AI CLUB IITM

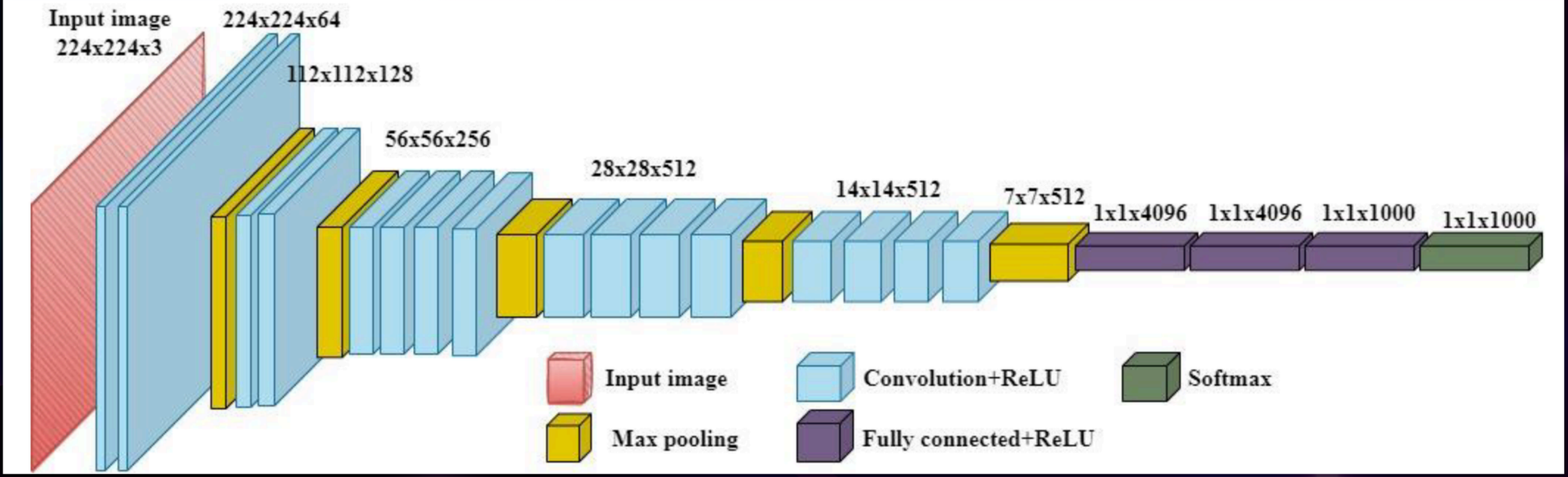
DC MINI-PROJECT

PIX2PIX-IMAGE CARTOONIZATION

(Ghibli?)

SUBGROUP 3.0

CONVOLUTIONAL NEURAL NETWORKS



VGG 19 Architecture

CNNs are deep learning models designed to process image data by learning spatial hierarchies of features.

Why CNNs for Our Project?

