INTEL UNNATI PROGRAM –2025

INTERNSHIP REPORT

On

Image sharpening using knowledge Distillation

By

TEAM DYNAMIC DUO

Submitted by:

Chappidi Theekshani

Kusam Varshitha

Degala Thanu Sree

Table Of Contents

DEMO VIDEO LINK: 3
Abstract: 3
Data source and choices: 3
Stage1: The Degradation Pipeline: 4
Stage2: Smart Training with "Patches": 6
Stage 3: Data Augmentation: 6
Model Architecture Selection: 7
The Teacher Model (Teacher_UNet_Wider): 7
Why a Custom U-Net for the Teacher? 7
Reasoning: 7
1.The Student Model (RRDBNet): 8
2. Low-Resolution Computation: 9
Training Methodology and Model Outcomes: 10
A. Training Setup 10
B. Response Based Knowledge Distillation Strategy 10
C. Quantitative Results 11
D. Qualitative Results 11
Final Performance Analysis : 13
The Final Challenge: 14
Optimization for Real-World Deployment: 14
Optimized Outcome Visuals: 15
Future Work: 16
Conclusion: 16

DEMO VIDEO LINK:

https://drive.google.com/drive/folders/1dGdrPOaKSbKqxdBE0wzFhUeeu3kqiYok

Abstract:

Low-bandwidth network conditions often degrade the quality of real-time video conferencing, leading to a poor user experience. This project tackles this challenge by developing a lightweight and efficient deep learning model to enhance video frames in real-time. Our approach uses **Knowledge Distillation**, where we trained a compact 'student' model (RRDBNet) to replicate the performance of a much larger 'teacher' model. To guide the training, we designed a multi-component loss function that included pixel, perceptual, gradient, frequency, and distillation losses. The results demonstrate that our student model successfully learned to generate sharp, high-resolution visuals from degraded input, offering a practical solution for improving video call quality on slow networks.

Data source and choices:

A robust model begins with high-quality, relevant data. After evaluating publicly available and popular datasets for this project, we decided to create a custom dataset from scratch for several key reasons:

1. **Task Specificity:** standard datasets like DIV2K or Flickr2K are great for general image enhancement but don't match the kind of problems we see in real video calls—like heavy compression, low resolution, and noise from poor network connections. Since our goal was to fix those exact issues, we needed data that reflects them.

2. Second, creating a custom dataset gives the full control from the collection of diverse scenarios to the degradation process, ensuring a perfect pairing of low resolution and high-resolution images for precise training.

To overcome this, we collected the data from Videos using our custom-made tool that takes the specified video links and processes them by generating the video frames from each video link at the regular intervals of the video time frame to ensure each frame is unique.

Using this pipeline, we can create a dataset that contains 4,464 high-resolution images (1920x1080).

Now, with our sharp, high-resolution images, how did we prepare the data for training?

Stage1: The Degradation Pipeline:

So, a huge part of this project was creating the right training data. To make our model smart, we couldn't just feed it simple blurry photos. We had to simulate what a *bad* video call looks like**.**

We believed that by training our model on diverse set of problems, it would become a much more flexible solution, capable of handling a range of real-world video quality issues. So, we designed a **robust image triplet preparation pipeline**. Each high-resolution (HR) image is converted into two degraded versions:

- A **low-resolution, compressed, and noisy version** intended as the student_input.
- An **upsampled counterpart** generated from the same degraded image, serving as the teacher_input.

The process includes the following steps:

1. **High-Resolution Ground Truth (HR)**
   Directly copied from the original image source.

2. **Student Input Generation**
    - The HR image is downscaled by a factor of 4 using a random resampling method (either **BICUBIC** or **NEAREST**). By exposing the model to both smooth blurs and blocky or pixelated look, we force it to learn how to reconstruct details from two fundamentally different types of resolution loss.
    - The downscaled image is then saved as **JPEG** with random quality between **5 and 35**, simulating compression artifacts. Since, to send video over the internet, it must be heavily compressed to save space. This compression is what creates the ugly, blocky artifacts we often see.
    - With 50% probability, **Gaussian noise** with variable strength is added to simulate sensor-level or transmission noise. Because sometimes, a bad connection or the camera sensor itself can add a fine layer of "static" or noise to the image.

