CSE 676Deep Learning Final Project Report

By - Shubham Soni(50593888), Hazel Mahajan(50592568), Piyush Modi(50606751)

Deep Image Refinement Using GANS

Introduction

Deep image refinement is the task of enhancing the quality of low-resolution or degraded images using deep learning techniques. In this project, we focus on using GANs to reconstruct high-quality (HQ) images from low-quality (LQ) inputs. This is especially relevant in fields such as super-resolution, denoising, and photo restoration.

This project aims to enhance degraded or low-resolution images using deep learning techniques. The goal is to reconstruct high-quality images by addressing noise and artifacts. The significance of this research lies in its broad applications, including medical imaging (MRI/X-ray enhancement), satellite image processing, old photo restoration, and improving security footage. By leveraging advanced neural networks, this project will improve image clarity and quality beyond traditional interpolation methods.

Phase-wise workflow

Phase 1: Baseline Exploration

- **Objective**: Understand foundational super-resolution techniques and establish performance baselines.
- **Approaches Used**:
  - **Autoencoders**: Implemented to learn direct image reconstruction from low-resolution to high-resolution. Results were limited, with outputs often blurry and lacking fine detail.
  - **Vanilla GAN**: Explored the use of a simple GAN for super-resolution. While it improved visual results slightly over autoencoders, it struggled with texture and sharpness.
- **Outcome**: Established that basic models were insufficient for high-fidelity image enhancement, motivating the need for advanced architectures.

Phase 2: Enhanced Training & Realism

- **Objective**: Improve model performance and perceptual quality using advanced GANs and data diversity.
- **Steps Followed**:
  - **Data Augmentation**: Expanded the dataset from 2560 to ~4000 images using geometric and photometric transformations (rotation, flip, zoom, etc.) to improve generalization.
  - **SRGAN Implementation**: Introduced perceptual and adversarial losses for better structure-aware generation. Achieved improvement over Phase 1, but generated images were still blurry and artifact-prone.
  - **ESRGAN Implementation**: Replaced SRGAN with ESRGAN, integrating Residual-in-Residual Dense Blocks (RRDB), removed batch normalization, and used a relativistic discriminator. This led to significantly sharper, more detailed, and perceptually realistic high-resolution images.

Dataset

We used the **Flickr 2K** dataset, which contains 2,560 high-resolution images scraped from Flickr.
Dataset Link: Kaggle - Flickr2K

Data Preprocessing & Augmentation

The training dataset was prepared by generating **Low-Quality (LQ)** and **High-Quality (HQ)** image pairs from the **Flickr 2K dataset**. These pairs serve as input-output examples for training a deep image refinement model, such as an autoencoder.

Preprocessing:

- We generated pairs of HQ and LQ images.
- LQ images were created using downsampling (e.g., bicubic interpolation or Gaussian blur + resizing).
- Data was split into training and validation sets for supervised learning

Post data preprocessing insights:

- High resolution images resized to 640 x 640 x 3
- Low resolution image resized to 160 x 160 x 3
- Images to normalized to mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]

Phase 1

Model Architecture

We implemented a **convolutional autoencoder** for learning image refinement. The autoencoder consists of two main parts:

- **Encoder**: Compresses the input LQ image into a low-dimensional representation
- **Decoder**: Reconstructs an HQ version from this latent vector

Model Overview

Autoencoder Training Framework

This implementation focuses on training an **autoencoder neural network** for image reconstruction using PyTorch. The primary goal is to compress and reconstruct images, minimizing the reconstruction loss using the Mean Squared Error (MSE) criterion

Auto encoder consists of 2 parts:

- Encoder inputs the 160 x 160 x 3 image and then extracts features while also downsampling across height and width
- Decoder accepts the output from the encoder and then upsamples it using bicubic transformation and scaling factor 4
- This results in an output image of dimension 640 x 640 x 3

Model Architecture

- The model, Autoencoder, is initialized based on the shape of the input data.

- It is assumed to follow a typical encoder-decoder structure, although the architecture is not shown in the script.
- The encoder consists of 5 convolutional layers and 2 max pooling layers which are going to extract features for the decoder.
- The decoder consists of 5 convolutional layers and 2 upsampling layers with a scaling factor of 4 which extrapolate the pixel using bicubic algorithms.

Loss Function

- **Mean Squared Error (MSE)** is used to measure reconstruction quality between input images and their reconstructions.

Optimizer

- **AdamW optimizer** is used for parameter updates, chosen for its better generalization over Adam due to decoupled weight decay.
- Learning rate is set to 0.0001.

Training & Validation Strategy

Train Loop

- Batches of images and their labels are loaded from train_loader.
- Labels are decremented by 1, likely due to label encoding from 1-based indexing.
- Model outputs are compared to ground truth using the MSE loss.
- The model is updated via backpropagation and optimiser steps.

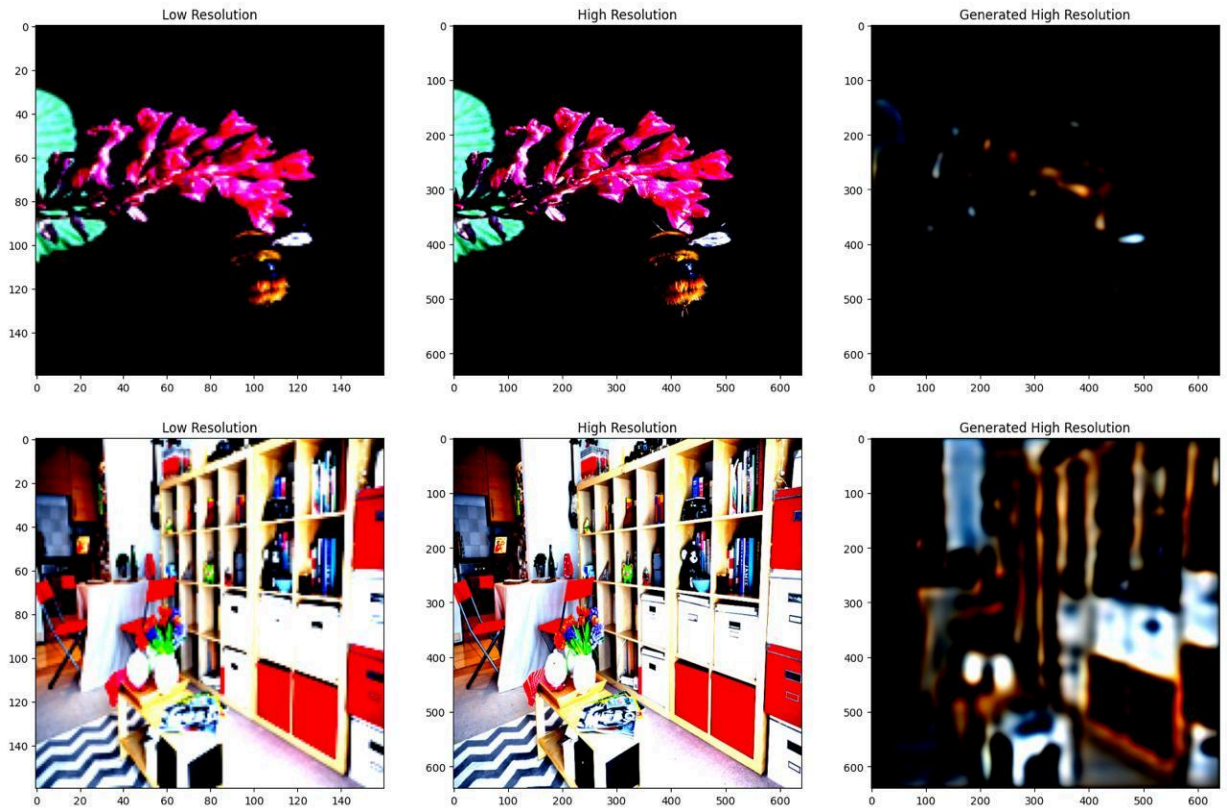
Validation Loop

- The model is evaluated without gradient computation to measure validation loss.
- Performance is logged after each epoch.

Training Setup

- **Loss Function**: Mean Squared Error (MSE) – to measure pixel-wise difference between predicted and ground truth images.
- **Optimizer**: AdamW with learning rate 0.0001
- **Hardware**: Trained on GPU if available.
- **Epochs**: 5

Results of Autoencoder



GAN Implementation

High-Level Summary

This is a simple **Generative Adversarial Network (GAN)** built using **PyTorch**.
It consists of:

1. A **Generator** — to create fake high-resolution (HR) images from low-resolution (LR) inputs.

2. A **Discriminator** — to classify real HR images vs. generated (fake) images.

The goal is to train the generator to fool the discriminator while the discriminator tries to distinguish real from fake images.

Components Breakdown

1. Generator

- The generator **upsamples** images from LR to HR using:
  - Convolutional layers.
  - PixelShuffle layers to increase resolution.
  - Non-linearities: PReLU, and final activation with Tanh.

  Purpose: Convert low-res images to high-res fakes.

2. Discriminator

- A CNN-based classifier that:
  - Takes an image and outputs the probability of being "real" or "fake".
  - Uses LeakyReLU, BatchNorm, AdaptiveAvgPool2d, and Sigmoid at the end.

Purpose: Distinguish between real HR images and generated fakes

Training Setup

- **Loss Function**: nn.BCELoss() (Binary Cross-Entropy) for both generator and discriminator.
- **Optimizers**: Adam optimizer with learning rate 1e-4.
- **Device**: Auto-detects GPU if available:

Training Loop

For each epoch:

1. **Train Discriminator:**
    - Classifies real images as 1, fakes as 0
    - Backpropagates combined loss from real and fake predictions
2. **Train Generator:**
    - Generates fake images
    - Tries to **fool the discriminator** by making discriminator output 1 for fakes
    - Backpropagates generator loss
3. **Validation Loop:**
    - Similar to training but with no gradients (torch.no_grad())
4. **Logging:**
    - Prints training and validation losses after every epoch

Results of the Implemented GAN

Why Autoencoders are not successful in generating high-resolution images

Autoencoders, while effective for tasks like denoising or dimensionality reduction, often struggle to generate high-resolution images due to their inherent architectural and loss function limitations. They work by compressing input data into a lower-dimensional latent space and then reconstructing it, typically optimizing a pixel-wise loss such as Mean Squared Error (MSE). This type of loss prioritizes overall pixel accuracy but fails to capture perceptual nuances, resulting in blurry or overly smooth outputs—especially evident when reconstructing high-resolution images that require fine textures and sharp edges. Moreover, the bottleneck in the architecture often leads to a loss of high-frequency information critical for photo-realism. Unlike GANs, autoencoders do not involve a discriminator to push the generator towards producing images indistinguishable from real samples, which further limits their capability in learning intricate, realistic details. This makes them unsuitable for tasks like super-resolution, where visual fidelity and fine detail reconstruction are crucial.

Data Augmentation Overview

1. Dataset Setup

- The script reads all .png images from the **Flickr 2K dataset** directory.
- The image paths are collected using glob.glob and sorted to ensure consistency.

2. Output Directory Structure

- Two output subdirectories are created under flickr2k_final_blur/:
  - HQ: Stores original high-quality images.
  - LQ: Stores synthetically degraded low-quality images.

- os.makedirs(..., exist_ok=True) ensures these folders are created without errors if they already exist.

Low-Quality Image Generation

Each HQ image undergoes two degradations to simulate real-world noise and artifacts:

1. Downscaling

- The image is resized to **1/4 of its original resolution** using bicubic interpolation.
- This simulates a loss of detail, common in compressed or low-resolution images.

2. Gaussian Blur

- A **Gaussian blur** with a radius of 1.5 is applied using ImageFilter.GaussianBlur
    - This adds realistic blur noise that models sensor imperfections or motion blur.

Image Pair Creation & Saving

- Each HQ–LQ image pair is saved with a matching filename (00000.png, 00001.png, ...)
- LQ and HQ images are stored in their respective directories for later use in model training
- A try-except block ensures robustness against corrupted or unreadable files

Script Parameters

Parameter	Value	Description
downscale_factor	**4**	Reduction factor for image resolution

blur_radius	**1.5**	Strength of gaussian blur
resample	**Image.BICUBIC**	High-quality interpolation during resizing

Progress Logging

- The script uses tqdm to display progress while generating image pairs.
- Informational and success messages provide clarity on processing status.

Output structure segregation post augmentation

/content/flickr2k_final_blur/

```
├── HQ/
│   ├── 00000.png
│   ├── 00001.png
│   └── ...
└── LQ/
    ├── 00000.png
    ├── 00001.png
    └── …
```

Next Steps

Now that we have successfully generated an augmented dataset consisting of 6,000 high-quality (HQ) and corresponding low-quality (LQ) image pairs, the next step is to utilize this data for training a Generative Adversarial Network (GAN).

