INTEL-UNNATI AI&ML JOURNAL
Number 2 (2025), pp. 1–9

Code:IUIT-2025

MACHINE LEARNING

Image Sharpening Using Knowledge Distillation

Team Axion

Arya Vinod, Ashlin Rose Manoj

Saintgits Group of Institutions, Kottayam, Kerala

Abstract: Achieving and maintaining image sharpness is a constant challenge, despite the fact that video conferencing is used by many people worldwide. A knowledge distillation framework is used in this project to further enhance image sharpening. Using the high-performance sharpened output of the teacher model as a guide, a pre-trained image sharpening model (Teacher) will be used to train a computationally efficient real-time model (Student). In this instance, the student model's objective is to maintain the teacher model stimulus's image quality while achieving frame rates of 30–60 frames per second in real time on easily accessible hardware. Bicubic and bilinear interpolation are used to upscale ground truth images in order to replicate realistic conditions. SSIM scoring will be used for evaluation, with benchmarks exceeding 90%. MOS testing will be used for additional subjective assessment. In the end, the produced architecture is able to use video conferencing situations with sparse bandwidths that are considerably confined on data accessibility since it conclusively exhibits pertinent correctness and visual precision together with substantial effectiveness.

Keywords: Image sharpening, Knowledge distillation, SSIM, Image Degradation, TensorFlow, Soft Targets, PSNR

1 Introduction

In this digital world, the quality of an image is one of the most vital things, especially today with the rise of video conferencing, surveillance cameras, and many more. These real-time tasks require high-quality images. But network limitations and hardware restrictions mostly lower the image quality, resulting in blurry and unsharpened images. This affects our effective communication, especially during a video conference. Traditional im-

age sharpening techniques are impractical because they require high computational power which sometimes cannot be met by real-time systems.

This project of ours addresses this issue and presents an image sharpening technique that uses knowledge distillation. In this model, a lightweight model called the student model learns from a more complex teacher model through knowledge transfer. This method focuses on sharpening the image while keeping computational costs low. With this model, it is possible to enhance images in real time by training on visual correction and fine details, even while having modest hardware and network conditions. Efficient use of computational resources improves the quality of images received during video streams.

2 Literature Review

This review builds on key breakthroughs in optimization, model compression, and visual quality assessment. We picked the Adam optimizer (Kingma & Ba 2015) because it adjusts learning rates for each parameter based on first and second gradient moments. It also has bias correction to help steady convergence, which helps when gradients are sparse. To keep our model small but still accurate, we use knowledge distillation (Hinton et al. 2015). In this approach, we train a smaller "student" model using both regular hard labels and softened outputs from a bigger "teacher" model. We create these soft targets by raising the softmax temperature. They contain info about how classes relate, which helps the model learn better. To check image quality, we use the Structural Similarity Index (SSIM) that Wang et al. (2004) came up with. SSIM measures how alike images look by comparing brightness, contrast, and layout. It works a lot like human eyes do. SSIM matches what people think better than Mean Squared Error (MSE) or Peak Signal-to-Noise Ratio (PSNR) so it's great for judging how well we've fixed up an image. Together, Adam optimization, knowledge distillation, and SSIM evaluation give us a solid base to build image models that work well and look good to people.

3 Libraries Used

In the project for various tasks, the following packages are used:

```
TensorFlow: Machine learning tasks.
OpenCV: Image processing.
NumPy: Numerical operations.
PIL (Pillow): Image loading, enhancement, noise, blur, and saving.
Matplotlib: To visualize the images.
TQDM: Training process visualization.
Scikit-learn: To access the quality of images.
```

4 Methodology

In order to enhance the blurry frames of any situation characterized by limited bandwidth, the algorithm establishes a system for creating and training data. This process majorly involves four libraries which are OpenCV, Pillow, NumPy, and Scikit-Learn for performing and handling of the images, mathematical functions, file manipulations, and progress

monitoring. For each pair of images under test, the system produces two images: a low-resolution blurred image and another sharp image from which the model learns. The blurred images are stored in separate directories, whereas the low-resolution images remain alongside the pristine scaled versions of the image. After that, the dataset is divided into training, validation, and testing sets. This meticulous separation aims at ensuring that the model performs well on new, unseen data. This method ensures reliable and consistent dataset generation for projects involving image restoration.

5 Implementation

5.1 Dataset Preparation And Degradation

Set	Number of Images	Percentage
Training	263	∼70.0%
Validation	57	∼15.1%
Test	57	∼15.1%
Total	**377**	**∼100%**

Table 1: Distribution of the image dataset.

The dataset consists of 377 images, which were split according to the ratios defined in the methodology. The final distribution is detailed in Table 1.