      We made this step probabilistic because not every bad video frame is noisy. By training on a mix of noisy and non-noisy images, the model learns to handle static when it's present but doesn't get confused or try to remove noise that isn't there.
3. **Teacher Input Generation**
    - The student_input is **resized back to HR resolution** using **BICUBIC interpolation**.
    - This upscaled version mimics a moderately degraded image, providing an intermediate supervision signal.

Dataset source link:

https://www.kaggle.com/datasets/chetanreddyc/ultralightweight-model-training-dataset-v8

Stage2: Smart Training with "Patches":

Instead of feeding the model entire 1920x1080 images, which is very slow and memory-intensive, we used a **patch-based strategy**. This technique allows us to generate thousands of unique training samples from a single image.

How it Works: For each training step, we would randomly cut out a small **128x128 pixel "patch"** by randomly selecting a starting coordinate (x, y) from the full-size HR image and its corresponding teacher input. For the smaller student input, we would cut out the perfectly aligned 32x32 patch extracted starting from a scaled-down coordinate (x/4, y/4)

This technique is like creating thousands of mini-training examples from a single image. It dramatically increases our data variety and allows the model to learn much faster and more effectively. It ensures the model focuses on learning the fine details in a manageable way.

Stage 3: Data Augmentation:

Finally, to make our model even tougher, we applied **data augmentation** to these patches with only 50% probability.

- **Geometric Augmentation:** We would randomly **flip** the patches horizontally or vertically and sometimes **rotate** them by 90 degrees. This teaches the model that a face is still a face, even if it's on the right side of the screen instead of the left. It learns to recognize features regardless of their orientation.
- **Color Augmentation:** We also randomly changed the **brightness, contrast, and saturation** of the *blurry* input patches, while keeping the HR "answer key" image perfect. This is a crucial step. It teaches the student

to restore the original, perfect colors even if its input feed is washed-out, too dark, or discoloured due to bad lighting or camera settings.

After passing through this three-stage pipeline, the result is a dictionary containing a triplet of perfectly aligned, augmented, and tensor-formatted image patches ready to be fed into the training loop.

This triplet structure is the foundation of our entire training pipeline, containing **4,017 training samples** and **447 validation samples**.

Model Architecture Selection:

The Teacher Model (Teacher_UNet_Wider):

A core principle of our knowledge distillation strategy was that the student can only be as good as its teacher. The teacher model serves as the source of all "knowledge," providing the high-quality target as output that the lightweight student model aims to replicate. Therefore, the selection of the teacher was one of the most critical decisions in this project.

Why a Custom U-Net for the Teacher?

Reasoning:

Our initial approach involved to explore and evaluate state-of-the-art, pre-trained super-resolution models to serve as our teacher. We investigated several well-known architectures, including prominent models like **Real-ESRGAN, Restormer**.

While these models are incredibly powerful for general-purpose image enhancement that includes tasks like denoising, deblurring., So, when we tested on our dataset images those are not giving the promising results that a person can visually be identified through naked eye. As, we researched on this we found these reasons:

Mismatch in Degradation Type: Most pre-trained super-resolution models are trained to enhance a simple, clean Bicubic downscaling. Our project, however, needed to address a much more complex degradations specific to video streaming like JPEG **compression artifacts**, **random noise**, and a mix of **blurry and pixelated** downscaling. The outputs from these pre-existing models, when applied to our specific type of degraded images, were often unsatisfactory and did not produce the clean, artifact-free results which we required.

Also, several other promising models we identified, pre-trained weights were not available which prevented us from using those models.

With these limitations we decided to **design and train our own expert Teacher model**. This approach gave us complete control over the architecture and its processing. By investing the time to train a high-quality, specialized Teacher, we created the strongest possible foundation for our knowledge distillation process. The Teacher became the ultimate source of truth, providing a clear and consistent goal for our lightweight Student model.

- **Architecture:** A U-Net with a wide feature base (features=96) was used as the expert teacher. Its encoder-decoder structure with skip connections is excellent for capturing both high-level context and fine-grained spatial detail.
- Trainable Parameters: 906,915 and achieved SSIM of 88.8% score.

1.The Student Model (RRDBNet):

Unlike the Teacher model, which was designed for maximum accuracy, the student model was engineered to maintain the perfect balance between performance and speed specifically for real-time video enhancement.

To achieve this, we selected RRDBNet (Residual-in-Residual Dense Network) — a lightweight architecture image super-resolution. Its inherent ability to

reconstruct realistic textures and fine details from degraded images made it an excellent choice for our task of restoring low-quality video frames.