The GAN will be trained to perform image refinement, where the generator learns to enhance or restore LQ inputs to closely match their HQ counterparts. This setup enables the model to learn complex mappings between degraded and clean images, leveraging the rich variety introduced during augmentation. By feeding the LQ images into the GAN and using the HQ images as ground truth, we aim to improve the model's ability to recover fine details and textures in real-world noisy or compressed visual data

Phase 2

Data Augmentation - generation of 4000 images

To enhance the training dataset and improve the robustness of the Super-Resolution GAN (SRGAN) model, we applied extensive data augmentation techniques. The original dataset consisted of 2560 high-resolution images, which were augmented to generate a larger, more diverse dataset of approximately **4000 images**. This expansion aimed to prevent overfitting, improve generalisation, and expose the model to a wider range of image transformations.

Augmentation Techniques Applied:

The augmentation process involved a combination of geometric and photometric transformations using ImageDataGenerator from Keras:

1. **Rotation (rotation_range=90)**
   Images were randomly rotated within a 90-degree range, helping the model learn orientation-invariant features.

2. **Width and Height Shifts (width_shift_range=0.2, height_shift_range=0.2)**
   Horizontal and vertical translations were applied, enabling the model to focus less on absolute object positions and more on textures and

patterns.

3. **Shearing (shear_range=0.2)**
   Applied a shear transformation to the image, which distorts the image along an axis, aiding in modeling geometric variations.

4. **Zooming (zoom_range=0.2)**
   Slight zoom in/out was applied to simulate varying camera distances and resolutions, helping the model recognize fine details at different scales.

5. **Horizontal Flipping (horizontal_flip=True)**
   Flipping images horizontally added mirror variants, useful for recognizing symmetrical features.

6. **Fill Mode (fill_mode='nearest')**
   When image pixels were shifted or rotated, empty areas were filled using the nearest pixel values, preserving continuity.

Augmentation Implementation:

```
datagen = ImageDataGenerator(
    rotation_range=90,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
```

Each image from the original set was augmented multiple times (controlled using a for loop with a sample cap), and the augmented images were saved into a new directory using .flow() generator combined with save_to_dir.

The augmentation loop for each image was:

```
i = 0
for batch in datagen.flow(x, batch_size=1, save_to_dir='path',
save_prefix='aug', save_format='jpg'):
    i += 1
    if i > 3:  # Generate 3 augmentations per image
        break
```

This setup ensured that for each original image, **three augmented variants** were generated. Hence, with ~2560 original images, the final dataset size increased to approximately **4000 total images**, combining original and augmented versions.

Purpose and Impact:

- Expanded data coverage without collecting new images.
- Made the model more resilient to scale, orientation, and positional variance.
- Helped improve the quality and realism of super-resolved outputs by training the generator on diverse image variants.

Overview - SRGAN and ESRGAN

SRGAN (Super-Resolution Generative Adversarial Network)

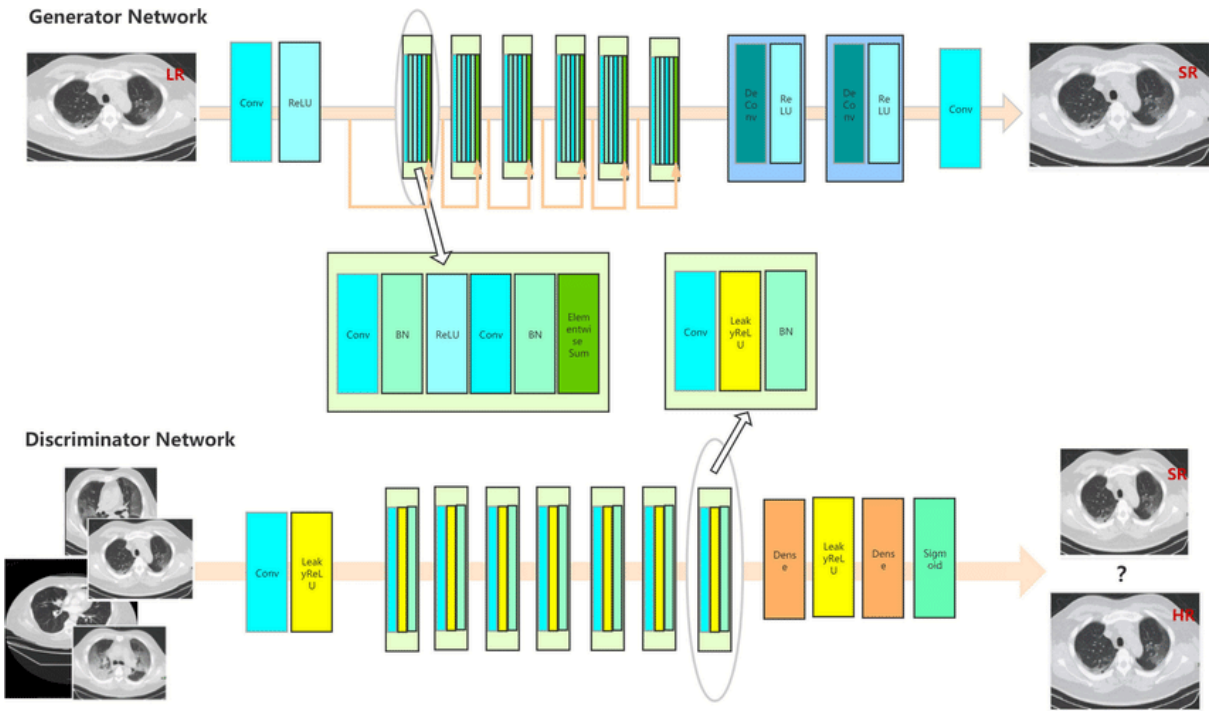
SRGAN is a deep learning model introduced by Ledig et al. in 2017 to generate high-resolution (HR) images from low-resolution (LR) inputs. It is the **first GAN-based approach for image super-resolution** that focuses not just on pixel-level accuracy but also on **perceptual quality**—i.e., how realistic and detailed the images look to humans.

Key Components:

- **Generator**: A deep convolutional neural network with residual blocks and upsampling layers (e.g., pixel shuffle) that takes LR images and generates HR images.
- **Discriminator**: A binary classifier CNN that tries to distinguish between real high-res images and the fake ones generated by the Generator.
- **Perceptual Loss**: Instead of only minimising pixel-wise loss (like MSE), SRGAN uses a **perceptual loss** composed of:
    - **Content loss**: Based on VGG feature space differences between generated and real images.
    - **Adversarial loss**: From the GAN framework to encourage photo-realistic textures.

Strengths:

- Produces more visually pleasing images with sharp textures compared to traditional interpolation or MSE-only models.
- Introduced the idea of **perceptual learning** in super-resolution.



ESRGAN (Enhanced Super-Resolution GAN)

ESRGAN, introduced by Wang et al. in 2018, is an improved version of SRGAN that significantly enhances image quality, especially around edges and textures. It was the winner of the **PIRM 2018 Super-Resolution Challenge**.

Enhancements Over SRGAN:

1. **Residual-in-Residual Dense Blocks (RRDB)**:
    - Instead of basic residual blocks, ESRGAN uses RRDBS, which combine **residual**, **dense**, and **skip connections**, enabling deeper feature extraction and better gradient flow.

2. **Relativistic Discriminator**:
    - Unlike SRGAN's binary discriminator, ESRGAN uses a **relativistic average GAN (RaGAN)** that predicts how real a generated image is **relative to** real images. This stabilises

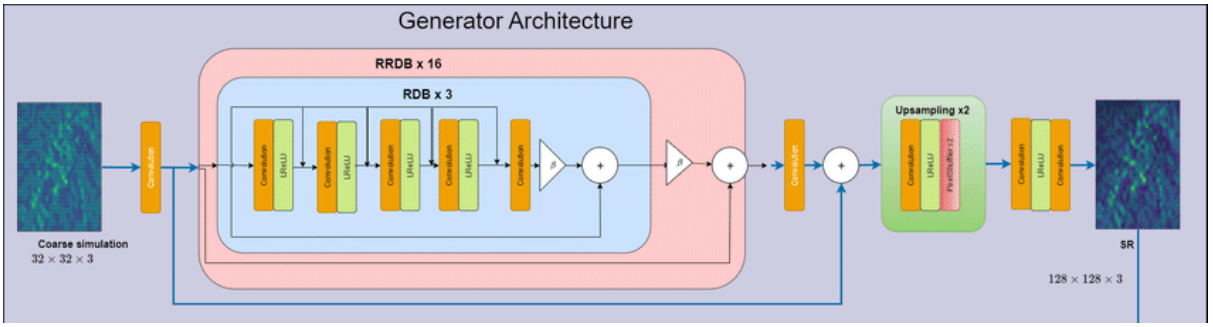
training and improves texture quality.

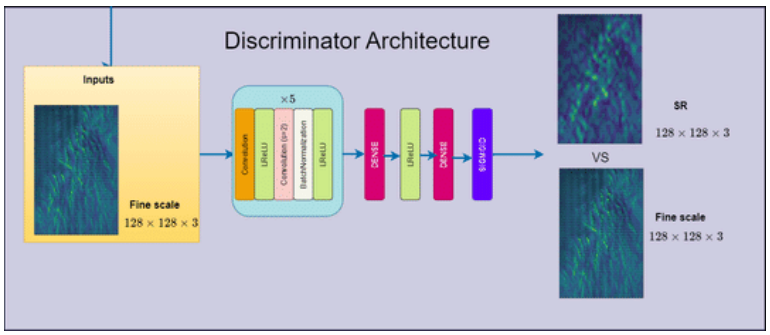
3. **Improved Perceptual Loss**:
    - ESRGAN computes the content loss using **features before activation** in the VGG network, capturing more detailed information.

4. **No Batch Normalisation**:
    - Removed to avoid artifacts and increase image sharpness.

Benefits:

- Produces **sharper, more realistic images**, especially effective in enhancing fine details like hair, grass, and textures.
- Addresses some of SRGAN's blurriness and over-smoothing issues.





Implemented Architectures

SRGAN Model Architecture

The SRGAN model implemented for this project comprises two primary components: the **Generator** and the **Discriminator**. These were designed to work adversarially, where the generator aims to produce realistic high-resolution images from low-resolution inputs, and the discriminator aims to distinguish between real and generated images.

1. Generator Architecture

The Generator network is responsible for converting low-resolution (LR) images into high-resolution (HR) images through a deep residual convolutional framework and upsampling layers. Below is a breakdown of the implemented architecture:

- **Input Layer**:
    - Takes a 3-channel RGB image.
    - First convolutional layer uses 64 filters with a kernel size of 9x9 and PReLU activation to extract low-level features.
- **Residual Blocks** (6 blocks total):
    - Each block consists of two convolutional layers (3x3), followed by Batch Normalization and PReLU activation.
    - These help in learning identity mappings and preserving spatial features.
- **Skip Connection**:
    - A skip connection adds the output of the first convolutional layer to the output of the last residual block to retain low-frequency information.

- **Upsampling Blocks**:
    - The number of upsampling blocks is determined by the scale factor (e.g., 2x, 4x).
    - Each block uses a PixelShuffle (in UpsampleBlock) to double the spatial resolution.
    - Followed by a convolution with 64 filters and PReLU activation.
    - Final convolution uses 3 filters (for RGB output) with a large kernel (9x9) to reconstruct the HR image.

Forward Flow Summary:

Input → Conv(9x9) + PReLU → 6x Residual Blocks → Skip Add → Conv + BN → Upsample Blocks → Final Conv → Output HR image

2. Discriminator Architecture

The Discriminator is a deep CNN that classifies images as either real (from dataset) or fake (from Generator). It uses progressively deeper convolutional layers to extract hierarchical features, downsampling the image and eventually reducing it to a classification score.

- **Conv Layers with LeakyReLU**:
    - Alternating layers of Conv2D (3x3 kernels), Batch Normalization, and LeakyReLU (slope=0.2) progressively double the feature channels from 64 to 512.
    - Stride of 2 is applied every alternate layer to downsample the image.

