Final Project Report

CISC6000 Fordham University
Xingrou Mei
Ryan L'Abbate
George Kuliner

Introduction

The goal of our project is to see if deep learning models can learn the animation styles of different studios or mediums. We chose Pixar and Studio Ghibli as our movie studios. We chose these studios because of their distinctive styles as well as each works in a single medium. Pixar films use 3-Dimensional (3D) animation and Studio Ghibli uses 2-Dimensional (2D) animation.

Datasets

Pixar 3D animations and Ghibli 2D animations:

- Pixar (3D)
  - Toy Story 3 (3.27GB)
  - Luca (1.58GB)
  - Inside Out (1.79GB)
  - Finding Nemo (2.54GB)
- Studio Ghibli (2D)
  - Ponyo (4.24GB)
  - Spirited Away (3.44GB)
  - Princess Mononoke (2.55GB)
  - My Neighbor Totoro (2.51GB)

The data comes from animation screen captures (i.e. screenshots) of various animated movies. The data can be found and downloaded at https://animationscreencaps.com/. The data is available for educational and non-commercial purposes. The website allows for direct zip downloads of the movies for free, however the speed is heavily limited to about 20-36 hours depending on the movie length as faster download speed is locked behind a paywall. To speed up the download process for free, we created a script, 'image_fetcher.py', this allowed us to download movies in 3-5 hours depending on the movie length. The script is located in this submission folder with directions to run in README word file. We were able to accomplish this using the requests library in python.

The dataset consists of JPG screenshots or frames of various movies. For every second of movie there are two frames (2 frames per second) as opposed to the normal 30-60 fps. The images will be downsized to the dimensions 256x256 and tagged with relevant information such as medium (2-D animation and 3-D animation), studio (Pixar and Ghibli). The tags will be used to train the model on different visual styles and generate new versions of the images in different styles.

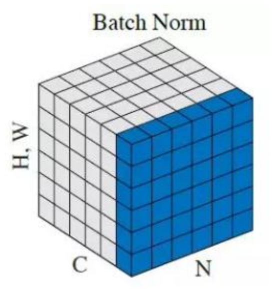
CycleGAN

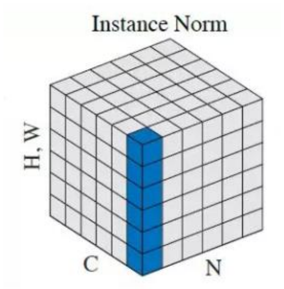
(based on the cycleGAN in Tensorflow tutorial)

This project uses the Kaggle environment to train the CycleGAN model with a GPU. The implementation in the Tensorflow tutorial is very similar to this project. The tutorial trains a model to transform images of horses to zebras and zebras to horses. For CycleGAN, it needs to have 2 generators and 2 discriminators.

For the generator, this model uses U-Net-based architecture. CycleGAN is very similar to CoGan, also called PixtoPix. The difference is CycleGAN uses instance normalization, and CoGAN uses batch normalization. CycleGAN also included two more loss functions besides generator loss and discriminator loss.

Instance normalization:
The mean and variance are calculated for each individual channel for each individual sample across both spatial dimensions as shown below.





There are 2 generators and 2 discriminators in cyclegan:
Generator G translates image from X to Y
$(G : X -> Y)$
Generator F translates image from Y to X
$(F : Y -> X)$

Discriminator Dx differentiate between image X and generated image of X that learns from image Y
Discriminator Dy differentiate between image Y and generated image of Y that learns from image X

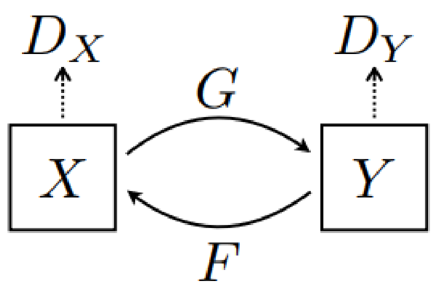


Image processing:
Before building the model, the images are reshaped the image size and normalized the images to a NumPy array. This project also cropped the images to the center of the images.

- Central crop: Crop the central region of the images
- Reshape to image size 256 x 256
- Normalized

Generator

Encoder (Downsampling):
Each block has a combination of below:
Convolution, Instance Normalization, Leaky ReLU.

All convolutional layers have the same kernel size of 4x4, strides of 2x2, and the same padding. All blocks have an instance normalization layer except the first block. The encoder uses LeakyReLU as an activation layer.

Input layer: shape = (256,256,3)
Convolutional layer: filters= 64, kernel_size = 4, strides = 2, padding = ‘same’ **Activation layer**: Leaky ReLU
Convolutional layer: filters= 128, kernel_size = 4, strides = 2, padding = ‘same’ **Instance normalization** **Activation layer:** Leaky ReLU
Convolutional layer: filters= 256, kernel_size = 4, strides = 2, padding = ‘same’ **Instance normalization** **Activation layer**: Leaky ReLU
Convolutional layer: filters= 512, kernel_size = 4, strides = 2, padding = ‘same’ **Instance normalization** **Activation layer**: Leaky ReLU
Convolutional layer: filters= 512, kernel_size = 4, strides = 2, padding = ‘same’ **Instance normalization** **Activation layer**: Leaky ReLU
Convolutional layer: filters= 512, kernel_size = 4, strides = 2, padding = ‘same’ **Instance normalization**

Activation layer: Leaky ReLU

Decoder (Upsampling):
Each block have a combination of below:
Transposed Convolution, Instance Normalization, Dropout, ReLU

Transposed Convolution layer: filters= 512, kernel_size = 4, strides = 2, padding = ‘same’
Instance Normalization
Dropout layer: rate = 0.5
Activation layer: ReLU

Transposed Convolution layer: filters= 512, kernel_size = 4, strides = 2, padding = ‘same’
Instance Normalization
Dropout layer: rate = 0.5
Activation layer: ReLU

Transposed Convolution layer: filters= 512, kernel_size = 4, strides = 2, padding = ‘same’
Instance Normalization
Activation layer: ReLU

Transposed Convolution layer: filters= 256, kernel_size = 4, strides = 2, padding = ‘same’
Instance Normalization
Activation layer: ReLU

Transposed Convolution layer: filters= 128, kernel_size = 4, strides = 2, padding = ‘same’
Instance Normalization
Activation layer: ReLU

Transposed Convolution layer: filters= 64, kernel_size = 4, strides = 2, padding = ‘same’
Instance Normalization
Activation layer: ReLU

Below is a plot_model diagram from Keras:
[Generator plot.pdf]
**diagram is large, use above pdf link, if it does not work, please go to “gan” folder, generator_model.jpg*

Discriminator

Each block have a combination of below:
Convolution, Instance Normalization, Leaky ReLU
All blocks below are similar to the generator, except it includes a layer of zero padding.

Input layer: shape = (256,256,3)

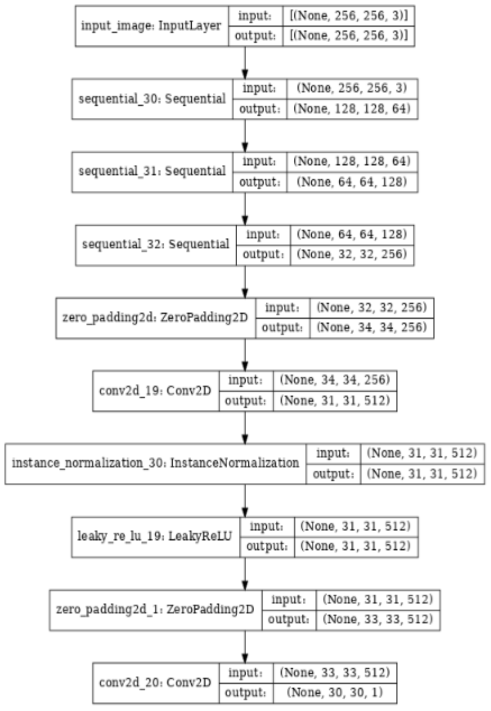
Convolutional layer: filters= 64, kernel_size = 4, strides = 2, padding = ‘same’
Activation layer: Leaky ReLU

Convolutional layer: filters= 128, kernel_size = 4, strides = 2, padding = ‘same’
Activation layer: Leaky ReLU

Convolutional layer: filters= 256, kernel_size = 4, strides = 2, padding = ‘same’
Activation layer: Leaky ReLU
Zero Padding

Convolutional layer: filters= 512, kernel_size = 4, strides = 2, padding = ‘same’
Instance normalization
Activation layer: Leaky ReLU
Zero Padding

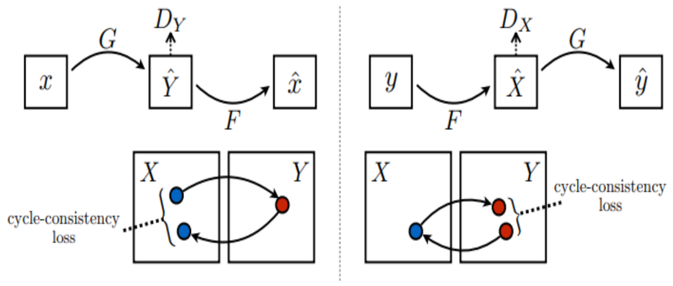
Below is a plot_model diagram from Keras:


input_image: InputLayer | input: [(None, 256, 256, 3)] | output: [(None, 256, 256, 3)]
sequential_30: Sequential | input: (None, 256, 256, 3) | output: (None, 128, 128, 64)
sequential_31: Sequential | input: (None, 128, 128, 64) | output: (None, 64, 64, 128)
sequential_32: Sequential | input: (None, 64, 64, 128) | output: (None, 32, 32, 256)
zero_padding2d: ZeroPadding2D | input: (None, 32, 32, 256) | output: (None, 34, 34, 256)
conv2d_19: Conv2D | input: (None, 34, 34, 256) | output: (None, 31, 31, 512)
instance_normalization_30: InstanceNormalization | input: (None, 31, 31, 512) | output: (None, 31, 31, 512)
leaky_re_lu_19: LeakyReLU | input: (None, 31, 31, 512) | output: (None, 31, 31, 512)
zero_padding2d_1: ZeroPadding2D | input: (None, 31, 31, 512) | output: (None, 33, 33, 512)
conv2d_20: Conv2D | input: (None, 33, 33, 512) | output: (None, 30, 30, 1)

Loss functions

In cycleGAN, it is an unpaired data training. Therefore, there is an additional loss, called cycle consistency loss, to help the network learn the correct mapping. Besides Discriminator loss and Generator loss, there is a cycle consistency loss.

Cycle consistency loss



For example, if one translates a sentence from English to French, and then translates it back from French to English. If the difference between the original English sentence is the same as the English that was translated from French, then this is a good generator.

Identity loss

If you run the Pixar-to-Ghibli style model on a Ghibli dataset or vice versa, it should not modify the image much.

Optimizers

The model is using Adam for generator and discriminator:

Learning_rate = 0.0002

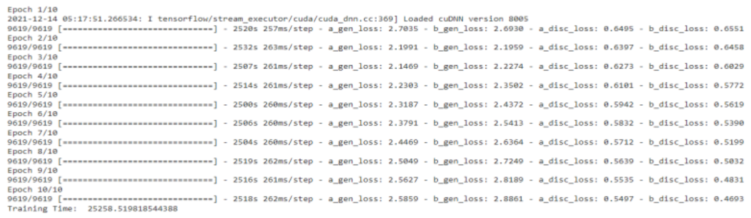
Beta_1 = 0.5 (the exponential decay rate for the 1st moment estimates)

Training the model:

Fitting the model with 10 epochs.
The generated images are somewhat learning the style from the other studio. 10 epochs on this model take about 7-9 hours to train. However, the result will be better if the model trains with more epochs.

Since Kaggle has 9 hours limits, this model cannot continue training with more epochs.

