# INTEL UNNATI INDUSTRIAL TRAINING 2025

# Image Sharpening Using Knowledge Distillation

**The Project Report**

**Submitted by:**

GONDI JAHNAVI BU22CSEN0101459 jgondi@gitam.in

BHUMA CHAITANYA DEEPIKA BU22CSEN0100501 cbhuma@giatm.in

RAVULA CHERUVU SAI SRUTHI BU22CSEN0300257 sravalac@gitam.in

**Under the esteemed guidance of**

**MEENA K**

**Professor**

**Department of Computer Science and Engineering**

**GITAM SCHOOL OF TECHNOLOGY**

**GANDHI INSTITUTE OF TECHNOLOGY AND MANAGEMENT**

**(Deemed to be University)**

**Bengaluru Campus.**

**TABLE OF CONTENTS**

**TOPICS PG-NO**

Title i

Table of contents ii

Abstract iii

1. Introduction 1
2. Methodology
   1. Data Collection 1
   2. Subset-Based Training Strategy 3
   3. Preprocessing Pipeline 4
   4. Teacher Model-Restormer 6
   5. Student Model-NAFNet Tiny 8
   6. Deployment 10
3. Experimental Analysis   
   3.1 Benchmarking and Validation Split of the Dataset 11  
   3.2 Measures of Evaluation 12  
   3.3 Student Performance 12   
   3.4 Evaluation of Visual Quality 14  
   3.5 Performance of Deployment in Real-Time Situations 15   
   3.6 Inference Speed and Model Efficiency 16
4. Conclusion 17
5. References 17

## **Abstract**

Sharpening of an image is a critical step in computer vision to enhance image sharpness through the enhancement of edges and fine details. Traditional sharpening methods don't succeed in balancing detail enhancement and noise suppression, especially in real-time or low-resource situations. This project explores a quick deep learning-based image sharpening technique under Knowledge Distillation (KD), where a lightweight, efficient student model is trained to mimic the function of a more capable, larger teacher model.

Here, we consider a high-capacity model like Restormer as the teacher, which has been seen to work best in image restoration applications, and a low-capacity model like NAFNet or TinyNAFNet as the student. The pre-training of the teacher model is done on high-resolution image sharpening tasks. Then, the student model is learned on ground truth supervision and soft targets provided by teachers so that it can infer the explicit and implicit features learned from the teacher network.

The student model that is distilled has a performance-competitive trade-off in terms of computational efficiency and can be deployed on edge devices as well as used for real-time applications like video conferencing or mobile imaging. Experimental results indicate that the student model preserves most of the teacher's sharpening quality, with the inference time being reduced by several orders of magnitude and memory consumption to a fraction of that of the teacher model.

This paper highlights the effectiveness of knowledge distillation in knowledge transfer from big models to small networks for high-quality and efficient image enhancement.

## **1. Introduction:**

In the field of real time visual communication as in video conferencing the quality of the interaction is what we see in the clarity and sharpness of the transmitted images. We see that blurred or out of focus visuals which at times is a result of motion, compression or lens issues present a great issue. While we have traditional image sharpening techniques like the use of Laplacian filters or unsharp masking we find that they tend to over enhance edges which in turn produces artifacts thus their practical value is limited.

A large complex **Restormer model** (teacher) which is a model that has a great deal of information and detail in it passes that knowledge on to a more simple and efficient**NAFNet-Tiny model** (student). What we have in the end is a very high quality output which also performs in real time.

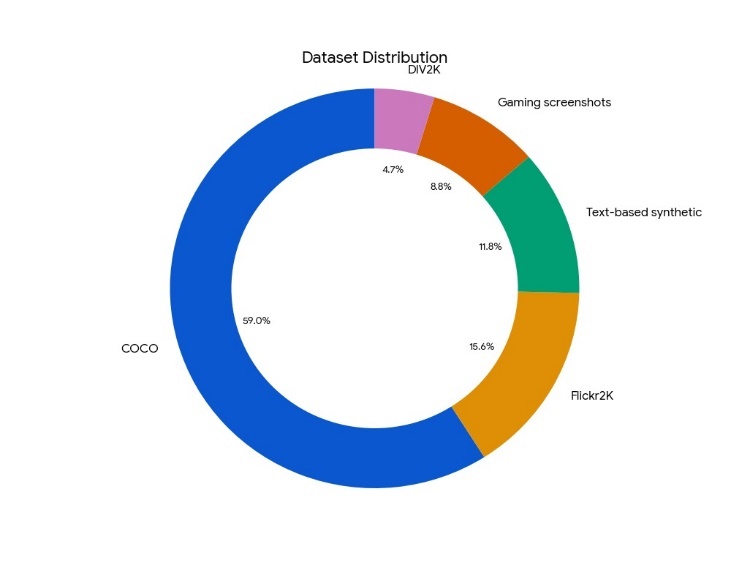
**2. Methodology**

## **2.1Dataset Collection:**

To ensure diversity and generalization, an extensive image dataset of around **17,000 high-resolution** images was extracted from a wide spectrum of domains. The sources and properties were as follows:

* **DIV2K**: 800 images — known for their ultra-high resolution and rich details
* **Flickr2K**: 2,650 images — varied scenes, typically 1080p or higher
* **COCO**: ~10,000 images — diverse real-world environments, all ≥640×480
* **Text-based synthetic datasets**: ~2,000 images — valuable for fine edge patterns and artifacts
* **Gaming screenshots**: ~1,500 images — dynamic lighting and texture-rich frames

All images were collected in their native resolutions and maintained in .png format throughout the pipeline. No resizing, compression conversion, or format changes were performed during collection to preserve raw fidelity.



