Facial Deocclusion - Remove objects covering your face

1. Definition

1.1. Domain Background

Facial deocclusion is the process of reconstructing occluded (covered) parts of a face in an image using AI techniques. This task holds significant value in various real-world scenarios, such as presentations, livestreaming, and security. By enhancing visibility of facial features, this technology can improve user experiences in situations where parts of the face may be obscured by objects like masks, sunglasses, hands, microphones, or cups. It aligns with existing innovations like Eyes Focus, aiming to bring clearer, more engaging interactions during live or recorded video feeds. Additionally, with further refinement, it has potential applications in security for facial recognition even under partial occlusions.

1.2 Problem Statement

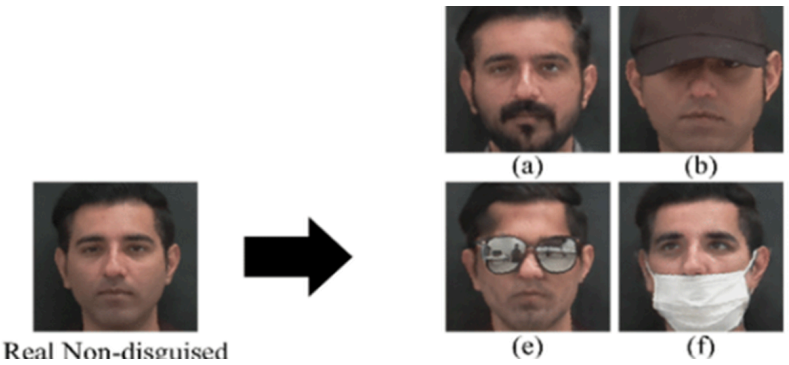
The goal of this project is to develop an AI-based system capable of removing occlusions from facial images, thereby reconstructing hidden facial parts as naturally and accurately as possible. The project aims to create a model that can restore facial features by intelligently filling in areas obscured by various objects. This would have practical applications in ensuring seamless presentations, livestreams, and video communications, especially when external elements partially cover the face.

1.3 Metrics

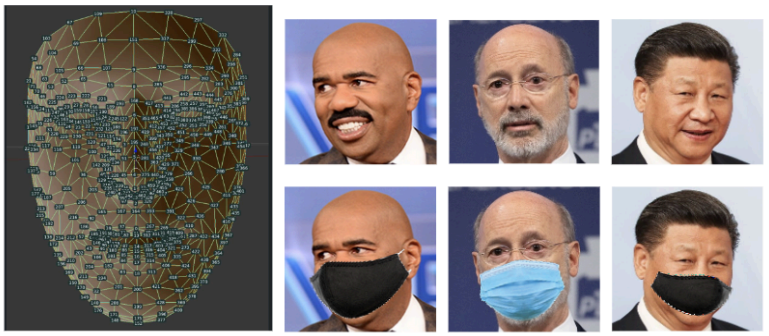
The project will primarily evaluate the model's performance using metrics such as Mean Squared Error (MSE), Structural Similarity Index (SSIM). SSIM will help assess the perceptual quality of the deoccluded images, while MSE will provide a more quantitative analysis of pixel-level accuracy. A higher SSIM, coupled with a lower MSE, will indicate better performance in restoring the occluded facial parts. In this Capstone project, I also use SSIM and MSE as loss functions when training the Generator model.

2. Analysis

Here my dataset's URL on Kaggle Facial-Deocculsion. At the beginning of the project, my initial plan was to find a dataset with pre-existing human faces (I used the ClebA-HQ with available face images) and search for transparent images of objects that often appear on faces. I would then overlay these onto the face images from the ClebA-HQ to create a dataset like the one shown below. However, due to limited time, I mostly gathered transparent images of face masks, which varied significantly in type, color, and angle (after COVID, such data became widely available and diverse due to the wave of training models for mask detection). This worked well with face images that also had multiple angles. Although I also collected images of glasses, beards, and hats, they lacked diversity in color, type, and angle, so I decided to use only face masks. This approach presents its own challenge, as masks cover a large portion of the face. However, if I can reconstruct a face behind a mask, I believe it should be possible and manageable to reconstruct a face obscured by smaller objects as well.

