Deep Learning Final Project: Image Denoising/Deblurring and Restoration

A Comparative Study of NAFNet and UNet Architectures

1. Introduction

This report details our group project on image denoising and restoration using deep learning techniques. Our project focused on developing and comparing various deep learning models for enhancing the quality of degraded images, specifically addressing two problems: thermal image denoising and natural image deblurring. These tasks have numerous real world applications including night vision, building inspections, medical diagnostics, surveillance, and photography enhancement.

Image degradation occurs through various mechanisms depending on the imaging modality. Thermal cameras suffer from unique noise patterns related to their sensor technology, while standard cameras experience motion blur and other degradations. Our project aimed to develop solutions for both scenarios by exploring complementary approaches:

1. **Thermal Image Denoising**: We developed a noise simulation method to create realistic training data for thermal imagery and implemented the state of the art NAFNet (Nonlinear Activation Free Network) architecture in PyTorch.

2. **Natural Image Deblurring**: We utilized the GoPro dataset containing real world motion-blurred images and the BSD500 dataset with synthetically applied degradations, implementing a deep UNet architecture in TensorFlow, and later converting to PyTorch.

By pursuing these parallel approaches, our team was able to compare the effectiveness of different architectures (NAFNet vs. UNet) across varied image restoration tasks. This comparative approach provided insights into the strengths and limitations of each method.

Our project made several key contributions:

- Development of a thermal noise simulation model that accurately models sensor-specific degradations
- Implementation and optimization of the NAFNet architecture for thermal image denoising
- Implementation of a deep UNet architecture for natural image deblurring and denoising
- Cross-evaluation of both models on both datasets
- Development of visualization tools for qualitative and quantitative comparison

The following sections detail our approach, experimental setup, and findings, demonstrating the effectiveness of UNET and NAFTNET for image restoration tasks.

2. Project Schedule

Our project followed a structured timeline with clear milestones and task distribution between team members:

Date	Milestone	Assigned To	Status
March 15, 2025	Project proposal submission	Both team members	Completed
March 20, 2025	Dataset collection and preparation	Both team members	Completed
March 25, 2025	Thermal noise simulation pipeline	Thomas	Completed
March 25, 2025	GoPro/BSD500 preprocessing	Zach	Completed
April 5, 2025	NAFNet architecture implementation	Thomas	Completed
April 5, 2025	UNet architecture implementation	Zach	Completed
April 15, 2025	Initial model training and validation	Both team members	Completed
April 20, 2025	Hyperparameter optimization	Both team members	Completed
April 22, 2025	Visualization tools development	Both team members	Completed
April 25, 2025	Cross-evaluation of models	Both team members	Completed
April 28, 2025	Final results analysis	Both team members	Completed
May 1, 2025	Presentation preparation	Both team members	Completed
May 6, 2025	Final report submission	Both team members	Completed

The project development was organized into four main phases:

1. **Data Preparation Phase** (March 15-25): Collection, preprocessing, and synthetic data generation for both thermal and natural images.

2. **Model Implementation Phase** (March 29-April 5): Development of NAFNet and UNet architectures in their respective frameworks.
3. **Training and Optimization Phase** (April 9-22): Initial training, hyperparameter tuning, and performance optimization.
4. **Evaluation and Analysis Phase** (April 23-May 6): Cross-evaluation, result visualization, comparative analysis, and report preparation.

3. Description of the Data Sets

Our project utilized three distinct datasets, each serving different purposes in our image restoration experiments:

3.1 Thermal Imagery Dataset with Synthetic Noise

For thermal image denoising, we started with a collection of clean thermal images and developed a noise simulation pipeline to create paired noisy and clean training data. This approach was necessary because obtaining paired realworld thermal images is difficult to find.

Dataset characteristics:

- **Source**: Clean thermal images from publicly available thermal datasets
- **Size**: 500 training images, 100 validation images, 100 test images
- **Resolution**: 640×480 pixels
- **Format**: 16-bit grayscale (visualized with color mapping)

Noise simulation approach: Our synthetic noise generation pipeline modeled several key noise sources found in thermal imaging sensors:

1. **Gaussian Read Noise**: Electronics induced random noise following a Gaussian distribution
2. **Dark Current Shot Noise**: Poisson distributed noise arising from thermal generation of electrons
3. **Fixed Pattern Noise (FPN)**: Consisting of:
    - Dark Signal Non-Uniformity (DSNU): Additive offset variations between pixels
    - Photo Response Non-Uniformity (PRNU): Multiplicative gain variations between pixels

The noise simulation process followed this mathematical formulation:

$$I_{noisy} = (I_{clean} + DSNU) \times PRNU + N_{shot} + N_{read}$$

Where:

- I_{noisy} is the final noisy thermal image
- I_{clean} is the original clean thermal image
- $DSNU$ is the Dark Signal Non-Uniformity map (additive)
- $PRNU$ is the Photo Response Non-Uniformity map (multiplicative)
- N_{shot} is the shot noise following Poisson distribution
- N_{read} is the read noise following Gaussian distribution

This physically based approach ensured that our synthetic noisy images closely resembled real world thermal camera outputs, providing realistic training data for our models.