The choice of RRDBNet is rooted in both its architecture and computational efficiency:

1. Deep but Efficient Feature Reuse: RRDBNet leverages a unique block structure

- **Residual-in-Residual Dense Blocks (RRDBs)** stack several Residual Dense Blocks (RDBs), where in our implementation each RDB block contains **5 densely connected convolutional layers** and the whole network's body consists of 4 RRDB blocks. As each RRDB block is composed of 3 ResidualDenseBlocks (RDBs).
- These layers perform **extensive feature reuse** because every layer gets information from **all** previous layers at once, allowing the network to "brainstorm" solutions across depths and extract rich features, even at shallow depths.

This design makes it possible for the model to **hallucinate high-frequency details** such as **skin pores**, **fabric weaves**, or **fine edges** from extremely low-resolution inputs.

2. Low-Resolution Computation:

Unlike the Teacher (a symmetrical U-Net) which performs multiple convolutions at higher resolutions, RRDBNet performs most of its heavy processing at **low resolution** feature map, which is computationally efficient:

- Only the **final stage** performs an expensive **4× upsampling** operation.
- This design significantly reduces memory and compute load, making it ideal for real-time inference scenarios.

- **Lightweight Parameters:** To ensure high speed, its architecture was configured with the following parameters:
    - Base Number of Features (nf): **32** (3x smaller than the teacher)
    - Number of RRDB Blocks (nb): **4** (a shallow depth)
    - Growth Channel Rate (gc): **16**
    - Trainable Parameters: 758,755

Training Methodology and Model Outcomes:

A. Training Setup

- **Epochs:** The model was trained for a total of **63 epochs**.
- **Batch Size:** A batch size of **32** was used.
- **Optimizer:** We used the **Adam** optimizer with a learning rate of **1e-4**.
- **Hardware:** Training was accelerated using **2 GPUs** with PyTorch's nn.DataParallel wrapper.

B. Response Based Knowledge Distillation Strategy

Our training approach was based on **response-based knowledge distillation**, where the student model learns to mimic both the **teacher's enhanced output** and the **true high-resolution ground truth**. A combined loss function was used to guide the student:

Total Loss = (α * L_gt) + (β * L_distill) + (γ * L_perceptual) + (ε * L_gradient) + (λ * L_frequency)

Ground Truth Loss ensures the output stays close to the real high-quality image. It ensures the student doesn’t copy any mistakes from the teacher.

Distillation Loss Compares the student's result with what the expert teacher model produced to imitate the output.

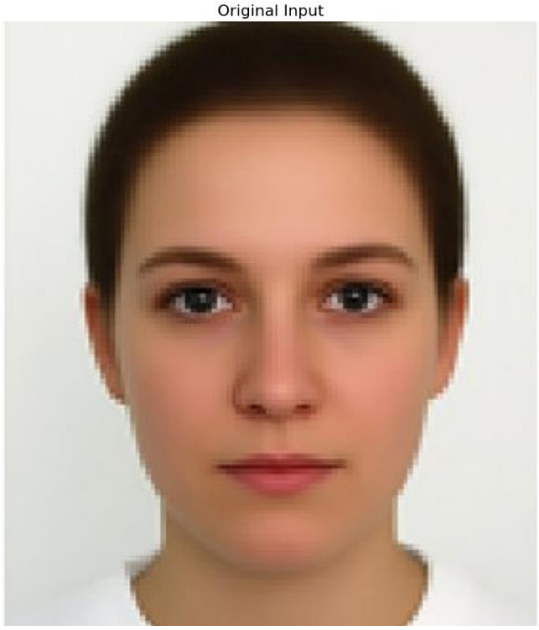
Perceptual, Gradient, and Frequency Losses help improve visual sharpness, edge clarity, and fine texture details.