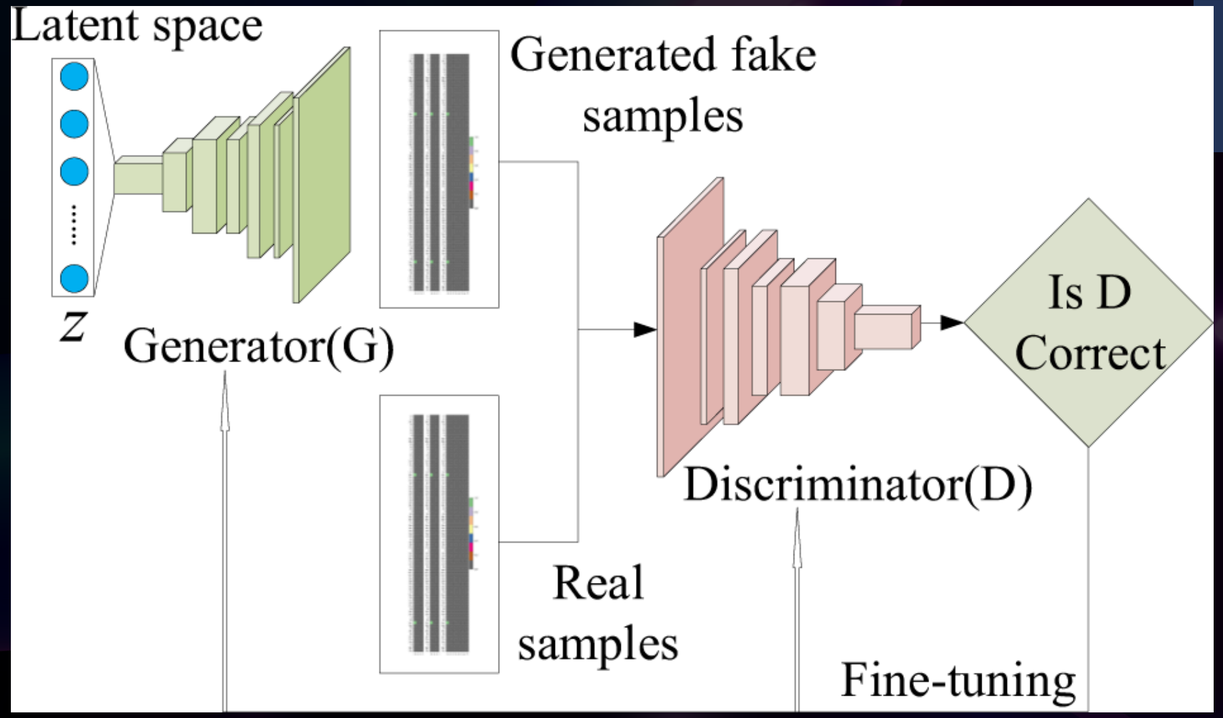
- Extract visual features from real and cartoon images
- Used in Generator & Discriminator Architecture.

Core Components:

- **Convolution Layer** – Detects local patterns (e.g., edges, textures) .
  -made up of Kernels(filters- multiple channels)

- **Pooling Layer** – Downsamples and highlights dominant features.
  -max value of each patch- max pooling)

- **Fully Connected Layer** – every neuron connects to all inputs
  -classification/ prediction



GANS

The "What"-

- **GAN stands for Generative Adversarial Network**
- **Developed as a new way to solve the age-old image generation problem.**

The "How"-

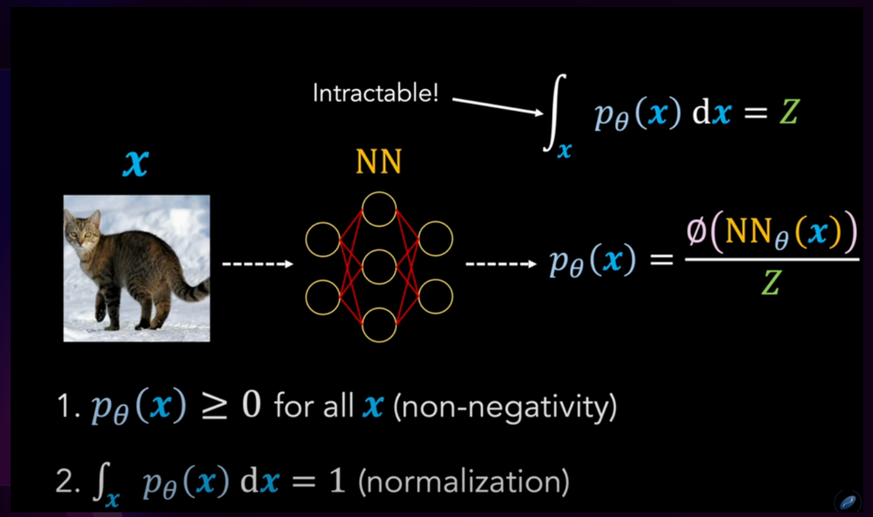
- **2 components - Discriminator and Generator**
- **Discriminator tries to detect fake images - Generator tries to fool it, and thus, both models are trained with backprop based on their outputs**
- **Aim to replicate the underlying probability distribution with the Generator as best as we can**

$$V(G,D) = \mathbb{E}_{x\sim p^*}[\log D(x)] + \mathbb{E}_{z\sim p_z}\left[\log\left(1 - D(G(z))\right)\right]$$

The Value Function in GANs

Here, D(x) represents the output of the Discriminator from our real dataset while D(G(z))) is the output when Generator outputs an image.
Our objective is to maximize log(D(x)) to improve the discriminator and minimize 1 - D(G(z)) for a better generator. We combine these objectives in the value function and train with backprop.