*Pie chart depicting the proportions of different dataset collected to make custom dataset*

### **2.1.1 Image Tracking and Naming Strategy**

To avoid any overlap or reuse across phases and benchmarks, we kept a special CSV metadata file. This file tracked how we used images in all subsets and benchmarks making sure we used them once in training and evaluation.

Also, we gave new names to all images before we processed them. We used prefixes based on where they came from and added numbers in order. This helped us trace them. Here are some examples of how we named them:

• Whole image: div2k\_0001.png coco\_0078.png flickr\_0123.png

• Parts of images: coco\_0078\_patch\_0000.png flickr\_0123\_patch\_0024.png

This organized way of doing things made sure our data process stayed the same throughout. It also let us identify and repeat each sample and where it came from.

### **2.1.2 Benchmark Set**

To make fair testing, a test group was made by keeping **10% (about 1,690 pictures)** from the full data set.

• This test group was split in half:

* **5% (around 845 pictures)** to check in the middle of training
* **5% (around 845 pictures)** for the last quality and number testing

• All test pictures were kept at high quality and not cut, showing how they would be used in real life.

## **2.2Subset-Based Training Strategy:**

Given the size and heterogeneity of the full dataset, a **phased subset training strategy** was adopted. Each subset featured a specific blurring technique to simulate various levels of image degradation, preparing the model for a wide range of real-world scenarios.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Subset | #Images | Blur Strategy | JPEG | Noise | #Patches | Padding | Description |
| 1 | 500 | 1.7× Bicubic ↓ + 1.2× Gaussian + JPEG | 60 | None | 6,250 train / 1,577 test | Yes | Heavy blur (initial phase) |
| 2 | 400 | 1.5× Bicubic ↓ + 0.8× Gaussian | None | None | 4,098 | No | Moderate blur |
| 3 | 500 | Mixed: Heavy, Medium, Light blur | None | None | 5,133 | No | Blur diversity |
| 4 | 350 | 1.7–2.0× Bicubic + 1.2–1.4× Gaussian + JPEG | 60 | None | 3,457 | No | Heavy blur reinforcement |
| 5 | 500 | Mixed + compression + noise | 30–60 (70%) | 40% prob. σ=2–5 | 5,231 | No | Realistic corruptions |
| 6 | 400 | 1.8–2.1× Bicubic + 1.3–1.6× Gaussian | 30–60 | None | 4,188 | No | High blur emphasis |
| 7 | 550 | Mixed: Heavy, Medium, Low + noise | 30–60 (70%) | 40% prob. σ=2–5 | 5,413 | No | Complex degradation |
| 8 | 450 | 1.8–2.1× Bicubic + 1.3–1.6× Gaussian | 30–60 | None | 4,723 | No | Final reinforcement |

This multi-phase structure allowed the model to progressively learn restoration patterns from increasingly varied and realistic degradations, combining downsampling, blurring, noise, and compression artifacts in a staged curriculum.

**2.3 Preprocessing Pipeline:**

**2.3.1 Blurring**

We applied the chosen blur technique to each subset's full-resolution images. We combined bicubic downsampling and Gaussian smoothing. Some subsets also got JPEG compression artifacts to mimic low-bandwidth video. This flexible approach helped us create many types of image degradation.



**Sharp Image**(640 \* 640)



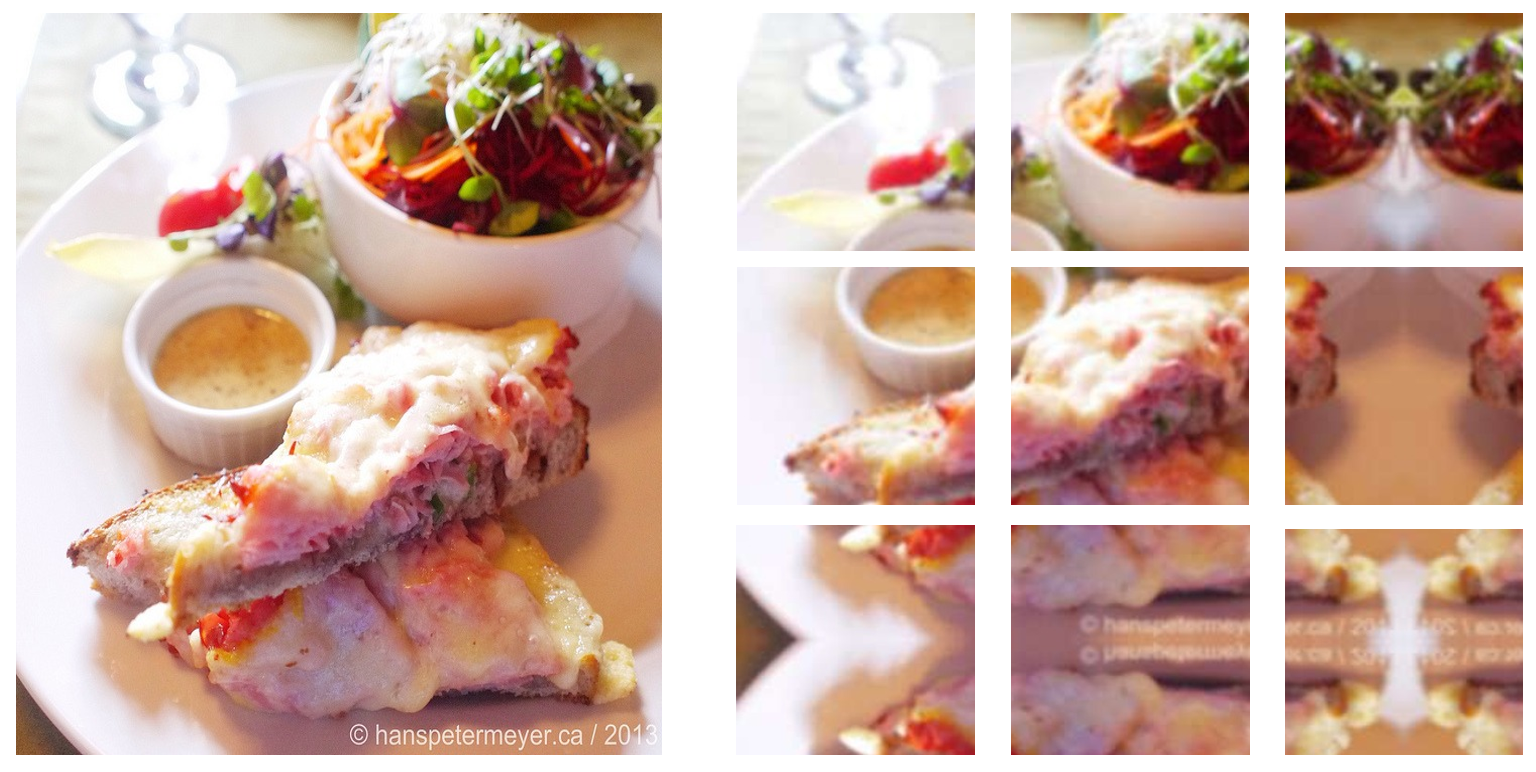
**Medium Blur**(1.5 bicubic + 0.9 gaussian)

