Project Report
Game Screenshot Upscaler

Hasan Asim

July 30th, 2025

Features

Deep Learning-Based Super-Resolution

- Uses an enhanced deep residual network (EDSR) architecture for high-quality 2× image upscaling.

Trained Specifically on Game Screenshots

- Dataset of cropped and downscaled screenshots from video games ensures domain-specific performance.

Multi-Loss Optimization

- MSE (pixel accuracy)
- SSIM (structural sharpness)
- Perceptual Loss (texture preservation via VGG16)
- Gradient Loss (edge and detail retention)

CUDA Acceleration

- Model leverages GPU computation (via PyTorch + CUDA) for faster training and real-time inference.

Visual Evaluation Support

- Includes tools for comparing low-res, output, and high-res images side-by-side.

Extendable

- Easily tweakable codebase — new losses, datasets, and UI components can be integrated with minimal changes.

Description

This project implements an advanced super-resolution pipeline based on Enhanced Deep Residual Network (EDSR) to train game screenshots based off their low- and high-resolution versions. The goal is to upscale low-resolution game images by a factor of 2 while preserving details, edges, and textures. The project is inspired by DLSS, and it includes carefully selected model components, training procedures, and loss functions to ensure visually high-quality results.

The preprocessing and dataset include a custom script (generate_pairs.py) that crops original game screenshots to a fixed size and creates a low-resolution pair using bicubic downscaling. During training the images go through synchronized augmentations such as random flipping, rotation and affine transforms using torchvision.transforms. This diversifies the training set and improves generalization. Lastly, they are converted to PyTorch tensors.

The core of this project is the EDSR model, a convolutional neural network designed to upscale low-resolution images while preserving detail. For each image in the dataset, the head module is used as a feature extraction where the model takes the 3-channel RGB image and applies a single convolution layer. This layer increases the number of feature maps so that the model can begin learning more abstract patterns in the image. Next the body module consists of 16 residual blocks, each consisting of two convolution layers, one ReLU, and a skip connection that adds the block's input back to its output. A residual block lets the model learn changes instead of absolute values. After the residual blocks, the original input features are added back to the output of the body. This helps the model retain the original low-level details and ensures it doesn't over-smooth or lose information during learning. Finally, the model takes the refined feature maps and turns them back into a high-resolution image using a convolution layer, nn.PixelShuffle, and a final convolution layer to convert the output into an RGB image.

A combination of four complementary losses is used to improve the trained model. Pixel Loss ensures the overall color and pixel-level fidelity. SSIM Loss captures perceptual similarity based on local luminance, contrast, and structure. It helps preserve visual sharpness and edges. Perceptual Loss uses VGG16 feature maps to compare high-level content and lastly Gradient Loss enforces similarity in image gradients to preserve texture.

The training process was straight forward. A dataset of 400 paired images, each sized at 576×576 pixels, was used for supervised learning. The model was trained over 350 epochs with a batch size of 4. After training, the test_sr_model.py script enables precise evaluation by allowing users to modify an index to visualize individual test results, showing the low-resolution input, the original image, and the model's prediction. For broader accessibility, a demo.py script provides a user-friendly web interface where users can upload any low-resolution screenshot and instantly view the enhanced output displayed side-by-side.

