Tensorflow Implementation of Adaptive Sparse Transformer with Attentive Feature Refinement for Image Restoration

Group Members:

Emma Chen: echen154
Katerina Nguyen: knguy155
Albert Dong: addong
Leanne Chia: lchia1

Introduction

This project aims to tackle the problem of image restoration, specifically focusing on removing real-world degradations in images such as raindrops and haze. This task is especially relevant because degraded images significantly impact the performance of numerous computer vision applications like object detection and autonomous driving.

Specifically, our project aims to reimplement the “Adapt of Perish: Adaptive Sparse Transformer with Attentive Feature Refinement for Image Restoration” paper (CVPR 2024) by Zhou et al., which proposes a Transformer-based architecture designed to effectively suppress irrelevant features in both spatial and channel domains. This is done through two core designs: an Adaptive Sparse Self-Attention (ASSA) block, and a Feature Refinement Feed-forward Network (FRFN).

We chose this paper because it presents an efficient mechanism that provides a solution to the shortcomings of dense attention mechanisms in image restoration, and demonstrates robust performance across multiple benchmark datasets. Furthermore, our project will reimplement this model in TensorFlow (as opposed to the original PyTorch version), and this provides a valuable opportunity to better understand attention design and architectural choices, while building a flexible version of the model.

This is a structured prediction problem, where the output is a pixel-level reconstruction of a clean image, and the quality of results is evaluated using perceptual and similarity-based metrics rather than classification accuracy.

Methodology

Data

For this project, we used the DID-MDN Rain Medium dataset (https://github.com/hezhangsprinter/DID-MDN), which provides a benchmark for rain removal tasks. Specifically, we worked with the Rain_Medium subset, containing 4,000 paired clean and rainy images. The dataset simulates diverse real-world rain degradations, making it suitable for evaluating image restoration models.

We performed minimal preprocessing beyond resizing and normalization. To improve generalization, we applied standard data augmentation techniques during training, including random flipping, following practices outlined in the original Adaptive Sparse Transformer (AST) paper.

Model Architecture

Our model follows a Transformer-based architecture specifically designed to suppress irrelevant features across both spatial and channel dimensions. It consists of two main components:

- Adaptive Sparse Self-Attention (ASSA) block
  - A hybrid attention block combining two parallel branches:
    - Sparse Self-Attention (SSA): Uses a ReLU^2 activation to selectively focus on informative features
    - Dense Self-Attention (DSA): Applies standard softmax attention across feature patches
  - The outputs of these two branches are fused through adaptive, learnable weights, allowing the model to dynamically balance sparse and dense attention at each layer.
- Feature Refinement Feed-forward Network (FRFN)
  - A specialized feed-forward block that refines features via an enhance-and-ease approach:
    - Partial Convolution enhances informative channels
    - Depthwise Convolution combined with a gating mechanism suppresses redundant channels

The overall network adopts a symmetric encoder-decoder structure with a bottleneck transformer stage, enabling it to capture both local and long-range dependencies. We structured attention computation around non-overlapping 8×8 patches to balance efficiency and accuracy.

Training the Model

The details of our model training process are as follows:

- Loss Function: Charbonnier loss, a smooth approximation of the L1 loss, which is less sensitive to outliers.
- Optimizer: Adam optimizer, selected for its adaptive learning rate capabilities and widespread effectiveness in deep learning models.
- Learning Rate: Starts at 0.0002 and decays to 0.000001 following a cosine decay schedule
- Batch Size: 2
- Window Size: Non-overlapping patches of 8x8 for attention

Results

We evaluated our Adaptive Sparse Transformer (AST) model both quantitatively and qualitatively. Training and testing metrics were recorded over 10 epochs.

Training Specifications:

- Trained for 10 epochs, with 1800 training steps and 200 validation steps per epoch
- Each epoch took approximately 15 minutes, totaling around 3 hours of training time

Quantitative Results

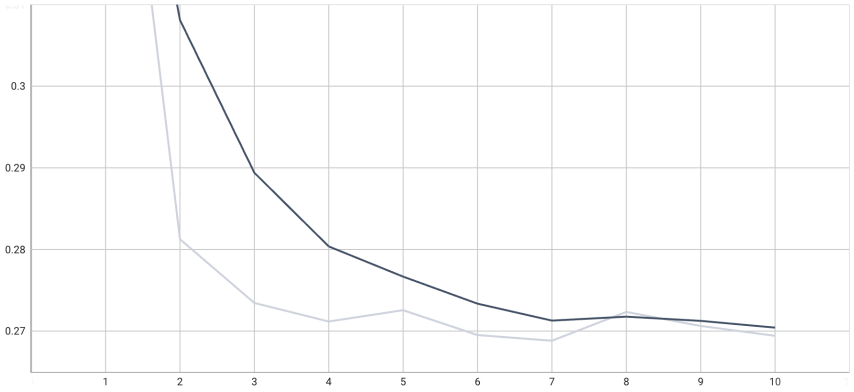

Figure 1: Training and validation loss curves over 10 epochs.