5.2 Model Architecture

The idea behind this project is to have a lightweight model, called the student model, and a deeper, complex model, called the teacher model. The teacher model is a large CNN with many layers and ReLU activation layers to perform high-quality image sharpening. The most optimal mode here is distillation, since it learns more intricate features in image restoration. The student model predicts an output which consists of the difference between the input and output, which implies that the structure has residual connections and has fewer filters in real-time applications. These structures facilitate a lesser computational environment and also guarantee that the student can mimic the performance of the teacher model.

5.3 Knowledge Distillation strategy

The code trains a lightweight image sharpening model (the student model) for real-time applications such as video conferencing using the knowledge distillation technique. A multi-layer deep convolutional neural network is used to train the teacher model, processing patches of sharp and blurry images. The teacher model is optimized using Mean Absolute Error (MAE) loss, also called L1 Loss. Following training, the "soft targets," or the teacher's outputs for the blurry patches, are created using the teacher model. The student model is a neural network with residual blocks, which makes it quicker and easier to use. In addition to the initial ground truth, it is trained using the teacher's output. The student

model tries to mimic the teacher's method and attain excellent image sharpening by using a blended loss function. This makes it appropriate for scenarios involving real-time or bandwidth constraints where the teacher model would be too slow or expensive; there we can deploy this lightweight student model. The code successfully strikes a perfect balance between quality and efficiency. The loss was calculated using the formula:

$$\mathcal{L}_{\text{total}} = (1 - \alpha) \cdot \mathcal{L}_{\text{hard}} + \alpha \cdot \mathcal{L}_{\text{soft}}$$

where:

- $\mathcal{L}_{\text{hard}}$ is the **hard loss**, which is the loss between the student model's prediction and the ground truth sharp image;
- $\mathcal{L}_{\text{soft}}$ is the **soft loss**, which is the loss between the student model's prediction and the teacher model's output;
- α is 0.5, the weighting factor used to balance the two losses.

5.4 Training Procedure

There are two phases to training. Initially, a deep teacher model is trained on pairs of sharp and blurry images with MAE loss to sharpen images. Soft targets are created in which, followed by the training, the teacher provides the estimated output to the given haze training data. The suggested loss is a mixed loss that weighs the contribution of each error measured between the student's output and hard and soft targets, since a lightweight student model is trained using the teacher output (soft targets) and target sharp images (hard targets). Distillation, thus, offers a more efficient and practical method for training a student to achieve teacher-level performance in real time.

5.5 Evaluation Setup

The student model that had been trained using knowledge distillation was evaluated on a set of unseen images that the model had not seen during the training phase. The model was evaluated on two metrics: Structural Similarity Index (SSIM) and Peak Signal-to-Noise Ratio (PSNR). SSIM estimates visual and structural similarity between the ground truth and the student's output, and PSNR estimates pixel-exact error. Predictions of the student model were compared with the blurred input, the teacher model's output, and the ground truth. The inference rate was also calculated in FPS to test the model's capability of running in real-time applications. Visual inspection confirmed that the student model generated sharp and clear images, showing the success of knowledge distillation to train a fast and efficient image enhancement model.

Structural Similarity Index (SSIM):

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

Peak Signal-to-Noise Ratio (PSNR):

$$\text{PSNR} = 10 \cdot \log_{10}\left(\frac{MAX_I^2}{\text{MSE}}\right)$$

6 Result and Discussion

Knowledge distillation was used successfully to create a student model that is lightweight and capable of sharpening images at high quality. Blurred images were utilized to evaluate the model and compare it against ground truth sharp images using the Structural Similarity Index (SSIM). The average SSIM score was high, indicating that the model produces visually correct and sharp outputs. The student model also demonstrated high speed, running on several images per second, and was therefore suitable for real-time applications like video conferencing or live video enhancement. While being smaller and quicker than the teacher model, the student model demonstrated comparable performance due to soft targets derived from training. The knowledge distillation method successfully balanced high image quality against fast inference time and was therefore well-suited for real-world deployment.