Block Diagram

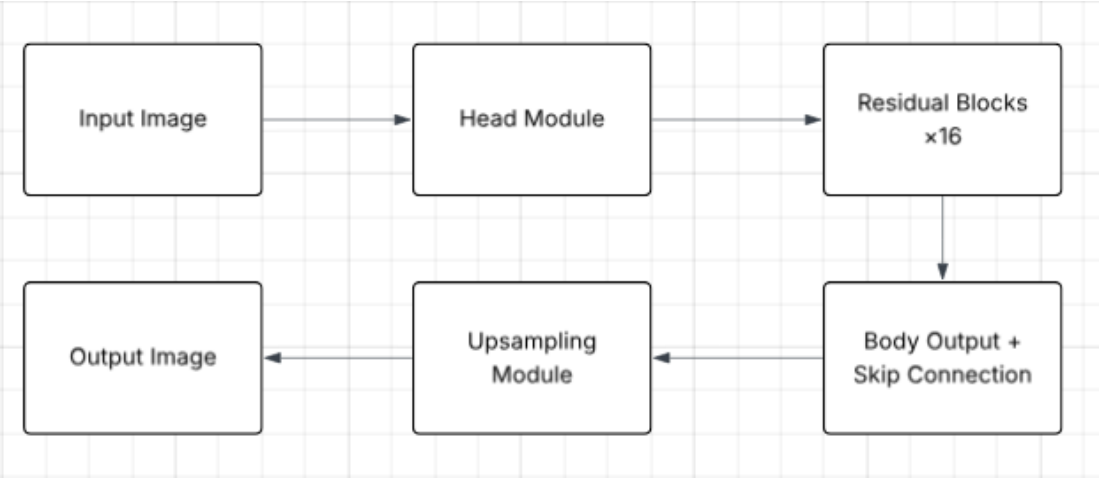

Figure 1: EDSR Model Architecture

Challenges & Iterations

Early in the project, the model outputs looked overly smooth and lacked fine details, especially in game textures like armor edges, grass, or UI text. I initially only tried pixel-wise losses like MSE and L1. After noticing the lack of sharpness, I iteratively added SSIM Loss, Perpetual loss and Gradient loss.

A large crop size like 576×576 helped with context of the entire screenshot for the model, however it made training very slow. It took almost 2 full days for 350 epochs. Smaller sizes trained faster but lost details. So, what I did was experiment with different crop sizes and reviewed output sharpness. Eventually settling with 432x432 which had a balance of training time and visual detail preservation.

At the very beginning of the project, I experimented with multiple architectures like SRCNN and FSRCNN. Despite their faster training, the output quality was poor and lacked definition. After testing I switched to EDSR, which allowed deeper residual blocks and produced noticeably better textures.

The initial model was overfitting as it had limited training diversity. To fix this I added data augmentation techniques such as random flips, rotations, color jitter, and affine transformations to the input and target images. This improved the model's ability to generalize to unseen screenshots and produced better outputs.