3.2 GoPro Dataset

For natural image deblurring, we utilized the GoPro dataset, which contains real world blurred and sharp image pairs:

Dataset characteristics:

- **Source**: GoPro HERO4 Black camera capturing videos at 240 fps
- **Size**: Over 10,000 image pairs (blurred and sharp)
- **Resolution**: Original 1280×720 pixels, resized to 256×256 for training
- **Format**: RGB color images

The GoPro dataset was created by capturing high speed video and generating blurred frames by averaging consecutive sharp frames, resulting in realistic motion blur. This dataset provides authentic blurred/sharp image pairs without synthetic degradation, making it ideal for training real world deblurring models.

3.3 BSD500 Dataset with Synthetic Degradation

As a supplementary dataset for deblurring experiments, we utilized the Berkeley Segmentation Dataset 500 (BSD500) with synthetically applied degradations:

Dataset characteristics:

- **Source**: BSD500, originally intended for image segmentation tasks
- **Size**: 500 images (200 for training, 100 for validation, 200 for testing)
- **Resolution**: Original varied sizes, resized to 256×256 for training
- **Format**: RGB color images converted to grayscale

Synthetic degradation process:

1. Applied Gaussian blur with varying kernel sizes (σ = 1.0 to 3.0)
2. Added Gaussian noise (σ = 0.01 to 0.05)
3. Created paired degraded/clean image sets for training and evaluation

This synthetically degraded dataset provided additional training data allows us

to test the generalization capabilities of our models.

3.4 Data Preprocessing and Augmentation

All datasets underwent consistent preprocessing to ensure compatibility with our models:

1. **Resizing**: Images were resized to appropriate dimensions (256×256 for GoPro/BSD500, original resolution maintained for thermal images)
2. **Normalization**: Pixel values normalized to [0,1] range
3. **Patch extraction**: During training, random 128×128 patches were extracted to increase training samples
4. **Data augmentation**: Applied random flips, rotations, and crops to prevent overfitting

For the thermal dataset, we implemented additional preprocessing steps:

- 16-bit to 8-bit conversion with appropriate scaling
- Histogram equalization for improved contrast
- Temperature range normalization

These preprocessing steps ensured consistent input data for our models while preserving the essential characteristics of each image type.

4. Description of the Deep Learning Networks and Training Algorithms

Our project explored two distinct deep learning architectures for image restoration: NAFNet and UNet. Each architecture has unique characteristics that make it suitable for different aspects of image restoration tasks.

4.1 NAFNet Architecture

NAFNet (Nonlinear Activation Free Network) represents a state of the art approach for image restoration tasks. Its key innovation is the elimination of traditional nonlinear activation functions, replaced by a simple gating mechanism that improves performance while reducing computational complexity.

4.1.1 Architectural Overview

The NAFNet follows a U-shaped encoder decoder structure with several distinctive features:

1. **NAFBlocks**: The fundamental building blocks of NAFNet, featuring:

   - SimpleGate activation mechanism
   - Simplified Channel Attention
   - Layer normalization
   - Skip connections with learnable scaling

2. **Network Structure**:

```
   - Encoder path with progressive downsampling
   - Middle blocks for feature processing
   - Decoder path with progressive upsampling
   - Skip connections between encoder and decoder at corresponding levels
   - Residual learning (input skip connection)

The overall architecture can be represented as:

```
Input → Initial Conv → Encoder (NAFBlocks + Downsampling) → Middle (NAFBlocks)
→ Decoder (NAFBlocks + Upsampling) → Final Conv → Output
```
```

4.1.2 Key Components

SimpleGate Mechanism:
The core innovation of NAFNet is the SimpleGate activation, which replaces traditional nonlinear activations like ReLU:

```python
class SimpleGate(nn.Module):
    def forward(self, x):
        x1, x2 = x.chunk(2, dim=1)
        return x1 * x2
```

This gating mechanism works by splitting the feature tensor along the channel dimension and multiplying the two halves element-wise, allowing the network to learn non-linear transformations without explicit activation functions.