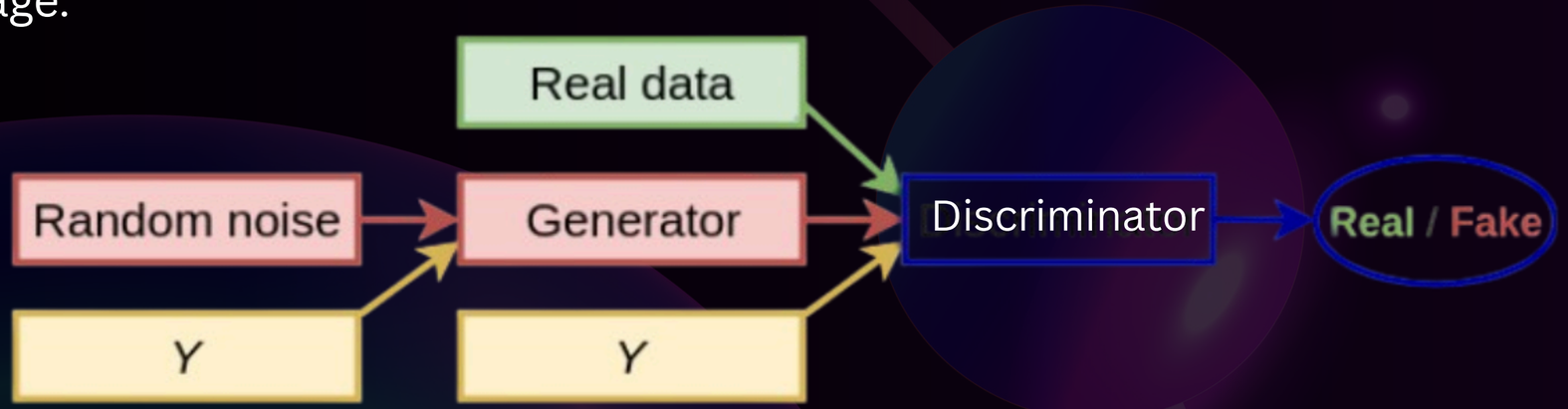
The "Why"-



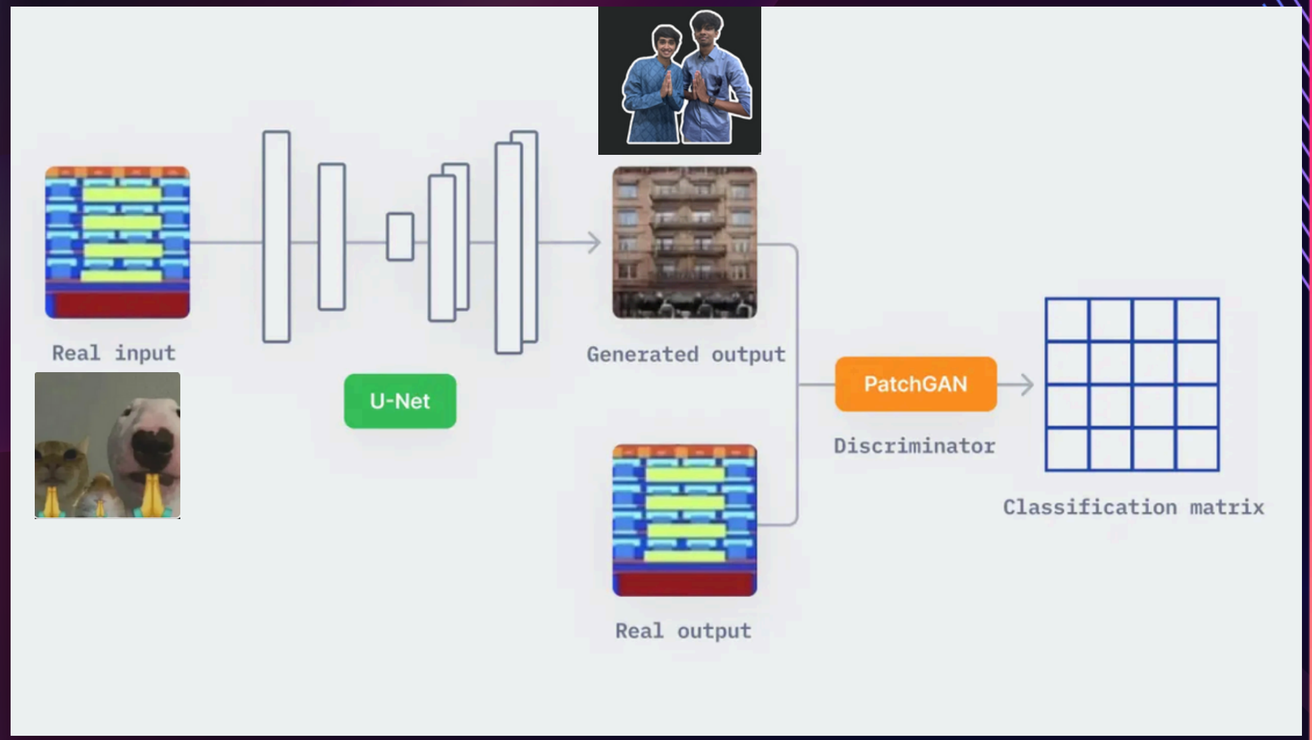
CONDITIONAL GENERATIVE ADVERSIAL NETWORKS