Results

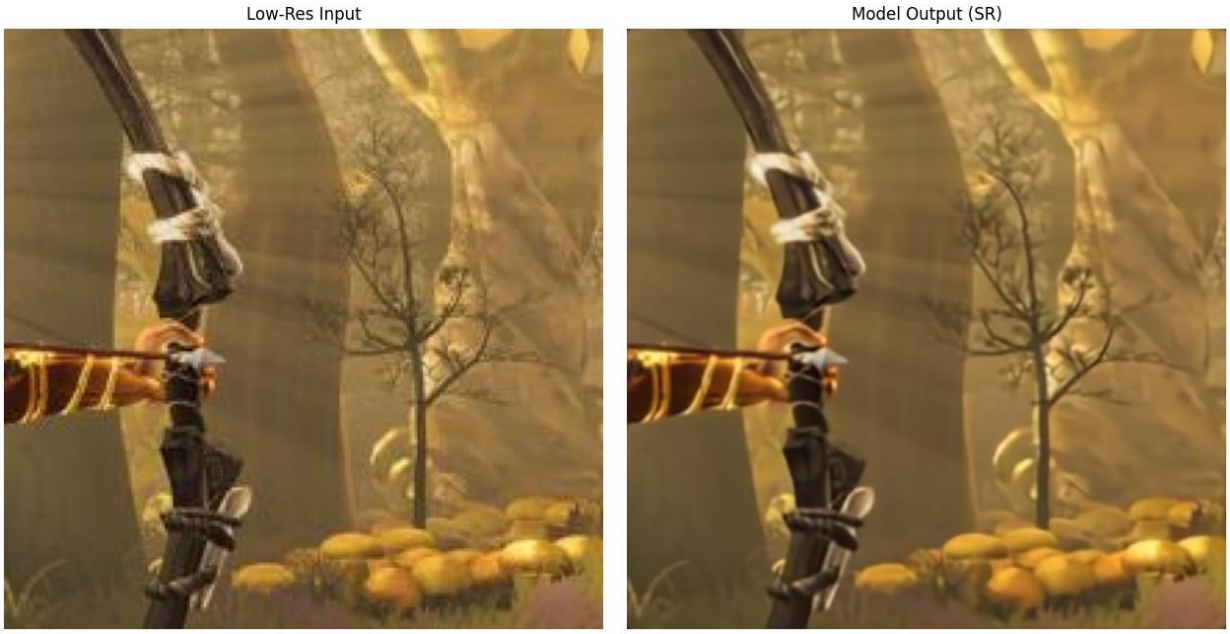

Figure 2: Result

Figure 3: Result

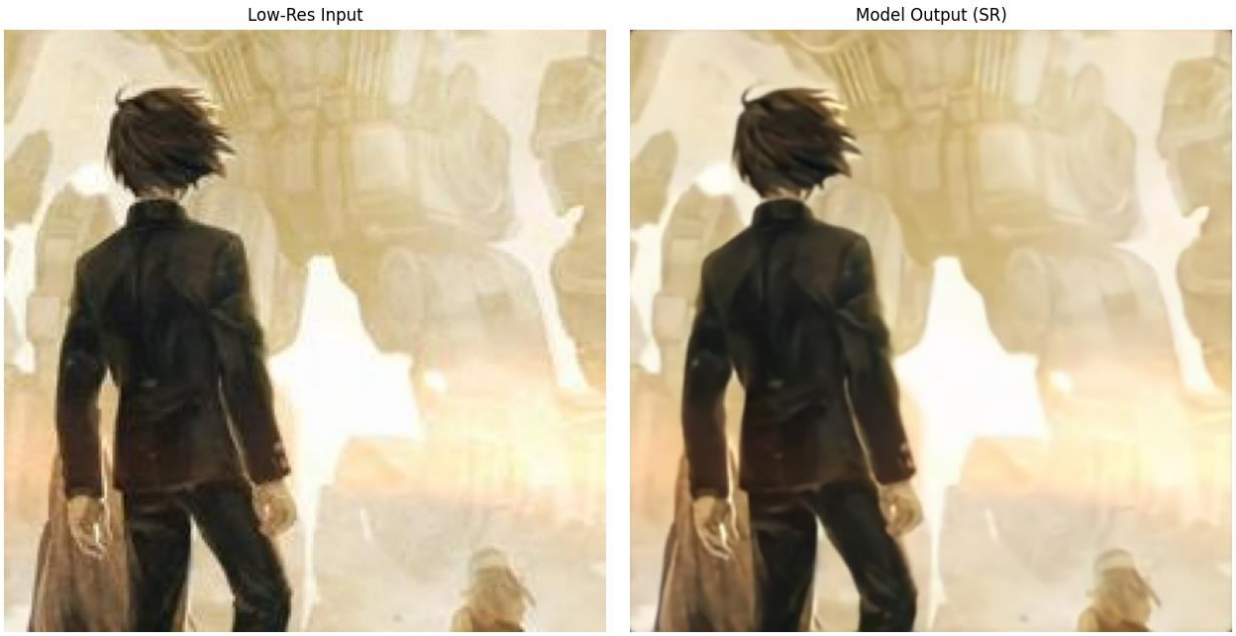

Figure 3: Result

Figure 4: Result

Instructions

See INSTRUCTIONS_TO_RUN.md

What I Learned

- **Deep Learning with EDSR:** I implemented and trained the Enhanced Deep Super-Resolution (EDSR) model in PyTorch, learning how residual blocks, up sampling, and a combination of loss functions impact image quality.
- **GPU Acceleration with CUDA:** I trained the model using a CUDA-enabled GPU, optimizing batch size, crop resolution, and training duration to fit compute constraints while improving runtime efficiency.
- **Debugging & Iterative Tuning:** I tackled over smoothing, blurry textures, and model instability by iteratively tuning losses, architecture, and evaluation techniques to improve perceptual quality and output sharpness.
- **Interface Development:** I built a Gradio-based UI that allows real-time super-resolution of user-submitted game screenshots, showcasing the model in an accessible and interactive way.

References

[1] B. Lim, S. Son, H. Kim, S. Nah, and K. M. Lee, “Enhanced Deep Residual Networks for Single Image Super-Resolution,” *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, 2017. [Online]. Available: https://arxiv.org/abs/1707.02921

[2] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, “Image Quality Assessment: From Error Visibility to Structural Similarity,” *IEEE Trans. Image Process.*, vol. 13, no. 4, pp. 600–612, Apr. 2004. [Online]. Available: https://ieeexplore.ieee.org/document/1284395

[3] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, “The Unreasonable Effectiveness of Deep Features as a Perceptual Metric,” *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2018. [Online]. Available: https://arxiv.org/abs/1801.03924

[4] PyTorch, “PyTorch Documentation,” 2023. [Online]. Available: https://pytorch.org/docs/stable/index.html

[5] TorchVision, “TorchVision Pretrained Models,” 2023. [Online]. Available: https://pytorch.org/vision/stable/models.html

[6] Gradio, “Gradio: Build Machine Learning Web Apps,” 2023. [Online]. Available: https://www.gradio.app/

[7] VainF, “pytorch-msssim: Structural Similarity (SSIM) Loss for PyTorch,” GitHub repository, 2023. [Online]. Available: https://github.com/VainF/pytorch-msssim

[8] OpenAI, “ChatGPT,” OpenAI, San Francisco, CA, USA, 2023. Used throughout project development for coding support.

[9] Pillow Developers, “Pillow (PIL Fork) Documentation,” 2023. [Online]. Available: https://pillow.readthedocs.io/en/stable/