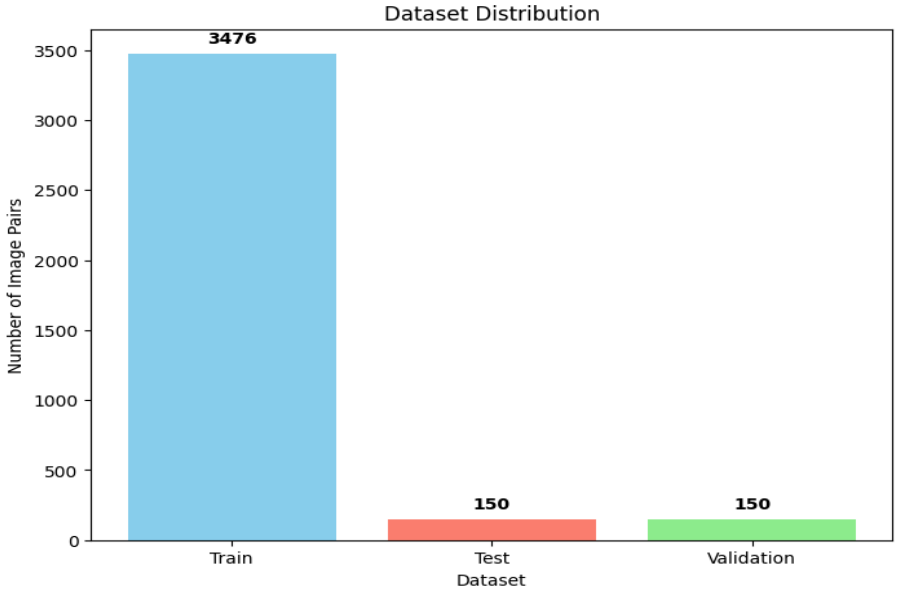


I used Mediapipe to detect facial landmarks, then calculated the viewing angle and face orientation based on vectors created by these points. This allows each face image to have a mask applied in a realistic and well-aligned manner.



I

In the Celeb dataset, there are multiple folders, each named after a celebrity and containing images of that particular celebrity's face. I randomly selected 3,776 celebrities and took only one image per celebrity to add a mask and include it in the dataset. Then, I split it into three sets—train/validation/test—with the following counts: 3,476 for training, 150 for validation, and 150 for testing. You can share the link to your Kaggle dataset, and I'll help ensure it's formatted or referenced correctly if needed.



Each train/test/validation folder contains two subfolders: Mask and Unmask. The Unmask folder includes the original images from CelebA with the format [image_index.png], while the Mask folder contains images with masks applied, formatted as [image_index_Mask.jpg]. An image pair in these two folders will share the same image_index.

Example Structure

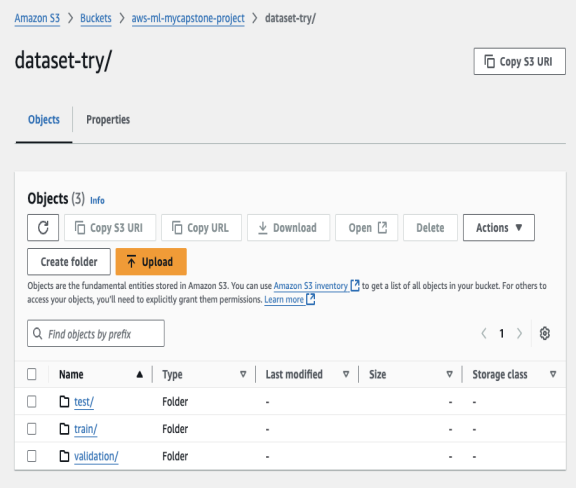
```
dataset/
├── train/
│   ├── Mask/
│   │   ├── 0001_Mask.jpg
│   │   ├── 0002_Mask.jpg
│   │   └── ...
│   └── Unmask/
│       ├── 0001.png
│       ├── 0002.png
│       └── ...
├── validation/
│   ├── Mask/
│   └── Unmask/
└── test/
    ├── Mask/
    └── Unmask/
```