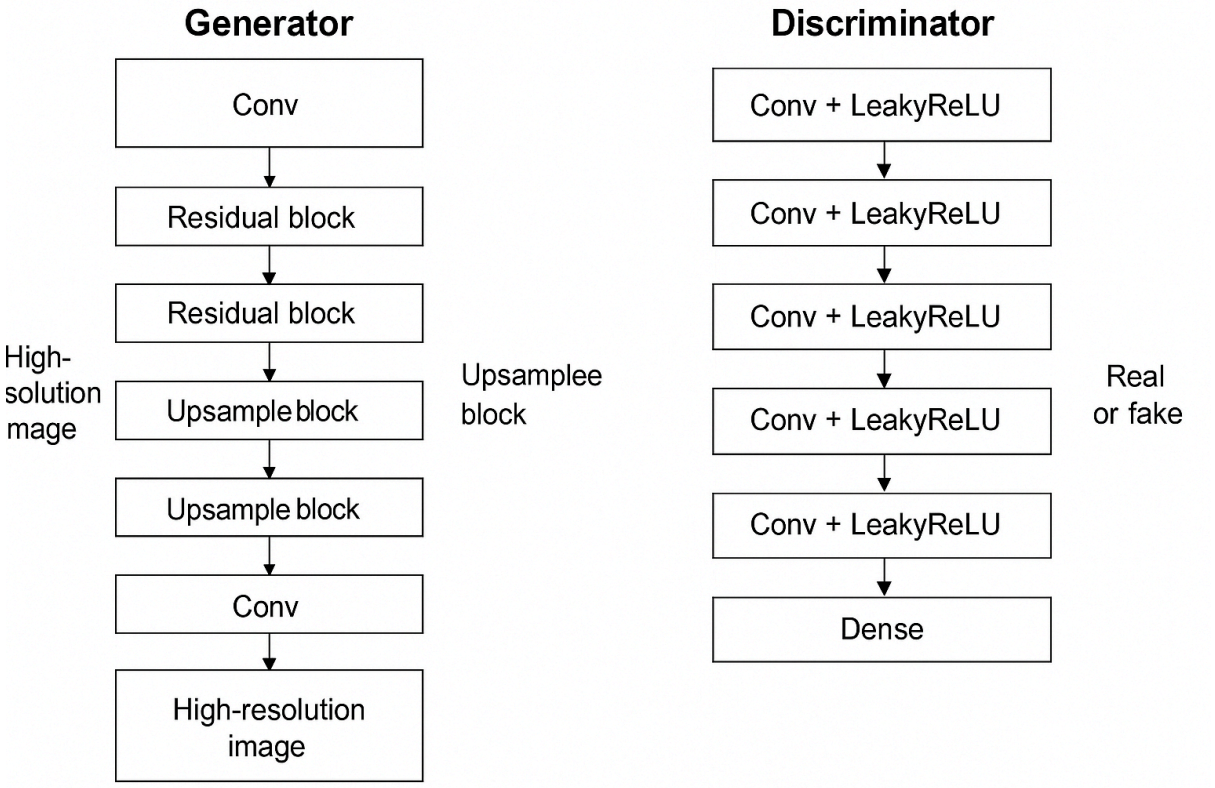
- **Fully Connected Layers**:
    - After global average pooling (AdaptiveAvgPool2d(1)), the image is passed through dense layers.
    - Ends with a 1x1 convolution that outputs the final classification score indicating real or fake.

Forward Flow Summary:

Input → [Conv + BN + LeakyReLU] x 8 → AdaptiveAvgPool → Dense Layers → Real/Fake Score

Training Overview

This SRGAN model was trained on a dataset of approximately **4000 images** (including original and augmented data). The Generator learns to create perceptually convincing images, while the Discriminator ensures those images resemble real HR samples. Together, they form an adversarial loop where both networks continuously improve.



ESRGAN Model Architecture

The **Enhanced Super-Resolution Generative Adversarial Network (ESRGAN)** builds upon SRGAN by addressing its limitations in texture reconstruction and visual sharpness. Your implementation closely follows the original ESRGAN architecture with custom-defined **Residual-in-Residual Dense Blocks (RRDBs)** in the Generator and a **strided convolutional Discriminator** for stability and texture fidelity

1. Generator Architecture (ESRGAN-Inspired)

The Generator is designed to extract hierarchical features from low-resolution inputs and reconstruct high-resolution images using a deeper network with advanced residual structures

Key Components:

- **Initial Convolution Layer**:
  - Conv2d with 3×3 kernel, stride=1, padding=1 to capture initial low-level features from the LR image.

- **RRDB Blocks (Residual-in-Residual Dense Blocks)**:
  - 23 stacked RRDBs enhance the representational capacity.
  - Each RRDB contains multiple convolutional layers with dense connections and residual scaling, allowing efficient feature reuse and gradient flow.

- **Conv Layer after RRDBs**:
  - A 3×3 convolution refines the features learned by RRDBs and adds the initial input feature map (skip connection).

- **Upsampling Blocks**:
  - Two Upsample layers to progressively upscale the feature maps to the target resolution.

    - Typically implemented using nearest-neighbor interpolation followed by a convolution + activation (e.g., LeakyReLU).

- **Final Layers**:
    - Final Conv2D with 3×3 kernel and LeakyReLU to refine textures.
    - Final Conv2D layer maps feature maps to 3-channel RGB image.

Forward Pass: Input → Conv → RRDB Blocks (x23) → Conv + Residual Add → Upsample (x2) → Final Conv Layers → Output HR Image

2. Discriminator Architecture- The Discriminator classifies images as real (from dataset) or fake (from Generator) and consists of a series of strided convolutions that progressively downsample the input while increasing the depth of feature maps.

Key Components:

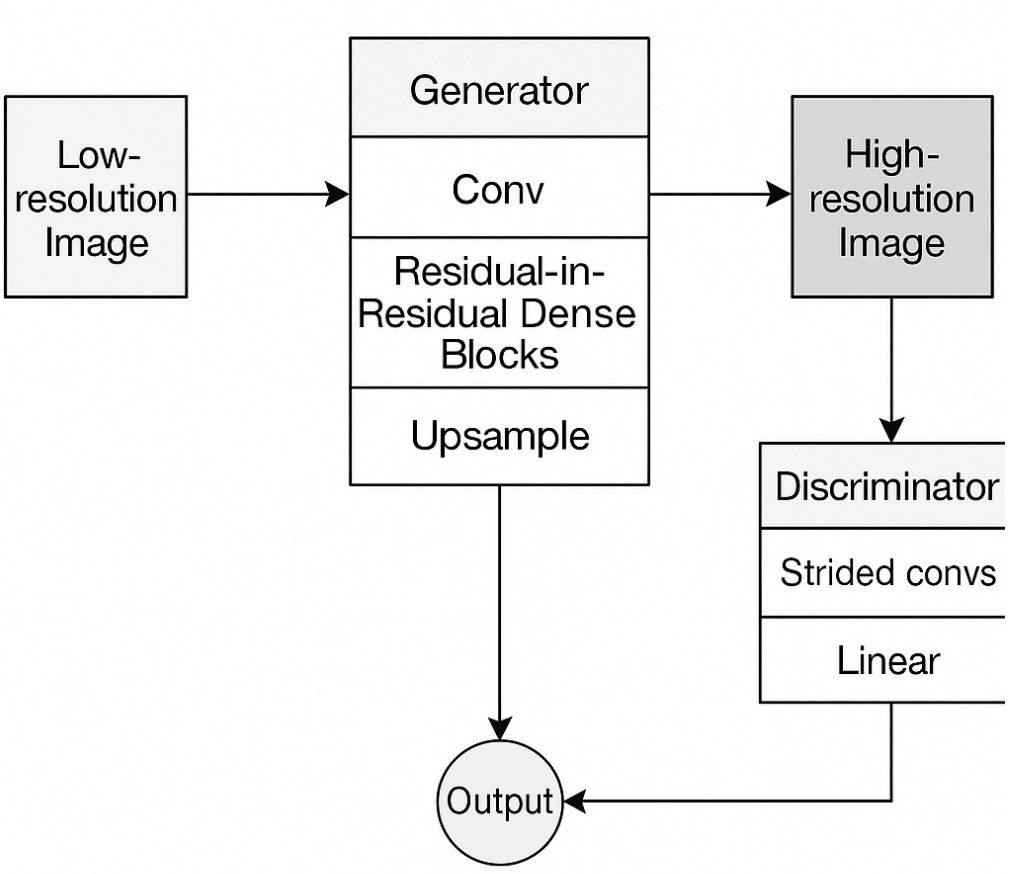
- **Convolutional Layers**:
    - A total of 8 convolutional blocks with increasing feature depths: [64, 64, 128, 128, 256, 256, 512, 512].
    - Alternate layers apply stride=2 to reduce spatial resolution, effectively increasing the receptive field.
    - LeakyReLU is used for non-linearity, and batch normalization or activation is optionally included based on use_act=True.

- **Classifier**:
    - After downsampling, a global AdaptiveAvgPool2d is applied, followed by:
        - Flatten → Dense (1024 units) + LeakyReLU → Dense (1 unit output)
    - Final output is passed through a sigmoid function for binary classification.

Forward Pass: Input → [Conv Blocks with Stride] → AdaptiveAvgPool → Flatten → Dense Layers → Sigmoid → Real/Fake

Highlights of Your Implementation

- Uses **23 RRDB blocks**, matching the official ESRGAN design.
- Includes **strided convolutions** in the discriminator, improving texture recognition.
- Upsampling is handled through modular Upsample() layers.
- Uses **LeakyReLU** activations with inplace=True for memory efficiency.
- Well-modularized for ease of expansion (e.g., for future use with perceptual/VGG loss).



ESRGAN Model Architecture

Training Pipeline

SRGAN

The training of the Super-Resolution Generative Adversarial Network (SRGAN) follows a **two-network adversarial learning setup**: the Generator (netG) learns to upsample low-resolution images to high-resolution ones, while the Discriminator (netD) learns to differentiate between real high-resolution images and generated ones. The training aims to optimize perceptual quality, not just pixel-wise accuracy.

Training Configuration

- **Epochs**: 150
- **Early Stopping Patience**: 4 rounds without improvement in validation loss
- **Training Dataset**: ~4000 images (original + augmented)
- **Framework**: PyTorch
- **Device**: CUDA GPU-enabled

Model Objectives

- **Generator (netG)**:
  - Minimize a combination of **content loss** (between generated and real HR image features) and **adversarial loss** (from the discriminator's feedback).
  - Objective: produce realistic high-res images from low-res inputs.

- **Discriminator (netD)**:

  - Distinguish between generated and real high-res images.

- Objective: assign higher scores to real images and lower scores to fake/generated ones.

Loss Functions Used

- **Generator Loss (g_loss)**:
  - A **custom loss function** combining:
    - **Adversarial loss**: Encourage realism via the discriminator's judgment.
    - **Content loss**: Typically computed using VGG-based perceptual loss to ensure semantic similarity.

- **Discriminator Loss (d_loss)**:
  - 1 - D(real) + D(fake): Penalizes the discriminator for being overly confident on fake images or not confident enough on real ones.

Training Phase (per batch):

1. **Forward Pass**:
   - Low-resolution image → Generator → Super-resolved output
   - Discriminator scores: D(real) and D(fake)
2. **Discriminator Update**:
   - d_loss is computed and backpropagated.
   - optimizerD.step() updates the discriminator weights.
3. **Generator Update**:
   - The generator is re-run to generate fake_img.
   - g_loss is computed using the adversarial and content losses.
   - optimizerG.step() updates the generator weights.

4. **Metrics Computed per Batch**:
   - **PSNR** (Peak Signal-to-Noise Ratio)
   - **SSIM** (Structural Similarity Index Measure)
   - Discriminator and Generator scores

Validation Phase:

- No gradients are computed.
- The generator is evaluated on a separate validation set.
- Same metrics (loss, PSNR, SSIM) are computed and stored.

Metric Tracking & Logging

Throughout training, the following values are logged each epoch for both training and validation:

- g_loss, d_loss
- PSNR, SSIM
- Discriminator Score (D(x) and D(G(z)))

Results are stored in a centralized results dictionary, enabling later plotting and analysis.

Checkpointing and Early Stopping

- The best generator model (best_G.pt) is saved when validation score improves.
- **Early stopping** is triggered if no improvement in generator validation loss for 4 consecutive rounds.
- The latest model (last_G.pt) is also saved at the end of each epoch.

Outcome of SRGAN Training

SRGAN training yielded noticeable improvements in visual fidelity over standard upsampling methods. However, certain fine details and edge sharpness were still lacking compared to ESRGAN, which addressed these shortcomings through architectural enhance

SRGAN Results & Insights

The SRGAN model, after training on the augmented 4000-image dataset, demonstrated its ability to upscale low-resolution images and recover general structures and textures. As shown in the visual comparison, the **Generated High-Resolution output** successfully captures the overall shape and placement of objects (e.g., the bee and flower), validating the impact of perceptual and adversarial losses used during training.

However, despite these strengths, the output lacks fine-grained details, sharpness, and natural color transitions when compared to the ground-truth high-resolution image. The generated result shows **muted colors, blotchy textures, and visible artifacts**, especially around edges and intricate regions, which are critical for high-quality image synthesis.

Training Metrics Analysis

The training and validation plots support this qualitative observation:

- **Discriminator & Generator Losses** (Top Row):
  - Show steady convergence, indicating that adversarial training remained stable.
  - Both training and validation losses plateaued without major divergence, suggesting no overfitting.