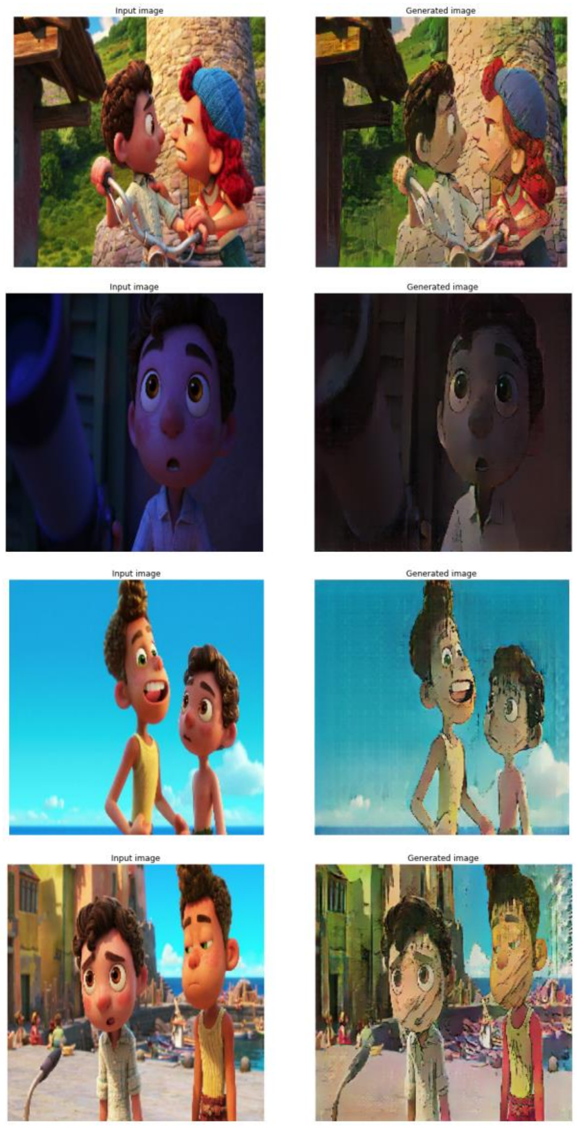
Luca and Spirited Away Training log:



```
Epoch 1/10
2021-12-14 05:17:51.266534: I tensorflow/stream_executor/cuda/cuda_dnn.cc:369] Loaded cuDNN version 8005
9619/9619 [==============================] - 2520s 257ms/step - a_gen_loss: 2.7035 - b_gen_loss: 2.6930 - a_disc_loss: 0.6495 - b_disc_loss: 0.6551
Epoch 2/10
9619/9619 [==============================] - 2532s 263ms/step - a_gen_loss: 2.1991 - b_gen_loss: 2.1959 - a_disc_loss: 0.6397 - b_disc_loss: 0.6458
Epoch 3/10
9619/9619 [==============================] - 2507s 261ms/step - a_gen_loss: 2.1469 - b_gen_loss: 2.2274 - a_disc_loss: 0.6273 - b_disc_loss: 0.6029
Epoch 4/10
9619/9619 [==============================] - 2514s 261ms/step - a_gen_loss: 2.2303 - b_gen_loss: 2.3502 - a_disc_loss: 0.6101 - b_disc_loss: 0.5772
Epoch 5/10
9619/9619 [==============================] - 2500s 260ms/step - a_gen_loss: 2.3187 - b_gen_loss: 2.4372 - a_disc_loss: 0.5942 - b_disc_loss: 0.5619
Epoch 6/10
9619/9619 [==============================] - 2506s 260ms/step - a_gen_loss: 2.3791 - b_gen_loss: 2.5413 - a_disc_loss: 0.5832 - b_disc_loss: 0.5390
Epoch 7/10
9619/9619 [==============================] - 2504s 260ms/step - a_gen_loss: 2.4469 - b_gen_loss: 2.6364 - a_disc_loss: 0.5712 - b_disc_loss: 0.5199
Epoch 8/10
9619/9619 [==============================] - 2519s 262ms/step - a_gen_loss: 2.5049 - b_gen_loss: 2.7249 - a_disc_loss: 0.5639 - b_disc_loss: 0.5032
Epoch 9/10
9619/9619 [==============================] - 2516s 261ms/step - a_gen_loss: 2.5627 - b_gen_loss: 2.8189 - a_disc_loss: 0.5535 - b_disc_loss: 0.4831
Epoch 10/10
9619/9619 [==============================] - 2518s 262ms/step - a_gen_loss: 2.5859 - b_gen_loss: 2.8861 - a_disc_loss: 0.5497 - b_disc_loss: 0.4693
Training Time:  25258.519818544388
```

Generated Images

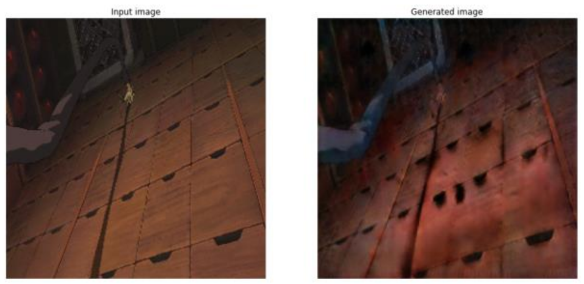
Luca(Pixar 3D) to Spirited Away(Ghibli 2D) style :

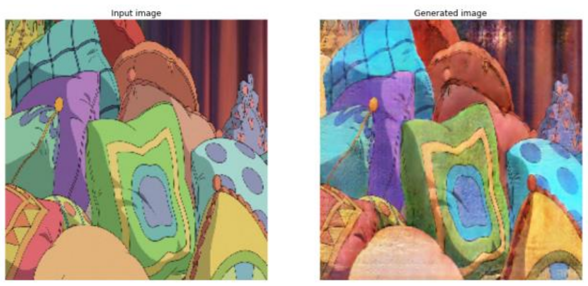


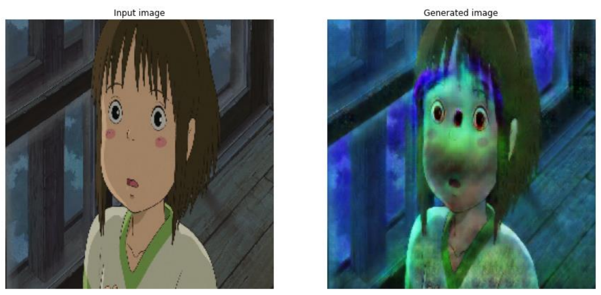
left = original image, right = generated image

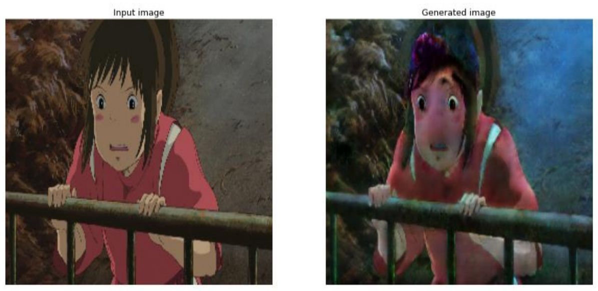
**More images at the end of the report, Appendix*

Spirited Away (Ghibli 2D) to Luca (Pixar 3D) style:





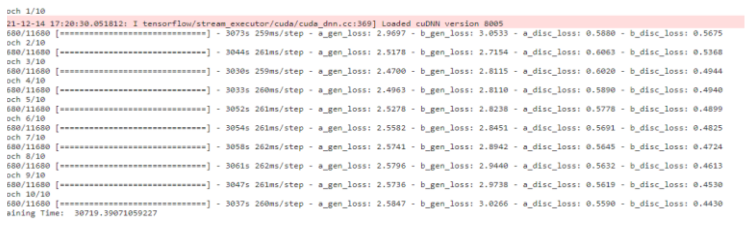




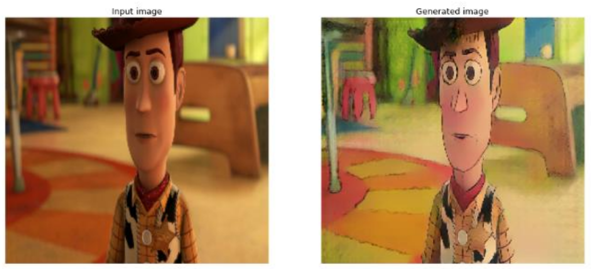
left = original image, right = generated image
**More images at the end of the report, Appendix*

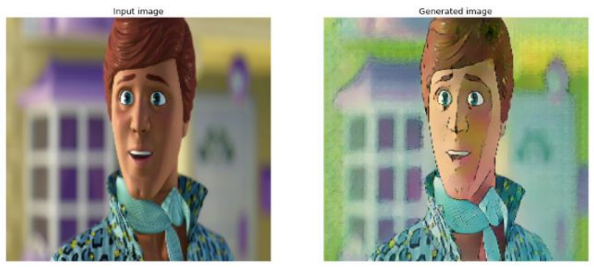
As shown above, some images are learning better than others. The images that are colorful and have more details are harder to train. It is very scratchy, but it flattens out the 3-dimensional style. For 2D to 3D style transfer, the images with faces look like Pixar 3D animation characters. The faces are more pop and puffy. The landscape images are not as visual as the face images. However, the landscape images look like it is in the process of translating to 3D animation. It needs more time to train the model to transform the images smoothly.

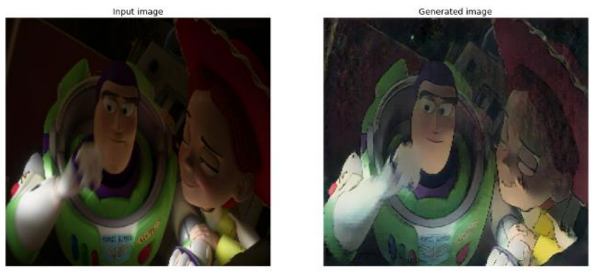
Toy Story 3 and Ponyo Training log:

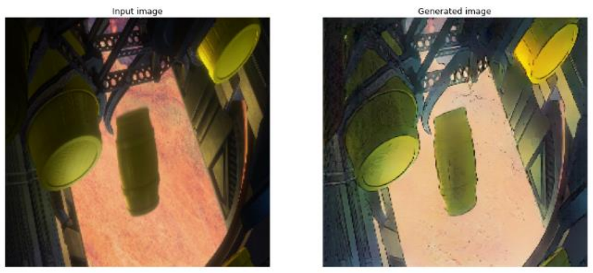

```
och 1/10
21-12-14 17:20:30.051812: I tensorflow/stream_executor/cuda/cuda_dnn.cc:369] Loaded cuDNN version 8005
680/11680 [==============================] - 3073s 259ms/step - a_gen_loss: 2.9697 - b_gen_loss: 3.0533 - a_disc_loss: 0.5880 - b_disc_loss: 0.5675
och 2/10
680/11680 [==============================] - 3044s 261ms/step - a_gen_loss: 2.5178 - b_gen_loss: 2.7154 - a_disc_loss: 0.6063 - b_disc_loss: 0.5368
och 3/10
680/11680 [==============================] - 3030s 259ms/step - a_gen_loss: 2.4700 - b_gen_loss: 2.8115 - a_disc_loss: 0.6020 - b_disc_loss: 0.4944
och 4/10
680/11680 [==============================] - 3033s 260ms/step - a_gen_loss: 2.4963 - b_gen_loss: 2.8110 - a_disc_loss: 0.5890 - b_disc_loss: 0.4940
och 5/10
680/11680 [==============================] - 3052s 261ms/step - a_gen_loss: 2.5278 - b_gen_loss: 2.8238 - a_disc_loss: 0.5778 - b_disc_loss: 0.4899
och 6/10
680/11680 [==============================] - 3054s 261ms/step - a_gen_loss: 2.5582 - b_gen_loss: 2.8451 - a_disc_loss: 0.5691 - b_disc_loss: 0.4825
och 7/10
680/11680 [==============================] - 3058s 262ms/step - a_gen_loss: 2.5741 - b_gen_loss: 2.8942 - a_disc_loss: 0.5645 - b_disc_loss: 0.4724
och 8/10
680/11680 [==============================] - 3061s 262ms/step - a_gen_loss: 2.5796 - b_gen_loss: 2.9440 - a_disc_loss: 0.5632 - b_disc_loss: 0.4613
och 9/10
680/11680 [==============================] - 3047s 261ms/step - a_gen_loss: 2.5736 - b_gen_loss: 2.9738 - a_disc_loss: 0.5619 - b_disc_loss: 0.4530
och 10/10
680/11680 [==============================] - 3037s 260ms/step - a_gen_loss: 2.5847 - b_gen_loss: 3.0266 - a_disc_loss: 0.5590 - b_disc_loss: 0.4430
aining Time:  30719.39071059227
```