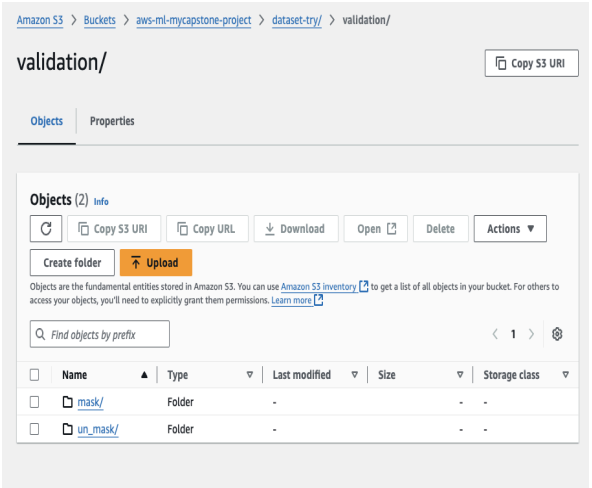
3. Methology

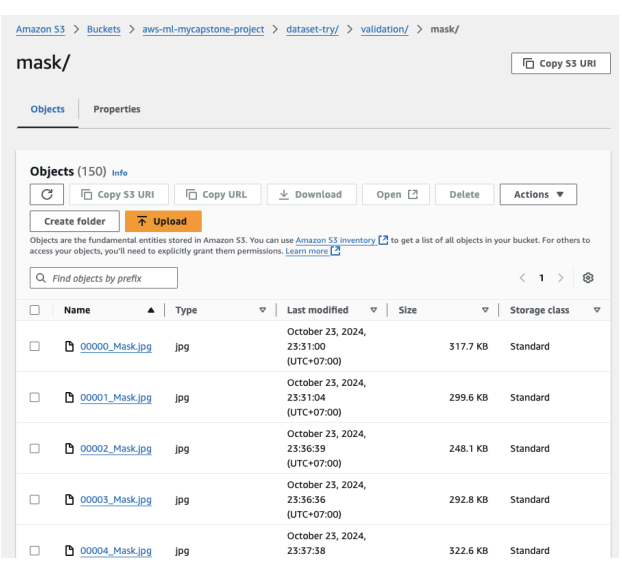
3.1. Data Preprocessing

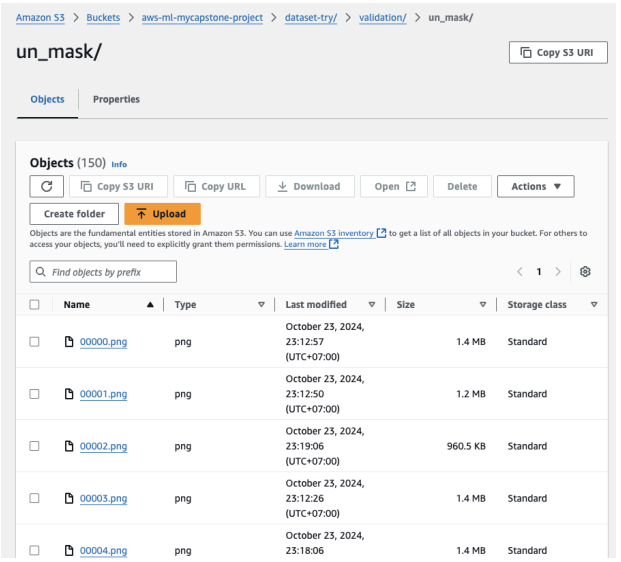
After creating the dataset as described in the Analysis section, I found it more convenient and time-saving to use it if I plan to train my model with a SageMaker Notebook or EC2 Instance, so I created a copy on S3.

My S3 Bucket screenshots about dataset:









I load each image pair from S3 using TensorFlow => Normalize to the range (-1,1) => Resize to dimensions (256, 256, 3). You can see more details in my source code.

