW_DATA
  --lfw_pair_file LFW_PAIR_FILE
  --pretrained_folder PRETRAINED_FOLDER
  --base_net BASE_NET   the file name of the pre-trained baseline network
  --wi_net WI_NET       the file name of the pre-trained wavelet-integrated network
  --sphere_net SPHERE_NET
                        the file name of the pre-trained sphere network
  --metrics METRICS [METRICS ...]
```

You can evaluate psnr, ssim, fid score and also verification rate by the “test.py” file. To do this, you have to put the test images in the corresponding folders in data root at first.

In order to evaluate the psnr and ssim of a wavelet-integrated pretrained model in scale of 8 and save the super-resolved results in folder of “./results/celeba”, you can write the following code in the command window:

```
>> python test.py --wavelet_integrated True --scale 8 --wi_net gen_net_8x --save_flag True --save_folder ./results/celeba --metrics psnr ssim
```

To estimate the fid score, you have to produce the super-resolved test images first. Therefore, if you have not generated the super-resolved images, you have to call **–metrics psnr ssim** with **fid** simultaneously. You can also add the acc option to the metrics to evaluate the verification rate of the model:

```
>> python test.py --wavelet_integrated True --scale 8 --wi_net gen_net_8x --save_flag True --save_folder ./results/celeba --metrics psnr ssim fid acc
```

In addition, we have developed a “demo.py” python file to demonstrate the results of some sample images in the “./sample_images/gt” directory. To run the demo file, simply write the following code in terminal:

```
>>python demo.py
```

By default, the images of “./sample_images/gt” folder will be super-resolved by the wavelet-integrated network by scale factor of 8 and the results will be saved in the “./sample_images/sr” folder. To change the scaling factor, one must alter not only the **–scale** option but also the corresponding **–wi_net** argument to import the relevant pretrained state dictionary.