Toy Story 3 (Pixar 3D) to Ponyo (Ghibli 2D) style :



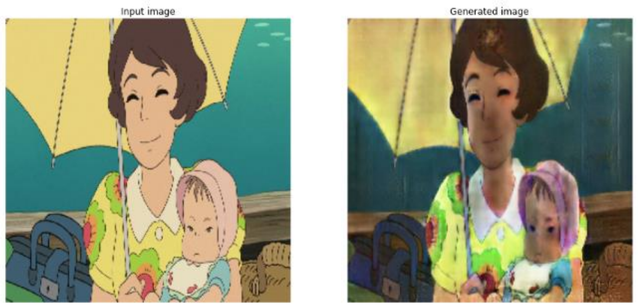


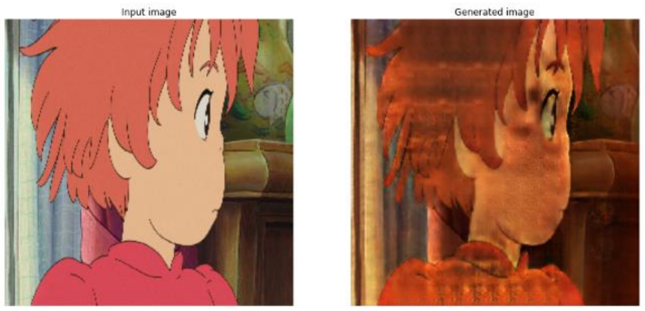


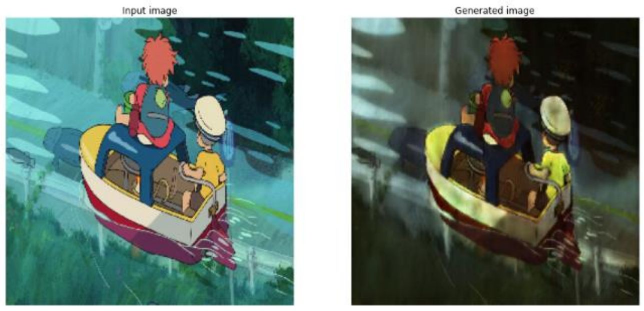


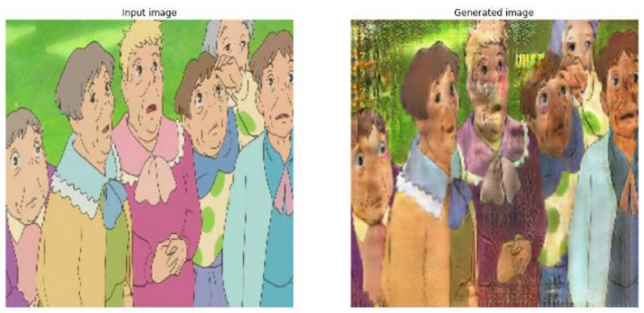
left = original image, right = generated image
**More images at the end of the report, Appendix*

Ponyo (Ghibli 2D) to Toy Story 3 (Pixar 3D) style:





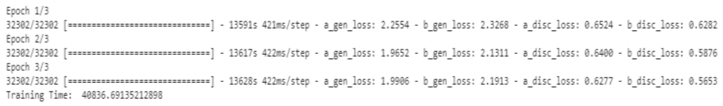




left = original image, right = generated image
**More images at the end of the report, Appendix*

For Toy Story 3 and Ponyo images translation, Toy Story transforms to
Ghibli style accurately. The black outline is clear for most images, and it is not as scratchy as Luca. However, Ponyo does not perform that well compared to Spirited Away.

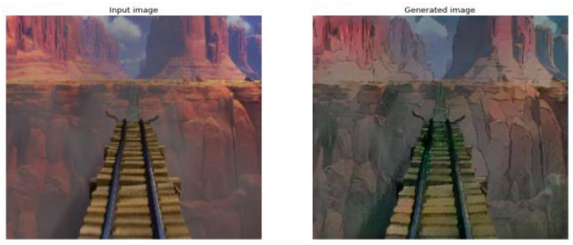
Training using Toy Story 3, Finding Nemo, Luca, Inside Out, Ponyo, Spirited Away, My Neighbor Totoro, and Princess Mononoke

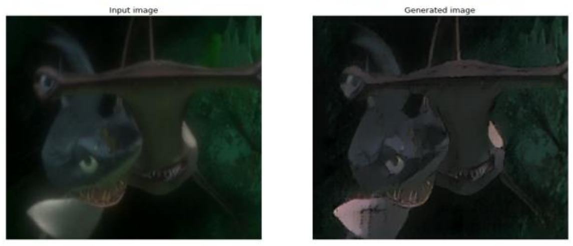


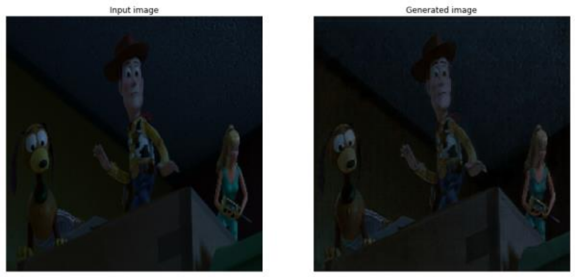
```
Epoch 1/3
32302/32302 [==============================] - 13591s 421ms/step - a_gen_loss: 2.2554 - b_gen_loss: 2.3268 - a_disc_loss: 0.6524 - b_disc_loss: 0.6282
Epoch 2/3
32302/32302 [==============================] - 13617s 422ms/step - a_gen_loss: 1.9652 - b_gen_loss: 2.1311 - a_disc_loss: 0.6400 - b_disc_loss: 0.5876
Epoch 3/3
32302/32302 [==============================] - 13628s 422ms/step - a_gen_loss: 1.9906 - b_gen_loss: 2.1913 - a_disc_loss: 0.6277 - b_disc_loss: 0.5653
Training Time:  40836.69135212898
```

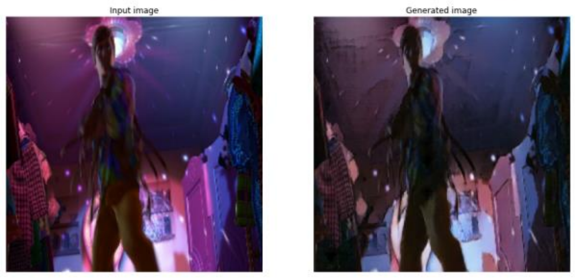
For this model we were only able to train for 3 epochs due to our limited resources, it took about 11 hours and 20 minutes to train for only 3, and that is with GPU boost and single gpu optimized code. However the results are promising as we can see below.

Pixar (3D) to Ghibli (2D):



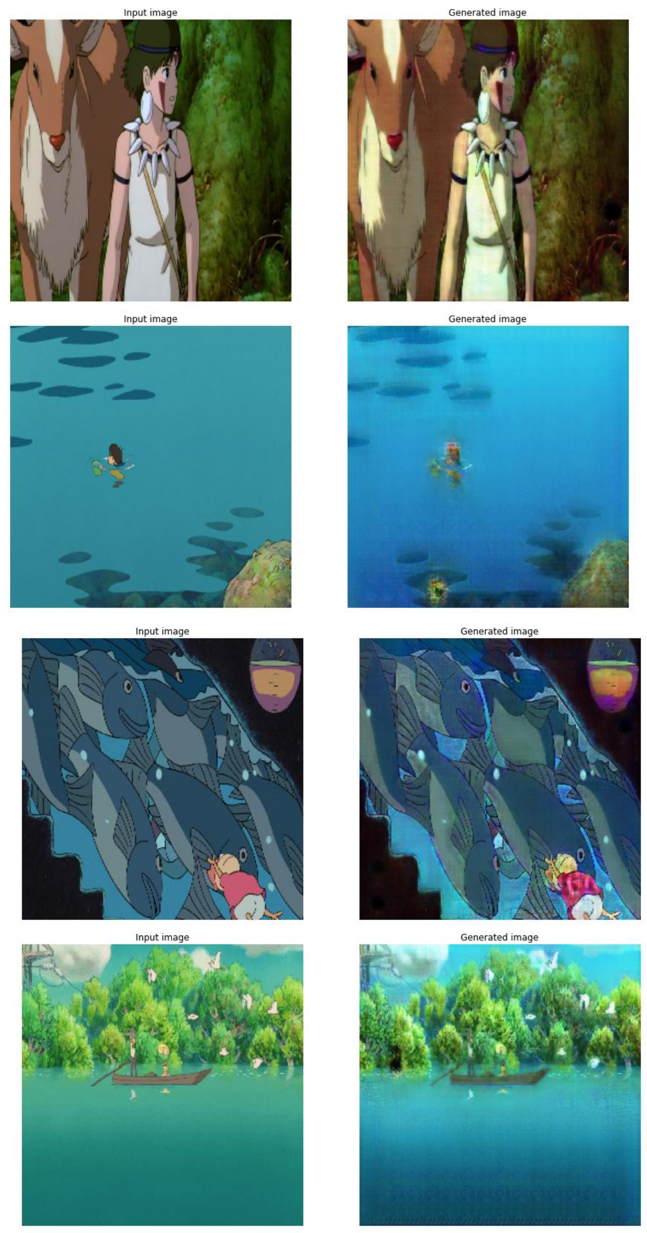






The model overall appears to learn well the ghibli style in general. However, in scenes that appear to be dark, we see a deterioration in performance as in the top right pair, the images look like in the same style as compared to the difference between the others.

Ghibli (2D) to Pixar (3D):



The results of this model seem to be poorer. What it does seem to be good at however is adding texture and shading to backgrounds that are a solid color when in 2D form helping give a slight appearance of depth.

Overall, the results of this model are fairly decent given that it was only able to train for 3 epochs. With better resources and more power, it would be interesting to see how this model might improve when being able to train with many more epochs.

<u>**Conditional Adversarial Autoencoder (CAAE)**</u>

Based on *Age Progression/Regression by Conditional Adversarial Autoencoder* by Zhang et. al

A conditional adversarial autoencoder (CAAE) was also used to solve the same issue as the Cycle GAN. This model would be able to handle multiple styles at once. This is because labels for the data are taken as input. These labels can be any dimension. As a result, multiple classes can be learned at once.

A CAAE uses an autoencoder as a generator for a GAN model. The CAAE features two discriminators: one for the latent vectors (output of encoder) and one for the generated images (output of decoder/generator). The purpose of the first discriminator's purpose is to push the distribution of image latent vectors into a uniform distribution. This should result in any latent vector decoding into a realistic image. This discriminator is analogous to a KL-divergence loss in a variational autoencoder (VAE) which pushes the VAE latent distribution to a normal distribution. Having the entire manifold space filled helps the model edit the image attributes in a realistic manner with minimal "ghosting"artifacts.

The second discriminator is used to force the decoder/generator to create realistic images by learning to distinguish between real and fake images. Because the image discriminator takes both the image and labels as input, it also learns what types of images are realistic for a given class.