**Heavy Blur**(2.0 bicubic + 0.9 gaussian)

**Low Blur**(1.2 bicubic + 0.4 gaussian)

**2.3.2 Padding Strategy**

To maintain inconsistency during the extraction of the sub -area 1, reflective filling used was divisible by 256. The applied logic was:  
w, h = img.size  
 - pad\_w = (patch\_size - (w % patch\_size)) % patch\_size  
 - pad\_h = (patch\_size - (h % patch\_size)) % patch\_size  
Other subfates avoided padding by simply excluding the sideseite in the picture border.



**2.3.3 Patching**  
Post-blurring and (if required) padding, each image was split into non-overlapping 256×256 patches. These patches served as atomic training units. Patch counts varied based on original resolution and subset size. Patch filenames retained the parent image name for traceability.



**2.3.4 Storage Format**  
To ensure compatibility and reusability across training and inference pipelines:  
• Blurred Input Patches: Stored in .png  
• Ground Truth Sharp Patches: Stored in .png  
• Teacher Model Outputs: Stored both as .pt tensors (for analysis) and .png images (used for training)  
This - two formats strategy - provides both visual inspection and effective data charging.

**2.3.5 Normalization**  
No special normalization is done for preprocessing. The student rules worked directly on the raw .png pixel values, allowing maximum transfer infection during the teacher output.

With not -normal and realistic data, we can move on to training. This provides an overview of how we prepare the restomer for offline knowledge dissillation. In the next sections, we will explain how we configured the teacher model (restormer), student model architecture (nafnet-tiny), offline knowledgedistillation pipeline and experimental evaluation.

**2.4 Teacher Model – Restormer**

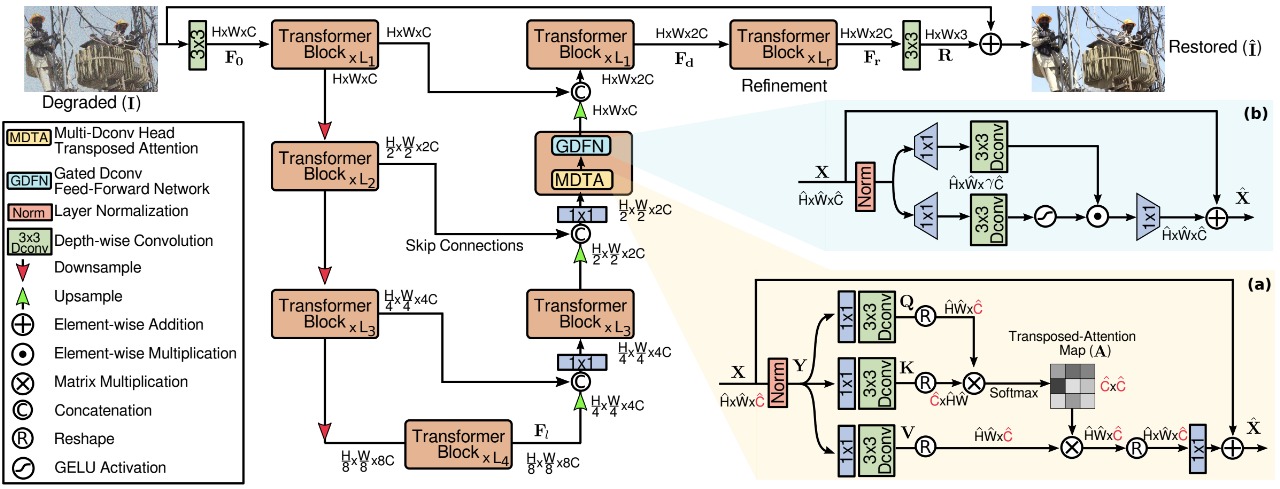
The teacher model used in this work is **Restormer**, a transformer-based network which is very efficient in high-quality image restoration tasks. It was selected considering its good results in several degradation areas such as defocus deblurring, motion blur, deraining, and denoising.

**2.4.1Architectural Outline**

Based on a hierarchical encoder-decoder structure with a self-attention mechanism at its foundation, Restormer (Restoration Transformer) is designed. Unlike conventional CNNs which depend only on convolutional kernels for feature extraction, Restormer makes use of multi-head self-attention with long-range context modeling. Its structure emphasizes:

* Multi-resolution token mixing
* Transformers with feed-forward and residual connections
* Learnable position encoding- Axial attention blocks for effective spatial processing

Deep feature representations at several levels are pulled by the encoder; the decoder then slowly rebuilt the restored image. Skip connections link feature maps across stages and help maintain spatial fidelity.