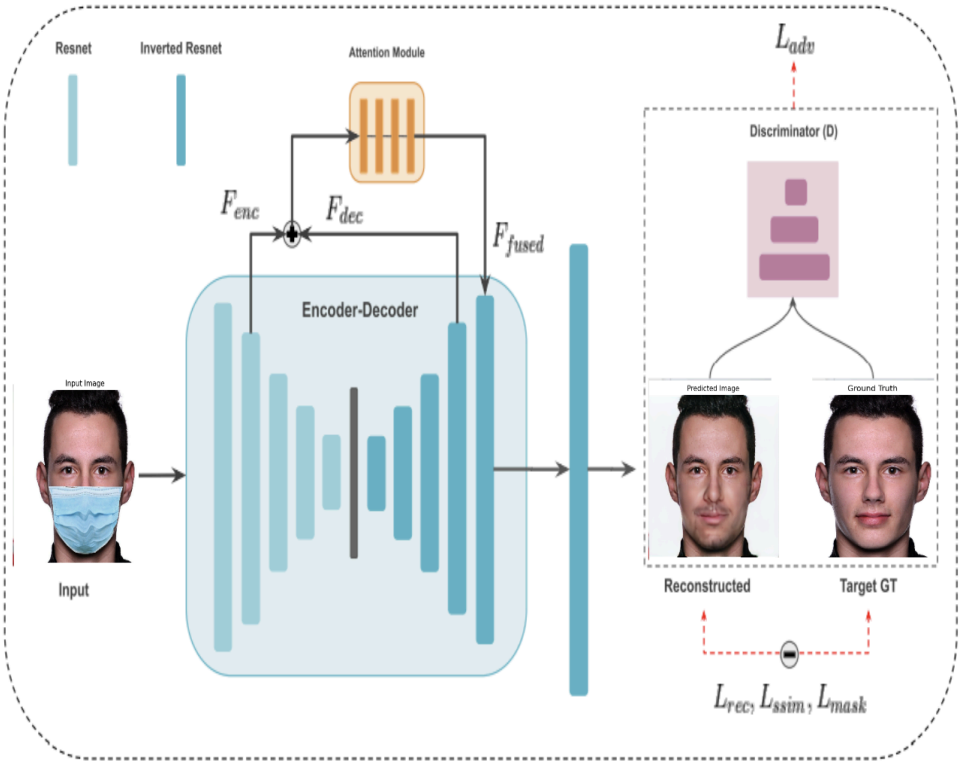
S3 Data Ingestion ——> (Image_Mask, Image_Unmask) ——> Image Normalization ——> Image Resizing ——> (Image_Mask_loaded, Image_Unmask_loaded).

3.2. Implementation:

3.2.1. Model Selected:

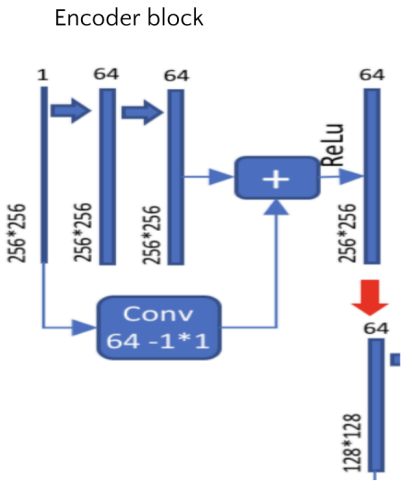
For this task, I consider it an image generation problem, so I apply the GAN concept to this problem.

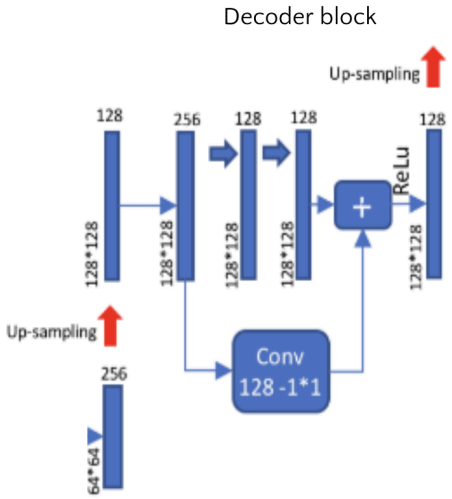
Model Architecture:



With Discriminator, It's simply a standard CNN classification model to distinguish between real and fake images.
With Generator model, we use a structure similar to a UNet network with two blocks: Encoder and Decoder. In addition, we combine it with an Attention Module to enhance content preservation in the images.