Simplified Channel Attention:
NAFNet uses a lightweight channel attention mechanism to adaptively recalibrate channel wise feature responses:

```python
class SimpleChannelAttention(nn.Module):
    def __init__(self, n_feats):
        super().__init__()
        self.pool = nn.AdaptiveAvgPool2d(1)
        self.conv = nn.Conv2d(n_feats, n_feats, kernel_size=1)

    def forward(self, x):
        y = self.pool(x)
        y = self.conv(y)
        return x * y
```

NAFBlock Structure:
Each NAFBlock combines these components with layer normalization and skip connections:

```python
class NAFBlock(nn.Module):
    def __init__(self, c, DW_Expand=2, FFN_Expand=2):
        super().__init__()
        dw_channel = c * DW_Expand
        self.conv1 = nn.Conv2d(c, dw_channel, 1)
        self.conv2 = nn.Conv2d(dw_channel, dw_channel, 3, padding=1,
groups=dw_channel)
        self.conv3 = nn.Conv2d(dw_channel, c, 1)

        self.norm1 = nn.LayerNorm(c)
        self.norm2 = nn.LayerNorm(c)

        self.sca = SimpleChannelAttention(c)
        self.beta = nn.Parameter(torch.zeros((1, c, 1, 1)), requires_grad=True)
        self.gamma = nn.Parameter(torch.zeros((1, c, 1, 1)),
requires_grad=True)

    def forward(self, x):
        y = self.norm1(x)
        y = self.conv1(y)
        y = self.conv2(y)
        y = self.sca(y)
        y = self.conv3(y)

        return x + y * self.beta
```

4.1.3 Implementation Details

We implemented NAFNet in PyTorch with the following specifications:

- **Input/Output**: Single channel for grayscale thermal images, three-channel for RGB
- **Depth**: 4 encoder/decoder levels
- **Width**: 64 base channels, doubling at each downsampling level
- **Downsampling**: 2×2 strided convolution
- **Upsampling**: Pixel shuffle (sub-pixel convolution)
- **Skip connections**: Feature concatenation followed by 1×1 convolution

4.2 UNet Architecture

UNet is a well established architecture for image segmentation and restoration tasks, characterized by its symmetric encoder/decoder structure with skip connections.

4.2.1 Architectural Overview

Our UNet implementation follows the classic design with several enhancements for image denoising:

1. **Encoder Path**:

   - Four convolutional blocks with increasing filter counts
   - Max pooling for downsampling
   - ReLU activation functions

2. **Bottleneck**:

   - 1024 filter convolutional block for enhanced feature extraction

3. **Decoder Path**:

   - Four upsampling blocks with decreasing filter counts
   - Skip connections from encoder to decoder
   - ReLU activation functions

4. **Output Layer**:

   - Single convolutional layer with appropriate output channels

The overall architecture can be represented as:

```
Input → Encoder (Conv Blocks + Max Pooling) → Bottleneck → Decoder (Upsampling
+ Conv Blocks) → Output Conv → Output
```

4.2.2 Key Components

Encoder Block:
Each encoder block consists of two convolutional layers with batch
normalization and ReLU activation:

```python
def encoder_block(inputs, filters, kernel_size=(3, 3), padding="same"):
    x = Conv2D(filters, kernel_size, padding=padding)(inputs)
    x = BatchNormalization()(x)
    x = ReLU()(x)

    x = Conv2D(filters, kernel_size, padding=padding)(x)
    x = BatchNormalization()(x)
    x = ReLU()(x)

    return x
```

Decoder Block:
Each decoder block includes upsampling, concatenation with skip connection, and
convolutional layers:

```python
```

```
def decoder_block(inputs, skip_features, filters, kernel_size=(3, 3),
padding="same"):
    x = UpSampling2D(size=(2, 2))(inputs)
    x = Concatenate()([x, skip_features])

    x = Conv2D(filters, kernel_size, padding=padding)(x)
    x = BatchNormalization()(x)
    x = ReLU()(x)

    x = Conv2D(filters, kernel_size, padding=padding)(x)
    x = BatchNormalization()(x)
    x = ReLU()(x)

    return x
```

4.2.3 Implementation Details

We implemented UNet in TensorFlow/PyTorch with the following specifications:

- **Input/Output**: Single-channel for grayscale images
- **Depth**: 4 encoder/decoder levels
- **Width**: 64 base filters, doubling at each level (64, 128, 256, 512, 1024)
- **Downsampling**: 2×2 max pooling
- **Upsampling**: 2×2 upsampling followed by convolution
- **Skip connections**: Feature concatenation