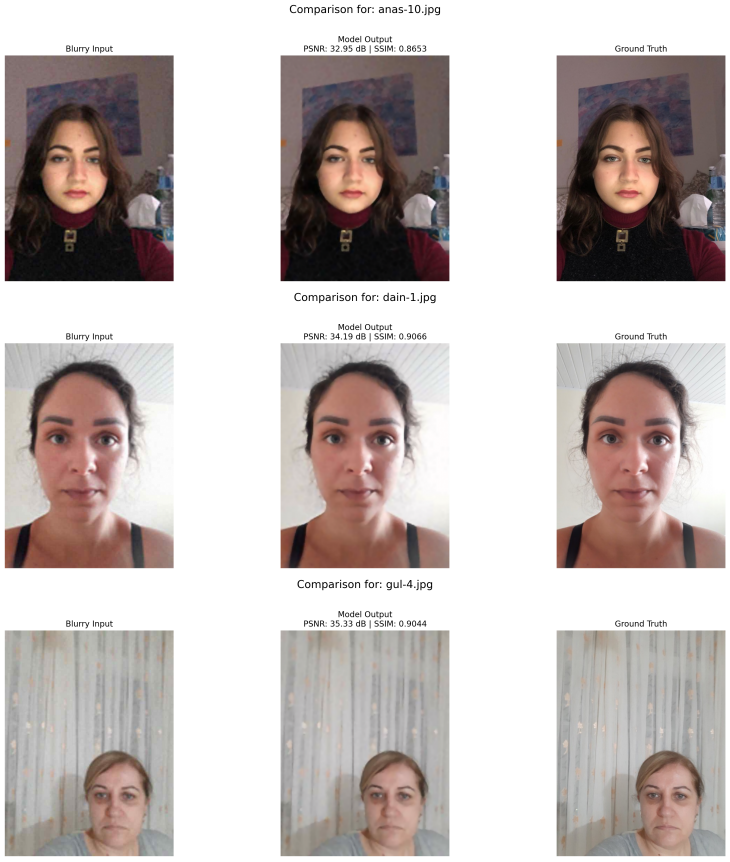

Figure 1: Visual comparison of image sharpening results for three test images.

7 Conclusions

The results prove that the provided code successfully implements a complete pipeline for real-time image sharpening by means of knowledge distillation. First of all, a large-capacity

teacher is trained on pairs of sharp and blurry images. Having learned to restore image clarity with high precision, the teacher model then generates soft targets that carry rich, learned information. In the next stage, the training of a lightweight student network is carried out. This student network learns both from the teacher's predictions as well as from the ground-truth sharp images. Hence, the student gets to mimic the teacher's performance. The combined loss function enables them to be smaller and much faster than the teacher model. During training, a trade-off is established between image quality (via SSIM) and inference speed (via FPS). The code ends by demonstrating how the student model retains a large percentage of teacher performance, but at a fraction of the cost. These findings prove that the student can be deployed in real-time scenarios where fast and efficient image enhancement is required, such as video conferencing.

Acknowledgments

We sincerely thank **Intel© Corporation** for believing in our vision and providing the stage to develop it through the *Intel© - Unnati Program*. Their generous support and constant encouragement turned our initial concept into a tangible working solution. We are especially appreciative of Er. Anish M George, and Mr. Siju Swamy our dedicated mentors, whose steady guidance, sharp insights, and unyielding encouragement have illuminated every step of this project. His expertise and timely feedback pushed us to refine our approach and improve at each milestone. We are also grateful to Saintgits College of Engineering and Technology for giving us the core knowledge, computational tools, and hands-on training needed to build this project. The supportive academic atmosphere and technical mentoring we found there were crucial to our growth. We extend our thanks to the wider artificial-intelligence and machine–learning community. The open research shared by countless developers and especially work on image enhancement and knowledge distillation–inspired our study and shaped its direction. Finally, we appreciate our institutional heads, program coordinators, and fellow students who cheered us on from start to finish. Their trust and encouragement kept us motivated and helped us deliver an outcome we hope will make a real impact.

References

[1] Opencv library. https://opencv.org. Accessed: 2025-07-04.

[2] HINTON, G., VINYALS, O., AND DEAN, J. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).

[3] KINGMA, D. P., AND BA, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)* (2015).

[4] WANG, Z., BOVIK, A. C., SHEIKH, H. R., AND SIMONCELLI, E. P. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing 13*, 4 (2004), 600–612.