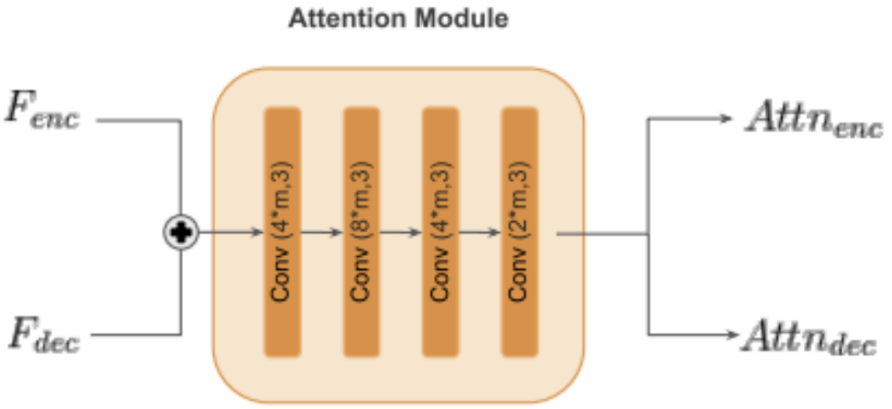
- Encoder: ResNet Block:
- Decoder: Inverted ResNet Blocks

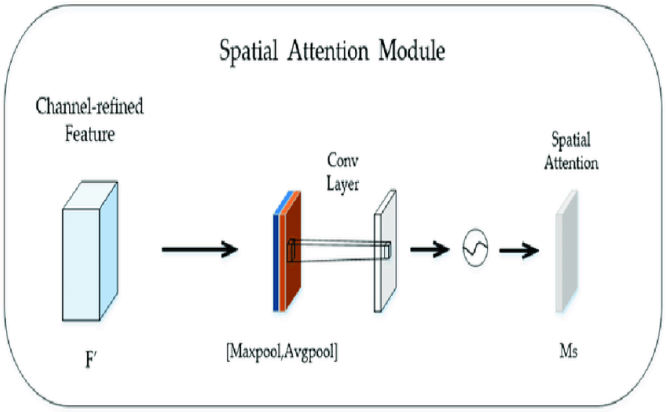




- The Attention Module is a block with four convolutional layers using Spatial Attention as input. (Read more about it)

$$F_{fused} = F_{enc} * Attn_{enc} + F_{dec} * Attn_{dec}$$





Reasons for choosing this model:

- The model meets the de-occlusion requirements.
- This approach is an end-to-end inpainting method that learns to fill in user-specific details.
- No need for 3D face-to-face reconstruction.
- Easy to train, customize, and fine-tune, and maybe it will be lightweight.

3.2.2. Loss Functions

I use three loss functions for training this GAN network.

3.2.2.1. Loss Reconstruction

First, in order to penalize reconstruction errors, we use pixel-based L1 loss.

$$L_{rec} = \left\| X_{gt} - X_{rec} \right\|_1$$

Where Xgt is image Ground Truth and Xrec is Image from Generator Model.
However, using only the L1 reconstruction loss produces blurry outputs.

3.2.2.2. Structural Similarity Index Measure (SSIM)

An image quality metric that assesses the visual impact of three characteristics of an image: luminance, contrast and structure.

$$SSIM(x, y) = [l(x, y)]^{\alpha} \cdot [c(x, y)]^{\beta} \cdot [s(x, y)]^{\gamma}$$

where

$$l(x, y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1},$$

$$c(x, y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2},$$

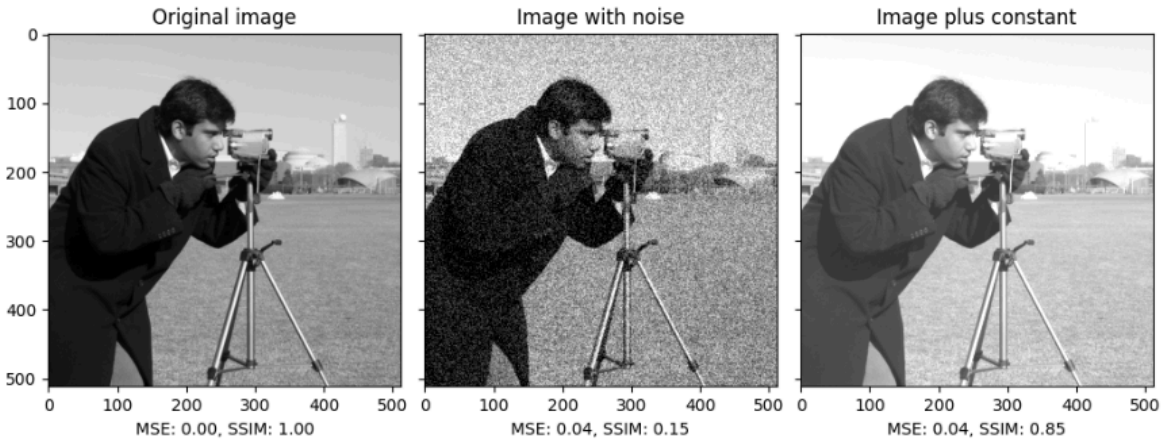
$$s(x, y) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3}$$