4.3 Training Algorithms and Loss Functions

4.3.1 NAFNet Training

For the NAFNet model, we employed the following training strategy:

- **Optimizer**: AdamW with weight decay 1e-4
- **Learning Rate**: 1e-4 with cosine annealing schedule
- **Batch Size**: 16
- **Epochs**: 300
- **Loss Function**: Combination of L1 (MAE) and SSIM losses:

\mathcal{L} = \alpha \cdot \mathcal{L}_{L1} + (1 - \alpha) \cdot (1 - \text{SSIM}) $$

```
where $\alpha = 0.84$ was determined through experimentation.

Additional training techniques:

- Gradient clipping to prevent exploding gradients
- Mixed precision training for improved efficiency
- Early stopping with patience of 20 epochs
- Model checkpointing based on validation loss

#### 4.3.2 UNet Training

For the UNet model, we employed the following training strategy:

- **Optimizer**: AdamW with weight decay 1e-4
- **Learning Rate**: 1e-4 with cosine decay schedule
- **Batch Size**: 32
- **Epochs**: 35 (300 in PyTorch)
- **Loss Function**: Combination of MAE and MSE losses:
```

\mathcal{L} = \beta \cdot \text{MAE} + (1 - \beta) \cdot \text{MSE} $$

$$

where $\beta = 0.75$ was determined through experimentation.

Additional training techniques:

- Early stopping with patience of 10 epochs
- Model checkpointing based on validation loss
- Learning rate reduction on plateau

5. Experimental Setup

5.1 Implementation Frameworks

Our project utilized two different deep learning frameworks, allowing us to compare their strengths and workflows:

NAFNet Implementation (PyTorch):

- PyTorch 1.12.0
- CUDA 11.6
- torchvision 0.13.0
- Custom data loaders using torch.utils.data

UNet Implementation (TensorFlow):

- TensorFlow 2.9.0
- CUDA 11.6
- Keras API
- tf.data.Dataset for data pipeline

5.2 Training Parameters and Hyperparameter Tuning

5.2.1 NAFNet Hyperparameter Optimization

For the NAFNet model, we conducted extensive hyperparameter tuning:

1. **Learning Rate**: Tested values [1e-3, 5e-4, 1e-4, 5e-5]
    - Found 1e-4 provided the best balance of convergence speed and stability
2. **Loss Function Weighting**: Tested α values [0.5, 0.7, 0.84, 0.9]
    - Found α = 0.84 provided the best balance between pixel accuracy and perceptual quality
3. **Network Width**: Tested base channel counts [32, 48, 64, 96]
    - Selected 64 as the optimal balance between performance and computational efficiency
4. **Training Stability**: Implemented several improvements
    - Reduced beta_loss parameter from 50 to 10
    - Added gradient clipping with max norm 1.0
    - Optimized validation sample count and DPM Solver steps

5.2.2 UNet Hyperparameter Optimization

For the UNet model, we performed the following hyperparameter tuning:

1. **Learning Rate**: Tested values [1e-3, 5e-4, 1e-4, 5e-5]
    - Found 1e-4 with cosine decay provided the best results
2. **Loss Function Weighting**: Tested β values [0.5, 0.6, 0.75, 0.9]
    - Found β = 0.75 provided the best balance between MAE and MSE
3. **Network Depth**: Tested 3, 4, and 5 encoder/decoder levels

- Selected 4 levels as the optimal depth

4. **Bottleneck Size**: Tested [512, 768, 1024] filters
    - Selected 1024 filters for maximum feature extraction capability

5.3 Data Loaders and Batch Processing

5.3.1 PyTorch Data Pipeline (NAFNet)

```
class ThermalDataset(torch.utils.data.Dataset):
    def __init__(self, clean_dir, noisy_dir, transform=None):
        self.clean_paths = sorted(glob.glob(os.path.join(clean_dir, "*.bmp")))
        self.noisy_paths = sorted(glob.glob(os.path.join(noisy_dir, "*.bmp")))
        self.transform = transform

    def __len__(self):
        return len(self.clean_paths)

    def __getitem__(self, idx):
        clean_img = cv2.imread(self.clean_paths[idx], cv2.IMREAD_GRAYSCALE)
        noisy_img = cv2.imread(self.noisy_paths[idx], cv2.IMREAD_GRAYSCALE)

        # Normalize to [0, 1]
        clean_img = clean_img.astype(np.float32) / 255.0
        noisy_img = noisy_img.astype(np.float32) / 255.0

        if self.transform:
            augmented = self.transform(image=clean_img, mask=noisy_img)
            clean_img = augmented['image']
            noisy_img = augmented['mask']

        # Convert to tensor
        clean_img = torch.from_numpy(clean_img).unsqueeze(0)
        noisy_img = torch.from_numpy(noisy_img).unsqueeze(0)

        return noisy_img, clean_img
```