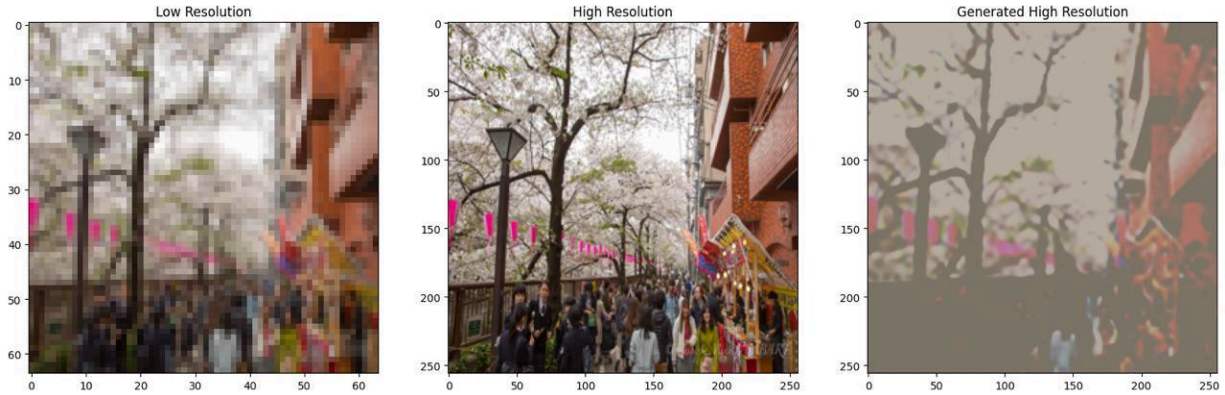
- **Discriminator and Generator Scores**:
  - Generator scores (D(G(z))) remained low, indicating the discriminator could easily distinguish fakes.
  - This aligns with the lack of realism seen in the generated images.

- **PSNR and SSIM Trends**:
  - PSNR and SSIM improved over epochs, showing that the model learned to produce pixel-wise better images.

- However, their relatively low absolute values reflect **limited perceptual quality** in fine details.

- **Combined Loss Trends**:
  - Overlaying G and D losses show that SRGAN training was stable, but the loss alone was insufficient for high-fidelity generation.

Output of SRGAN



Transition to ESRGAN

While SRGAN laid the groundwork for learning structure-aware image super-resolution, the results confirm that it **struggles with preserving high-frequency details**, resulting in **blurry and artifact-prone outputs**. These shortcomings stem from SRGAN's reliance on **shallow residual blocks, batch normalization**, and a basic discriminator design. To overcome these limitations, we now proceed to train an **ESRGAN (Enhanced SRGAN)** model. ESRGAN introduces **Residual-in-Residual Dense Blocks (RRDBs)**, **removes batch normalization**, and uses a **relativistic average discriminator**, significantly improving texture sharpness, realism, and overall visual fidelity.

Training Pipeline - ESRGAN

The ESRGAN (Enhanced Super-Resolution GAN) training framework is designed to improve upon SRGAN by producing sharper, more realistic, and detail-rich high-resolution images. Your ESRGAN training loop builds on adversarial learning while integrating advanced residual learning (via RRDB blocks) and deep feature discrimination (via a strided convolutional discriminator).

Training Configuration

- **Epochs**: Typically 150 or until convergence
- **Batch Size**: Based on available memory (e.g., 16–32)
- **Loss Functions**: Composite losses integrating both pixel-level and perceptual-level differences
- **Training Setup**: Alternating optimization of Generator (netG) and Discriminator (netD)
- **Device**: GPU-enabled (CUDA)

Generator Objective

The **ESRGAN Generator** is built using:

- An initial convolutional layer,
- A deep stack of **23 Residual-in-Residual Dense Blocks (RRDB)**,
- Post-residual convolution,
- Two **upsampling blocks** to scale up the image,
- A final convolutional output layer with LeakyReLU.

Purpose: Extract deep hierarchical features from low-resolution inputs, upscale them progressively, and reconstruct natural-looking, high-resolution images.

Discriminator Objective

The **ESRGAN Discriminator** consists of:

- Deep **strided convolutional layers** with increasing feature depth (up to 512),
- An adaptive average pooling and two fully connected layers,
- A **final sigmoid output** for real/fake classification.

Purpose: Evaluate whether generated high-resolution images are perceptually close to real ones.

Loss Functions Used

1. **Adversarial Loss** (L_adv):
   Encourages the generator to produce outputs indistinguishable from real HR images using discriminator feedback.

2. **Content Loss (Perceptual Loss)** (L_content):
   Extracts high-level features using a pre-trained VGG network (before activation) and computes feature space differences between generated and ground truth HR images.

3. **Pixel Loss (optional)**:
   Sometimes included (e.g., L1 or L2 loss) for additional low-level structure enforcement.

Final Generator Loss:

$L_{total} = \lambda 1 * L_{content} + \lambda 2 * L_{adv}$

Where $\lambda 1$ and $\lambda 2$ are tunable weights (e.g., 1.0 and 0.005 respectively).

Training Workflow

Per Epoch:

1. **Forward pass (Generator)**:

    - Generate super-resolved image from low-resolution input.

2. **Update Discriminator**: Compute D(real) and D(fake), then: d_loss = 1 - D(real) + D(fake)
    - Backpropagation & optimizer step.

3. **Update Generator**:
    - Recompute fake_img
    - Compute g_loss using perceptual and adversarial losses.
    - Backpropagation & optimizer step.

4. **Compute metrics**:
    - PSNR (Peak Signal-to-Noise Ratio)
    - SSIM (Structural Similarity Index Measure)
    - Discriminator/Generator scores

Logging & Checkpointing

- Track: g_loss, d_loss, psnr, ssim, D(real), D(fake) each epoch.
- Save **best performing generator** (best_G.pt) when validation performance improves.
- Use **early stopping** if no improvement is seen for a defined number of epochs (e.g., 5 rounds).
- Save **last generator checkpoint** after each epoch (last_G.pt).

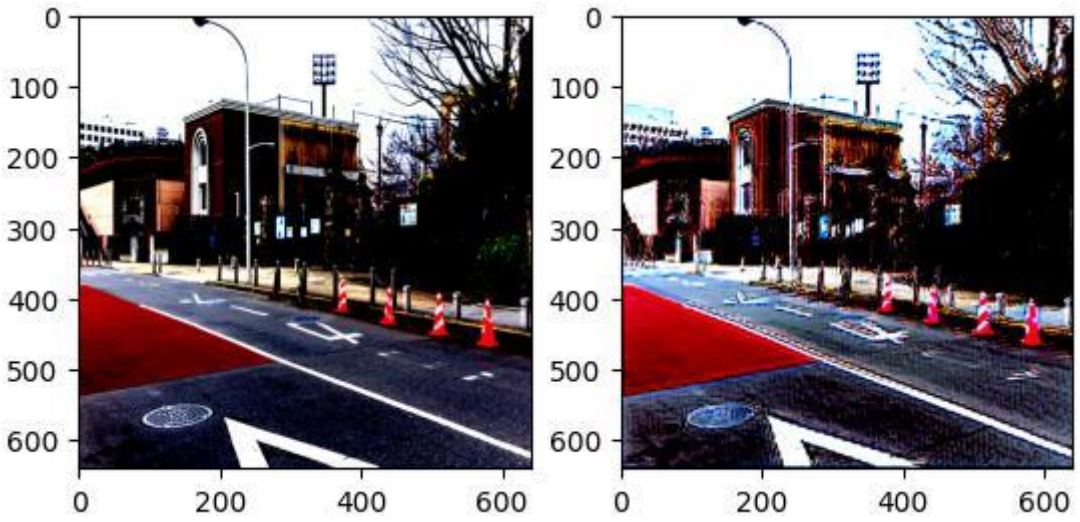
Summary of ESRGAN Improvements Over SRGAN

- **Deeper Network**: ESRGAN uses **23 Residual-in-Residual Dense Blocks (RRDBs)** for better feature extraction, unlike SRGAN's

shallow residual blocks.

- **No Batch Normalization**: ESRGAN **removes batch normalization** to avoid texture distortion and improve detail preservation.
- **Advanced Discriminator**:ESRGAN employs a **deeper discriminator with strided convolutions**, enabling better texture discrimination
- **Enhanced Loss Functions**: Uses **Relativistic average GAN loss** and **VGG-based perceptual loss (before activation)** for more realistic textures
- **Superior Image Quality**: Produces **sharper, more natural, and artifact-free images** compared to SRGAN's blurrier outputs.

