Introduction

This project explores the application of deep learning for grayscale image denoising, using real-world blurred images from the GoPro dataset and synthetically altered images from the BSD500 dataset. Our team decided to split our focus to compare results. My partner implemented a custom model using PyTorch on a separate dataset, while here, I focus on developing a deep UNet architecture using TensorFlow. We evaluated the performance of our models individually and in crossover scenarios, testing each on the other's dataset to identify the best model.

This report outlines the steps taken to design, implement, and train the deep UNet model. It details dataset processing, training methodology, loss functions, and evaluation metrics such as PSNR and SSIM. My goal was to create a model capable of generalizing across varied image degradations. In the end, we compared results across all model configurations and select a final model that leverages the strengths of both approaches.

Description of My Individual Work

My individual contribution is the building of the deep UNet model in TensorFlow for grayscale image restoration. I created the implementation pipeline starting with preprocessing data, to designing the network architecture, training, and evaluation.

I began by writing custom data loaders for two datasets. The GoPro dataset includes over 10,000 real-world motion-blurred images with corresponding ground truths. The BSD500 dataset, originally meant for segmentation tasks, was adapted by applying Gaussian blur and synthetic noise to create degradation. All images were resized to 256×256 and normalized to the [0, 1] range to ensure consistency and compatibility with the model.

The UNet follows a symmetrical encoder/decoder architecture. The encoder uses four convolution blocks paired with max pooling layers to downsample spatial information and capture hierarchical features. The decoder mirrors the encoder, using upsampling and skip connections to restore detailed image features. A 1024 filter bottleneck layer in the center enhances the models ability to learn more abstract features and representations before reconstruction.

```
# Encoder
c1 = layers.Conv2D(64, 3, activation='relu', padding='same')(inputs)
c1 = layers.Conv2D(64, 3, activation='relu', padding='same')(c1)
p1 = layers.MaxPooling2D((2, 2))(c1)  # 128x128

c2 = layers.Conv2D(128, 3, activation='relu', padding='same')(p1)
c2 = layers.Conv2D(128, 3, activation='relu', padding='same')(c2)
p2 = layers.MaxPooling2D((2, 2))(c2)  # 64x64

c3 = layers.Conv2D(256, 3, activation='relu', padding='same')(p2)
c3 = layers.Conv2D(256, 3, activation='relu', padding='same')(c3)
p3 = layers.MaxPooling2D((2, 2))(c3)  # 32x32

c4 = layers.Conv2D(512, 3, activation='relu', padding='same')(p3)
c4 = layers.Conv2D(512, 3, activation='relu', padding='same')(c4)
p4 = layers.MaxPooling2D((2, 2))(c4)  # 16x16

# Bottleneck
bn = layers.Conv2D(1024, 3, activation='relu', padding='same')(p4)
bn = layers.Conv2D(1024, 3, activation='relu', padding='same')(bn)

# Decoder
u4 = layers.UpSampling2D((2, 2))(bn)
u4 = layers.Concatenate()([u4, c4])
c5 = layers.Conv2D(512, 3, activation='relu', padding='same')(u4)
c5 = layers.Conv2D(512, 3, activation='relu', padding='same')(c5)

u3 = layers.UpSampling2D((2, 2))(c5)
u3 = layers.Concatenate()([u3, c3])
c6 = layers.Conv2D(256, 3, activation='relu', padding='same')(u3)
c6 = layers.Conv2D(256, 3, activation='relu', padding='same')(c6)

u2 = layers.UpSampling2D((2, 2))(c6)
u2 = layers.Concatenate()([u2, c2])
c7 = layers.Conv2D(128, 3, activation='relu', padding='same')(u2)
c7 = layers.Conv2D(128, 3, activation='relu', padding='same')(c7)

u1 = layers.UpSampling2D((2, 2))(c7)
u1 = layers.Concatenate()([u1, c1])
c8 = layers.Conv2D(64, 3, activation='relu', padding='same')(u1)
c8 = layers.Conv2D(64, 3, activation='relu', padding='same')(c8)

outputs = layers.Conv2D(1, 1, activation='sigmoid', padding='same')(c8)
```

Figure 1: Architecture of the deep UNet used for grayscale denoising.

For the loss function, I combined Mean Absolute Error (MAE) and Mean Squared Error (MSE). This hybrid approach is to balance sensitivity to large errors (MSE) with overall pixel level consistency (MAE), making it more adapt to noise and outliers. I trained the model using the AdamW optimizer, which helps prevent overfitting by decoupling weight decay from gradient updates. Additionally, I implemented a cosine decay learning rate schedule to gradually reduce the learning rate, improving convergence stability. To

further prevent overfitting, I used early stopping and a model checkpoint callback that restored the best weights based on validation loss.

To evaluate denoising performance, I computed PSNR (Peak Signal-to-Noise Ratio) and SSIM (Structural Similarity Index) across validation and test sets. These metrics provided both pixel and perceptual evaluations of output quality. Last, I visualized predictions and performed cross verification with the dataset evaluations to gauge how well the model generalized to unseen types of noise and blur.

Detailed Description of My Work

Starting with data preparation, I wrote Python scripts to preprocess the GoPro dataset by iterating through structured directories of blurred and sharp image pairs, resizing and converting them to grayscale. For the BSD500 dataset, which originally lacked blurry samples, I generated noisy-blurred inputs by applying a Gaussian blur followed by Gaussian noise using PIL and NumPy libraries. This allowed me to synthetically degrade the data for training.

Using TensorFlow's tf.data.Dataset API, I designed a data pipeline where Batching, prefetching, and shuffling were implemented to support scalable training on GPU. I also split a validation set from the training data using train_test_split to monitor generalization.