*From official lrepo -swz30/Restormer*

**2.4.2 Pretrained Models Used**

For this project, no fine-tuning or architectural modifications were made to **Restormer.** Instead, the following official **pretrained weights** were used directly:

* ✅**Single Image Defocus Deblurring** — primary model used throughout most training phases
* ✅**Gaussian Color Denoising (Blind)** — used only for noise-injected subsets (Subset 5 and 7)

These weights were sourced from the official Restormer GitHub repository.

The table below summarizes SSIM performance of all available pretrained Restormer models from the original paper:

|  |  |  |
| --- | --- | --- |
| **Task** | **Dataset** | **SSIM** |
| ✅ Single Image Defocus Deblurring | GoPro | **0.960** |
| ✅ Gaussian Color Denoising (Blind) | SIDD Benchmark | **0.902** |
| Motion Deblurring | GoPro | 0.926 |
| Deraining | Rain100H | 0.842 |
| Image Deblurring (non-defocus) | RealBlur-J | 0.889 |

**Only the two highlighted tasks were used in this project.** Inferenceruns for all experiments were done on a T4 GPU with PyTorch. The output softtargets for knowledge transfer were saved per image. Nofine-tuning or architectural changes to Restormer were made in this work. Justthe following **pretrained weight** sets were used directly:

* **Single Image Defocus Deblurring** model — the main checkpoint that was used for most of the experiments phases
* **Gaussian Color Denoising (Blind)**model — a few instances were chosen for noise removal tasksRestormer

GitHub repository is the source of these models, and they were loaded into acustom inference script that was run on Google Colab with T4 GPU support.

**NOTE :**only .png soft targets were utilized during student model training despite both formats being kept.

This process was a Sharp teacher outputs were saved in both .png and .pt formats3.These teacher outputs served as the distillation targets for the NAFNet-Tiny student model

**This approach formed the basis of an offline knowledge distillation pipeline, where the student model learns from both:**

. The ground truth (sharp patch)

. The teacher output (Restormer prediction)

By using Restormer's refined outputs as supervision signals, the student model benefits from

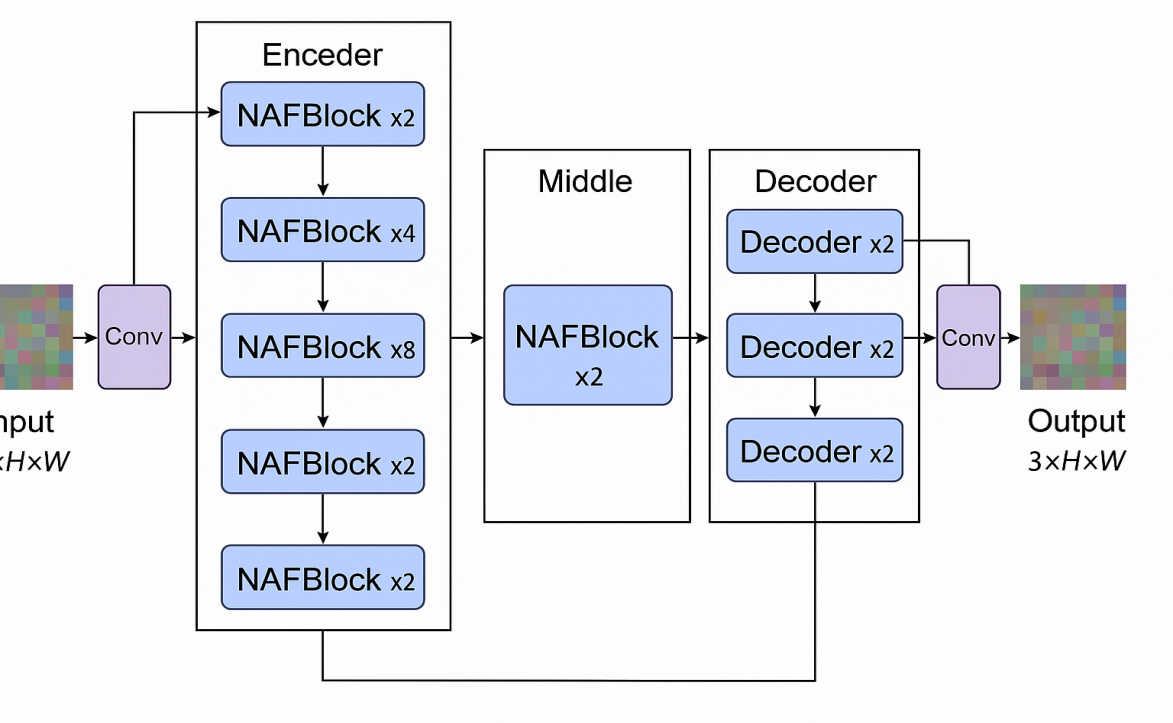
**2.5 Student Model :NAFNet-Tiny Model**

**NAFNet-Tiny**nodel is used as the student model. student model is a light version of the NAFNet (Nonlinear Activation Free Network) architecture, pre-trained on image sharpening or enhancement tasks. It is composed of modular test, train, and utility function components. The following is the description of what each file does and what it is :

**2.5.1 NAFNet\_tiny.py**– Model Structure

This paper introduces the Tiny variant of NAFNet model, an efficient and light-weighted variant that can be deployed on resource-limited devices (e.g., real-time applications or webcam inference).  
Key features:

* Initializes the model's blocks and layers (e.g., residual blocks, convolutional layers).
* Input image channels : 3(RGB)
* Feature width(base number of channels) : 16
* Encoder blocks : [2, 2, 4, 8]
* Middle blocks : 2
* Decoder blocks : [2, 2, 2, 2]
* Does fewer operations to cut down on parameters and computation.
* Is CPU-inferable and ONNX-exportable.