where μ_x, μ_y, σ_x,σ_y, and σ_{xy} are the local means, standard deviations, and cross-covariance for images x, y. If $\alpha = \beta = \gamma = 1$ (the default for `Exponents`), and $C_3 = C_2/2$ (default selection of C_3) the index simplifies to:

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

A generated image is good if:

- The pixels have different levels of light and dark, and the more dark levels there are and the more details there are => Good quality photos
- A photo is not the higher the contrast, the better, but the balance between light and dark. => Skin variety.



SSIM has a value between -1 and 1, reaching a value of 1 in the case of two identical tuples. The larger this index is, the better it corresponds to the model.

$$\mathcal{L}_{SSIM} = 1 - SSIM$$

3.2.2.3. The Adversarial Loss

Discriminator in the architecture to compute the adversarial loss. This Adversarial Loss term forces the encoder-decoder to reconstruct high-fidelity outputs by sharpening the blurred images. For Discriminator models we adopt the architecture of the DCGAN discriminator.

$$L_{adv} = log(D(X_{gt})) + log(1 - D(X_{rec}))$$

3.2.2.4. Loss total

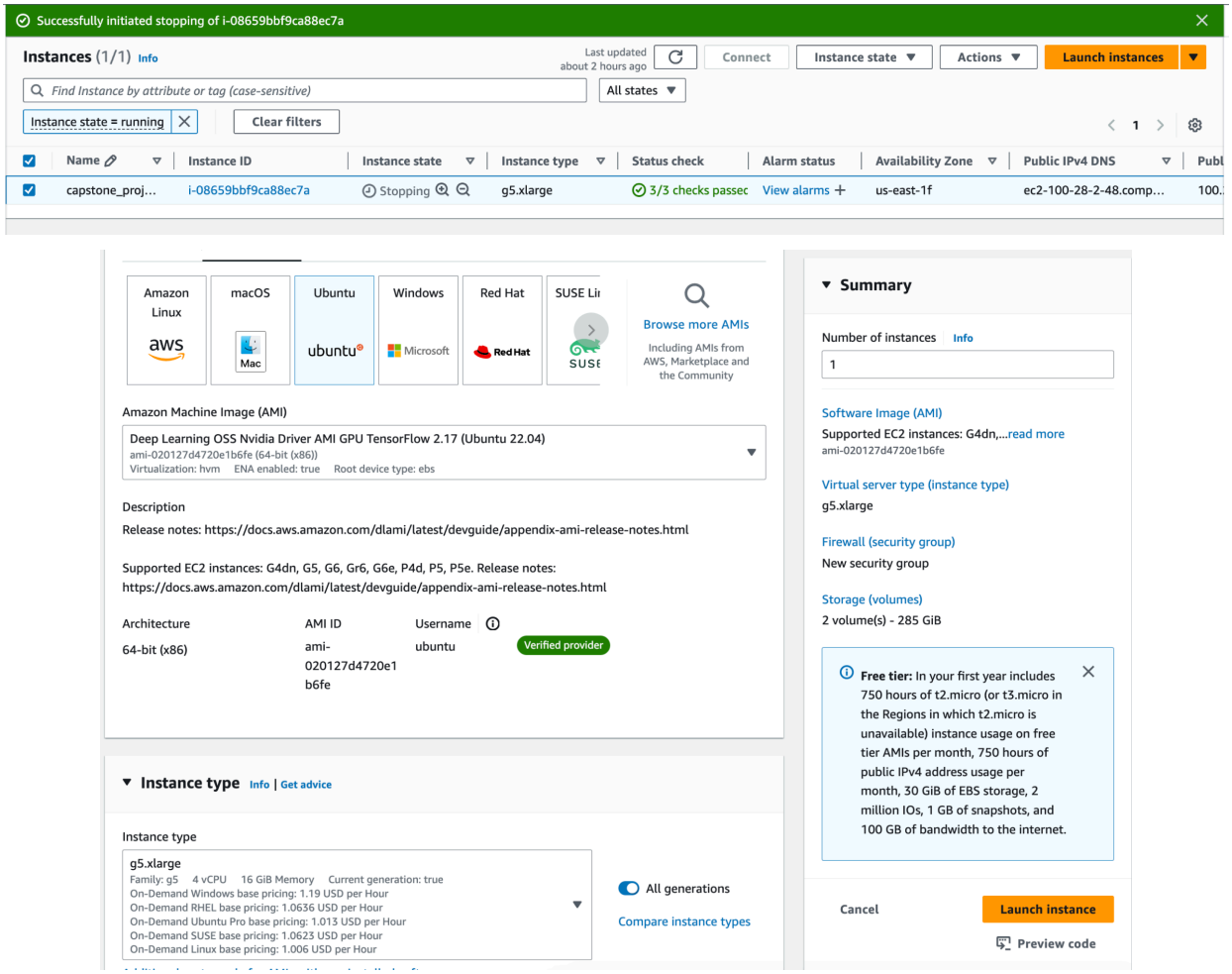
The final training objective loss function can be written as below formula. Where λrec, λadv, λssim are the corresponding weight parameters for each loss term. Throughout the training and testing process, these three values remained constant: λrec = 1.2, λadv = 0.5, and λssim = 80, to facilitate comparison across experiments. Value of λssim is the largest because I wanted it to have the most significant impact on my training process.

