IPR PROJECT-1

MetaF2N: Blind Image Super-Resolution by
Learning Efficient Model Adaptation from Faces

Raj Agrawal (210809)

1 Dataset Description

1.1 Training Dataset

Regarding the dataset, the original study used 30,000 images from the FFHQ dataset and 3,500 images from the DF2K dataset. After processing, the data size amounted to nearly 70GB. Due to hardware and time limitations, I reduced the dataset by cropping and selecting 10,000 images from FFHQ and 1,000 images from DF2K. These datasets, consisting of RGB images in `.PNG` format, were preprocessed prior to training the model.

1.2 Dataset Preprocessing

Once the environment was set up, I proceeded to prepare the dataset. I selected 250 images from the DF2K dataset and generated multiscale images using the `generate_multiscale.py` script, resulting in 1,000 DF2K_multiscale images. Additionally, 10,000 images from the FFHQ dataset were prepared. The next step was generating `.tfrecord` files using the `generate_tfrecord.py` script to ensure compatibility with TensorFlow. The FFHQ images were converted into a `wholeFace.tfrecord` file, while the DF2K images were converted into a `back400.tfrecord` file. Therefore, the final processed dataset for training consisted of these two `.tfrecord` files. The preprocessed data can be found on the google drive- **Preprocessed data**.

1.3 Testing Dataset

For testing, I employed six datasets: `FFHQ_iid`, `FFHQ_ood`, `FFHQ_multiiid`, `FFHQ_multiood`, `CelebA_iid`, and `CelebA_ood`. Each dataset consisted of 1,000 images, except for `FFHQ_iid`, which included 100 images. Testing took approximately 3.5 to 4 hours to complete. After the testing phase, I computed evaluation metrics such as FID (Fréchet Inception Distance) loss and LPIPS (Learned Perceptual Image Patch Similarity) loss by comparing the generated images with the ground truth data. The test data can be accessed via Google Drive at the following link: **Test Data**.

2 Implementation Details

2.1 GPU requirements and Epochs details

In the original research paper, the authors utilized an NVIDIA A6000 GPU with 48GB of memory and trained their model over a span of seven days. However, due to the unavailability of such hardware and time constraints, I adapted the setup by using a 24GB GPU and trained the model for 10 hours. On this configuration, the training process took approximately 300 seconds for every 10 epochs, and I executed a total of 1200 epochs.

2.2 Environment Setup

The first step involved setting up the environment by creating a Python 3.9 virtual environment in Anaconda, as this version was most suitable for the project requirements. Afterward, all necessary packages were installed using the `requirements.txt` file.

2.3 Training

I updated the `configure.py` script, setting the number of training iterations, batch size, and learning rate. I then trained the model using the `main.py` script, running 1,200 epochs over a period of 10 hours. The trained model was saved in the weights directory.

2.4 Testing

For testing, I employed various datasets, each consisting of four image types: LQ (Low Quality), HQ (High Quality), and GT (Ground Truth). FaceLQ and FaceHQ images were generated using the `generate_test_faces.py` script. I executed the `test.py` script to evaluate the model on the selected datasets, and the results were stored in the respective directories. Finally, I computed evaluation metrics, including FID and LPIPS, using the `calculate_metric.py` script.

3 GitHub Link

The code for the implementation, the requirements.txt file for environment setup, and the results can be found in the following GitHub repository: **MetaF2N**

4 Results: Tables and Figures

The results of the test data images on which I tested the model can be found on Google Drive at the following link: **Result Data**.

4.1 Figures

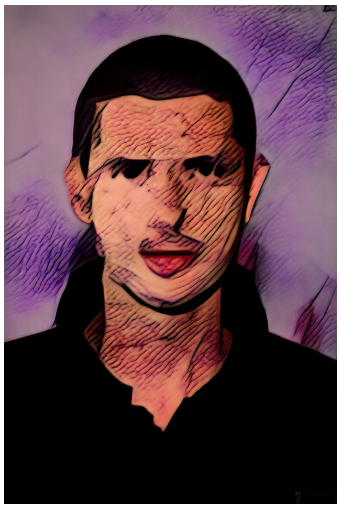
Some of the pairs provided below serve as references for evaluating the performance of my model.For more you can check **Result Data**.

(a) LQ_celebA_iid

(b) LQ_celebA_ood

(c) LQ_FFHQ_multi_iid

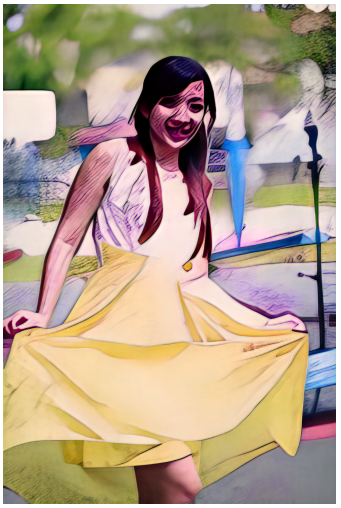
(d) LQ_FFHQ_iid



(e) Result_celebA_iid

(f) Result_celebA_ood

(g) Result_FFHQ_multi_iid



(h) Result_FFHQ_iid

Figure 1: Test Results

4.2 Loss Metric

Due to the network driver error on the GPU provided I were unable to calculate metrics for losses, But it can be calculated using `calculate_metric.py`. I will upload separate file on Github after resolving this issue or try running my code on another GPU.

4.3 Table

Losses during Training and Iteration time

Epoch No.	Epoch Time (s)	Epoch Loss (Post)
1	66.83	0.644
10	192	0.5280
100	211	0.173
200	215	0.03
500	239	0.01
800	214	0.02
1200	216	0.003

Table 1: Performance metrics for each epoch during training.

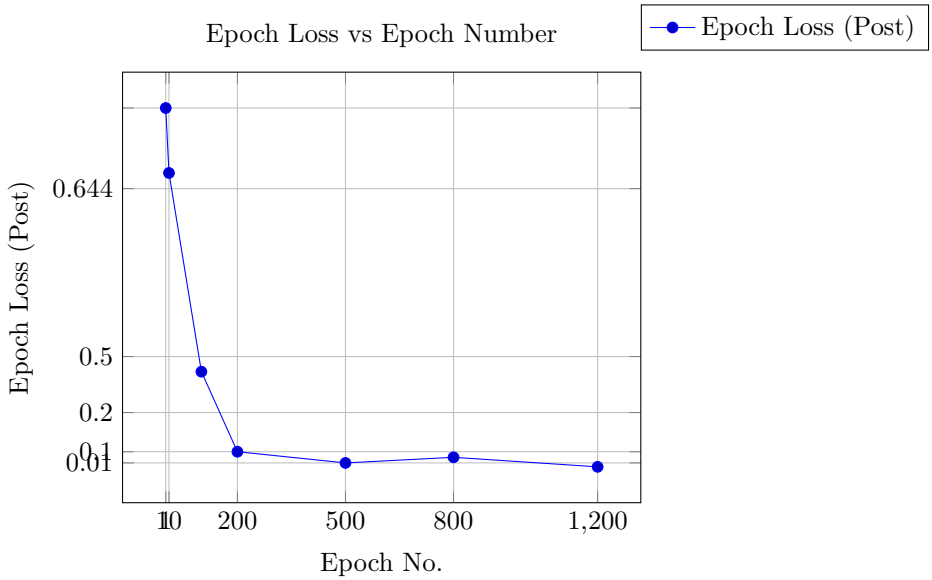


Figure 2: Graph of Epoch Loss (Post) over Epoch Number