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x}|\boldsymbol{y})] + \mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z}|\boldsymbol{y})))].$$

- Conditional GANs are a type of GAN where both the generator and discriminator are given extra information (called a condition) to guide the output.
- This setup allows for controlled generation, meaning the model can generate specific types of data based on the condition.
- The generator tries to create outputs that not only look real but also match the given condition.
- The discriminator evaluates both the authenticity of the data and whether it aligns with the condition.
- The condition can be a one-hot-encoded vector (e.g., for digit labels in MNIST) or an image itself, like in Pix2Pix.
- In Pix2Pix, the input image acts as the condition, allowing for tasks like edge → photo or sketch → colored image.



PIX-2-PIX



Key Features :

- U-Net model (Encoder + Decoder + Skip Connections)
- Patch GANs
- Conditional Input (CGAN)
- They have altered the CGAN loss function to include L1 loss as well

Why Pix-2-Pix?

- Pix2Pix is a modified version of CGANs, designed specifically for **image-to-image** conversion where the input and output are spatially aligned — a feature enabled by the **U-Net architecture** and not supported by standard CGANs.
- CGANs-Stochastic whereas Pix2Pix- **Deterministic** approach i.e. generator takes only the input image without any added noise.
- As a result, feeding the same image multiple times produces consistent and similar outputs.

Generator

- The generator in Pix2Pix uses a **U-Net architecture** - an encoder-decoder model with skip connections - preserves spatial features.
- Skip connections links downsampling layer to the corresponding upsampling layer, making the output sharper and more accurate.

Discriminator

- The Discriminator uses **Patch GANs** -divides the image into patches and evaluates if each of these patches is generated or original, this matrix is called the classification matrix.

Loss Function

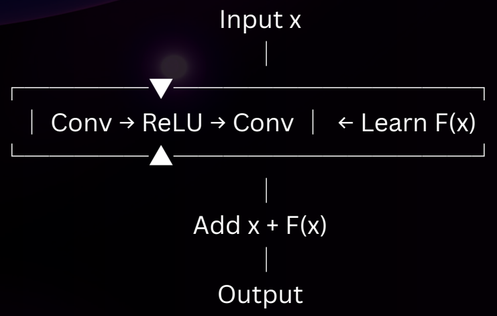
- We add **L1 loss** because CGAN loss alone is not enough.
- CGAN loss tells the generator:
  → "Fool the discriminator by making outputs look real."
- L1 loss tells the generator:
  → "Make the output as close to the real target image as possible."

CARTOONIFY IMPLEMENTATION- CartoonGAN

GENERATOR ARCHITECTURE

- CNN- Initial convolution block--> 8 residual blocks(Conv+ Norm+ReLU+Conv+Norm)-->2 Up-conv--->final 7x7 conv layer to output image
- PatchGAN

Inspired by **ResNet**- Residual block:



LOSS FUNCTIONS

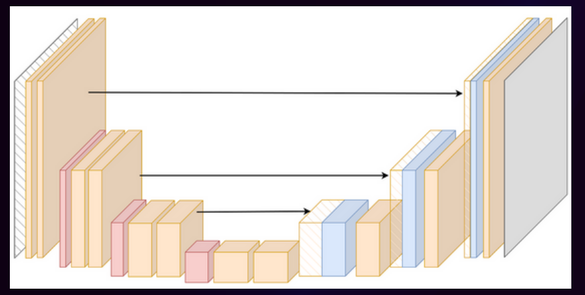
$$\mathcal{L}(G, D) = \mathcal{L}_{adv}(G, D) + \omega \mathcal{L}_{con}(G, D),$$