$$Loss = \lambda_{rec} * L_{rec} + \lambda_{adv} * L_{adv} + \lambda_{ssim} * L_{ssim}$$

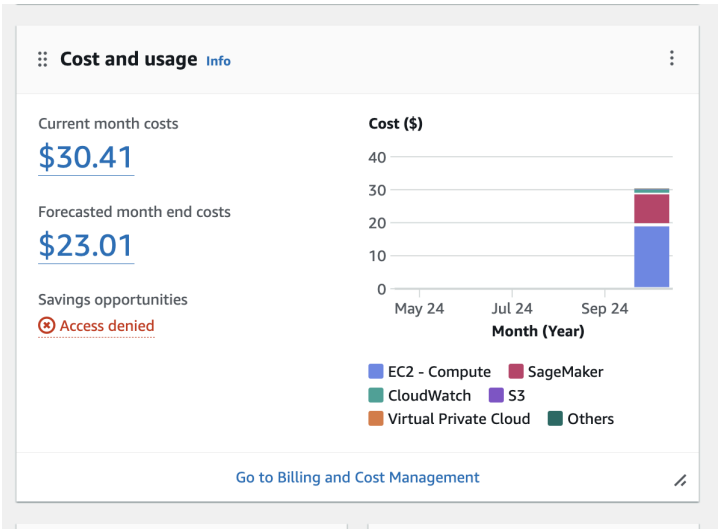
Here is the formula used when training the Generator model, which corresponds to training a Gen model based on L1, SSIM loss, and the Discriminator's judgment capability, thereby leading to improved training outcomes for the Generator.

3.3. Refinement

At the beginning of the training process, I used a SageMaker notebook. However, since I needed to modify and experiment multiple times, I created an EC2 instance Ubuntu with the g5.xlarge type, 16GB RAM, and a GPU. This setup has made it more convenient for training my model. All my source code is using Tensorflow so in I also choose AMI have GPU Tensorflow too.



I have used $30 out of the $75 allowed by the course for this project.



3.3.1 Traning Error

In the first training, the definition of Loss SSIM function was wrong, so the model could not generate images with correct color. Loss_ssim = (SSIM) with SSIM in range (-1,1) so loss can't approaches 0.

=> Fix the loss function SSIM: Loss_ssim = (SSIM) => Loss_ssim = (1-SSIM).

After that, i have another error when training. At the output layer, the output image is normalized in the range (0,1) but the input image for training is normalized (-1,1). The generated model image does not have the required amount of darkness.

=> Changed the activation function at the output layer from ReLU() to Tanh().