To build the deep UNet architecture, I used a modular Keras layers and Model APIs. The encoder section used sequential convolution layers with ReLU activation, capturing increasingly abstract representations of the input. The decoder used upsampling layers and skip connections, which helped preserve spatial details by concatenating feature maps from earlier layers. This skip connection design enables fine details lost in downsampling to be recovered during reconstruction. I also added a bottleneck with 1024 filters to give the network better abstraction ability.

My training loop included a hybrid loss function that combined MAE and MSE. This loss balances global pixel accuracy and robustness to outliers, which I found particularly useful for very distorted images. I paired this with the AdamW optimizer to prevent overfitting and used a cosine decay learning rate to gradually prevent premature convergence. I monitored validation loss for early stopping, and used a model checkpoint callback to save the best weights.

For evaluation, I implemented Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index (SSIM) to examine perceptual and structural quality. These metrics were calculated on both validation and test datasets. I also created loss visualizations over training epochs and sample visual comparisons of noisy, denoised, and ground truth images.

Cross-evaluation played a major role I tested my U-Net on my partner's dataset and vice versa. This allowed us to explore how models behave in out-of-distribution settings and which designs are more robust. In this phase, I also assisted in documenting visual outputs, comparing metrics, and providing interpretation for joint analysis.

Results

The deep UNet model showed consistent performance across both datasets, despite the differences in how noise and blur were introduced. On the GoPro test set, the model achieved a PSNR of approximately 31.33 and an SSIM of 0.9226, indicating strong pixel level recovery and structural fidelity. On the BSD500 test set, which contained synthetically degraded images, the model still

performed well with a PSNR of 24.42 and SSIM of 0.6290, confirming its ability to generalize to synthetic distortions.

Training and validation losses over 35 epochs showed smooth convergence with minimal overfitting, due to early stopping and regularization through the use of the AdamW optimizer. The cosine decay learning rate scheduler allowed for better stabilization in later epochs, resulting in improved generalization on unseen data. Loss curves and metric trends demonstrated that the model not only learned to denoise but retained fine details and edges in the reconstructed output.

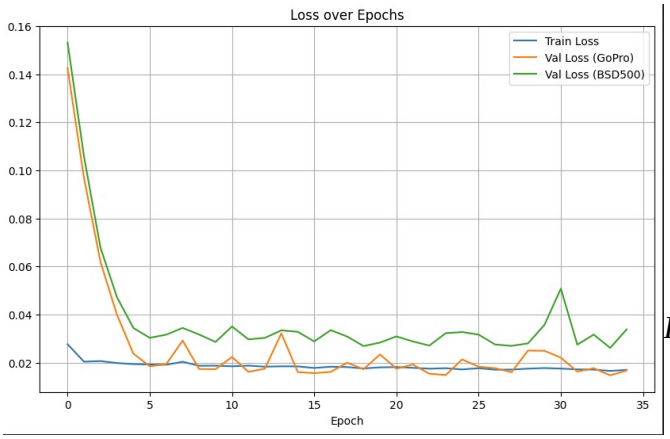

Figure 2: Hybrid MAE+MSE average loss with cosine learning rate scheduler over epochs

Figure 3: Clean

Figure 4: Noisy

When visually comparing the predicted images to ground truth samples, the UNet consistently removed motion blur and noise while preserving facial and object contours. These qualitative results aligned well with the quantitative metrics, supporting the architectural decisions and training strategy used. Additionally, running the UNet on the alternate dataset used by my partner further confirmed its robustness, making it a strong candidate for the final merged approach.

Figure 5: Denoised for 35 epochs

Summary and Conclusions

This project demonstrated the value of deep learning for grayscale image denoising by comparing two distinct approaches. While my partner implemented a custom model in PyTorch, I developed a deep U-Net using TensorFlow, trained on both real-world blurred GoPro images and synthetically degraded BSD500 images. My pipeline emphasized clean preprocessing, a robust encoder–decoder architecture, and a carefully tuned training setup, including combined MAE + MSE loss, cosine learning rate scheduling, and early stopping.

Our final comparisons showed that each model had unique strengths: mine performed consistently across varying noise conditions, while my partner's model excelled on their chosen dataset. Cross-evaluation helped highlight which components generalized best. Ultimately, we selected a hybrid configuration that leveraged our strongest architectural and training insights.

Through this process, I gained deeper experience in model development, data engineering, and performance evaluation. More importantly, I learned how collaborative model diversity and critical benchmarking lead to stronger, more adaptable solutions. Future improvements could include

extending to color images, experimenting with adversarial loss functions, or incorporating transformer-based denoisers for enhanced context awareness.

References

[1] Eko J. Salim, *"PRES Denoising (U-Net based autoencoder with PSNR + SSIM)"*, Kaggle Notebook. https://www.kaggle.com/code/ekojsalim/pres-denoising*State-of-the-Art Deblurring on GoPro Dataset*, Papers with Code.

[2] https://paperswithcode.com/sota/deblurring-on-goproUtkarsh Saxena, *"Image Denoising with Autoencoders"*, Kaggle Notebook.

[3] https://www.kaggle.com/code/utkarshsaxenadn/image-denoising-with-auto-encodersKaggle Discussion, *"Old-School Denoising Example"*.

[4] https://www.kaggle.com/discussions/questions-and-answers/553140Seungjun Nah, *GoPro Dataset Description Page*.

[5] https://seungjunnah.github.io/Datasets/goproBalraj Singh, *Berkeley Segmentation Dataset 500 (BSDS500)*, Kaggle Dataset.

[6] https://www.kaggle.com/datasets/balraj98/berkeley-segmentation-dataset-500-bsds500?resource=download