**Parameters :**

Total NAFBlocks: 16 (Encoder) + 2 (Middle) + 8 (Decoder) = 26 blocks Based on the original NAFNet implementation, each block at 16 channels contains approximately 26,848 trainable parameters. Final Parameter Count: Total Parameters=Total NAFBlocks×Parameters per Block = 26 × 26 , 848 = 698 , 048 =26×26,848=698,048 📊Total Trainable Parameters: ≈ 698K ~ 0.7M

While the original NAFNet has 17.02M

**2.5.2 Utils.py**– Support Functions

There are support functions in the utility module useful for training and prediction.  
Support functions in general used include:

* Image loading, normalization, and image ↔ tensor format conversions.
* Saving model output or prediction to disk.
* Computation of metrics (e.g., PSNR, SSIM).

**2.5.3 Dataset.py**– Custom Dataset Loader

1. Responsible for the way data is accessed and preprocessed for training and testing.
2. Primary functions:
   * + Defines a Dataset class (probably by using PyTorch’s Dataset module).
     + It will remove the corrupted images from the dataset folders.
     + Loads image pairs (input/target) from folders or a CSV.
     + Implements transformations like resizing, normalization, or augmentation.
     + Makes sure that data is compatible with training(e.g., CHW format, float32 tensors).

**2.5.4 Train\_student.py**– Student Model Training ScriptExecutes student model training. The model was trained upto**110 epochs**.   
Key features:

* Loads the training and validation data through dataset.py.
* Builds optimizer and student model (nafnet\_tiny.py).
* Knowledge distillation is optional in case of a teacher model.
* Verifies and prints loss, accuracy, PSNR, etc.
* Saves checkpoints of the trained model for future inference.

**2.5.5 Test\_student.py-**Evaluation

Testing model will test the SSIM, MSE(Mean Square error) for the benchmark dataset by using **student\_epoch\_110.pth**path which IS the best path got in Training.

* It will work on the loaded model weight.
* This will run the Inference on the benchmark data
* It will saves the outputs to **test\_outputs**folder.

**2.6 Deployment**

To make the Student\_model usable in the real-time apllications like **zoom,Googlemeet, or OBS.** Two deployment scripts are used to deploy the model into the real time web-cam video. The script converts the whole **pytorch** model into **ONNX** version of model.

**2.6.1 Deployment.py–**deploy pipeline

This script is a straightforward and efficient deployment pipeline for the trained ONNX model.

Principal duties:

* Loads the ONNX model with ONNX Runtime.
* Gathers the default webcam with OpenCV.
* Open each frame, preprocess it (resize, normalize).
* Inference of the frame by the ONNX model.
* Run output (denormalization, conversion to the displayable format).
* Displays the real-time improved output in an OpenCV window.

After running this python deployment.py the whole model is converted into the onnx model. The new onnx model is saved as**student\_model.onnx**in the present nafnet-main folder

**2.6.2 Deploy\_sharpening.py –**virtual camera is on

This will integrate the virtual camera, it will load the model and after the model is loaded the virtual camera will run. We have to install OBS camera in pc to run successfully. While running it will calculate both **inference time** and**FPS**for each frame

**3.Experimental Analysis :**

All the codes were run on a system equipped with :

**Processor :** Intel Core i5,

**RAM :**Intel Core i5 (e.g., i5-1135G7, 11th Gen, 4 Cores / 8 Threads)

**CPU :**8 GB / (commonly)

**OS :**Windows 10/ (64-bit)

**Programming Language :**Python (with Pytorch)

**3.1 Benchmarking and Validation Split of the Dataset :**

In the first step that is data collection we have taken different kinds of data sets from different sources as mentioned above. We have separated the data in train data, teat data and benchmark data. For train data we divided the images into patches and we made blur images for that patched data. We have name the sharp patch and blur patch with same image name to get easily identified by the model. We make images blur with different parameters as the result , it gives heavy blur, medium blur , and low blur images as mention above

Some of the images are :



**3.2 Measures of Evaluation :**

After completion of blurring process, we have cloned the RESTORMER model and the blurred images are passed through the RESTORMER teacher model. In this model we used the pre trained weights like Single Image Defocus Deblurring and Gaussian Color Denoising (Blind). As it is mentioned in the github official Restormer repo the ssim score for this is 90.

Some of the teacher outouts are:



**3.3 Student Model Performance** :

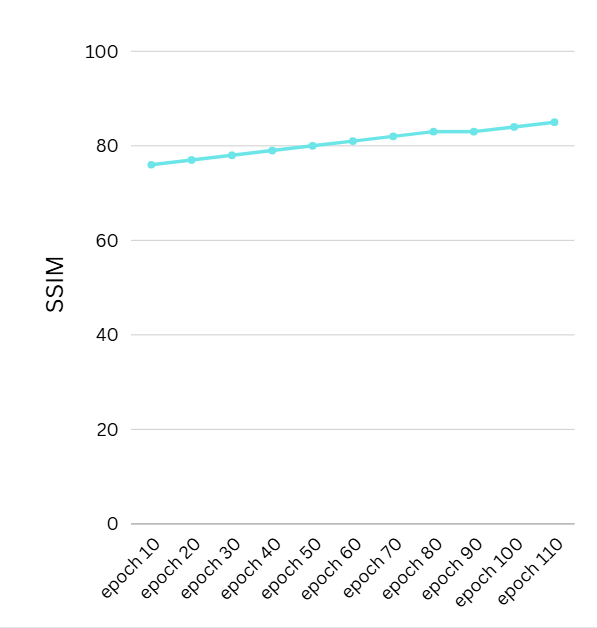
As we took NAFNet model as the student model, we cloned the model and we reduced the parameters and we configured to a **tiny** and a light-weighted version. We trained the model up to 110 epochs. As the epochs are increasing, the SSIM score was gradually increasing.