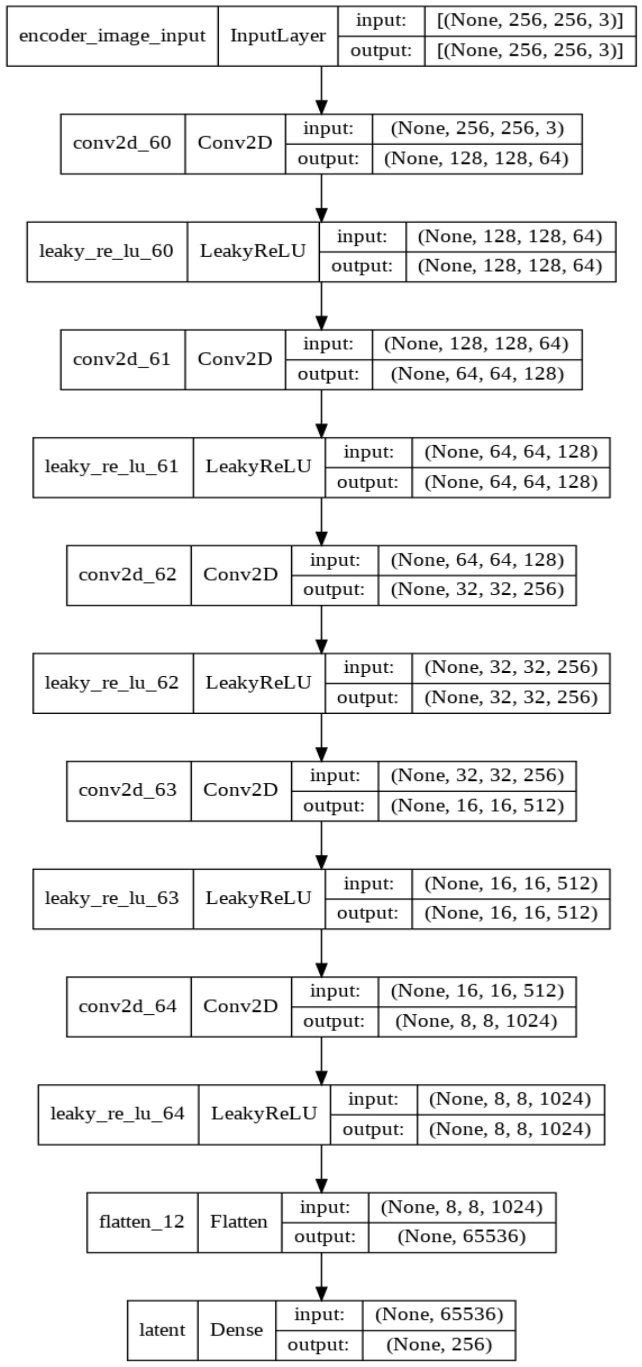
Encoder and decoder inputs and outputs are normalized to the range of -1 to 1. Tanh is used for the output activation layer of the encoder and decoder. The image inputs are normalized to the range of -1 to 1. The autoencoder uses a combination of L1 and total variation for loss functions with weights of 100 and 10 respectively. The discriminators both use binary cross entropy as the loss function and sigmoid as the output activation function. All intermediate activation functions are Leaky ReLu. The learning rate for all models was set at 0.002. The latent dimension for the autoencoder was 256.

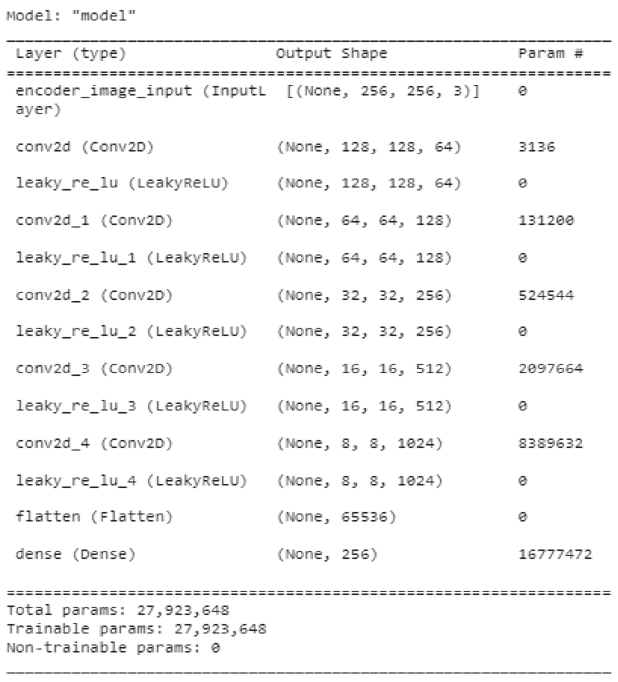
The architecture of this CAAE model was based on a paper by Zhang et. al titled *Age Progression/Regression by Conditional Adversarial Autoencoder*. However, there were some adjustments made to suit our problem:

1. ***An extra convolutional layer of stride 2 was added to the encoder and image discriminator.*** This was done so that the image width and height before the latent layer was the same (256x256 input in our problem vs 128x128 input in the original paper). The image depth immediately before the latent layer will be double that of the original model from the paper.
2. ***The latent dimension was increased from 50 to 256.*** This was done because the original problem was only concerned with images of faces while this problem is concerned with many different types of images based on scenarios that occur in the movies. The latent dimension was increased to allow for this more varied and complex space to be represented.
3. ***Leaky ReLu was used as the activation function for intermediate layers instead of ReLu.*** This change was due to vanishing gradients that occurred while training the GAN.
4. ***L1 loss was used instead of L2 loss for the autoencoder.*** Experimentally, L1 loss led to sharper images while L2 loss led to blurry images.

Model Architecture Diagrams

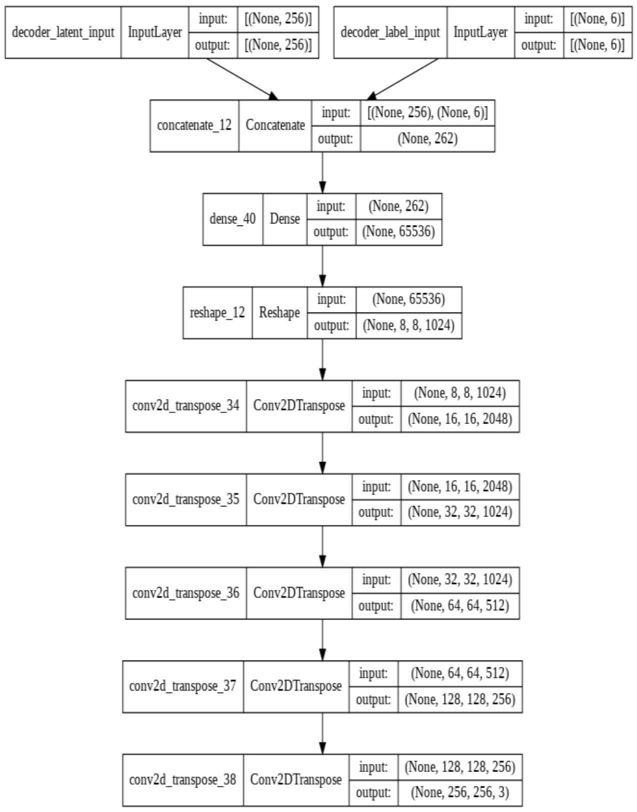
Encoder


encoder_image_input | InputLayer | input: [(None, 256, 256, 3)] | output: [(None, 256, 256, 3)]
conv2d_60 | Conv2D | input: (None, 256, 256, 3) | output: (None, 128, 128, 64)
leaky_re_lu_60 | LeakyReLU | input: (None, 128, 128, 64) | output: (None, 128, 128, 64)
conv2d_61 | Conv2D | input: (None, 128, 128, 64) | output: (None, 64, 64, 128)
leaky_re_lu_61 | LeakyReLU | input: (None, 64, 64, 128) | output: (None, 64, 64, 128)
conv2d_62 | Conv2D | input: (None, 64, 64, 128) | output: (None, 32, 32, 256)
leaky_re_lu_62 | LeakyReLU | input: (None, 32, 32, 256) | output: (None, 32, 32, 256)
conv2d_63 | Conv2D | input: (None, 32, 32, 256) | output: (None, 16, 16, 512)
leaky_re_lu_63 | LeakyReLU | input: (None, 16, 16, 512) | output: (None, 16, 16, 512)
conv2d_64 | Conv2D | input: (None, 16, 16, 512) | output: (None, 8, 8, 1024)
leaky_re_lu_64 | LeakyReLU | input: (None, 8, 8, 1024) | output: (None, 8, 8, 1024)
flatten_12 | Flatten | input: (None, 8, 8, 1024) | output: (None, 65536)
latent | Dense | input: (None, 65536) | output: (None, 256)

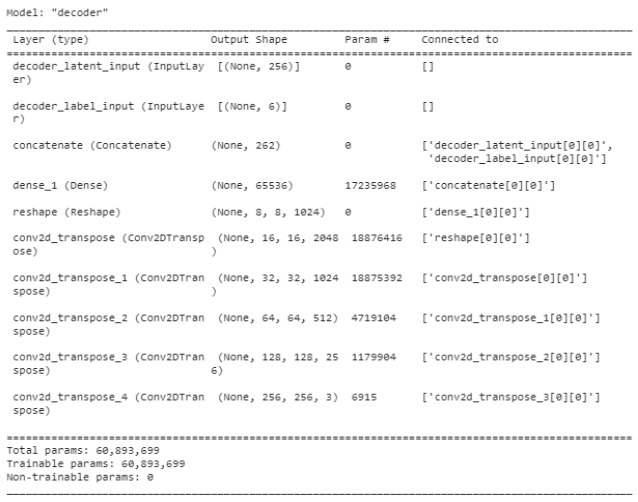


```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 encoder_image_input (InputL  [(None, 256, 256, 3)]    0
 ayer)

 conv2d (Conv2D)             (None, 128, 128, 64)      3136

 leaky_re_lu (LeakyReLU)     (None, 128, 128, 64)      0

 conv2d_1 (Conv2D)           (None, 64, 64, 128)       131200

 leaky_re_lu_1 (LeakyReLU)   (None, 64, 64, 128)       0

 conv2d_2 (Conv2D)           (None, 32, 32, 256)       524544

 leaky_re_lu_2 (LeakyReLU)   (None, 32, 32, 256)       0

 conv2d_3 (Conv2D)           (None, 16, 16, 512)       2097664

 leaky_re_lu_3 (LeakyReLU)   (None, 16, 16, 512)       0

 conv2d_4 (Conv2D)           (None, 8, 8, 1024)        8389632

 leaky_re_lu_4 (LeakyReLU)   (None, 8, 8, 1024)        0

 flatten (Flatten)           (None, 65536)             0

 dense (Dense)               (None, 256)               16777472

=================================================================
Total params: 27,923,648
Trainable params: 27,923,648
Non-trainable params: 0
_________________________________________________________________
```