C. Quantitative Results

- **Structural Similarity Index (SSIM):** The student model was evaluated on a benchmark dataset of over 86 images that were degraded using downscaling, JPEG compression, and noise where the student model achieved an SSIM of **87.82 %** score**.**
- In terms of performance, the lightweight student model was tested on a GPU to ensure its feasibility for real-time applications. It achieved a **model FPS of 58.5 frames per second (FPS)**,

D. Qualitative Results

Original Input

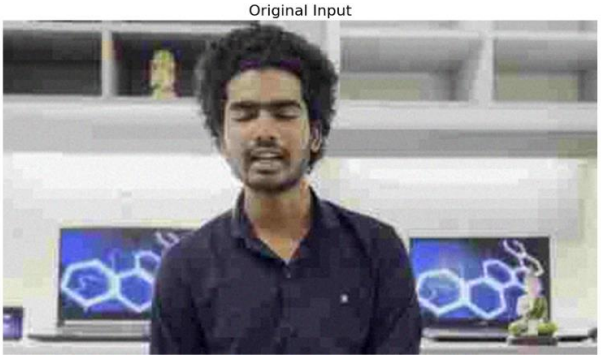


Enhanced Output

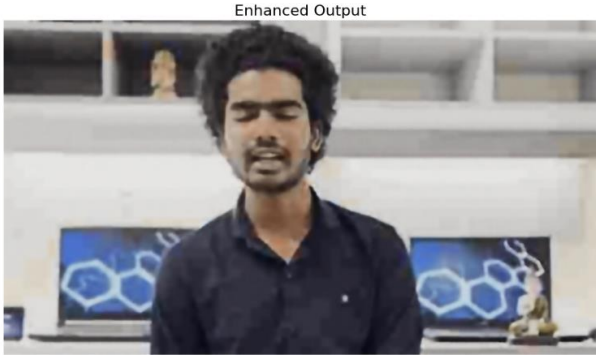
Original Input

Enhanced Output

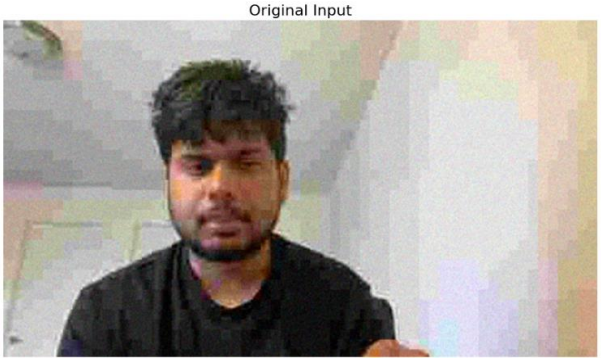
Original Input



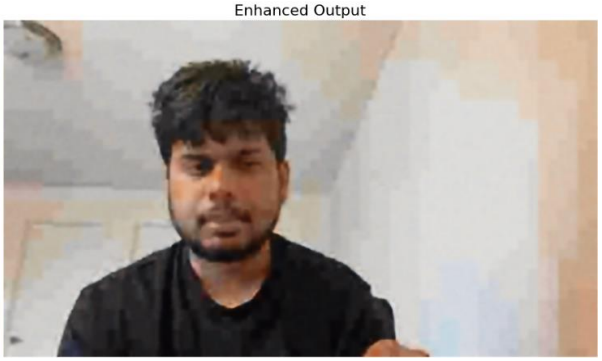
Enhanced Output



Original Input



Enhanced Output



Final Performance Analysis :

The development of our final student model was the result of an iterative process spanning nine distinct versions, each refining the balance between architectural complexity, model size, and performance. The quantitative results presented—achieving a **Model FPS of ~58 on a GPU** and a **Model FPS of ~25 on an optimized CPU**—are not arbitrary figures. They represent the practical performance limits of this highly tuned architecture A key takeaway from this is that the model's speed is directly tied to the computational power of the hardware it runs on.

We have engineered the RRDBNet architecture to be as efficient as possible for the task of real-time image sharpening. The "Model FPS" metric isolates the pure "thinking speed" of the network, and this speed is a direct reflection of how many operations the hardware can execute per second.

Therefore, the performance we have measured is the result of a deliberate engineering trade-off. We have pushed this lightweight design to its practical limits based on the data it was trained on and the task it must perform. The model effectively utilizes all the power the hardware provides, delivering the maximum possible framerate for its size. We conclude that we have successfully created an efficient model capable of real-time performance, whose final speed will scale directly with the capability of the deployment hardware.

The Final Challenge:

Optimization for Real-World Deployment:

Our initial benchmarks proved the success of our model's architecture. On a powerful Colab GPU, the lightweight Student model achieved an impressive **~58 FPS**, confirming its high efficiency which is far beyond from reality. But here's the problem: most people don't have a dedicated GPU. Real-world deployment means running on a standard CPU. When we tested the original PyTorch model on a CPU, it barely reached 1.2 FPS which is practically unusable.

This created a critical challenge: how do we bridge the massive gap between our model's potential and its practical performance on common hardware for everyday devices?