8 main code sections for the solution

8.1 data loading and preprocessing

```
def data_patch_generator(blurry_dir, gt_dir, patch_size, stride, target_img_width,
                                        shuffle=True):
    blurry_image_paths = sorted(glob.glob(os.path.join(blurry_dir, "*.jpg")))
    gt_image_paths = sorted(glob.glob(os.path.join(gt_dir, "*.jpg")))

    path_pairs = list(zip(blurry_image_paths, gt_image_paths))
    if shuffle:
        random.shuffle(path_pairs)

    for blurry_path, gt_path in path_pairs:
        # Load and resize images
        blurry_img_orig = load_image(blurry_path)
        gt_img_orig = load_image(gt_path)
        if blurry_img_orig is None or gt_img_orig is None: continue

        blurry_img_resized = cv2.resize(blurry_img_orig, (target_img_width,
                                                IMG_HEIGHT), interpolation=cv2.
                                                INTER_AREA)
        gt_img_resized = cv2.resize(gt_img_orig, (target_img_width, IMG_HEIGHT),
                                                interpolation=cv2.INTER_AREA)

        # Create and yield patches
        b_patches_img, g_patches_img = create_patches_from_single_image(
            blurry_img_resized, gt_img_resized, patch_size, stride
        )
        if b_patches_img is not None:
            indices = np.arange(len(b_patches_img))
            if shuffle: np.random.shuffle(indices)

            for idx in indices:
                yield b_patches_img[idx], g_patches_img[idx]

def create_tf_dataset_for_patches(blurry_dir, gt_dir, patch_size, stride,
                                        target_img_width, batch_size, shuffle=
                                        True, repeat_dataset=False):
    output_signature = (
        tf.TensorSpec(shape=(patch_size, patch_size, IMG_CHANNELS), dtype=tf.
                                                float32),
        tf.TensorSpec(shape=(patch_size, patch_size, IMG_CHANNELS), dtype=tf.
                                                float32)
    )
    dataset = tf.data.Dataset.from_generator(
        lambda: data_patch_generator(blurry_dir, gt_dir, patch_size, stride,
                                                target_img_width, shuffle=shuffle
                                                ),
        output_signature=output_signature
    )
    if shuffle:
        dataset = dataset.shuffle(buffer_size=max(1000, batch_size * 50))
    if repeat_dataset:
        dataset = dataset.repeat()

    dataset = dataset.batch(batch_size)
    dataset = dataset.prefetch(tf.data.AUTOTUNE)
```

```
    return dataset
```

8.2 model architecture

8.2.1 teacher model

```
def build_teacher_model(input_shape):
    inputs = Input(shape=input_shape)
    x = Conv2D(64, (3, 3), padding='same')(inputs); x = ReLU()(x)
    x = Conv2D(64, (3, 3), padding='same')(x); x = ReLU()(x)
    x = Conv2D(128, (3, 3), padding='same')(x); x = ReLU()(x)
    x = Conv2D(128, (3, 3), padding='same')(x); x = ReLU()(x)
    x = Conv2D(64, (3, 3), padding='same')(x); x = ReLU()(x)
    outputs = Conv2D(input_shape[-1], (3, 3), activation='sigmoid', padding='same'
                                                )(x)
    return Model(inputs, outputs, name="teacher_model")
```

8.2.2 student model

```
   def build_student_model_residual(input_shape):
    inputs = Input(shape=input_shape)
    s = Conv2D(24, (3, 3), activation='relu', padding='same')(inputs)
    s = Conv2D(24, (3, 3), activation='relu', padding='same')(s)
    residual = Conv2D(input_shape[-1], (3, 3), activation='linear', padding='same'
                                                )(s)
    outputs_sum = Add()([inputs, residual])
    outputs = ReLU(max_value=1.0)(outputs_sum)
    return Model(inputs, outputs, name="student_model_residual")
```

8.3 final evaluation

```
    # Note: This code assumes final_student_model, X_test_full, and Y_test_full
                                                are loaded.
test_psnr_scores = []
test_ssim_scores = []
student_test_preds = []

print("Predicting on test images and calculating metrics...")
for i in tqdm(range(len(X_test_full))):
    blurry_test_img = np.expand_dims(X_test_full[i], axis=0)
    sharpened_output = final_student_model.predict(blurry_test_img, verbose=0)[0]
    sharpened_output = np.clip(sharpened_output, 0.0, 1.0)
    student_test_preds.append(sharpened_output)

    gt_img_test = Y_test_full[i]

    # Calculate PSNR
    psnr_score = tf.image.psnr(gt_img_test, sharpened_output, max_val=1.0).numpy()
    test_psnr_scores.append(psnr_score)

    # Calculate SSIM
    win_size = min(gt_img_test.shape[0], gt_img_test.shape[1], 7)
```

```
    if win_size % 2 == 0: win_size -= 1
    ssim_score = structural_similarity(gt_img_test, sharpened_output, channel_axis
                                       =-1, data_range=1.0, win_size=
                                       win_size)
    test_ssim_scores.append(ssim_score)

# Report average scores
if test_ssim_scores and test_psnr_scores:
    average_psnr = np.mean(test_psnr_scores)
    average_ssim = np.mean(test_ssim_scores)
    print(f"\nAverage PSNR on Test Set: {average_psnr:.2f} dB")
    print(f"Average SSIM on Test Set: {average_ssim:.4f}")
```