Decoder

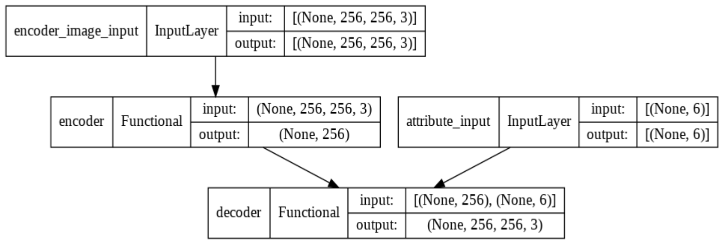


decoder_latent_input | InputLayer | input: [(None, 256)] | output: [(None, 256)]
decoder_label_input | InputLayer | input: [(None, 6)] | output: [(None, 6)]
concatenate_12 | Concatenate | input: [(None, 256), (None, 6)] | output: (None, 262)
dense_40 | Dense | input: (None, 262) | output: (None, 65536)
reshape_12 | Reshape | input: (None, 65536) | output: (None, 8, 8, 1024)
conv2d_transpose_34 | Conv2DTranspose | input: (None, 8, 8, 1024) | output: (None, 16, 16, 2048)
conv2d_transpose_35 | Conv2DTranspose | input: (None, 16, 16, 2048) | output: (None, 32, 32, 1024)
conv2d_transpose_36 | Conv2DTranspose | input: (None, 32, 32, 1024) | output: (None, 64, 64, 512)
conv2d_transpose_37 | Conv2DTranspose | input: (None, 64, 64, 512) | output: (None, 128, 128, 256)
conv2d_transpose_38 | Conv2DTranspose | input: (None, 128, 128, 256) | output: (None, 256, 256, 3)



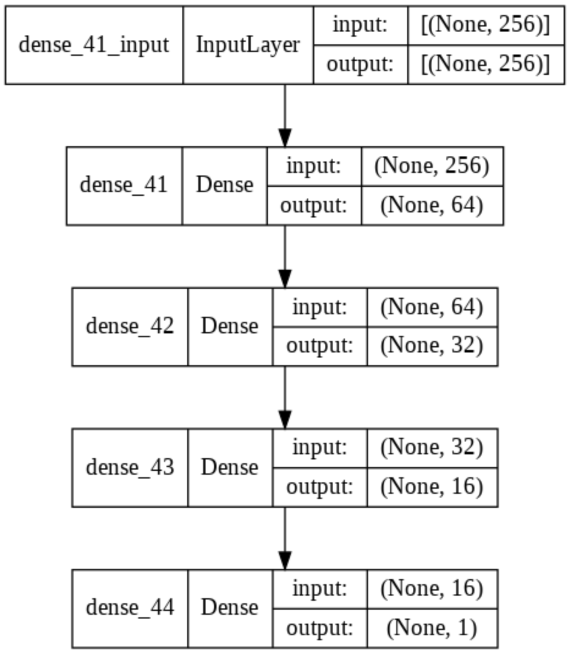
```
Model: "decoder"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 decoder_latent_input (InputLay  [(None, 256)]       0           []
 er)

 decoder_label_input (InputLaye  [(None, 6)]         0           []
 r)

 concatenate (Concatenate)      (None, 262)          0           ['decoder_latent_input[0][0]',
                                                                  'decoder_label_input[0][0]']

 dense_1 (Dense)                (None, 65536)        17235968    ['concatenate[0][0]']

 reshape (Reshape)              (None, 8, 8, 1024)   0           ['dense_1[0][0]']

 conv2d_transpose (Conv2DTransp  (None, 16, 16, 2048  18876416   ['reshape[0][0]']
 ose)                           )

 conv2d_transpose_1 (Conv2DTran  (None, 32, 32, 1024  18875392   ['conv2d_transpose[0][0]']
 spose)                         )

 conv2d_transpose_2 (Conv2DTran  (None, 64, 64, 512)  4719104    ['conv2d_transpose_1[0][0]']
 spose)

 conv2d_transpose_3 (Conv2DTran  (None, 128, 128, 25  1179904    ['conv2d_transpose_2[0][0]']
 spose)                         6)

 conv2d_transpose_4 (Conv2DTran  (None, 256, 256, 3)  6915       ['conv2d_transpose_3[0][0]']
 spose)

==================================================================================================
Total params: 60,893,699
Trainable params: 60,893,699
Non-trainable params: 0
__________________________________________________________________________________________________
```

Autoencoder

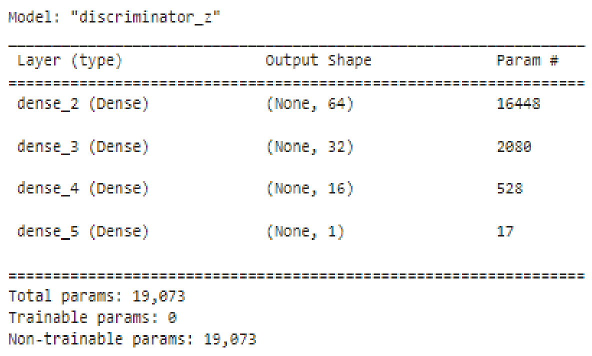


encoder_image_input | InputLayer | input: [(None, 256, 256, 3)] | output: [(None, 256, 256, 3)]
encoder | Functional | input: (None, 256, 256, 3) | output: (None, 256)
attribute_input | InputLayer | input: [(None, 6)] | output: [(None, 6)]
decoder | Functional | input: [(None, 256), (None, 6)] | output: (None, 256, 256, 3)

Latent Discriminator

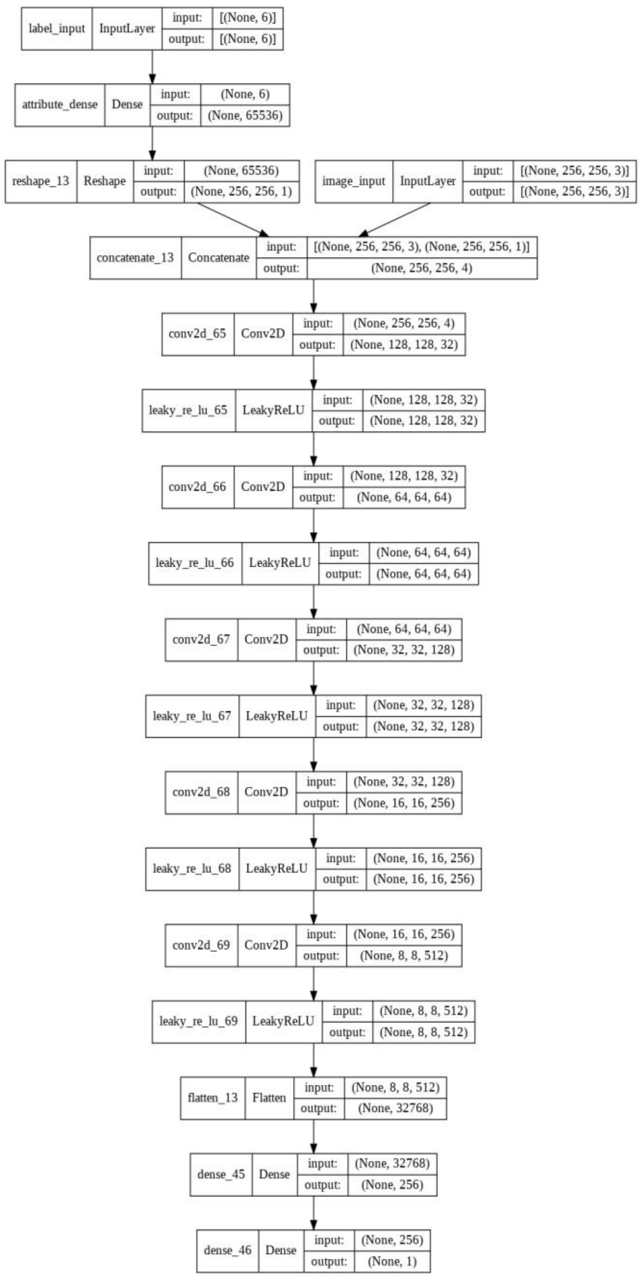


dense_41_input | InputLayer | input: [(None, 256)] | output: [(None, 256)]
dense_41 | Dense | input: (None, 256) | output: (None, 64)
dense_42 | Dense | input: (None, 64) | output: (None, 32)
dense_43 | Dense | input: (None, 32) | output: (None, 16)
dense_44 | Dense | input: (None, 16) | output: (None, 1)

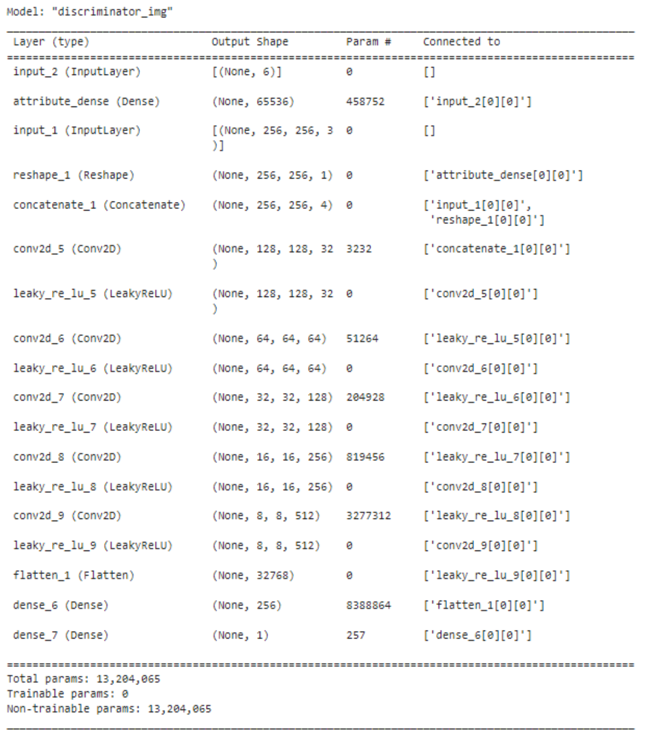


```
Model: "discriminator_z"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_2 (Dense)             (None, 64)                16448

 dense_3 (Dense)             (None, 32)                2080

 dense_4 (Dense)             (None, 16)                528

 dense_5 (Dense)             (None, 1)                 17

=================================================================
Total params: 19,073
Trainable params: 0
Non-trainable params: 19,073
```

Image Discriminator

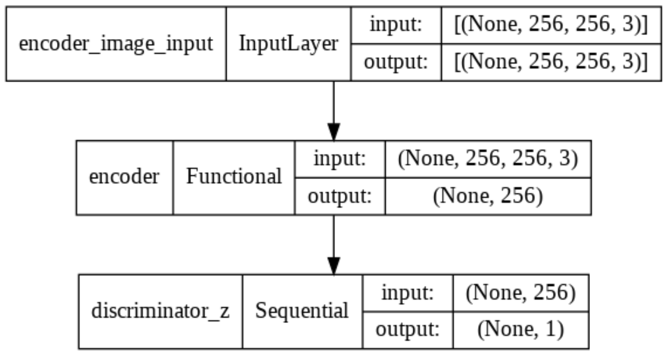


label_input | InputLayer | input: [(None, 6)] | output: [(None, 6)]
attribute_dense | Dense | input: (None, 6) | output: (None, 65536)
reshape_13 | Reshape | input: (None, 65536) | output: (None, 256, 256, 1)
image_input | InputLayer | input: [(None, 256, 256, 3)] | output: [(None, 256, 256, 3)]
concatenate_13 | Concatenate | input: [(None, 256, 256, 3), (None, 256, 256, 1)] | output: (None, 256, 256, 4)
conv2d_65 | Conv2D | input: (None, 256, 256, 4) | output: (None, 128, 128, 32)
leaky_re_lu_65 | LeakyReLU | input: (None, 128, 128, 32) | output: (None, 128, 128, 32)
conv2d_66 | Conv2D | input: (None, 128, 128, 32) | output: (None, 64, 64, 64)
leaky_re_lu_66 | LeakyReLU | input: (None, 64, 64, 64) | output: (None, 64, 64, 64)
conv2d_67 | Conv2D | input: (None, 64, 64, 64) | output: (None, 32, 32, 128)
leaky_re_lu_67 | LeakyReLU | input: (None, 32, 32, 128) | output: (None, 32, 32, 128)
conv2d_68 | Conv2D | input: (None, 32, 32, 128) | output: (None, 16, 16, 256)
leaky_re_lu_68 | LeakyReLU | input: (None, 16, 16, 256) | output: (None, 16, 16, 256)
conv2d_69 | Conv2D | input: (None, 16, 16, 256) | output: (None, 8, 8, 512)
leaky_re_lu_69 | LeakyReLU | input: (None, 8, 8, 512) | output: (None, 8, 8, 512)
flatten_13 | Flatten | input: (None, 8, 8, 512) | output: (None, 32768)
dense_45 | Dense | input: (None, 32768) | output: (None, 256)
dense_46 | Dense | input: (None, 256) | output: (None, 1)