Our Solution: Optimization with Intel® OpenVINO™ Toolkit

To solve this, we turned to **Intel's OpenVINO™ (Open Visual Inference & Neural Network Optimization) toolkit**.

OpenVINO doesn't change the model, it transforms how it runs. It is a high-performance inference engine and optimization tool designed specifically to accelerate deep learning models on Intel hardware.

it takes the existing neural network and optimizes everything under the hood such as fusing layers, adjusting memory access, and reducing precision where possible all to get more speed out of the same hardware.

The Process:

Think of OpenVINO as an expert tuner for our student model. We took our fully trained PyTorch model and put it through the OpenVINO optimization pipeline.

1. **Conversion:** The model was first converted to the standard ONNX format (Open Neural Network Exchange).
2. **Optimization:** The OpenVINO Model Optimizer then took this ONNX file and performed a series of advanced optimizations, such as **graph fusion** (merging multiple layers into a single, more efficient one) and **precision calibration**.
3. **Final Output:**

   This process generated a pair of highly optimized files:

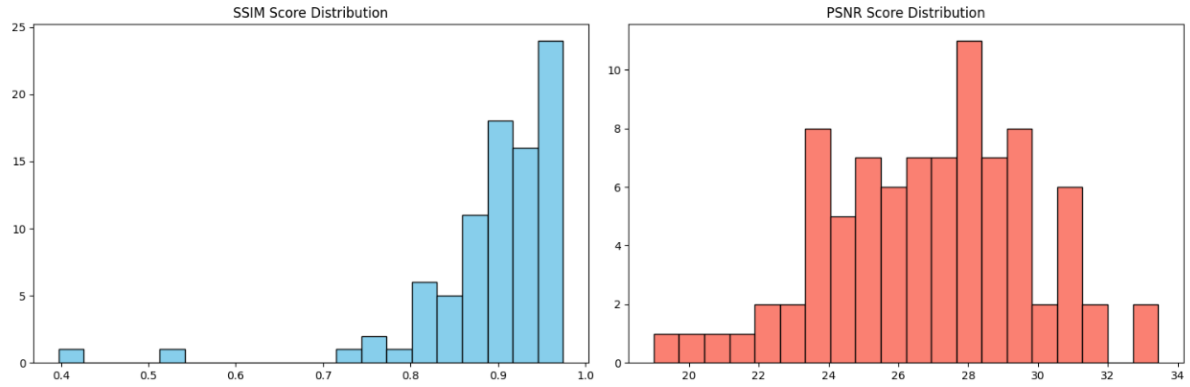
   - model_v9.xml: The blueprint describing the new, optimized network structure.
   - model_v9.bin: The file containing the trained weights, ready for the optimized engine.

Optimized Outcome Visuals:

Best Case Example: SSIM = 0.9745, PSNR = 30.91 dB

```
==========================================
📊 CPU BENCHMARK (OpenVINO) - NUMERICAL SUMMARY 📊
==========================================
Average SSIM: 0.8937 (Std Dev: 0.0859)
Average PSNR: 26.89 dB (Std Dev: 2.92)
Average FPS (System Throughput): 0.10 FPS
==========================================

==========================================
📈 CPU BENCHMARK (OpenVINO) - GRAPHICAL ANALYSIS 📈
==========================================
```



Future Work:

On the CPU, we saw that even with a powerful tool like OpenVINO giving us a 3x speedup, the final framerate in this Colab environment was low. And this is an important finding. It shows us that the performance of any AI model is fundamentally tied to the hardware it's running on. This leads us perfectly into Future Work.The clear next step is to take this optimized OpenVINO model and run it on a modern desktop CPU, where we fully expect it to meet the real-time 30 FPS target. We can also explore even more advanced optimization techniques to push the model speed even further.

Conclusion:

This project demonstrates the effectiveness of using knowledge distillation to build a lightweight yet high-performing model for real-time video enhancement. By training a compact **RRDBNet** student model to replicate a custom **U-Net** teacher, and carefully designing a degradation pipeline tailored to real-world video issues, we achieved both quality and efficiency. With an **SSIM of 87.82%** and real-time speeds of **58 FPS on GPU** , our model proves to be both accurate and light weight. OpenVINO optimization further enhanced CPU performance,

making the solution practical for everyday devices. This work lays a strong foundation for deploying image sharpening model in real-world applications like video calls and streaming.