From Figure 1, we can see that the loss decreases rapidly during the initial epochs and gradually plateaus, indicating that the model quickly learns major patterns and then gradually fine-tunes its predictions, therefore showing successful convergence of the model during training. The small

gap between training and validation losses suggests good generalization without significant overfitting.

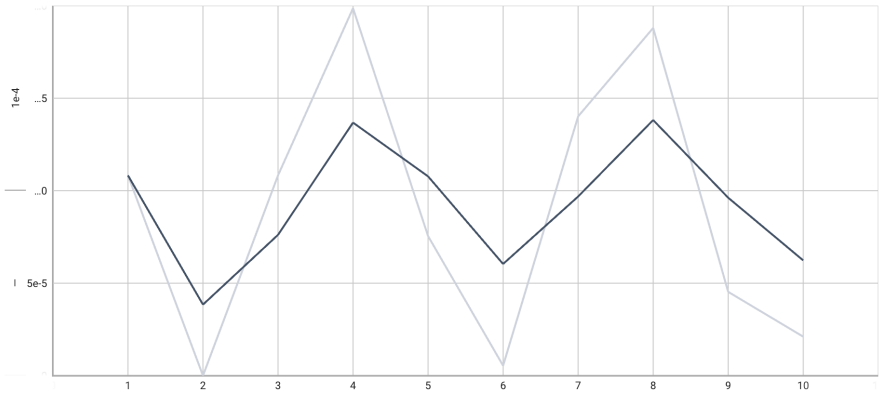

Figure 2: Learning rate across 10 epochs
(where the x-axis shows the number of epochs)

From Figure 2, we can see that rather than remaining constant, the learning rate is dynamically adjusted to oscillate downwards following a cosine-shaped decay pattern. This dynamic learning rate schedule helps to stabilize training, encourage better convergence, and improve final model generalization.

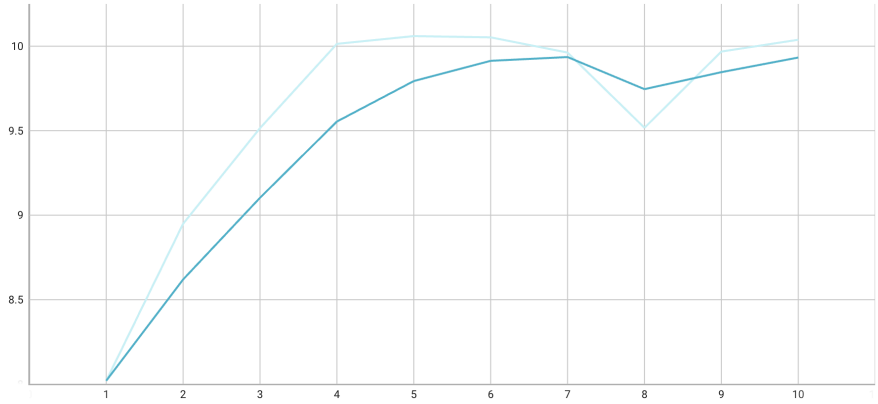

Figure 3: Peak Signal-to-Noise Ratio (PSNR) over 10 epochs
(where the x-axis shows the number of epochs)

PSNR is a critical metric for evaluating image restoration tasks: higher PSNR values indicate better image reconstruction, with lower levels of distortion compared to the ground truth. From Figure 3, we can see that the PSNR steadily increases during training, indicating that the model progressively reduces image distortion and restores cleaner images. Minor fluctuations near the end reflect typical fine-tuning dynamics as the model approaches convergence.

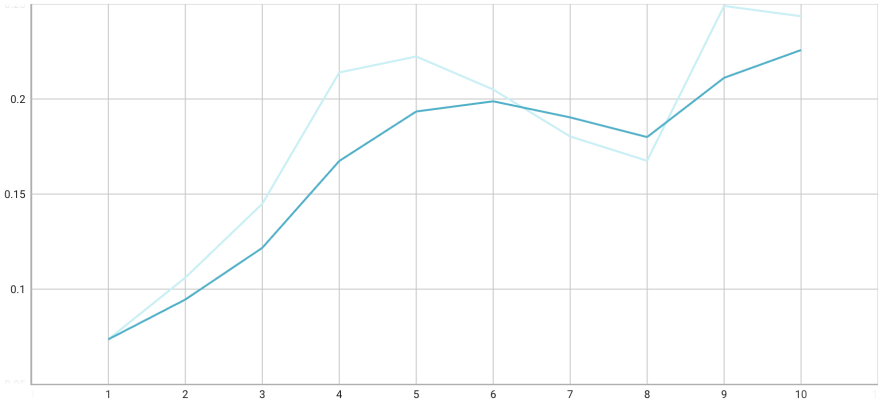

Figure 4: Structural Similarity Index Measure (SSIM) across 10 epochs
(where the x-axis shows the number of epochs)