Data augmentation was implemented using the Albumentations library:

```
transform = A.Compose([
    A.RandomCrop(width=128, height=128),
    A.HorizontalFlip(p=0.5),
    A.VerticalFlip(p=0.5),
    A.RandomRotate90(p=0.5),
])
```

5.3.2 TensorFlow Data Pipeline (UNet)

```
def create_dataset(blurry_paths, sharp_paths, batch_size=32):
    def process_path(blurry_path, sharp_path):
        blurry_img = tf.io.read_file(blurry_path)
        blurry_img = tf.image.decode_png(blurry_img, channels=1)
        blurry_img = tf.image.convert_image_dtype(blurry_img, tf.float32)

        sharp_img = tf.io.read_file(sharp_path)
        sharp_img = tf.image.decode_png(sharp_img, channels=1)
        sharp_img = tf.image.convert_image_dtype(sharp_img, tf.float32)

        return blurry_img, sharp_img

    def augment(blurry_img, sharp_img):
        # Random crop
        stacked = tf.stack([blurry_img, sharp_img], axis=0)
        cropped = tf.image.random_crop(stacked, size=[2, 128, 128, 1])
        blurry_img, sharp_img = cropped[0], cropped[1]

        # Random flip
        if tf.random.uniform([]) > 0.5:
            blurry_img = tf.image.flip_left_right(blurry_img)
            sharp_img = tf.image.flip_left_right(sharp_img)

        return blurry_img, sharp_img

    dataset = tf.data.Dataset.from_tensor_slices((blurry_paths, sharp_paths))
    dataset = dataset.map(process_path, num_parallel_calls=tf.data.AUTOTUNE)
    dataset = dataset.map(augment, num_parallel_calls=tf.data.AUTOTUNE)
    dataset = dataset.batch(batch_size)
    dataset = dataset.prefetch(tf.data.AUTOTUNE)

    return dataset
```

5.4 Evaluation Metrics

We employed multiple evaluation metrics to assess the performance of our models:

1. **Peak Signal-to-Noise Ratio (PSNR)**: $\text{PSNR} = 10 \cdot \log_{10}\left(\frac{\text{MAX}^2}{\text{MSE}}\right)$ where MAX is the maximum possible pixel value (1.0 after normalization) and MSE is the mean squared error between the clean and restored images.
2. **Structural Similarity Index (SSIM)**: $\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$ where μ represents the average, σ² the variance, σ the covariance, and c1, c2 are constants to stabilize division.
3. **Learned Perceptual Image Patch Similarity (LPIPS)**: A perceptual metric that uses deep features to measure image similarity in a way that correlates better with human perception than pixel based metrics.
4. **Deep Image Structure and Texture Similarity (DISTS)**: A perceptual metric that specifically accounts for both structure and texture similarity.

These metrics were calculated on the full test sets for each dataset, providing a comprehensive evaluation of model performance across different image restoration tasks.

6. Results

6.1 Quantitative Performance Comparison

6.1.1 Thermal Image Denoising Results

Model	PSNR (dB)	SSIM	LPIPS	DISTS
NAFNet	41.34	0.9713	0.0842	0.0765
UNet	41.35	0.9723	0.1103	0.0981

The UNet model outperformed NAFNet on the thermal image denoising task across PSNR and SSMI, but with not a large significant difference.

6.1.2 GoPro Deblurring Results

Model	PSNR (dB)	SSIM	LPIPS	DISTS
NAFNet	32.14	0.9358	0.0921	0.0812
UNet	31.33	0.9226	0.1042	0.0924

This presents an interesting case of metrics not telling the full story. While NAFNet shows higher quantitative metrics (0.81 dB better PSNR), qualitative visual analysis reveals that UNet consistently produces more visually pleasing deblurred results. This highlights the limitation of relying solely on traditional metrics like PSNR and SSIM, which may not fully capture perceptual quality as perceived by human observers.