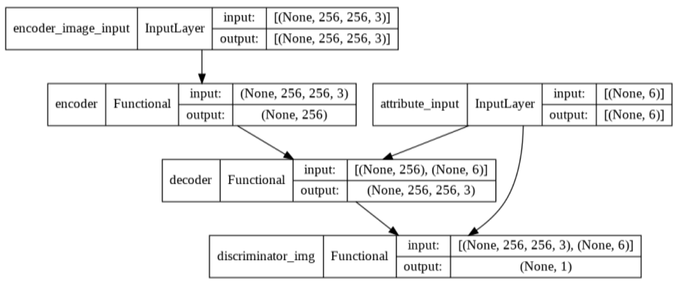
```
Model: "discriminator_img"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_2 (InputLayer)           [(None, 6)]          0           []

 attribute_dense (Dense)        (None, 65536)        458752      ['input_2[0][0]']

 input_1 (InputLayer)           [(None, 256, 256, 3  0           []
                                )]

 reshape_1 (Reshape)            (None, 256, 256, 1)  0           ['attribute_dense[0][0]']

 concatenate_1 (Concatenate)    (None, 256, 256, 4)  0           ['input_1[0][0]',
                                                                  'reshape_1[0][0]']

 conv2d_5 (Conv2D)              (None, 128, 128, 32  3232        ['concatenate_1[0][0]']
                                )

 leaky_re_lu_5 (LeakyReLU)      (None, 128, 128, 32  0           ['conv2d_5[0][0]']
                                )

 conv2d_6 (Conv2D)              (None, 64, 64, 64)   51264       ['leaky_re_lu_5[0][0]']

 leaky_re_lu_6 (LeakyReLU)      (None, 64, 64, 64)   0           ['conv2d_6[0][0]']

 conv2d_7 (Conv2D)              (None, 32, 32, 128)  204928      ['leaky_re_lu_6[0][0]']

 leaky_re_lu_7 (LeakyReLU)      (None, 32, 32, 128)  0           ['conv2d_7[0][0]']

 conv2d_8 (Conv2D)              (None, 16, 16, 256)  819456      ['leaky_re_lu_7[0][0]']

 leaky_re_lu_8 (LeakyReLU)      (None, 16, 16, 256)  0           ['conv2d_8[0][0]']

 conv2d_9 (Conv2D)              (None, 8, 8, 512)    3277312     ['leaky_re_lu_8[0][0]']

 leaky_re_lu_9 (LeakyReLU)      (None, 8, 8, 512)    0           ['conv2d_9[0][0]']

 flatten_1 (Flatten)            (None, 32768)        0           ['leaky_re_lu_9[0][0]']

 dense_6 (Dense)                (None, 256)          8388864     ['flatten_1[0][0]']

 dense_7 (Dense)                (None, 1)            257         ['dense_6[0][0]']

==================================================================================================
Total params: 13,204,065
Trainable params: 0
Non-trainable params: 13,204,065
__________________________________________________________________________________________________
```

Latent GAN



encoder_image_input | InputLayer | input: [(None, 256, 256, 3)] | output: [(None, 256, 256, 3)]
encoder | Functional | input: (None, 256, 256, 3) | output: (None, 256)
discriminator_z | Sequential | input: (None, 256) | output: (None, 1)

Image GAN



encoder_image_input | InputLayer | input: [(None, 256, 256, 3)] | output: [(None, 256, 256, 3)]
encoder | Functional | input: (None, 256, 256, 3) | output: (None, 256)
attribute_input | InputLayer | input: [(None, 6)] | output: [(None, 6)]
decoder | Functional | input: [(None, 256), (None, 6)] | output: (None, 256, 256, 3)
discriminator_img | Functional | input: [(None, 256, 256, 3), (None, 6)] | output: (None, 1)

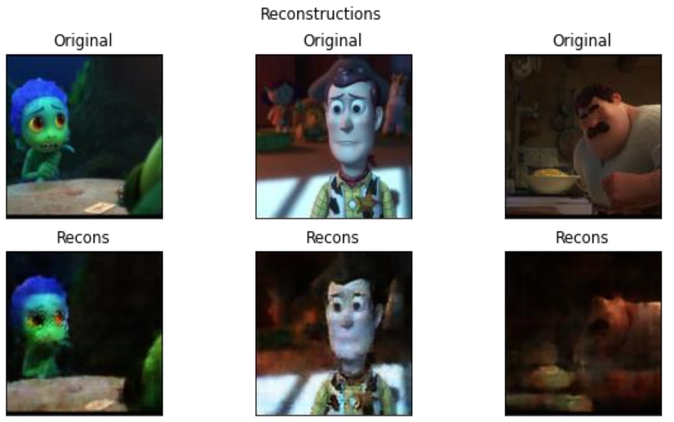
CAAE Model Training

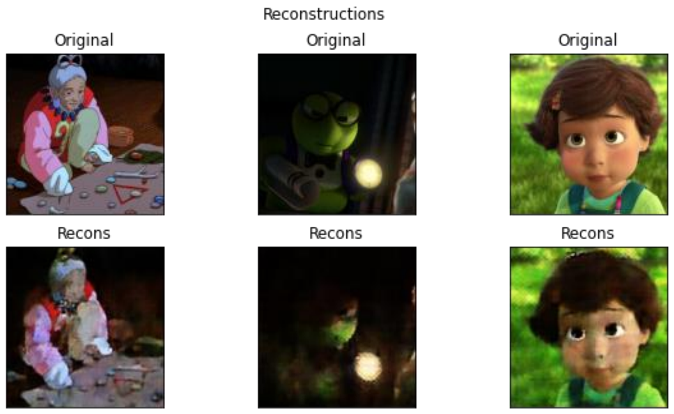
The entire CAAE architecture was trained for 60 epochs to achieve the final results.

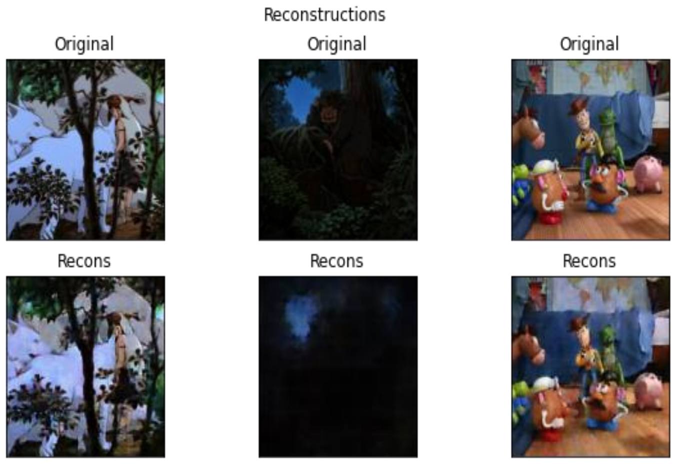
Several issues were encountered during the training of the entire CAAE. The issues encountered and steps taken to rectify them are listed below:

1. ***Vanishing Gradients in image discriminator***. The dense layer before the output layer was giving all 0 outputs, resulting in roughly 0.5 output for the discriminator probability for any input. This was rectified by changing the activation function of the dense layer from ReLu to Leaky ReLu.
2. ***Discriminator was overpowering the generator.*** The discriminator would achieve very high accuracy before the generator had a chance to learn how to generate convincing images. This was rectified in three ways:
    - ***a.*** The number of nodes in the z-discriminator and the number of filters in the image discriminator were reduced to weaken the discriminators.
    - ***b.*** The latent dimension was increased from 128 to the final value of 256 to give the generator more representational power.
    - ***c.*** The autoencoder section of the model was pretrained before training the entire CAAE model. This led to a more stable training outcome.

Reconstructions

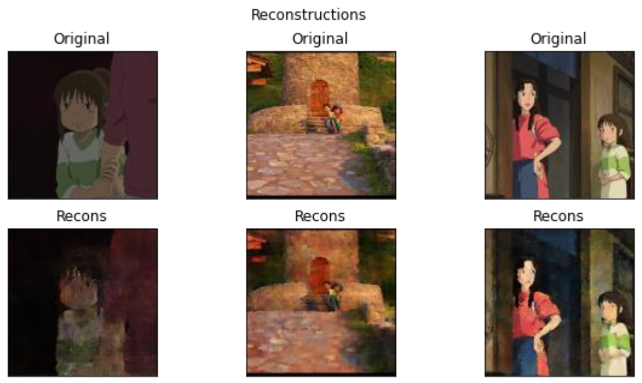


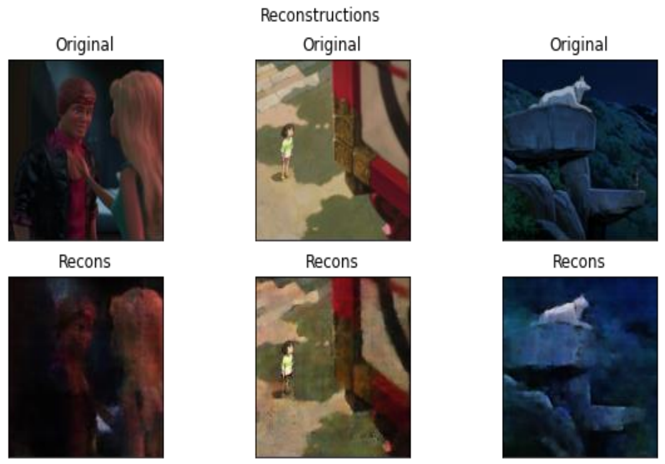




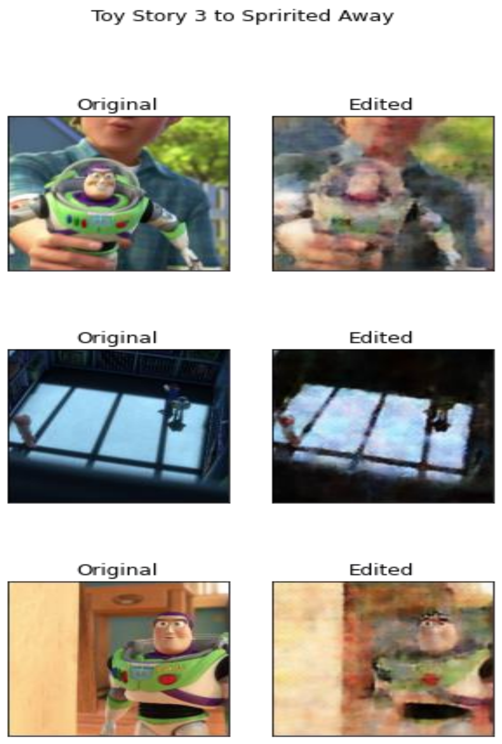
Results

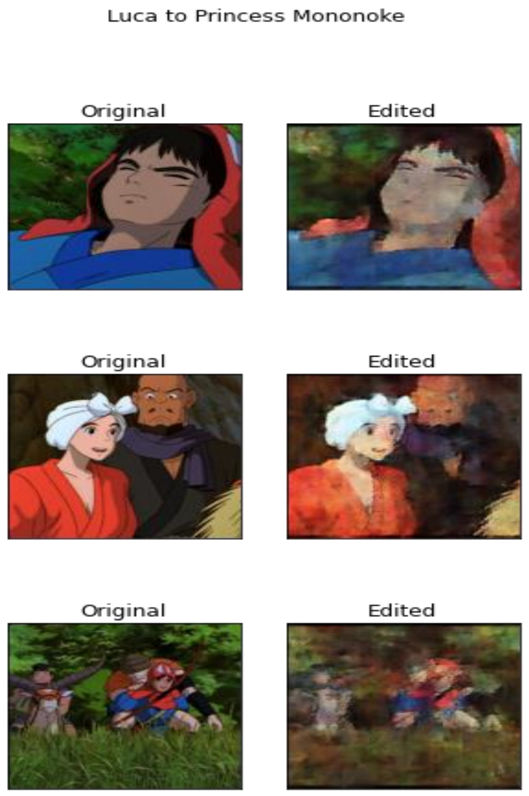
Model training histories may be found in the appendix.