SSIM measures the perceptual similarity between the predicted (restored) images and the clean ground truth images. Specifically, in contrast to PSNR, SSIM captures luminance, contrast, and structural consistency, making it more aligned with human visual perception. From Figure 4, we can see that the SSIM values increase over training, indicating that the model progressively improves in preserving structural details and perceptual quality between the predicted and ground truth images.

Final Quantitative Results:

- **Training result:**
  - Loss: 0.2694
  - PSNR: 10.20
  - SSIM: 0.2435
- **Testing Result**:
  - Paper PSNR: 17.27
  - Our PSNR: 10.14
  - Paper SSIM: 0.57
  - Our SSIM: 0.26

Qualitative Results

Figure 5: Original clear testing image

Figure 6: Original rainy testing image

Figure 7: Model testing image result
(Restored image from model output)

Figures 5, 6, and 7 show a visual comparison of the model's deraining performance on a sample image from the test set.

Figure 5 is the ground truth clean image, free from any rain degradation. It serves as the ideal target output for the model during training and evaluation.

Figure 6 presents a noticeable amount of synthetic rain streaks, particularly visible across uniform surfaces like the garage door, walls, and sky region. Rain significantly reduces the clarity and perceptual quality of the image, making it an appropriate challenge for testing deraining models.

Figure 7 is the output generated by the model. It shows partial removal of rain artifacts, but there are severe color distortions and loss of fine details. The restored image appears over-saturated and posterized (showing unnatural sharp transitions between color regions), indicating that the model struggles to preserve original color distributions, and has difficulty reconstructing texture details effectively.

Therefore, while the model learned some notion of rain removal (there are fewer rain streaks visible compared to the input), the significant degradation in color and structure highlights that there still exists improvement to be made to the model's learning.

Challenges/Discussion

Visual Explanation of Model Behavior

One notable visual issue in our outputs was that models trained with Charbonnier loss tend to predict the pixel-wise mean of plausible outputs. As a result, high-frequency rain streaks were often "washed out" into flat, over-bright patches. Additionally, because network outputs are clipped at 1.0, any predicted values above 1.0 saturated into stark white regions, further exaggerating these artifacts and leading to unnatural restored images.

Challenges

Tensorflow Reimplementation

Adapting the original PyTorch-based AST model to TensorFlow/Keras proved to be one of the most difficult parts of the project. Implementing custom attention mechanisms like Adaptive Sparse Self-Attention (ASSA), which fuses sparse ($ReLU^2$) and dense (softmax) branches, involved significant challenges, particularly related to maintaining numerical stability and avoiding NaNs during training.

System Integration

Ensuring compatibility across all custom components (ASSA, FRFN, the encoder-decoder wrapper, and data loading pipeline) required careful debugging. Minor tensor shape mismatches, dtype inconsistencies, and window partitioning errors were recurrent early problems.

Training Infrastructure Limitations

Training deep models even on relatively small patches can take hours to days without high-end GPUs. Specifically, training deep models like AST on our team's MacBook computers required reliance on TensorFlow-Metal for GPU acceleration, which, while helpful in terms of runtime, fell short compared to the authors' CUDA-powered training speeds.

We had to significantly lower batch sizes and crop sizes to fit memory constraints, leading to slower convergence and limited model expressiveness.

Furthermore, even with GPU acceleration, training and evaluation were much slower than expected. Therefore, full training on larger datasets like RESIDE was infeasible within available time and resources, forcing a reduction to a 4,000-image subset of DID-MDN. We were limited in how much hyperparameter tuning, model experimentation, and retraining we could realistically perform.

These limitations highlighted the importance of setting realistic milestones based on available computing resources, and planning for overnight or multi-day experiments.

Data Preprocessing

Early on, a bug in the augmentation pipeline caused input and ground truth patches to misalign during random cropping. This led to extremely poor accuracy despite correct model logic, wasting several training runs before discovery.