Final Output of ESRGAN

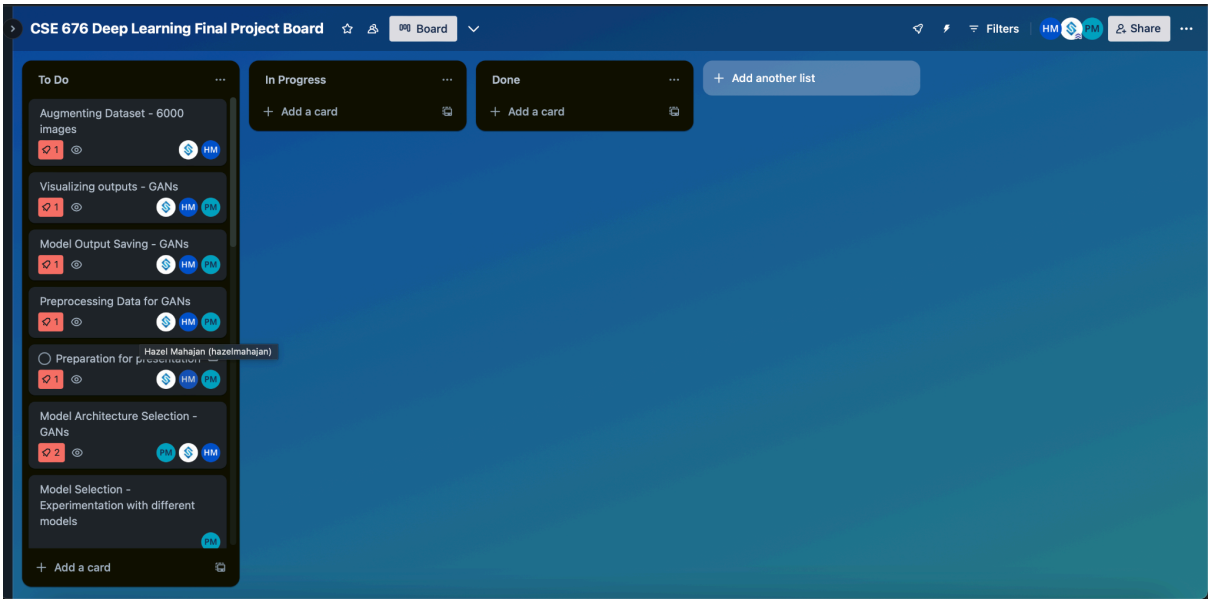


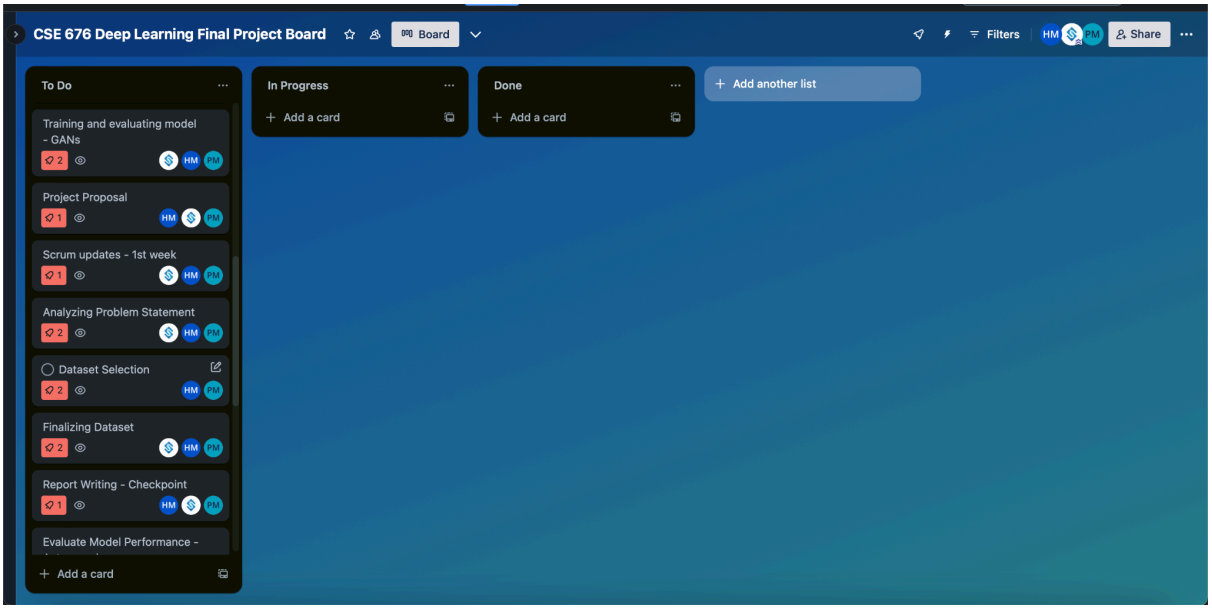
Conclusion

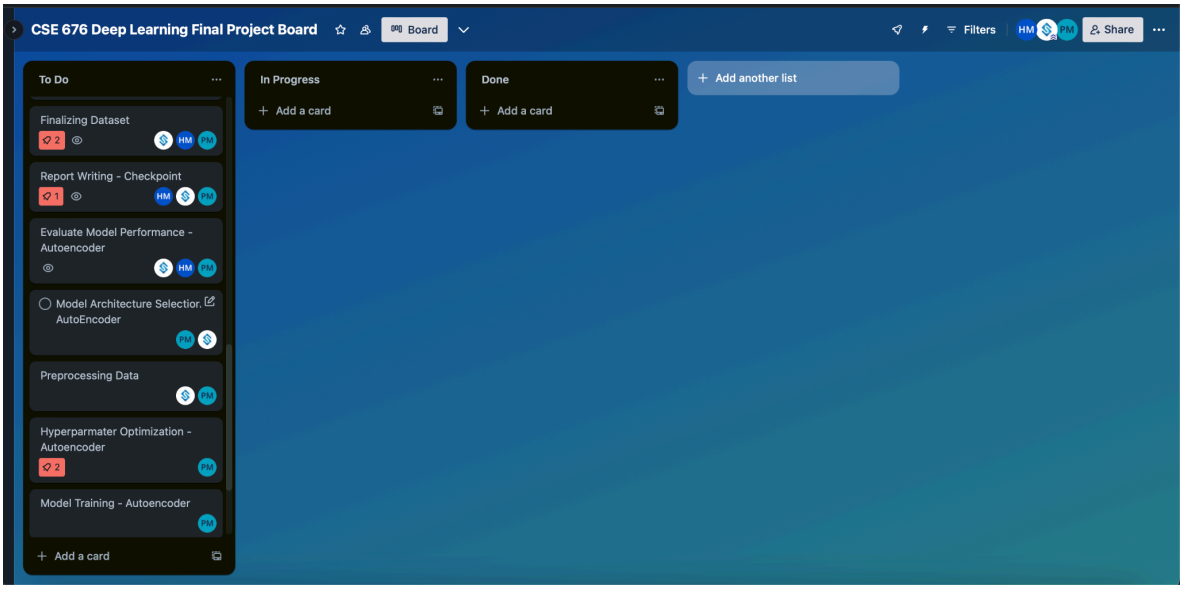
Based on the training results and visual output quality, ESRGAN has proven to be the most effective model for super-resolution and image refinement tasks in this project. Unlike traditional interpolation methods or even earlier deep learning models like SRGAN, ESRGAN is capable of producing remarkably sharp, detailed, and perceptually convincing high-resolution images. The use of Residual-in-Residual Dense Blocks (RRDBs) enables deep feature extraction without degradation, while the relativistic discriminator ensures texture realism. Through our experiments, ESRGAN consistently outperformed other approaches in capturing fine details, restoring edges, and preserving content structure—making it the best-suited architecture for high-fidelity image super refinement.

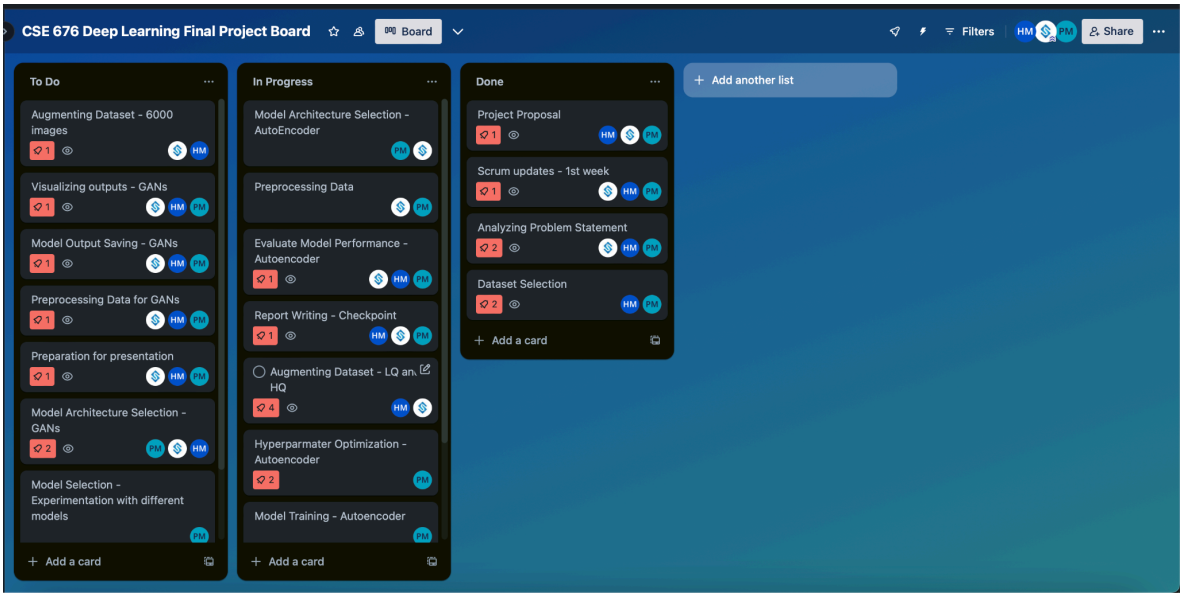
Trello Project Board Link

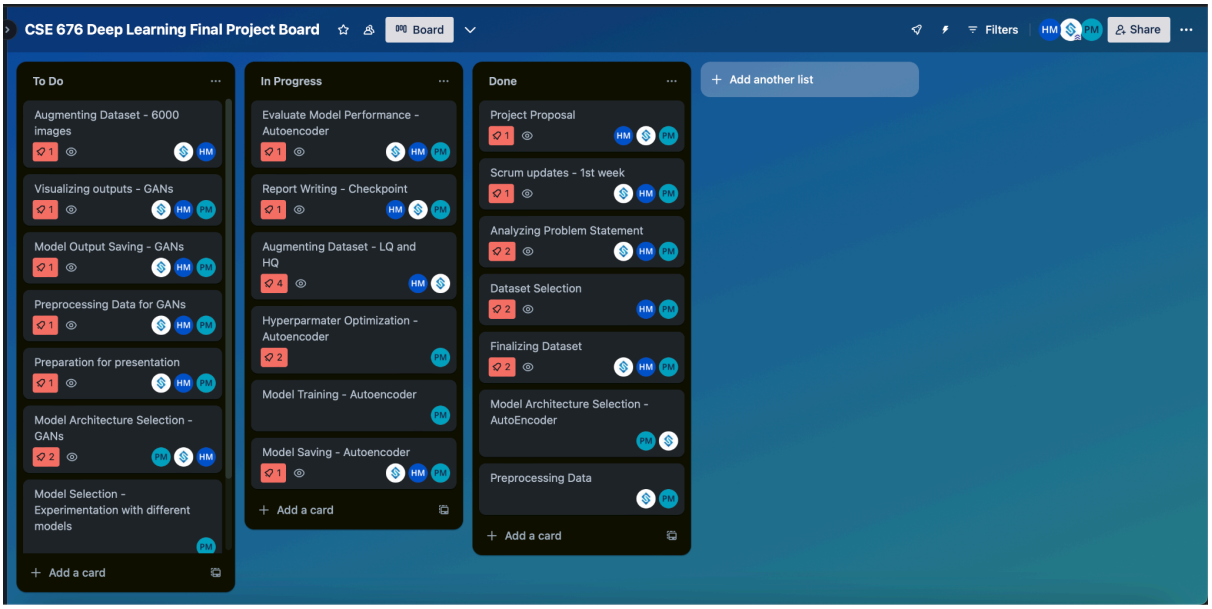
https://trello.com/b/p9g4wgtL/cse-676-deep-learning-final-project-board