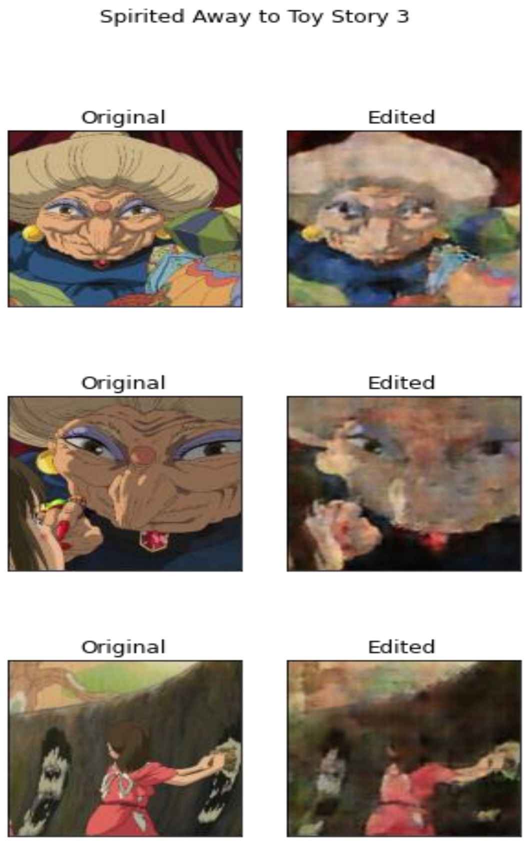


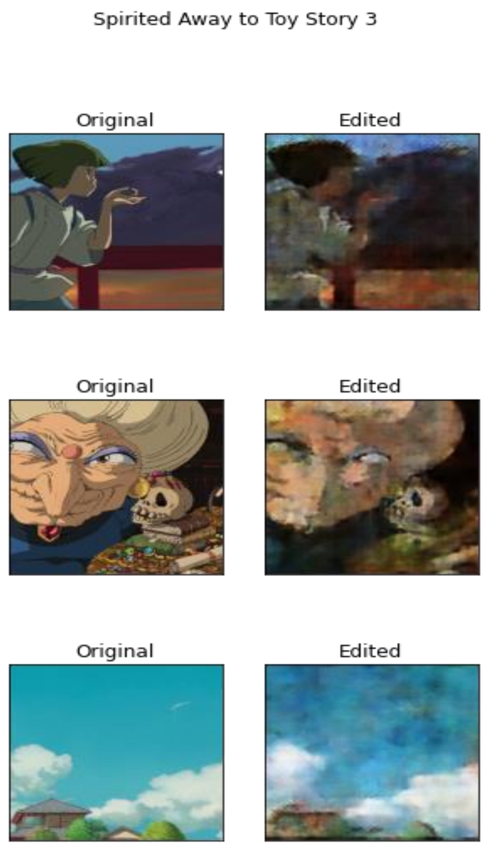


Change Style









A qualitative analysis of the images generated in new styles suggests that the model had an easier time translating 3D animation to 2D animation than doing the opposite. A similar trend was observed with the CycleGAN. This may be because 3D animation features more complex geometry and shading. This focus on shading may explain why the 2D images reconstructed in 3D seem to come out darker than the originals.

The presence of ghosting artifacts can be observed in the reconstructions. This may indicate that the loss on the latent discriminator should have been weighted more heavily. More training data may also help spread the latent distribution to be uniform which could make the style-adjusted images more realistic.

During training, the generator took many epochs to learn to fool the discriminator. The learning rate of the generator could possibly be increased to help mitigate this issue in any future experiments.

<u>Further Qualitative Analysis</u>

For further analysis we wanted to see how the models performed in a movie ordered sequence of pictures, as the data originally came from movies. As the same objects and backgrounds might be in adjacent frames. This would help take a look at the accuracy of the model, does the object in multiple images look the same after it is generated in each image it is in, in an ordered sequence. That is, does the model generate consistently. To achieve this, we first selected ordered sequences of 61 frames, 30.5 seconds of the film from 2 FPS, into folders (see README for folder location). Then we loaded the models back in and generated over these sample sequences and saved them into new folders. After this, we created an ipbny file ‘picture2movie.ipynb’ that will read from the folder locations the files and stitch them together into AVI video format, creating a clip of the movie. We kept the video to use 2FPS as it was what the original data was in and gives enough view time to see if the model stays consistent. To accomplish this we used Open-CV, Numpy, and Glob libraries. Please see some examples of these clips in the ‘clips’ folder. The overall results were fairly good, the model stayed consistent for 3D to 2D with great generation. For 2D to 3D the model was consistent but the generation was still not great as indicated by the sample stills we have seen previously.

<u>Potential Future Work</u>

The work done in this project could be expanded upon further by training with more data for longer periods of time, thought this may require optimization of code for better GPU usage. Additionally, live actions images could also be included in the style transfer GANs to transfer between live action and 2D or 3D animation. The results shown so far have been promising so more could likely be achieved with more computing time and training data.

Reference

Tensorflow Core. (2021, Nov. 25). Cyclegan. https://www.tensorflow.org/tutorials/generative/cyclegan

TensorFlow Core. (2021, Nov. 11) Pix2pix: Image-to-image translation with a conditional GAN. https://www.tensorflow.org/tutorials/generative/pix2pix

Synced. (2015, Mar. 23). Facebook AI Proposes Group Normalization Alternative to Batch Normalization. Medium. https://medium.com/syncedreview/facebook-ai-proposes-group-normalization-alternative-to-batch-normalization-fb0699bffae7

Ulyanov, D., Vedaldi, A., Lempitsky, V. (2017, Nov. 6). Instance Normalization:
The Missing Ingredient for Fast Stylization. https://arxiv.org/pdf/1607.08022.pdf

Agarwal, M. (2020, Aug. 7). Batch Normalization, Instance Normalization, Layer Normalization: Structural Nuances. https://becominghuman.ai/all-about-normalization-6ea79e70894b

Zhang, Z. et al, Age Progression/Regression by Conditional Adversarial Autoencoder https://openaccess.thecvf.com/content_cvpr_2017/papers/Zhang_Age_ProgressionRegression_by_CVPR_2017_paper.pdf

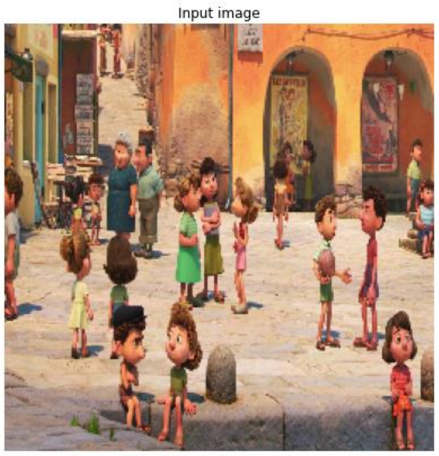
Kang, Pankaj. Singh, Atul Krishna Singh. Creating Video From Images Using OpenCV-Python. The AI Learner. https://theailearner.com/2018/10/15/creating-video-from-images-using-opencv-python/

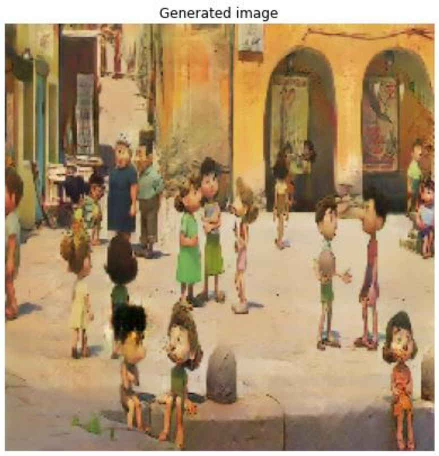
Stack Overflow. web scraping - How to save an image locally using Python whose URL address I already know? - Stack Overflow

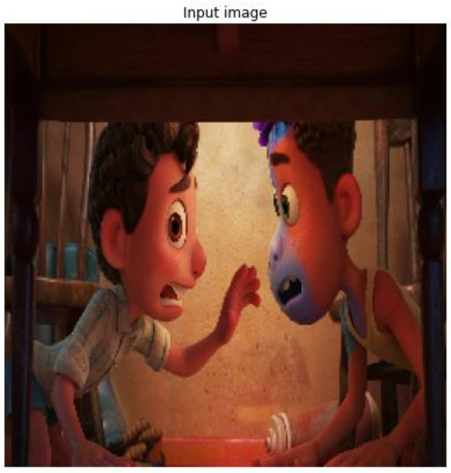
Appendix

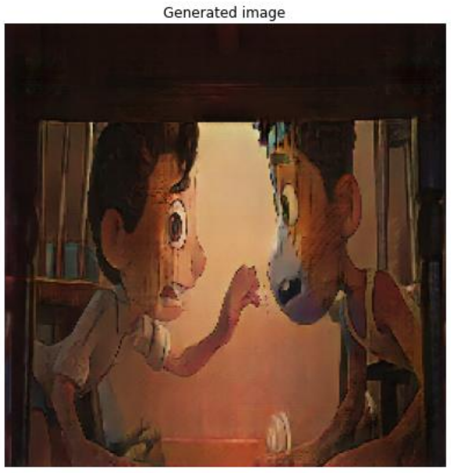
More generated images from CycleGAN

Luca(Pixar 3D) to Spirited Away(Ghibli 2D) style :