3.3.2. Experiment

3.3.2.1 Training without Attention Module

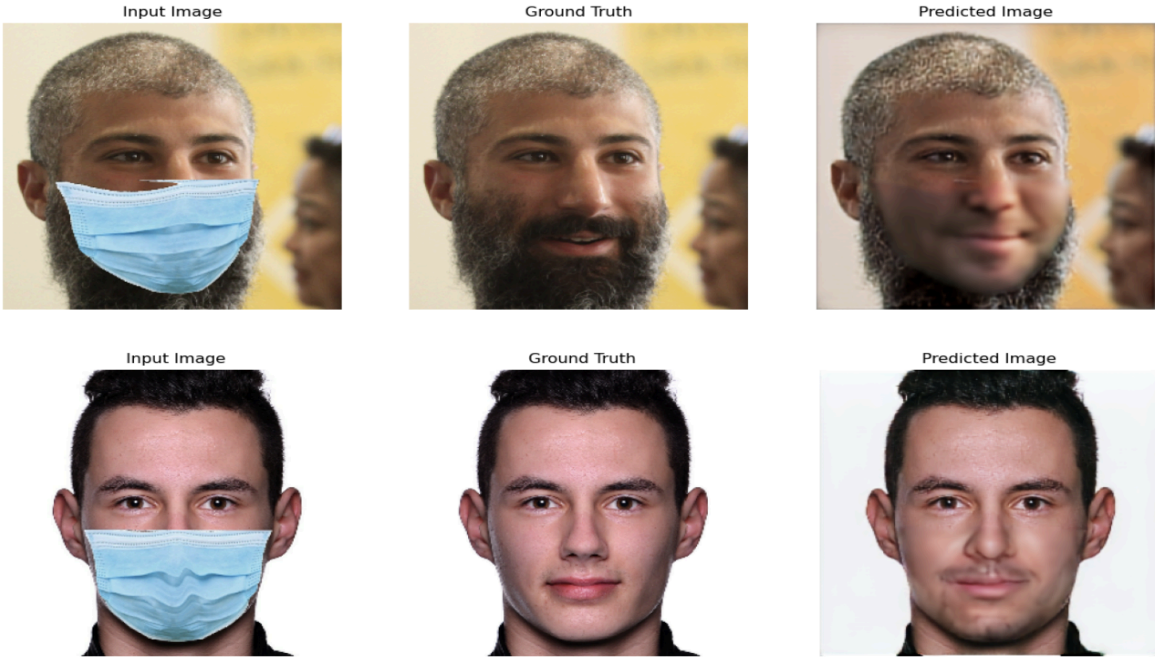
First the group trained the model with no Attention module with 100 epoch, batch size 8. The generated image fills the lips and nose quite harmoniously. However, the image quality is blurry, not clear. I called it the Base Model.

3.3.2.2 Traning with Attention Module

After that, I trained model 300 epoch, batch size 8 with Attention module. The generated image is clear and detailed, but there is a white part of the mouth that cannot be filled.
I think this issue arises because the model lacks the ability to decide or, more precisely, is still unsure whether to recreate a smiling mouth with visible teeth or a closed mouth.
Meanwhile, the model without attention that I previously trained didn't encounter this issue.

3.3.2.3. Fine - Tune Based Model with Attention module

Based on my judgment, I loaded the untrained Base model with the previous Attention model's knowledge using Class Name and retrained it for 200 epochs. Fortunately, this solved the issue, and now the model is both sharp and able to fill in the gap in the mouth area.



4. Results

4.1. Model Evaluation and Validation

Below is a table summarizing the loss function values from the training results across different attempts.

Training Attempt	val_total_loss	val_ssim	val_rec	val_adv
Base model without Attention Module	6.55	0.11	0.045	0.15
Training from scratch with Attention Module	6.55	0.11	0.045	0.15
Load base model, Attention by class name	5.64	0.103	0.055	0.21

As for the recognition results, you can view the output images from each process listed above.

4.2. Justification

I have prepared images of people wearing masks collected from the internet; however, the results were not as expected.

It seems the model is overfitting on my dataset, even though the distribution, position, and distance of faces in these images differ from the training data (original images in the CelebA dataset usually have similar face size, position, and scale). In the future, I plan to focus on the masked area only, training the model to fill just that specific region. This approach will speed up training and help the model concentrate on the critical areas that genuinely need filling.

Additionally, I may expand the dataset with a variety of objects, not just masks.
Through the training and evaluation process, I believe the current solution has high practical potential, although more time is needed for improvement.