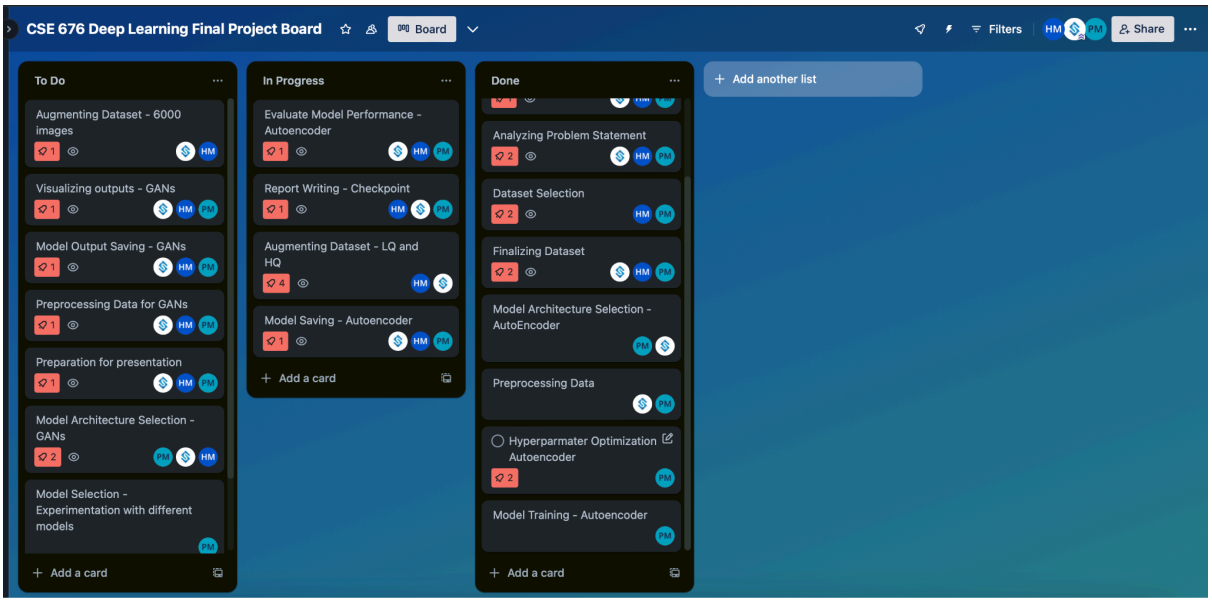


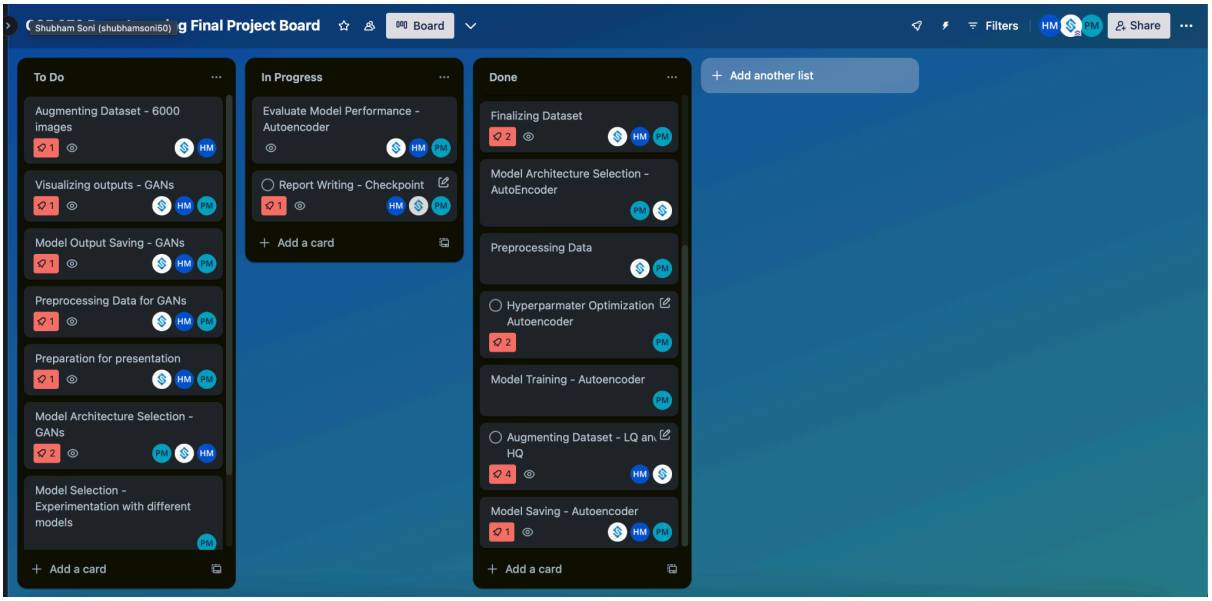


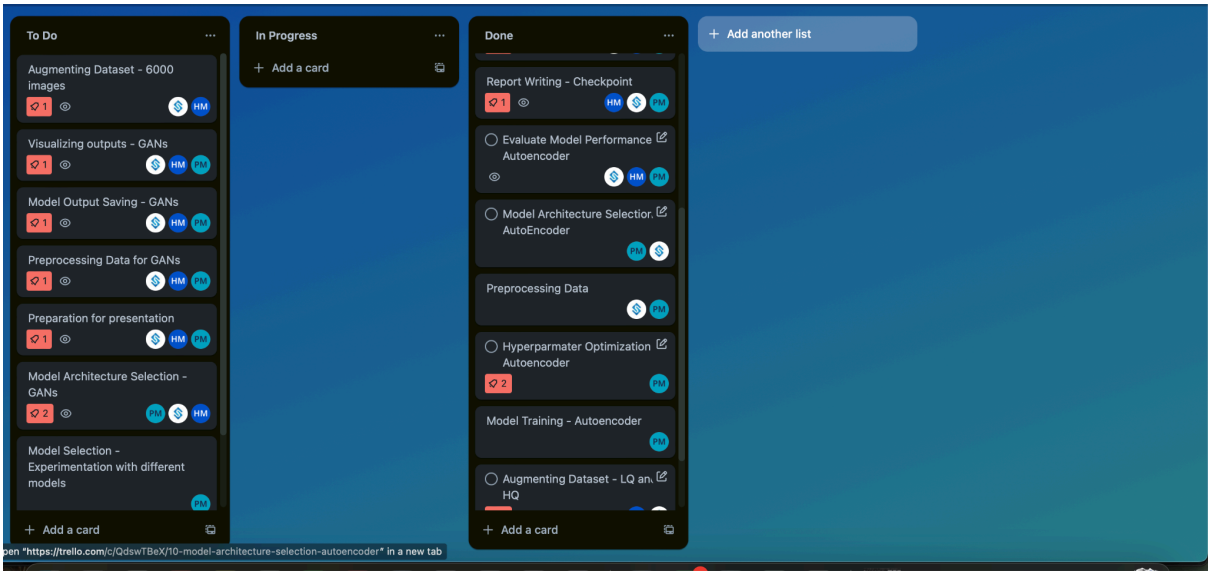

CSE 676 Deep Learning Final Project Board
Board
Filters
Share
To Do
Finalizing Dataset
Report Writing - Checkpoint
Evaluate Model Performance - Autoencoder
Model Architecture Selection - AutoEncoder
Preprocessing Data
Hyperparmater Optimization - Autoencoder
Model Training - Autoencoder
Add a card
In Progress
Add a card
Done
Add a card
Add another list

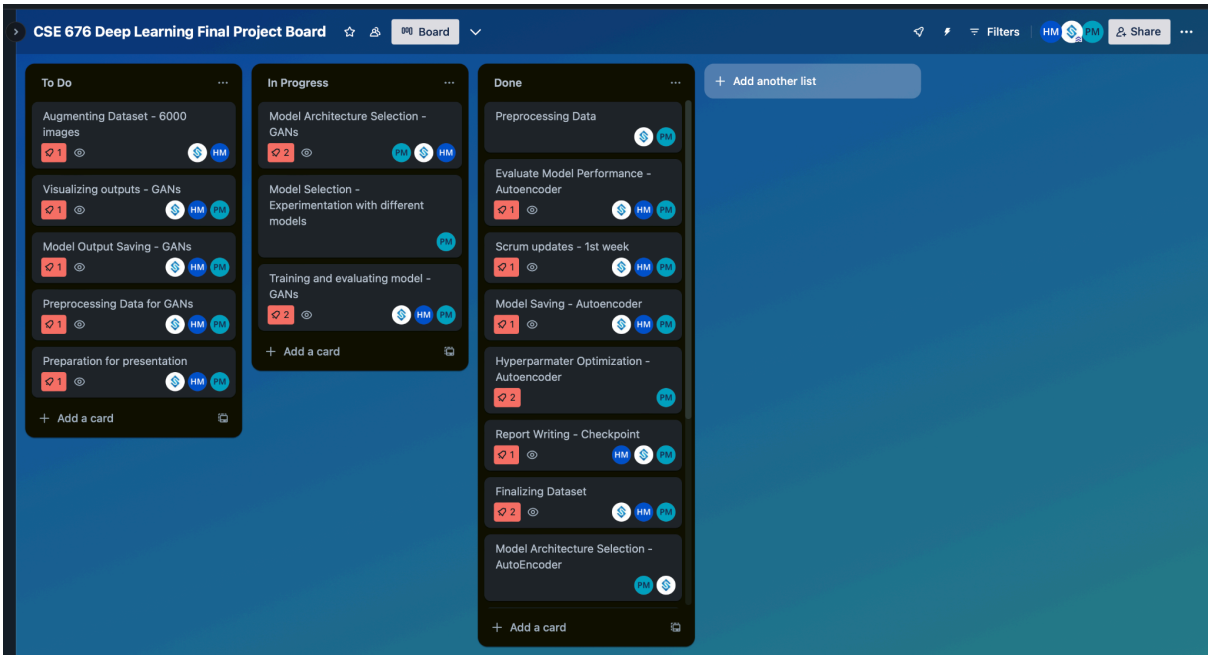

CSE 676 Deep Learning Final Project Board
Board
Filters
Share
To Do
Augmenting Dataset - 6000 images
Visualizing outputs - GANs
Model Output Saving - GANs
Preprocessing Data for GANs
Preparation for presentation
Model Architecture Selection - GANs
Model Selection - Experimentation with different models
Add a card
In Progress
Model Architecture Selection - AutoEncoder
Preprocessing Data
Evaluate Model Performance - Autoencoder
Report Writing - Checkpoint
Augmenting Dataset - LQ an HQ
Hyperparmater Optimization - Autoencoder
Model Training - Autoencoder
Add a card
Done
Project Proposal
Scrum updates - 1st week
Analyzing Problem Statement
Dataset Selection
Add a card
Add another list

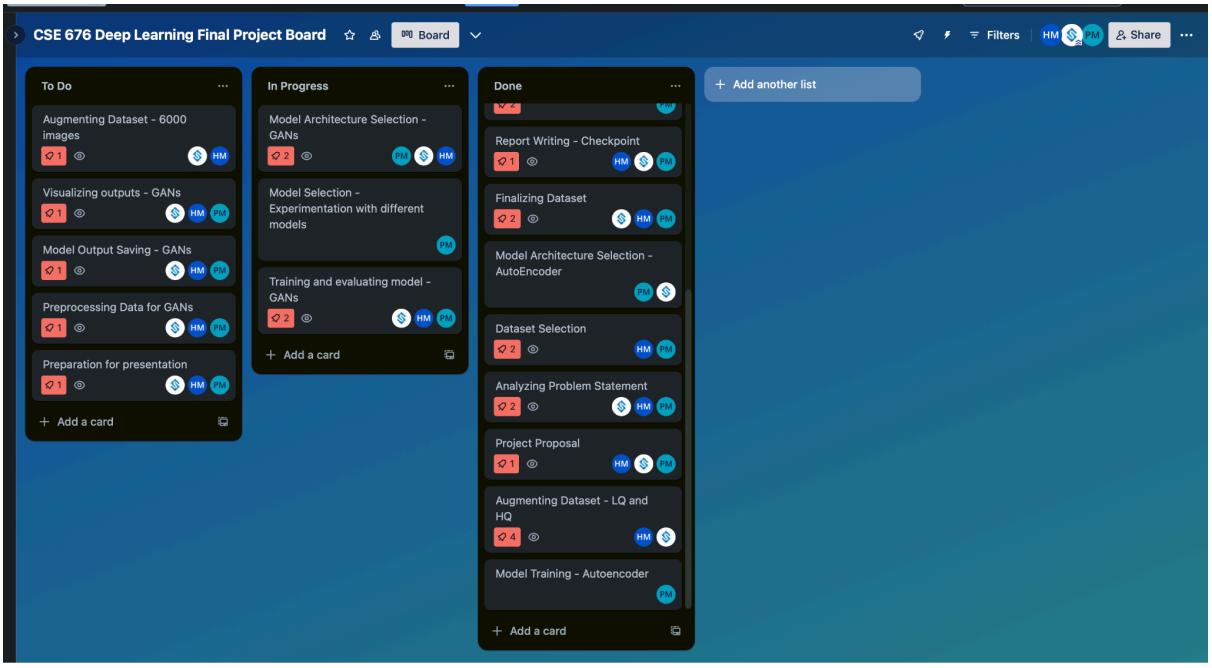

CSE 676 Deep Learning Final Project Board
Board
Filters
Share
To Do
Augmenting Dataset - 6000 images
Visualizing outputs - GANs
Model Output Saving - GANs
Preprocessing Data for GANs
Preparation for presentation
Model Architecture Selection - GANs
Model Selection - Experimentation with different models
Add a card
In Progress
Evaluate Model Performance - Autoencoder
Report Writing - Checkpoint
Augmenting Dataset - LQ and HQ
Hyperparmater Optimization - Autoencoder
Model Training - Autoencoder
Model Saving - Autoencoder
Add a card
Done
Project Proposal
Scrum updates - 1st week
Analyzing Problem Statement
Dataset Selection
Finalizing Dataset
Model Architecture Selection - AutoEncoder
Preprocessing Data
Add a card
Add another list


CSE 676 Deep Learning Final Project Board
Board
Filters
Share
To Do
Augmenting Dataset - 6000 images
Visualizing outputs - GANs
Model Output Saving - GANs
Preprocessing Data for GANs
Preparation for presentation
Model Architecture Selection - GANs
Model Selection - Experimentation with different models
Add a card
In Progress
Evaluate Model Performance - Autoencoder
Report Writing - Checkpoint
Augmenting Dataset - LQ and HQ
Model Saving - Autoencoder
Add a card
Done
Analyzing Problem Statement
Dataset Selection
Finalizing Dataset
Model Architecture Selection - AutoEncoder
Preprocessing Data
Hyperparmater Optimization Autoencoder
Model Training - Autoencoder
Add a card
Add another list

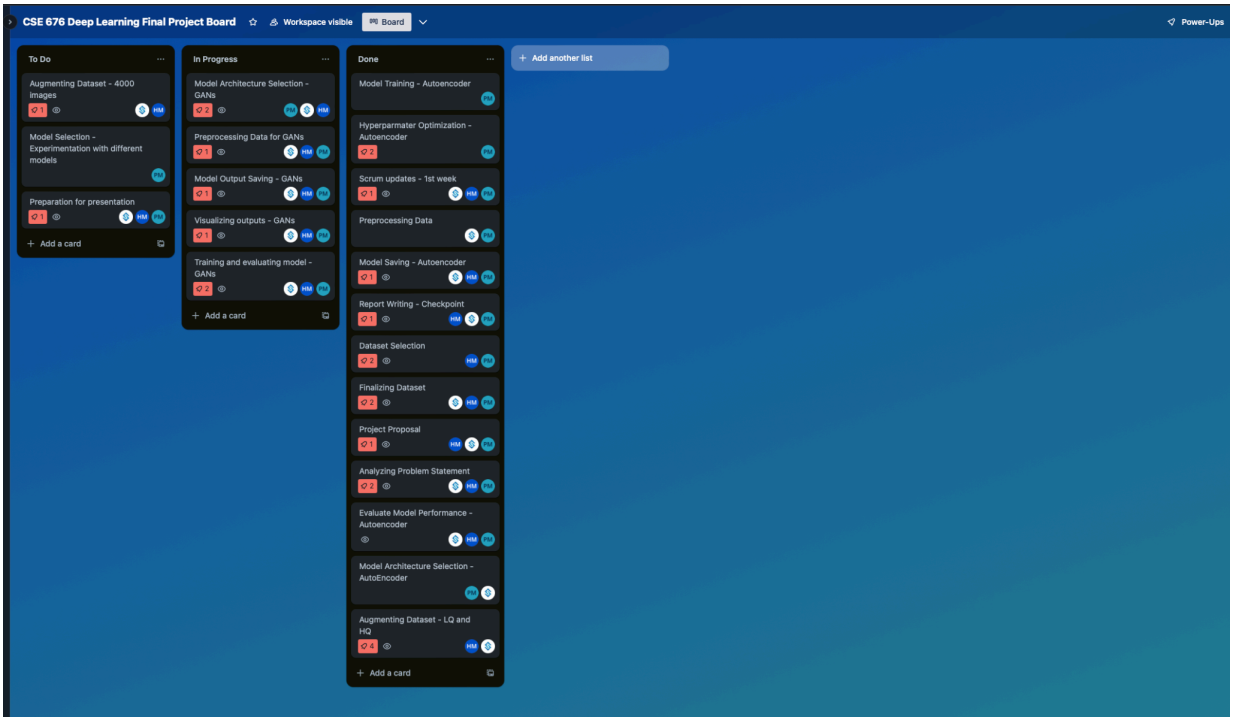

Shubham Soni (shubhamsoni50)
g Final Project Board
Board
Filters
Share
To Do
Augmenting Dataset - 6000 images
Visualizing outputs - GANs
Model Output Saving - GANs
Preprocessing Data for GANs
Preparation for presentation
Model Architecture Selection - GANs
Model Selection - Experimentation with different models
Add a card
In Progress
Evaluate Model Performance - Autoencoder
Report Writing - Checkpoint
Add a card
Done
Finalizing Dataset
Model Architecture Selection - AutoEncoder
Preprocessing Data
Hyperparmater Optimization Autoencoder
Model Training - Autoencoder
Augmenting Dataset - LQ an
HQ
Model Saving - Autoencoder
Add a card
Add another list