In training the configurations which I have used are :

* Loss = 0.7\*loss\_groundtruth + 0.3\*loss\_teacher
* SSIM = SSIM(Student output, Ground truth)

While we run the train\_student.py model the each epoch will save one image output for our reference. After completion of the epoch the epoch path will be saved in .pthfile.if we stop in the middle of the epoch the code will automatically save the checkpath\_latest.pth file, so we can continue from where we stopped only if the epoch path is saved. After completion For every epoch the code will give SSIM score of that epoch and it will also validate the bench mark data if we give the validation data.

The below graph shows the SSIM for each epochs :



Epochs for Heavy data

SSIM :

for student\_epoch\_110.pth

for patch images

- for heavy blur = SSIM : 87

- for medium blur = SSIM : 92

- for low blur = SSIM : 94

for full images

- for heavy blur = SSIM : 88

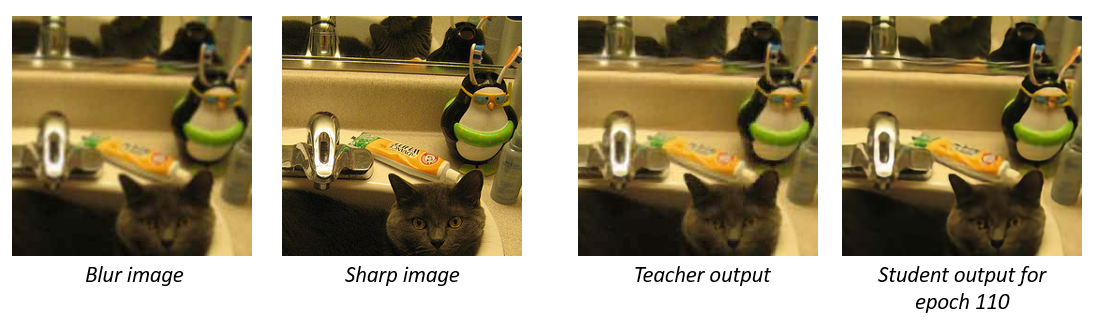
- for medium blur = SSIM : 93

- for low blur = SSIM : 95

We can compare the output of the student model for the epoch 1 and for epoch 110 :



We can also compare with the student model output with the teacher model :



**3.4 Evaluation of Visual Quality :**

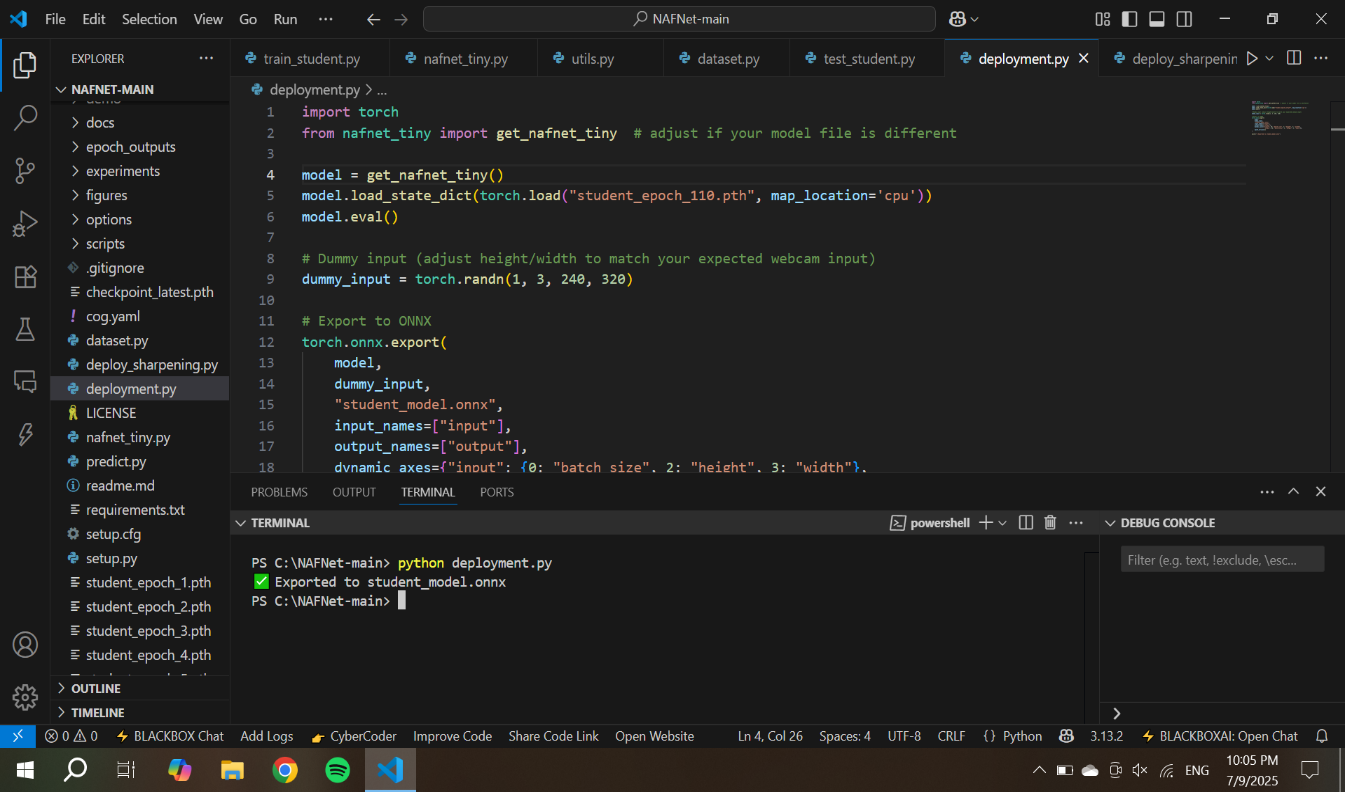
We conducted a survey by taking ratings out of 5 for 4 different images from the people. The survey results are:



