Mini Project 2: Image Deblurring

GNR638: Deep Learning for Image Analysis

April 10, 2024

GROUP MEMBERS
Sanika Padegaonkar (20D070069)
Sanjhi Priya (20D070070)

Contents

1 Objective

The objective of this project is to develop a deep learning model for image deblurring, aiming to enhance the quality of blurred images through state-of-the-art techniques. This involves:

1. Preprocessing a dataset of high-resolution images by downscaling and generating a varied set of blurred images using Gaussian filters.
2. Designing and implementing a neural network architecture capable of effectively deblurring images while adhering to a parameter constraint.
3. Evaluating the performance of the deblurring model using the Peak Signal-to-Noise Ratio (PSNR) metric on a test dataset, alongside the provided evaluation script.
4. Ensuring compatibility and consistency in preprocessing and evaluation processes by utilizing specified library versions.

2 Approach Used:

2.1 Dataset

The dataset consists of 240 classes, each containing 100 images of random scenes in daily life.

Figure 1: Sample Images

2.2 Model Architecture

The model architecture used is a **convolutional autoencoder** designed to process images.

2.2.1 Encoder

The encoder component takes **256x448x3** images as input and progressively reduces their dimensions through convolutional layers with increasing numbers of filters (8, 16, 32) followed by max-pooling. The output of the encoder is then flattened into a 128-dimensional latent space representation.

2.2.2 Decoder

On the other hand, the decoder component reconstructs the original images from the latent space representation. It first expands the latent vector through a dense layer, reshapes it into a **32x56x32 tensor**, and then applies three transpose convolutional layers to upsample it back

to the original image size. The final layer outputs reconstructed images of size 256x448x3, matching the input shape.

```
Model: "encoder"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 encoder_input (InputLayer)  [(None, 256, 448, 3)]     0

 conv2d (Conv2D)             (None, 128, 224, 8)       224

 conv2d_1 (Conv2D)           (None, 64, 112, 16)       1168

 conv2d_2 (Conv2D)           (None, 32, 56, 32)        4640

 flatten (Flatten)           (None, 57344)             0

 latent_vector (Dense)       (None, 128)               7340160

=================================================================
Total params: 7346192 (28.02 MB)
Trainable params: 7346192 (28.02 MB)
Non-trainable params: 0 (0.00 Byte)
```

(a) Encoder Architecture

```
Model: "decoder"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 decoder_input (InputLayer)  [(None, 128)]             0

 dense (Dense)               (None, 57344)             7397376

 reshape (Reshape)           (None, 32, 56, 32)        0

 conv2d_transpose (Conv2DTr  (None, 64, 112, 32)       9248
 anspose)

 conv2d_transpose_1 (Conv2D  (None, 128, 224, 16)      4624
 Transpose)

 conv2d_transpose_2 (Conv2D  (None, 256, 448, 8)       1160
 Transpose)

 decoder_output (Conv2DTran  (None, 256, 448, 3)       219
 spose)

=================================================================
Total params: 7412627 (28.28 MB)
Trainable params: 7412627 (28.28 MB)
Non-trainable params: 0 (0.00 Byte)
```

(b) Encoder Architecture

2.2.3 Autoencoder

The autoencoder combines both the encoder and decoder components. During training, it learns to minimize the mean squared error between the reconstructed images and the original inputs using the Adam optimizer. Accuracy and peak signal to noise ratio are monitored as metrics.

```
Model: "autoencoder"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 encoder_input (InputLayer)  [(None, 256, 448, 3)]     0

 encoder (Functional)        (None, 128)               7346192

 decoder (Functional)        (None, 256, 448, 3)       7412627

=================================================================
Total params: 14758819 (56.30 MB)
Trainable params: 14758819 (56.30 MB)
Non-trainable params: 0 (0.00 Byte)
```

Figure 3: Final Autoencoder Architecture

Method:

- The input images are first resized to the desired size of (256,448,3).
- Three different Gaussian filters are applied to the sharp images to create a dataset of blurred images which is thrice the size of the sharp images dataset.
- These blurred images are passed to the autoencoder along with sharp images which act as ground truth values for training.
- The autoencoder is trained with pixel-wise mse as the loss function.
- The blurred images from the custom test set are fed to the autoencoder and the predictions of the autoencoder i.e. the de-blurred images are saved.
- These de-blurred images are input to the evaluation script provided and the PSNR between de-blurred and sharp images is computed and observed.