To Do
Augmenting Dataset - 6000 images
Visualizing outputs - GANs
Model Output Saving - GANs
Preprocessing Data for GANs
Preparation for presentation
Model Architecture Selection - GANs
Model Selection - Experimentation with different models
Add a card
In Progress
Add a card
Done
Report Writing - Checkpoint
Evaluate Model Performance Autoencoder
Model Architecture Selection AutoEncoder
Preprocessing Data
Hyperparmater Optimization Autoencoder
Model Training - Autoencoder
Augmenting Dataset - LQ an
HQ
Add a card
Add another list
pen "https://trello.com/c/QdswTBeX/10-model-architecture-selection-autoencoder" in a new tab

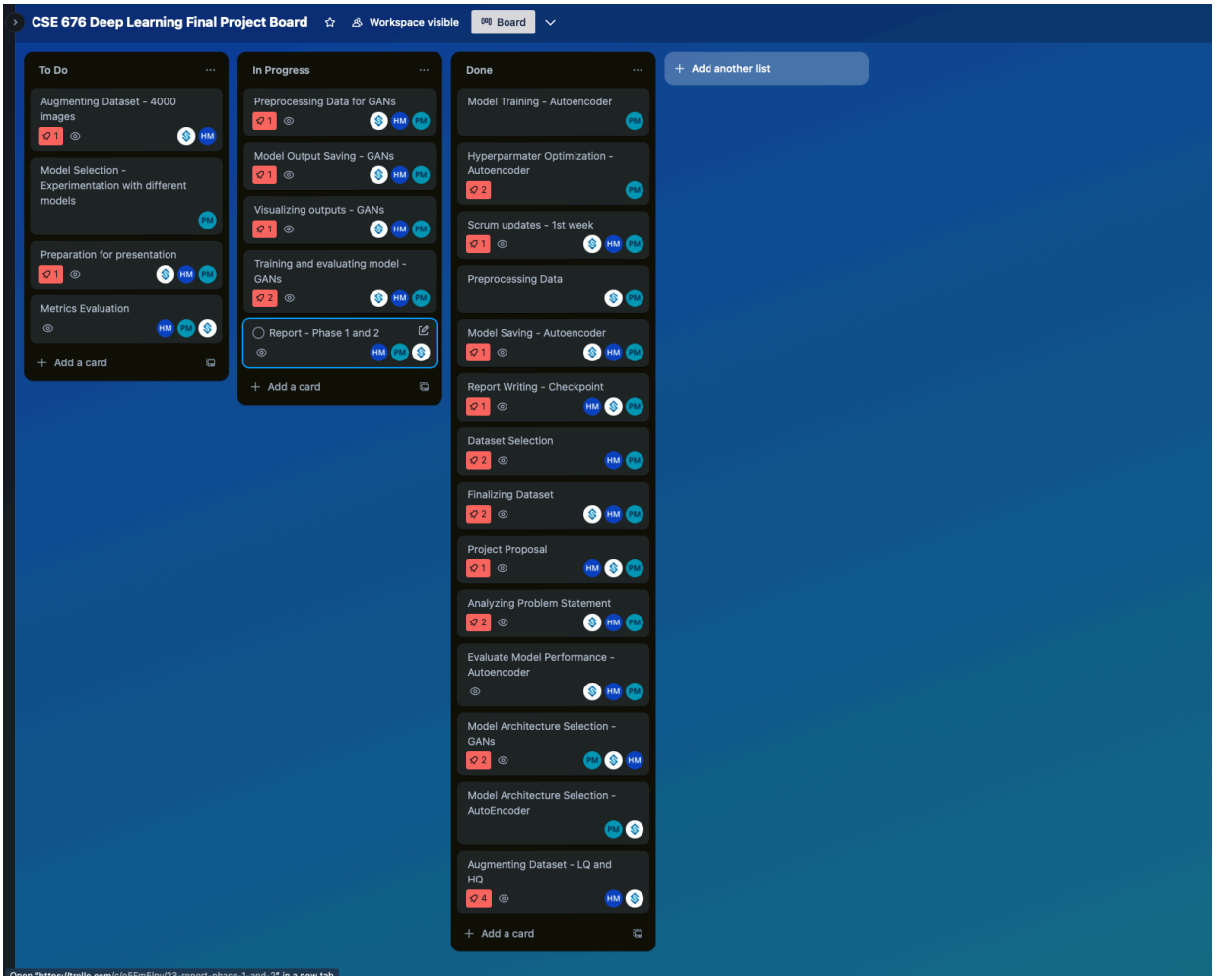

CSE 676 Deep Learning Final Project Board
Board
Filters
Share
To Do
Augmenting Dataset - 6000 images
Visualizing outputs - GANs
Model Output Saving - GANs
Preprocessing Data for GANs
Preparation for presentation
Add a card
In Progress
Model Architecture Selection - GANs
Model Selection - Experimentation with different models
Training and evaluating model - GANs
Add a card
Done
Preprocessing Data
Evaluate Model Performance - Autoencoder
Scrum updates - 1st week
Model Saving - Autoencoder
Hyperparmater Optimization - Autoencoder
Report Writing - Checkpoint
Finalizing Dataset
Model Architecture Selection - AutoEncoder
Add a card
Add another list


CSE 676 Deep Learning Final Project Board
Board
Filters
Share
To Do
Augmenting Dataset - 6000 images
Visualizing outputs - GANs
Model Output Saving - GANs
Preprocessing Data for GANs
Preparation for presentation
Add a card
In Progress
Model Architecture Selection - GANs
Model Selection - Experimentation with different models
Training and evaluating model - GANs
Add a card
Done
Report Writing - Checkpoint
Finalizing Dataset
Model Architecture Selection - AutoEncoder
Dataset Selection
Analyzing Problem Statement
Project Proposal
Augmenting Dataset - LQ and HQ
Model Training - Autoencoder
Add a card
Add another list

Phase - 2


CSE 676 Deep Learning Final Project Board
Workspace visible
Board
Power-Ups
To Do
Augmenting Dataset - 4000 images
Model Selection - Experimentation with different models
Preparation for presentation
Add a card
In Progress
Model Architecture Selection - GANs
Preprocessing Data for GANs
Model Output Saving - GANs
Visualizing outputs - GANs
Training and evaluating model - GANs
Add a card
Done
Model Training - Autoencoder
Hyperparmater Optimization - Autoencoder
Scrum updates - 1st week
Preprocessing Data
Model Saving - Autoencoder
Report Writing - Checkpoint
Dataset Selection
Finalizing Dataset
Project Proposal
Analyzing Problem Statement
Evaluate Model Performance - Autoencoder
Model Architecture Selection - AutoEncoder
Augmenting Dataset - LQ and HQ
Add a card
Add another list

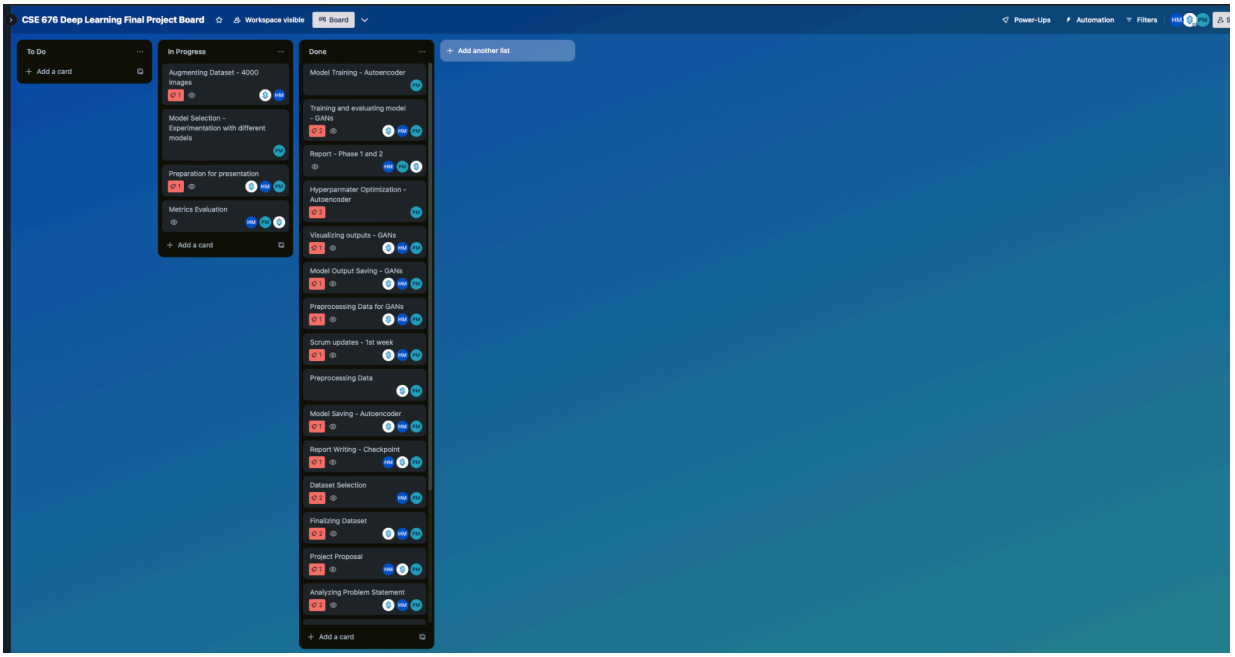

CSE 676 Deep Learning Final Project Board
Workspace visible
Board
To Do
Augmenting Dataset - 4000 images
Model Selection - Experimentation with different models
Preparation for presentation
Add a card
In Progress
Preprocessing Data for GANs
Model Output Saving - GANs
Visualizing outputs - GANs
Training and evaluating model - GANs
Add a card
Done
Model Training - Autoencoder
Hyperparmater Optimization - Autoencoder
Scrum updates - 1st week
Preprocessing Data
Model Saving - Autoencoder
Report Writing - Checkpoint
Dataset Selection
Finalizing Dataset
Project Proposal
Analyzing Problem Statement
Evaluate Model Performance - Autoencoder
Model Architecture Selection - GANs
Model Architecture Selection - AutoEncoder
Augmenting Dataset - LQ and HQ
Add a card
Add another list

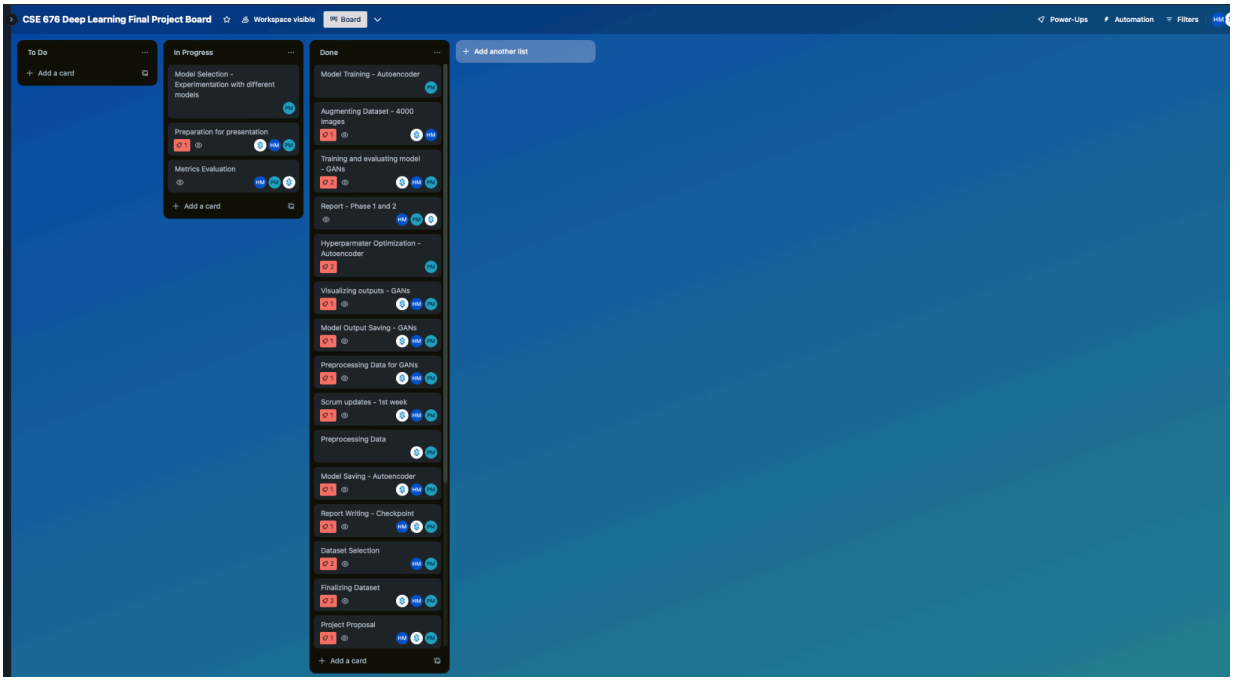

CSE 676 Deep Learning Final Project Board
Workspace visible
Board
To Do
Augmenting Dataset - 4000 images
Model Selection - Experimentation with different models
Preparation for presentation
Metrics Evaluation
Add a card
In Progress
Preprocessing Data for GANs
Model Output Saving - GANs
Visualizing outputs - GANs
Training and evaluating model - GANs
Report - Phase 1 and 2
Add a card
Done
Model Training - Autoencoder
Hyperparmater Optimization - Autoencoder
Scrum updates - 1st week
Preprocessing Data
Model Saving - Autoencoder
Report Writing - Checkpoint
Dataset Selection
Finalizing Dataset
Project Proposal
Analyzing Problem Statement
Evaluate Model Performance - Autoencoder
Model Architecture Selection - GANs
Model Architecture Selection - AutoEncoder
Augmenting Dataset - LQ and HQ
Add a card
Add another list