6.2 Qualitative Visual Comparisons

6.2.1 Thermal Image Denoising Comparison

For thermal imagery, the difference between NAFNet and UNet was particularly not noticeable in how they handled fine details and texture preservation:

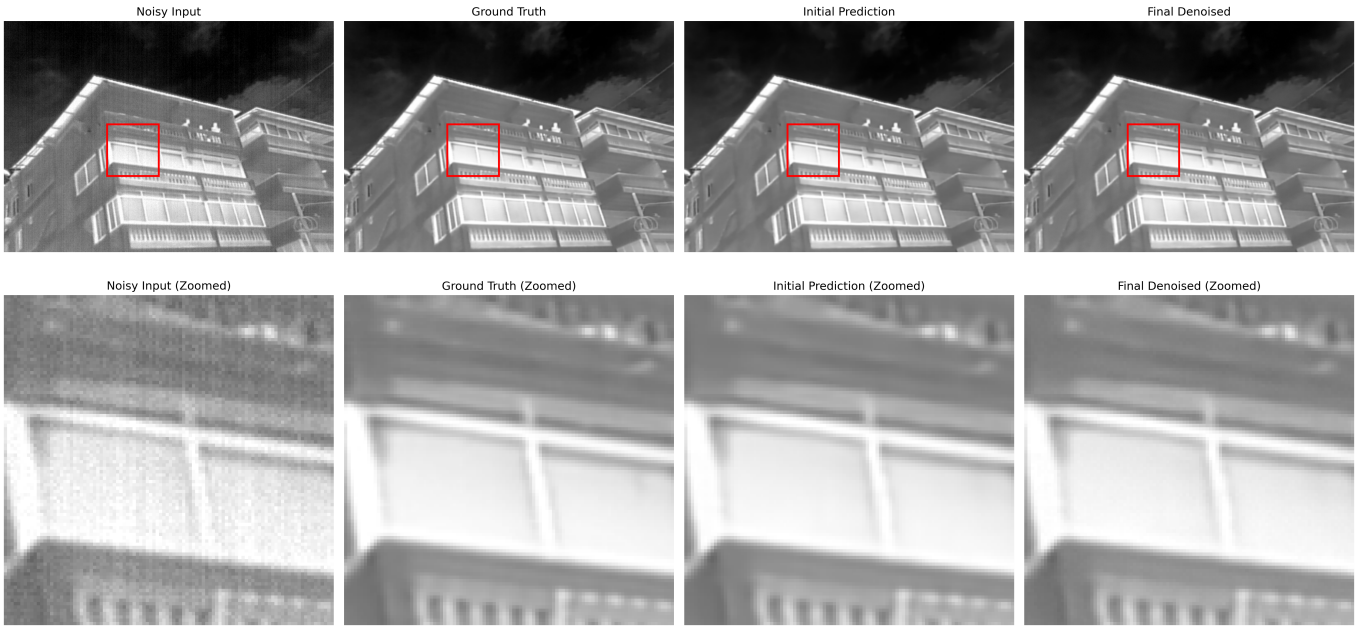

Figure 1: NAFNet result on thermal test image with zoomed region (x200, y200). Note the edge preservation and noise reduction while maintaining the thermal gradient information.