3 Training Details

- The base model used is an **autoencoder**. The **encoder** has 3 convolutional layers followed by a fully connected layer which maps the convolution output to a latent space of dimension **latent_dim** which was set to **256**. The **decoder** is approximately the inverse of the encoder.
- The model was trained for **25 epochs** and the images were reshaped to have a size of **(256,448,3)** as recommended.
- The **loss function** used was **pixel-wise MSE** and the **metrics** used were **accuracy** and **peak signal to noise ratio**.
- The **learning rate scheduler** used was **ReduceLROnPlateau** and the **optimizer** used was **Adam**.
- The checkpoint of the best model was saved using keras.callbacks.ModelCheckpoint.

4 Results and Analysis

4.1 Results

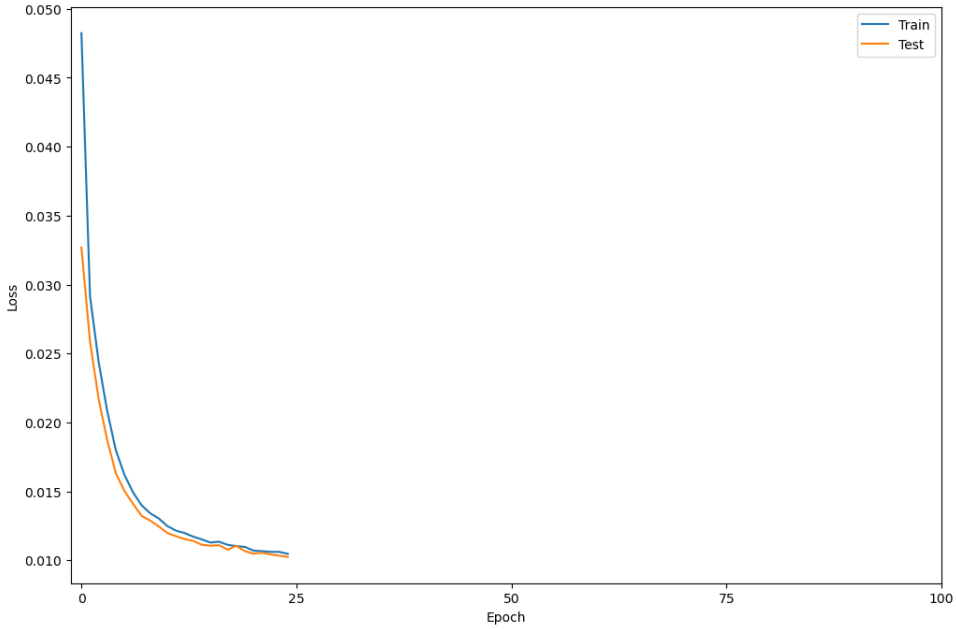

Figure 4: Training and validation loss vs No. of epochs

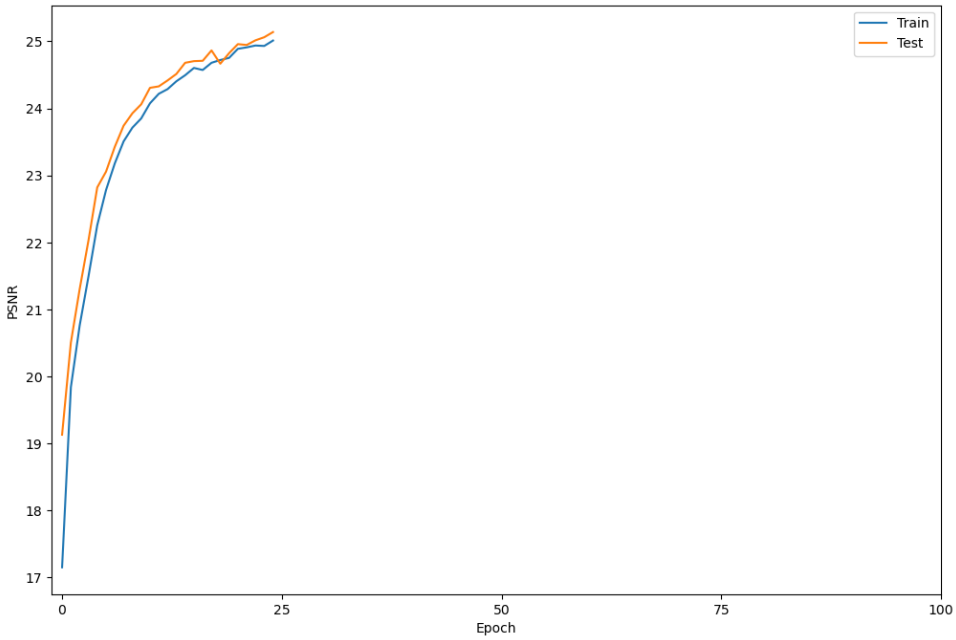

Figure 5: Training and validation PSNR vs No. of epochs

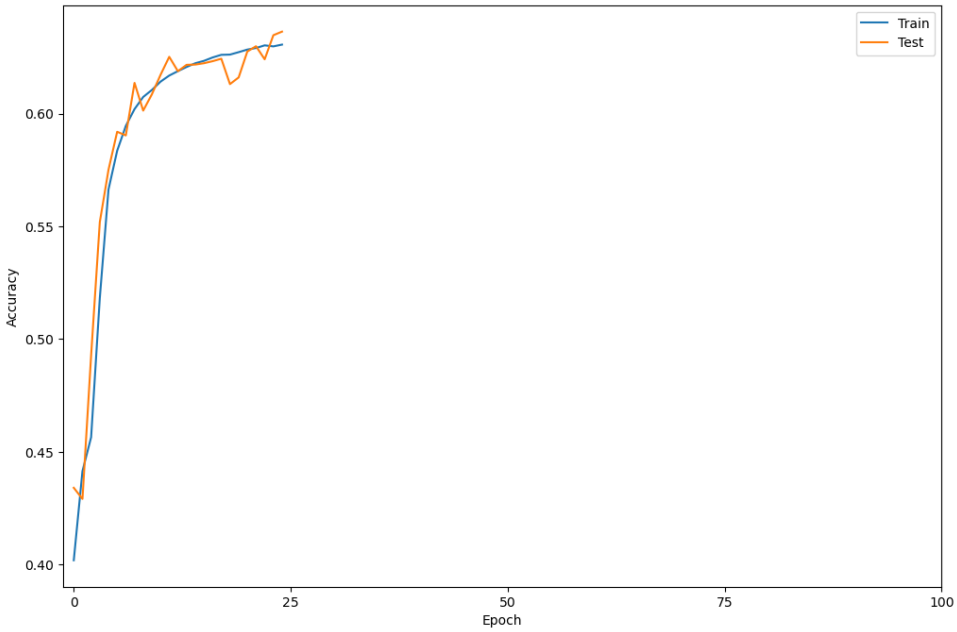

Figure 6: Training and validation accuracy vs No. of epochs

4.1.1 Sample outputs

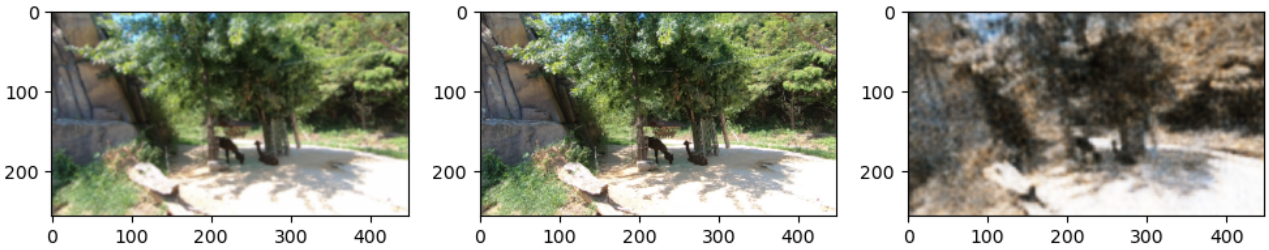

Figure 7: Blurred input(left), Sharp ground truth (middle), De-blurred output (right)

4.2 Analysis

	PSNR (in dB)	Accuracy
Train	25.0123	0.6306
Validation	25.1389	0.6364
Testing	15.519613317153283	N/A

Table 1: Metrics

The final **PSNR** obtained on the custom test set was **15.519613317153283 dB**.
The test PSNR would have increased if the entire trainig set could be used instead of just 10% images from each folder.
Number of autoencoder parameters = 14758819

5 Conclusion

In this project, we employed an autoencoder architecture for image de-blurring using a dataset of sharp images. Our model, designed to stay within the 15 million parameter limit, yielded a notable **training PSNR** of **25.0123 dB**, **validation PSNR** of **25.1389 dB** and **testing PSNR** of **15.519613317153283 dB**.

Computational Constraints: Due to computational limitations, our model was constrained to use only 10% of the images from all 240 training folders. Despite this constraint, we achieved a commendable PSNR of 25.0123 dB on the train set, showcasing the efficiency of our model.

Parameter Utilization: We optimized the model within the specified limit, utilizing only 14.76 million parameters out of the allowed 15 million. This deliberate parameter efficiency choice demonstrates the potential for further improvement in accuracy with additional model complexity. Given more computational resources, extending the model's capacity could lead to even higher performance in fine-grained image classification tasks.

Our conclusion underscores the success of the autoencoder model in fine-grained image de-blurring, emphasizing its potential for accurate and parameter-efficient solutions.

6 Checkpoint - Link

Link to the checkpoint

7 Project Link

Link to all files