Overfitting and Monitoring

We continued training even after validation metrics plateaued, inadvertently allowing minor overfitting and wasting compute cycles. Future training runs should include more robust early stopping mechanisms or close monitoring of validation loss/metrics to prevent unnecessary overfitting and

Limitations of Our Implementation

Hardware Constraints

Training was significantly slower than expected even with TensorFlow-Metal, restricting hyperparameter searches and architectural experiments.

Reduced Dataset Size

We intended for the original model to be trained on a dataset of ~12,000 images, but we had to downscale to 4,000 images to make training feasible on local hardware.

Memory and compute limitations also restricted batch sizes and total training epochs. Deep models like AST have many parameters and need large amounts of data to reach full potential. With less variation in input images, the model struggled to generalize across different rain thicknesses, backgrounds, and lighting conditions.

Model Size vs. Dataset Size Mismatch

The AST model has a large number of parameters and is designed for large datasets. Training it on a smaller dataset without adjusting the model size likely contributed to instability and artifacts in outputs.

Future Work

Training on the Full Dataset

Access to higher-end GPUs would allow training on the full DID-MDN and RESIDE datasets, enabling better generalization across various degradations.

Data Augmentation Improvements

Incorporating augmentations like color jittering, brightness shifts, and contrast adjustments could help the model better handle variations in input images.

Hyperparameter Tuning

Conducting systematic searches over learning rates, batch sizes, window sizes, and optimizer choices could lead to significantly better results and improved accuracy.

Post-Processing Enhancements

Adding simple post-processing steps (e.g., guided filtering, contrast enhancement) after de-raining could mitigate the color distortions and posterization seen in model outputs and boost perceptual quality without retraining the model.

Smaller Model Variants

Exploring lighter versions of AST or dynamic window sizes could make training more tractable on modest hardware without significantly sacrificing performance.

Reflection

Overall Outcome

Overall, we are proud of the technical progress we made in adapting and training a complex Transformer-based architecture like AST from scratch in TensorFlow. Despite the many challenges we encountered, including limited computational resources and the difficulty of reimplementing specialized attention mechanisms, we successfully built a complete working model capable of learning to reduce rain artifacts and restore clearer images.

Although our model did not meet our goals in terms of PSNR/SSIM matching the authors' original paper, we observed consistent improvements in both PSNR and SSIM over the course of our training. This steady increase in quantitative metrics, combined with qualitative improvements in the output images, shows that the model was effectively learning and refining its predictions even within a constrained setting.

Given the significant limitations on dataset size, batch size, and training time, we view the convergence of the model, the upward trends in performance metrics, and the successful restoration of key image structures as a meaningful technical achievement. Ultimately, this project demonstrated our ability to reimplement and train a challenging deep learning model under real-world constraints, deepening our understanding of Transformer architectures, attention mechanisms, and the practical demands of developing end-to-end deep learning workflows.

Model Performance vs. Expectations

Our model behavior was broadly aligned with expectations: it was able to remove some rain artifacts, but qualitative outputs and quantitative metrics showed that full high-fidelity reconstruction was not achieved.

Color distortions and limited structural recovery were larger than anticipated, highlighting the need for longer training, larger datasets, and more capacity tuning.

Pivots and Changes

We initially planned to work with both DID-MDN and RESIDE datasets, but pivoted to focusing solely on a subset of DID-MDN due to computational constraints. We also planned to perform more aggressive model tuning, but reduced the scope after encountering slow training speeds and GPU memory issues on our computers.

If we were to redo the project, we would:

- Spend more time early on setting up smaller baseline experiments (e.g., on smaller comprehensive datasets) to validate model structure.
- Implement earlier validation checks to avoid silent misalignment bugs.
- Try to secure access to higher-end hardware sooner to avoid bottlenecks in terms of runtime and memory.

Areas for Further Improvement

If given more time, we would:

- Retrain on larger datasets with better augmentation.
- Optimize the model architecture for small-sample settings.
- Conduct broader hyperparameter searches to find better learning schedules.
- Experiment with mixed-loss training, such as combining Charbonnier loss with perceptual loss.

Biggest Takeaways

Custom model reimplementation is challenging but achievable with enough patience and careful debugging. Throughout this project, we realized that hardware resources are a critical bottleneck. Specifically, scaling deep learning realistically requires thoughtful GPU planning and access to powerful computing resources.

We also learned that data alignment and preprocessing are just as important as model architecture; even minor bugs in data handling can severely impact model performance. Additionally, early validation monitoring is crucial for saving time and resources, as it helps catch potential issues before they escalate.

Despite the many difficulties we encountered, this project significantly deepened our understanding of Transformer architectures, attention mechanisms, TensorFlow engineering, and the practical realities of developing deep learning workflows from scratch.