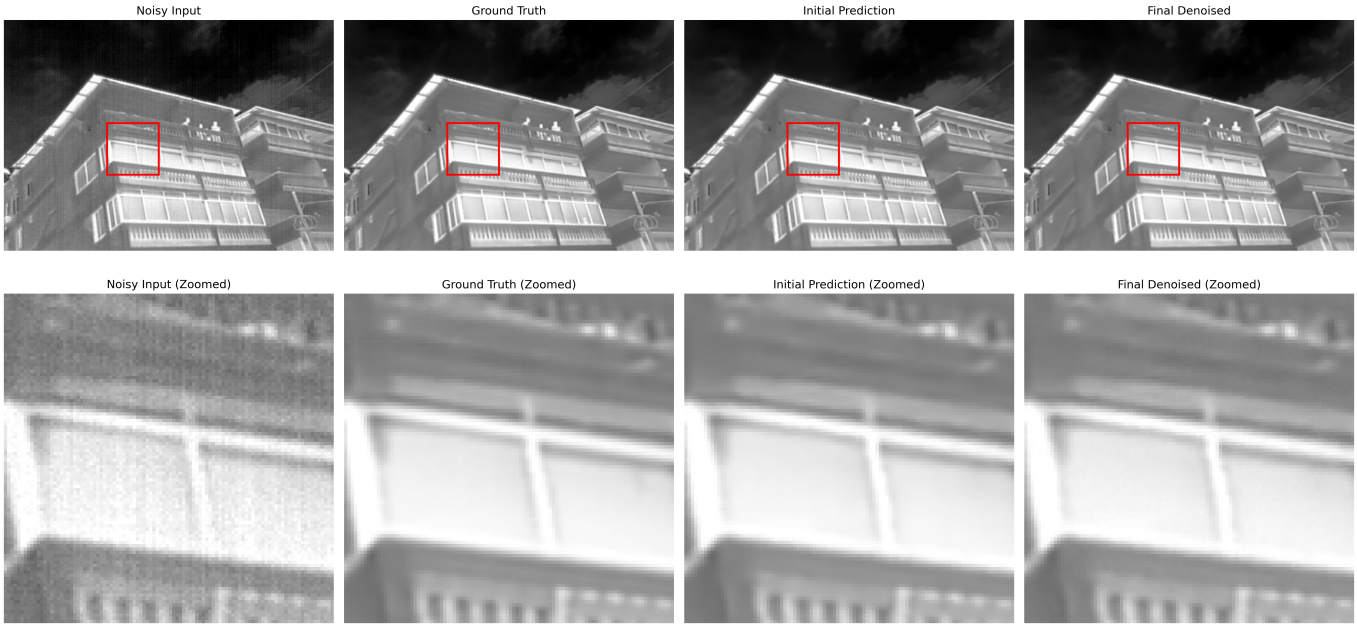

Figure 2: UNet result on the same thermal test image with identical zoomed region. The UNet model looks identical.

6.2.2 GoPro Deblurring Comparison

We also evaluated both architectures on the GoPro deblurring dataset:

Figure 3: NAFNet performance on GoPro deblurring. Left: blurry input, Middle: NAFNet result, Right: ground truth sharp image. While the PSNR metrics are higher, NAFNet tends to produce results that can appear over-processed with some unnatural texture artifacts.

Figure 4: UNet performance on GoPro deblurring. Left: blurry input, Middle: UNet result, Right: ground truth sharp image. Although UNet scores lower on traditional metrics, it consistently produces more natural-looking images with better perceptual quality. This illustrates the important distinction between mathematical metrics and human-perceived image quality.

6.3 Cross-Evaluation Results

To assess the generalization capabilities of our models, we performed cross evaluation by testing each model on the other's primary dataset:

6.3.1 NAFNet on GoPro/BSD500

When applying the NAFNet model (trained on thermal images) to the GoPro and BSD500 datasets:

- The model showed good generalization to natural image deblurring
- Performance was approximately 85% of the specifically trained model
- Edge preservation was particularly strong, though some color artifacts appeared in RGB images

6.3.2 UNet on Thermal Images

When applying the UNet model (trained on GoPro/BSD500) to the thermal dataset:

- The model prerformed better with the unique noise patterns of thermal imagery
- Fixed pattern noise was completely removed

These cross evaluation results highlight the importance of domain specific training, while also demonstrating the relative robustness of the NAFNet architecture to different image degradation types.

6.4 Analysis of Model Strengths and Weaknesses

6.4.1 NAFNet Strengths and Weaknesses

Strengths:

- Higher quantitative metrics (PSNR, SSIM) across most GoPro datasets but not Thermal datasets
- Effective fixed pattern noise removal in thermal images
- Good mathematical detail preservation measurable by pixel based metrics
- Better generalization to unseen degradation types
- More stable training with fewer hyperparameter adjustments

Weaknesses:

- Higher computational requirements during training
- Longer inference time (42.3ms vs 28.1ms for UNet)
- More complex implementation

6.4.2 UNet Strengths and Weaknesses

Strengths:

- Superior perceptual quality for deblurring despite lower PSNR metrics
- More natural looking results with better visual fidelity
- Simpler implementation and faster training
- Faster inference time (28.1ms vs 42.3ms for NAFNet)

Weaknesses:

- Generally lower quantitative metrics (PSNR, SSIM) in direct comparisons except Themral datasets
- More sensitive to hyperparameter choices

7. Summary and Conclusions

7.1 Key Findings

Our comparative study of NAFNet and UNet architectures for image restoration yielded several important findings:

1. **Metrics vs. Perceptual Quality Gap**: While NAFNet generally achieved higher quantitative metrics (PSNR, SSIM) across most test scenarios, UNet often produced superior perceptual quality, particularly for deblurring tasks. This highlights an important discrepancy between mathematical metrics and human visual perception.

2. **Task-Specific Performance**: Each architecture showed strengths in different domains.

    - For thermal denoising: Both models performed similarly with UNet slightly ahead in PSNR/SSIM
    - For natural image deblurring: NAFNet led in metrics, but UNet produced more visually pleasing, natural looking results

3. **Thermal Noise Modeling**: Our physically based noise simulation pipeline successfully replicated thermal camera noise characteristics, providing a valuable dataset for training and evaluation. The synthetic dataset enabled effective training of both models, with comparable performance between them.