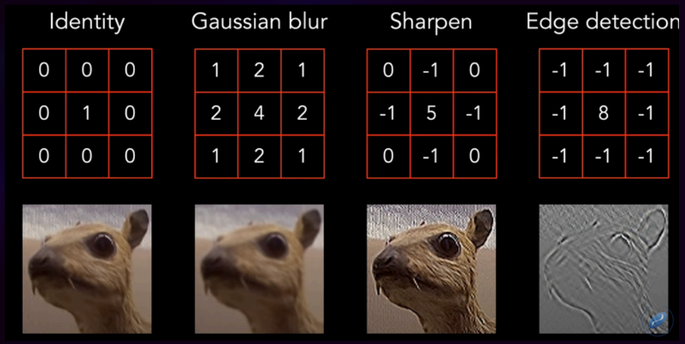
Edge Promoting Adversial loss-

- **Canny Edge detector** to find outlines-Dilate the edges- blur with **Gaussian smoothing**
- Discriminator learns to identify these as fake and thus generator generates images with sharp edges.
- Hence,Edge- Promoting Adversial loss-helps generator to achieve cartoonization.

- ***Content loss***- retains image content during cartoon stylization
- We use pre-trained VGG19, freeze the weights initially and later enable updation
- We use L1 loss- sparser, drastic change in style, L1 handles better.

UNet





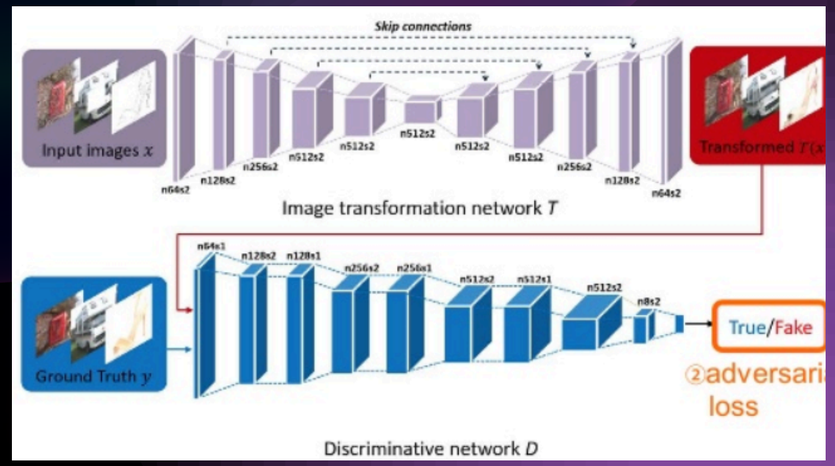
RESULTS

DISCRIMINATOR ARCHITECTURE

- CNN: Conv(3x3)+ Leaky ReLU- extract local features
- Strided Conv - Downsampling
- Final Conv+ Sigmoid- Outputs whether patch is real or fake
- Uses **PatchGAN-** Instead of sayihng this image is fake, it says this patch of image is fake.
- This helps model focus on fine details like edges, texture and shading