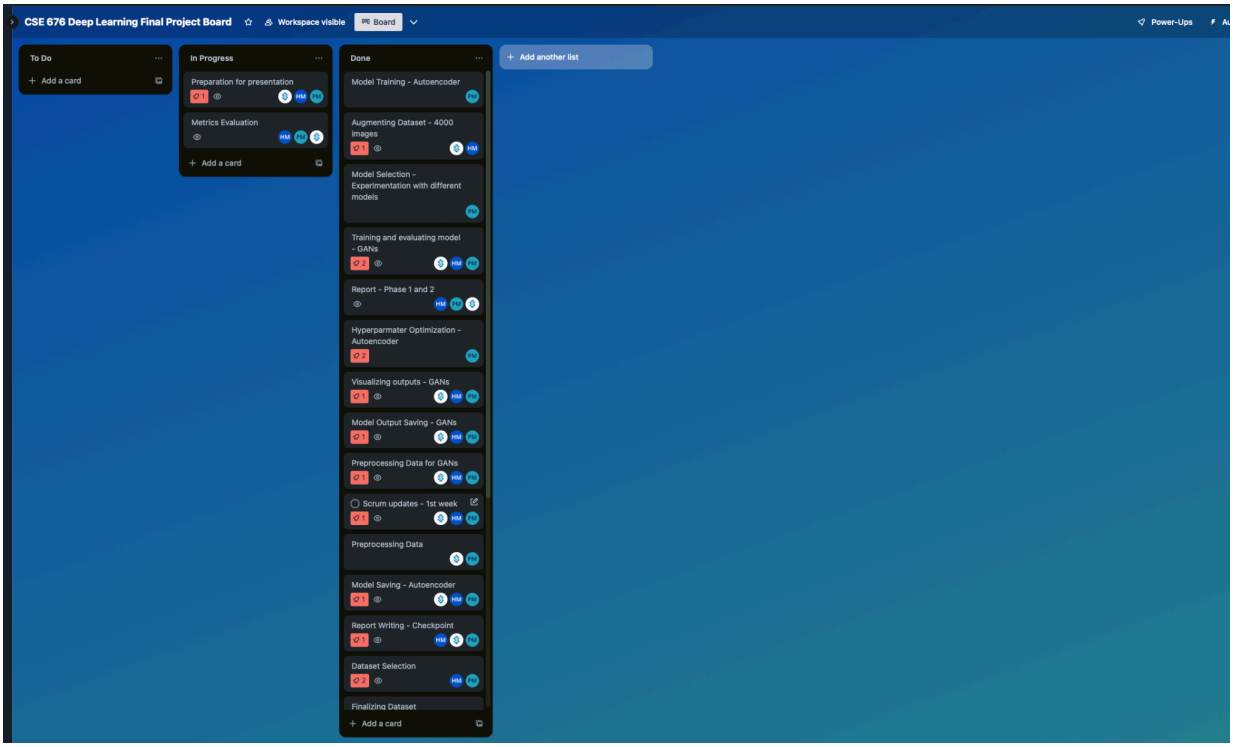

CSE 676 Deep Learning Final Project Board
Workspace visible
Board
Power-Ups
Automation
To Do
Augmenting Dataset - 4000 images
Model Selection - Experimentation with different models
Preparation for presentation
Metrics Evaluation
Add a card
In Progress
Visualizing outputs - GANs
Training and evaluating model - GANs
Report - Phase 1 and 2
Add a card
Done
Model Training - Autoencoder
Hyperparmater Optimization - Autoencoder
Model Output Saving - GANs
Preprocessing Data for GANs
Scrum updates - 1st week
Preprocessing Data
Model Saving - Autoencoder
Report Writing - Checkpoint
Dataset Selection
Finalizing Dataset
Project Proposal
Analyzing Problem Statement
Evaluate Model Performance - Autoencoder
Model Architecture Selection - GANs
Model Architecture Selection - AutoEncoder
Add a card
Add another list

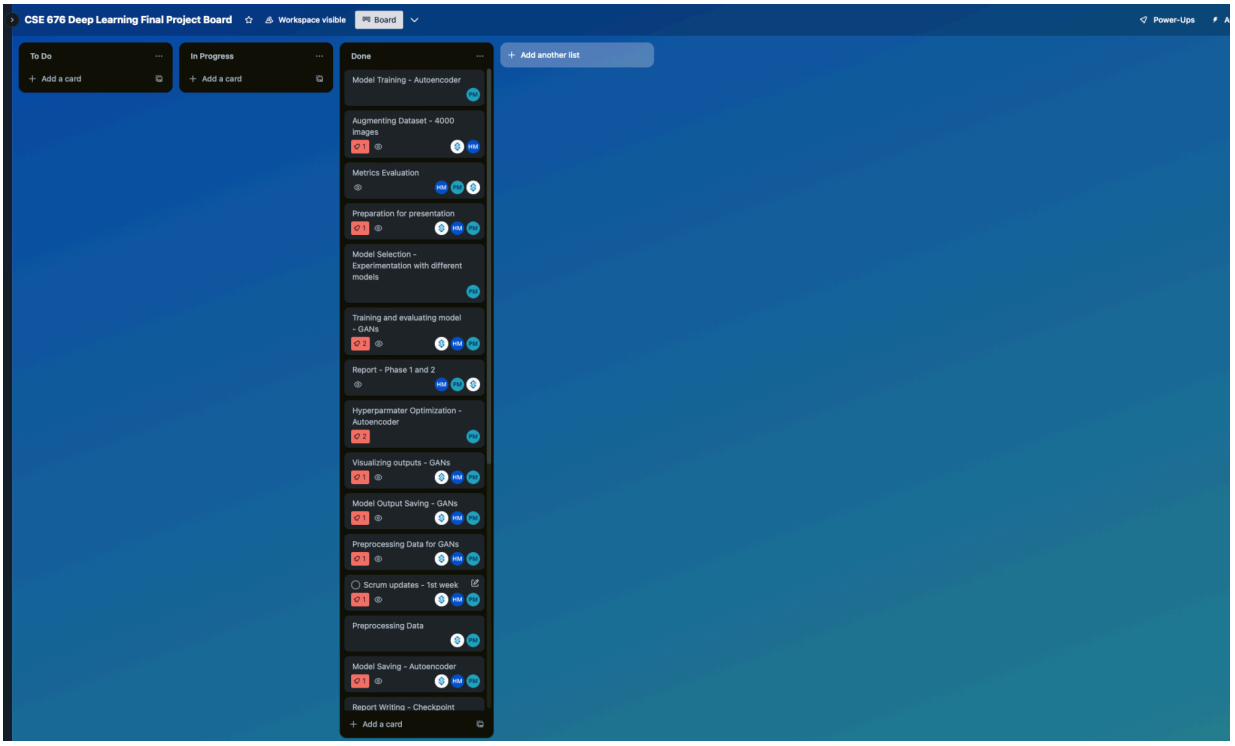

CSE 676 Deep Learning Final Project Board
Workspace visible
Board
Power-Ups
Automation
To Do
Augmenting Dataset - 4000 images
Model Selection - Experimentation with different models
Preparation for presentation
Metrics Evaluation
Add a card
In Progress
Report - Phase 1 and 2
Add a card
Done
Model Training - Autoencoder
Training and evaluating model - GANs
Hyperparmater Optimization - Autoencoder
Visualizing outputs - GANs
Model Output Saving - GANs
Preprocessing Data for GANs
Scrum updates - 1st week
Preprocessing Data
Model Saving - Autoencoder
Report Writing - Checkpoint
Dataset Selection
Finalizing Dataset
Project Proposal
Analyzing Problem Statement
Evaluate Model Performance - Autoencoder
Add a card
Add another list


CSE 676 Deep Learning Final Project Board
Workspace visible
Board
Power-Ups
Automation
Filters
To Do
Model Selection - Experimentation with different models
Preparation for presentation
Metrics Evaluation
Add a card
In Progress
Augmenting Dataset - 4000 images
Add a card
Done
Model Training - Autoencoder
Training and evaluating model - GANs
Report - Phase 1 and 2
Hyperparmater Optimization - Autoencoder
Visualizing outputs - GANs
Model Output Saving - GANs
Preprocessing Data for GANs
Scrum updates - 1st week
Preprocessing Data
Model Saving - Autoencoder
Report Writing - Checkpoint
Dataset Selection
Finalizing Dataset
Project Proposal
Analyzing Problem Statement
Add a card
Add another list


CSE 676 Deep Learning Final Project Board
Workspace visible
Board
Power-Ups
Automation
Filters
To Do
Add a card
In Progress
Augmenting Dataset - 4000 images
Model Selection - Experimentation with different models
Preparation for presentation
Metrics Evaluation
Add a card
Done
Model Training - Autoencoder
Training and evaluating model - GANs
Report - Phase 1 and 2
Hyperparmater Optimization - Autoencoder
Visualizing outputs - GANs
Model Output Saving - GANs
Preprocessing Data for GANs
Scrum updates - 1st week
Preprocessing Data
Model Saving - Autoencoder
Report Writing - Checkpoint
Dataset Selection
Finalizing Dataset
Project Proposal
Analyzing Problem Statement
Add a card
Add another list


CSE 676 Deep Learning Final Project Board
Workspace visible
Board
Power-Ups
Automation
Filters
To Do
Add a card
In Progress
Model Selection - Experimentation with different models
Preparation for presentation
Metrics Evaluation
Add a card
Done
Model Training - Autoencoder
Augmenting Dataset - 4000 images
Training and evaluating model - GANs
Report - Phase 1 and 2
Hyperparmater Optimization - Autoencoder
Visualizing outputs - GANs
Model Output Saving - GANs
Preprocessing Data for GANs
Scrum updates - 1st week
Preprocessing Data
Model Saving - Autoencoder
Report Writing - Checkpoint
Dataset Selection
Finalizing Dataset
Project Proposal
Add a card
Add another list


CSE 676 Deep Learning Final Project Board
Workspace visible
Board
Power-Ups
To Do
Add a card
In Progress
Preparation for presentation
Metrics Evaluation
Add a card
Done
Model Training - Autoencoder
Augmenting Dataset - 4000 images
Model Selection - Experimentation with different models
Training and evaluating model - GANs
Report - Phase 1 and 2
Hyperparmater Optimization - Autoencoder
Visualizing outputs - GANs
Model Output Saving - GANs
Preprocessing Data for GANs
Scrum updates - 1st week
Preprocessing Data
Model Saving - Autoencoder
Report Writing - Checkpoint
Dataset Selection
Finalizing Dataset
Add a card
Add another list


CSE 676 Deep Learning Final Project Board
Workspace visible
Board
Power-Ups
To Do
Add a card
In Progress
Add a card
Done
Model Training - Autoencoder
Augmenting Dataset - 4000 images
Metrics Evaluation
Preparation for presentation
Model Selection - Experimentation with different models
Training and evaluating model - GANs
Report - Phase 1 and 2
Hyperparmater Optimization - Autoencoder
Visualizing outputs - GANs
Model Output Saving - GANs
Preprocessing Data for GANs
Scrum updates - 1st week
Preprocessing Data
Model Saving - Autoencoder
Report Writing - Checkpoint
Add a card
Add another list

References

[1] “Flickr2K,” Kaggle, Jan. 29, 2023. https://www.kaggle.com/datasets/daehoyang/flickr2k

[2] K. Chauhan, “Super-Resolution using autoencoders and TF2.0 - Analytics Vidhya - Medium,” Medium, Dec. 14, 2021. [Online]. Available: https://medium.com/analytics-vidhya/super-resolution-using-autoencoders-and-tf2-0-505215c1674

[3] M. Busquet, “Understanding Image Generation: A Beginner’s Guide to Generative Adversarial Networks,” OVHcloud Blog, Sep. 05, 2023. https://blog.ovhcloud.com/understanding-image-generation-beginner-guide-generative-adversarial-networks-gan/