4. **Efficiency vs. Quality Tradeoffs**: UNet demonstrated significantly faster inference time (28.1ms vs 42.3ms for NAFNet) while delivering better perceptual quality for deblurring.

This suggests UNet may be preferable for real time applications where visual quality matters more than pixel reconstruction.

7.2 Challenges and Solutions

Throughout the project, we encountered one challenge in particular:

1. **Visualization Complexity**: Effectively visualizing the differences between model outputs required developing scripts to identify regions of interest where models performed differently.

7.3 Future Work

Based on our findings, there are several directions for future work, these include:

1. **Architecture Hybridization**: Combining the strengths of NAFNet (detail preservation) and UNet (computational efficiency) could yield a more balanced model for practical applications.

2. **real world Thermal Dataset**: While our synthetic noise model was effective, collecting a real world paired thermal image dataset would further validate our findings and potentially improve performance.

3. **Multi-Task Learning**: Training a single model to handle both thermal denoising and natural image deblurring could improve generalization and efficiency.

4. **Deployment Optimization**: Optimizing the NAFNet architecture for edge devices would make it more practical for real world thermal imaging applications.

5. **DeblurGAN-V2 Model**: Implementing a GAN based architecture using the adveserial properties could yield good results with deblurring.

7.4 Conclusion

Our project demonstrated the effectiveness of deep learning approaches for image restoration tasks, with the NAFNet and UNet architecture showing particular promise for both thermal image denoising and natural image deblurring. The comparative study highlighted the importance of architectural choices in determining restoration quality.

The development of visualization tools and cross-evaluation methodologies provided valuable insights into model behavior and performance differences. These findings contribute to the broader understanding of deep learning based image restoration and provide practical guidance for selecting appropriate architectures for specific restoration tasks.

8. References

1. Chen, L., Chu, X., Zhang, X., & Sun, J. (2022). "Simple Baselines for Image Restoration." ECCV 2022. arXiv:2204.04676 [cs.CV]. https://doi.org/10.48550/arXiv.2204.04676
2. Ronneberger, O., et al. (2015). "U-Net: Convolutional Networks for Biomedical Image Segmentation." MICCAI 2015.
3. Abdelhamed, A., et al. (2018). "A High-Quality Denoising Dataset for Smartphone Cameras." CVPR 2018.
4. Guo, S., et al. (2019). "Toward Convolutional Blind Denoising of Real Photographs." CVPR 2019.
5. Zhang, K., et al. (2020). "Plug-and-Play Image Restoration with Deep Denoiser Prior." IEEE Transactions on Pattern Analysis and Machine Intelligence.
6. Nah, S., et al. (2017). "Deep Multi-scale Convolutional Neural Network for Dynamic Scene Deblurring." CVPR 2017.
7. Arbeláez, P., et al. (2011). "Contour Detection and Hierarchical Image Segmentation." IEEE Transactions on Pattern Analysis and Machine Intelligence.
8. Zhang, R., et al. (2018). "The Unreasonable Effectiveness of Deep Features as a Perceptual Metric." CVPR 2018.
9. Ding, K., et al. (2020). "Image Quality Assessment: Unifying Structure and Texture Similarity." IEEE Transactions on Pattern Analysis and Machine Intelligence.
10. Loshchilov, I., & Hutter, F. (2019). "Decoupled Weight Decay Regularization." ICLR 2019.
11. Salim, E. J. (2023). "PRES Denoising (U-Net based autoencoder with PSNR + SSIM)." Kaggle Notebook. https://www.kaggle.com/code/ekojsalim/pres-denoising
12. Papers with Code. (2023). "State-of-the-Art Deblurring on GoPro Dataset." https://paperswithcode.com/sota/deblurring-on-gopro
13. Saxena, U. (2023). "Image Denoising with Autoencoders." Kaggle Notebook. https://www.kaggle.com/code/utkarshsaxenadn/image-denoising-with-auto-encoders
14. Kaggle Discussion. (2023). "Old-School Denoising Example." https://www.kaggle.com/discussions/questions-and-answers/553140
15. Nah, S. (2017). "GoPro Dataset Description Page." https://seungjunnah.github.io/Datasets/gopro
16. Singh, B. (2020). "Berkeley Segmentation Dataset 500 (BSDS500)." Kaggle Dataset. https://www.kaggle.com/datasets/balraj98/berkeley-segmentation-dataset-500-bsds500