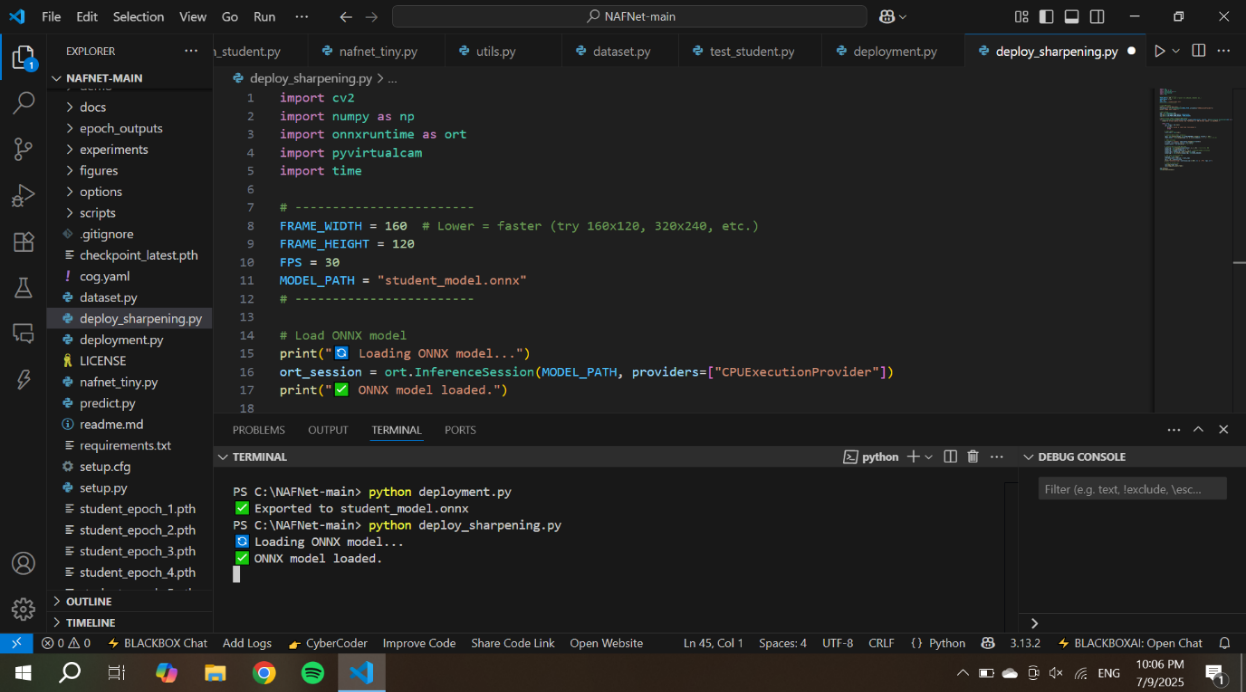
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S.NO** | **NAMES** | **BLURRED IMAGE** | **SHARP**  **IMAGE** | **TEACHER OUTPUT** | **STUDENT OUTPUT** |
| 1 | Hadya | 3 | 5 | 3.5 | 4 |
| 2 | Tharun | 2 | 4.5 | 3 | 4 |
| 3 | Preethi | 3.5 | 5 | 4 | 4.5 |
| 4 | Mythri | 3 | 5 | 4.5 | 4.5 |
| 5 | Shiva | 2.5 | 4.5 | 3 | 4 |
| 6 | Aishwarya | 3 | 5 | 3.5 | 4.5 |
| 7 | Vishal | 2 | 4.5 | 3 | 4 |
| 8 | Kruthi | 3.5 | 5 | 4 | 4.5 |
| 9 | Ravi | 3 | 5 | 3.5 | 4 |
| 10 | Pavan | 3.5 | 5 | 4 | 4.7 |

**3.5 Performance of Deployment in Real-Time Situations :**

The deployment.py will convert the whole Pytorch student model into ONNX model. The model is saved as **student\_model.onnx.**