CARTOONIFY IMPLEMENTATION- Pix2Pix

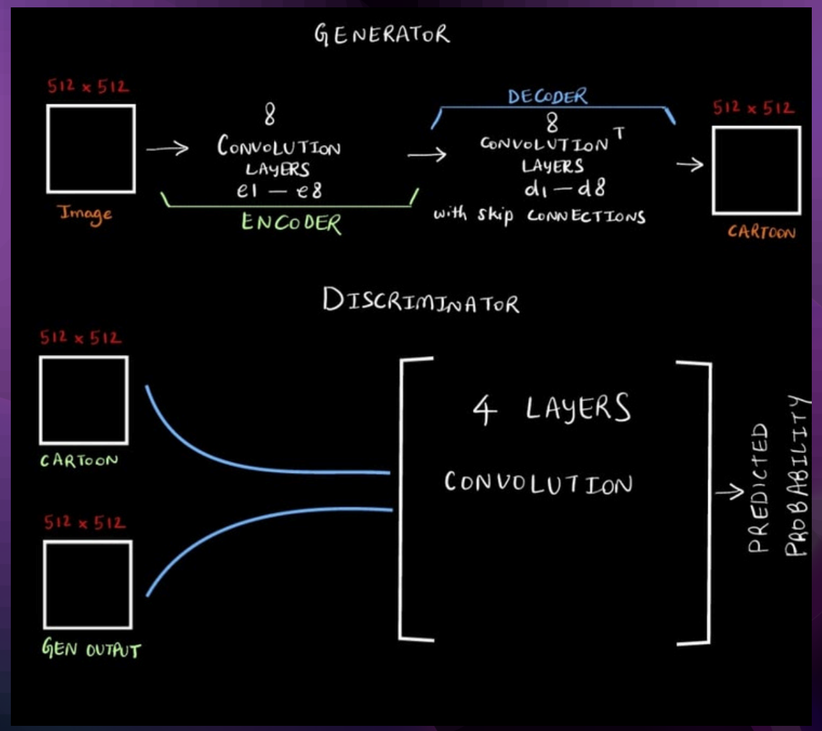
OG REPO ARCHITECTURE



Uses a **Generator** consisting of **Encoder** and **Decoder**
Encoder - **Conv2D layers** that reduce input dimension in terms of H and W and typically generates output that has many channels.
Decoder - Output of Encoder → **Transpose Conv layers** and **skip connections** → reconstructs final image dimension

OUR INITIAL ARCHITECTURE

Generator :
Input image converted to 512*512 and normalised to (-1,1)
8 layers of Conv2D as encoding in **2*2*512**
8 layers of **transpose conv2D** layers with skips decoding to **RGB image(3 channels)**
Mirrors encoder's layer count to match with the dimensions for concatenation
Uses **LeakyReLU** everywhere except final layer to normalise, then denormalise



AND FINALLY

Converted the final kernel of encoeder alone to size 5*5 to get better features

Added Multi-Headed Attention (8 heads) to the last layer of encoder to allow better generalization within each channel of encoder's last layer

The problems with an imbalanced dataset were evident when testing the model(consistently worse and blurred outputs when darker images were fed)

Without MHA, a little bit of smoothening was externally applied using cv2, for better outputs

OUTPUTS (with MHA from training logs)

Initial stages of training with Multi-Head Attention(8) applied to the generator's encoder

Output after 800 steps(1st epoch)

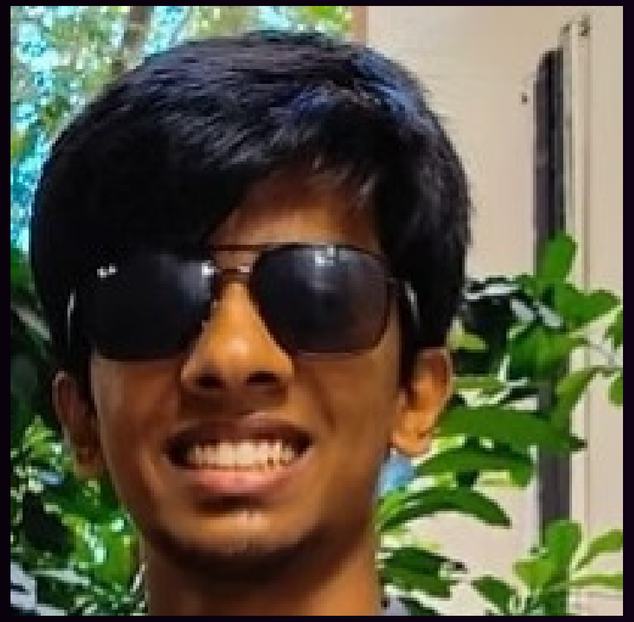
Midway through training with Multi-Head Attention(8) applied to the generator's encoder

Output after 20,000 steps(5th epoch)

Trained with Multi-Head Attention(8) applied to the generator's encoder

Output after 42,000+ steps(6th epoch)

Results (with Attention)



Honorable mentions: Shailesh and Musk

REFERENCES

GANs and their applications

https://arxiv.org/pdf/1701.0010

GANs Paper

https://arxiv.org/abs/1406.2661

CGANs Paper

https://arxiv.org/abs/1411.1784

Pix2Pix Paper

https://arxiv.org/abs/1611.07004

CartoonGAN

https://paperswithcode.com/paper/cartoongan-generative-adversarial-networks

Video on GANs

https://youtu.be/RAa55G-oEuk?si=5SC0Z60-Svtu8zEd

Our repo :)

https://github.com/AnshumanMishra21345/Pix2Pix-Implementation

THANK YOU!