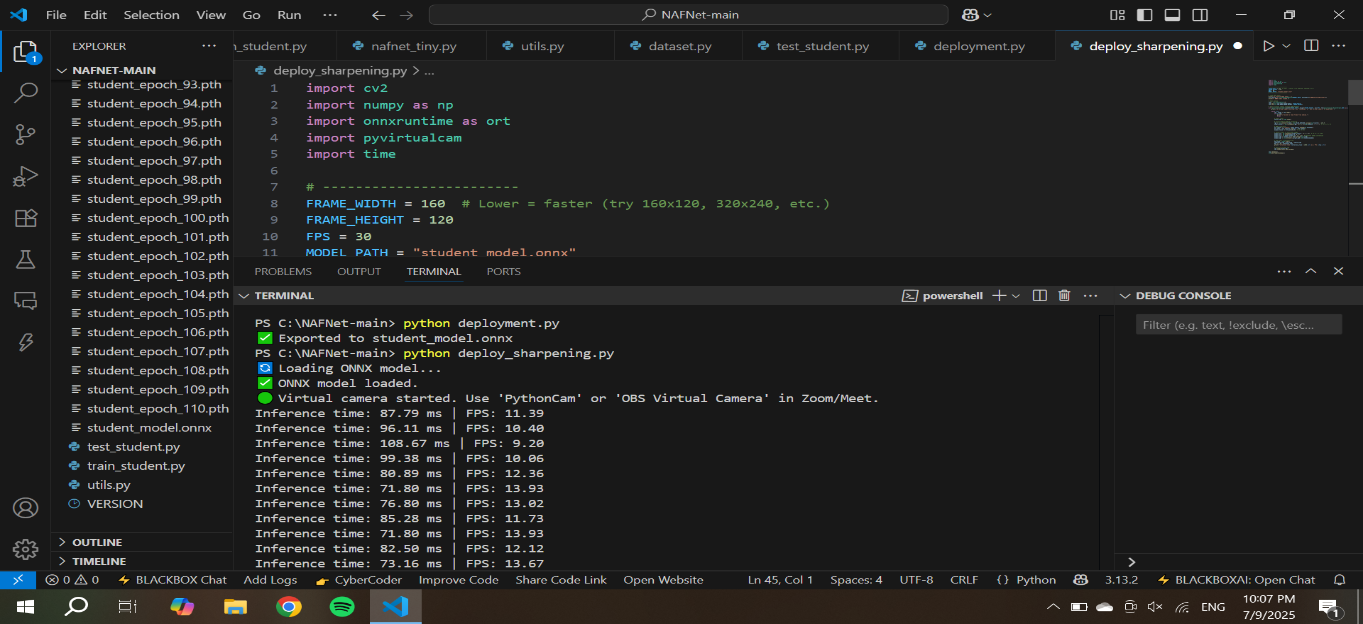
****

Then we have to install the OBS studio (Open Broadcaster Software Studio) . The converted ONNX model is used in **deploy\_sharpening.py**and this will calculate the inference time and FPS. Once the code is executed successfully the OBS webcam will open and it will run automatically. Now we can use any video conferencing apps like zoom, google meet, etc….



**3.6 Inference Speed and Model Efficiency :**

As we are running this on CPU the FPS is around 15 – 20, if we run on high GPU it will give higher FPS ( Above 30)



**4. Conclusion :**

We actually pulled off a real-time image sharpening thing for video calls using this fancy “knowledge distillation” trick. The gist? Take a monster transformer model (Restormer—yeah, sounds like a Marvel villain) that’s really good at cleaning up images, and teach a much smaller, faster model (NAFNet-Tiny) to copy its moves. The small one’s quick enough to run on stuff like Zoom, Google Meet, or even your OBS camera feed, without turning your laptop into a toaster.

We started out by cobbling together this massive, high-res dataset—seriously, like 17,000 images from all over the place: DIV2K, COCO, Flickr2K, random video game screenshots, and some faked-up synthetic stuff. Then, we went wild with the blurring—motion blur, Gaussian noise, JPEG artifacts, you name it—to make sure the model could handle all sorts of real-world messiness.

Restormer played the role of “teacher,” spitting out super sharp versions of the blurry images. Then, NAFNet-Tiny, the “student,” learned by copying both the teacher’s outputs and the original sharp images, but with way less computational muscle. Kind of like learning to play guitar by watching YouTube videos instead of going to Berklee.

We checked how well it did with SSIM, MSE, and even asked a few humans for their thoughts (user survey—nothing too scientific, but hey, it counts). Turns out, the little model held its own. It practically matched the teacher in sharpness, way better than the original blurry junk, and the numbers kept going up after each training round.

Last step: we converted the trained model to ONNX, tossed it onto a regular CPU, and—boom—it actually ran in real time. No fancy GPU, no overheating, just solid frame rates, ready to drop into a video call.

**5. Reference :**

Zamir, S. W., Arora, A., Khan, S., Hayat, M., Khan, F. S., Yang, M. H., & Shao, L. (2022). Restormer: Efficient Transformer for High-Resolution Image Restoration. CVPR 2022.   
GitHub: <https://github.com/swz30/Restormer>  
  
Chen, L., Liu, S., Wang, J., Zeng, H., & Liu, Y. (2022). NAFNet: Nonlinear Activation Free Network for Image Restoration. ECCV 2022.   
GitHub: <https://github.com/megvii-research/NAFNet>

Bevilacqua, M., Roumy, A., Guillemot, C., & Alberi-Morel, M. L. (2012). Low-complexity single-image super-resolution based on nonnegative neighbor embedding. BMVC 2012. (DIV2K Dataset used, by the way.)  
  
Lin, T. Y., Maire, M., Belongie, S., et al. (2014). Microsoft COCO: Common Objects in Context. ECCV 2014.   
[https://cocodataset.org](https://cocodataset.org/)  
  
Timofte, R., Agustsson, E., Van Gool, L., et al. (2017). NTIRE Challenge on Image Super-Resolution. (For the Flickr2K dataset—yeah, that's the one.)  
  
Intel®. (2024). Intel® Unnati Industrial Training Program.   
<https://www.intel.com/content/www/us/en/education/unnati.html>  
  
Paszke, A., Gross, S., Massa, F., et al. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. NeurIPS 2019.   
[https://pytorch.org](https://pytorch.org/)  
  
OpenCV Team. (2023). Open Source Computer Vision Library.   
[https://opencv.org](https://opencv.org/)

ONNX Runtime. (2023). Accelerating Machine Learning Across Frameworks. [https://onnxruntime.ai](https://onnxruntime.ai/)

OBS Studio. (2023). Open Broadcaster Software.

<https://obsproject.com>