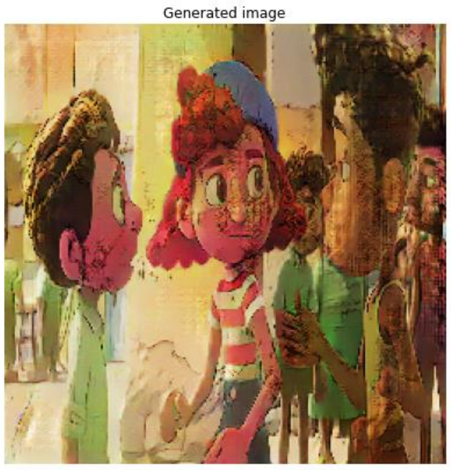


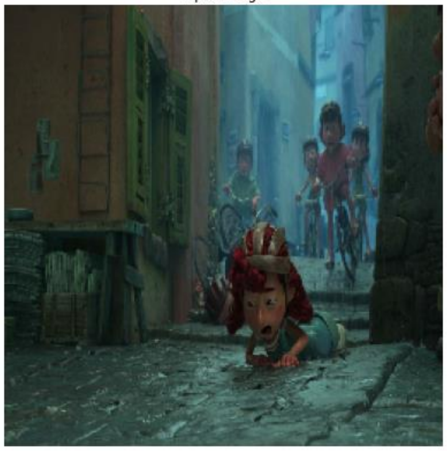


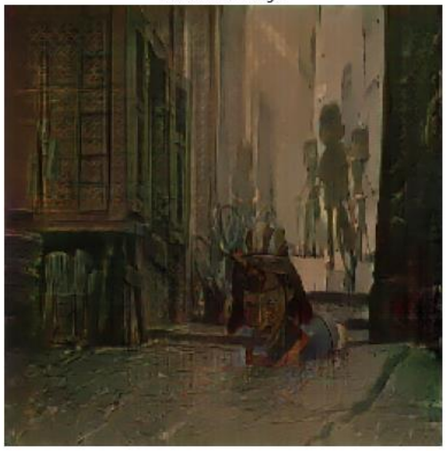


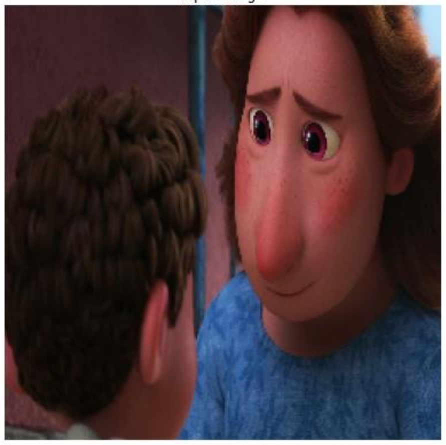


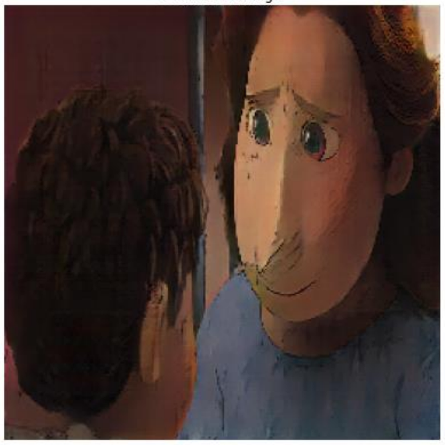
Input image

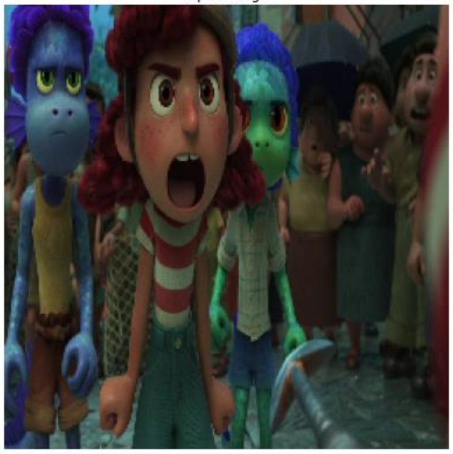


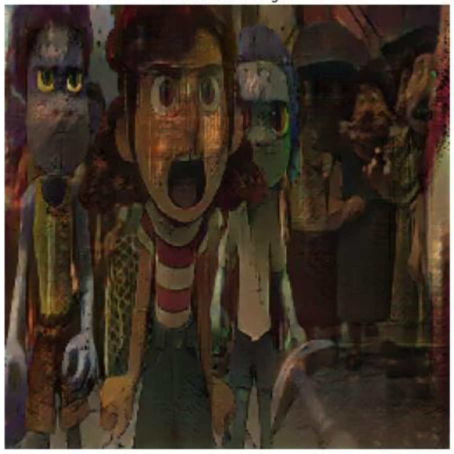






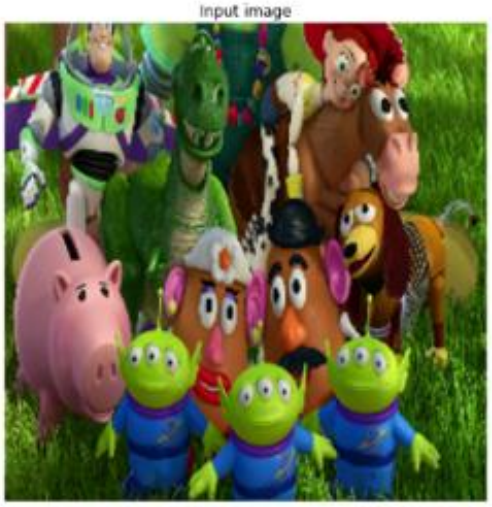




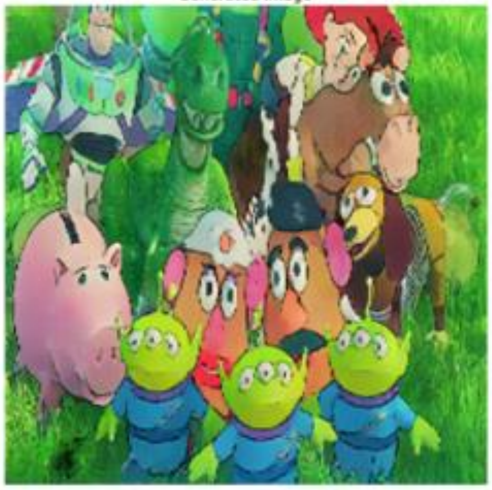


Toy Story 3 (Pixar 3D) to Ponyo (Ghibli 2D) style :

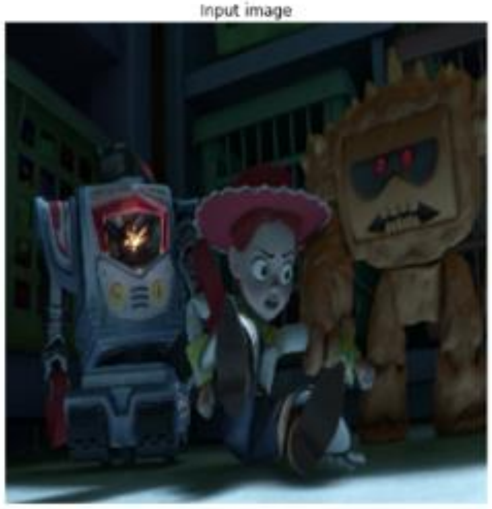
Input image



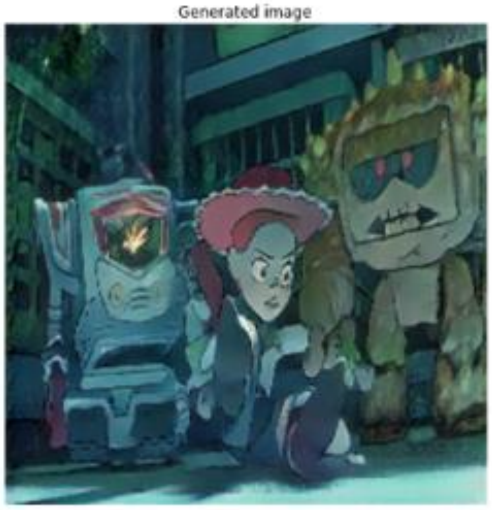
Generated image



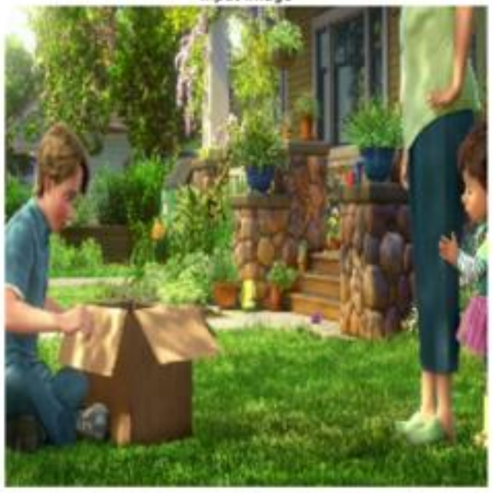
Input image



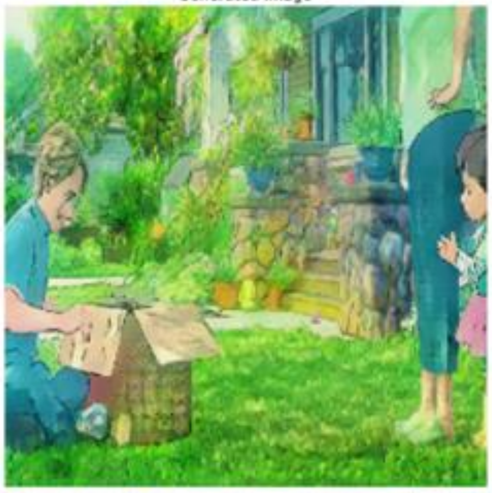
Generated image

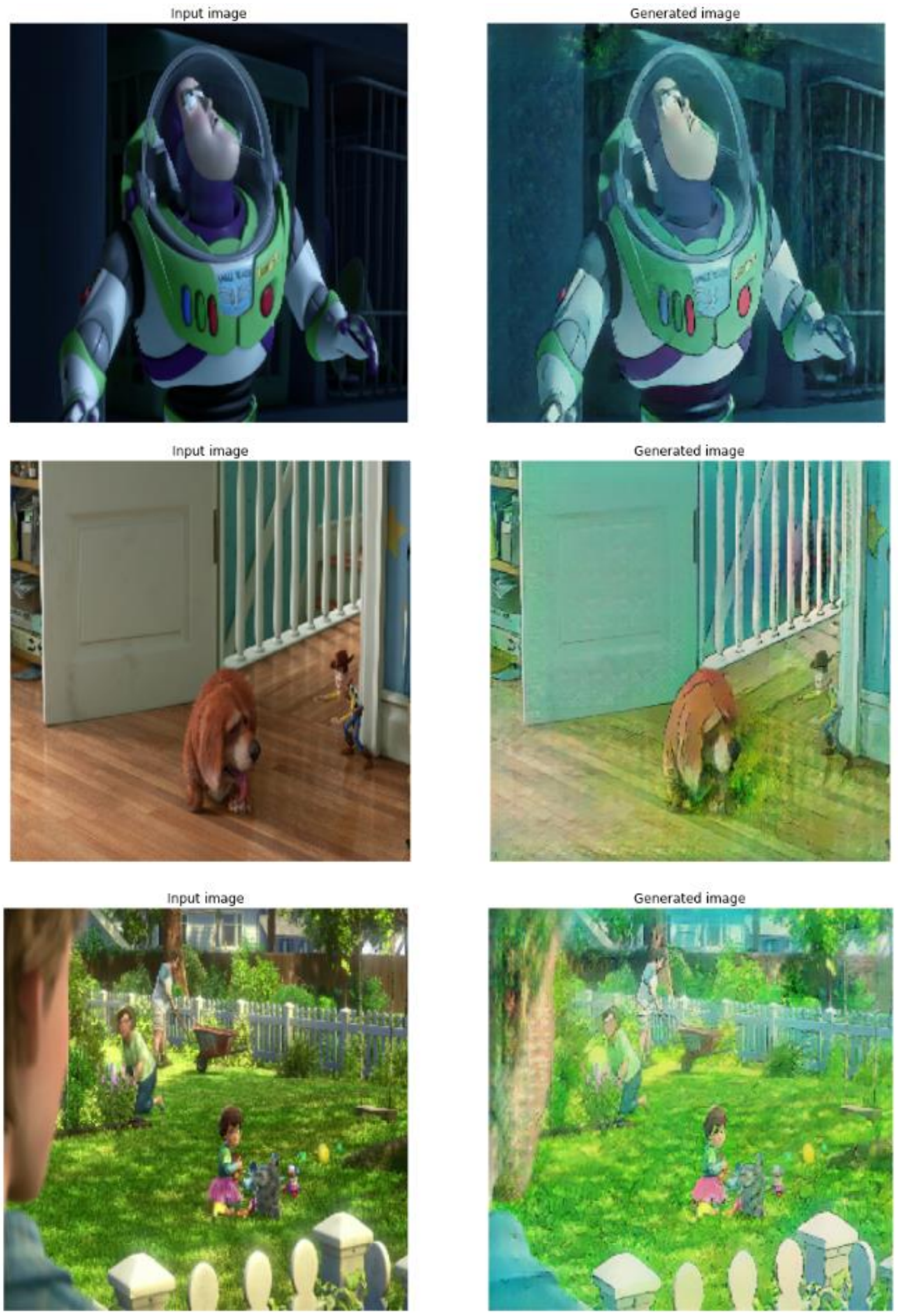


Input image

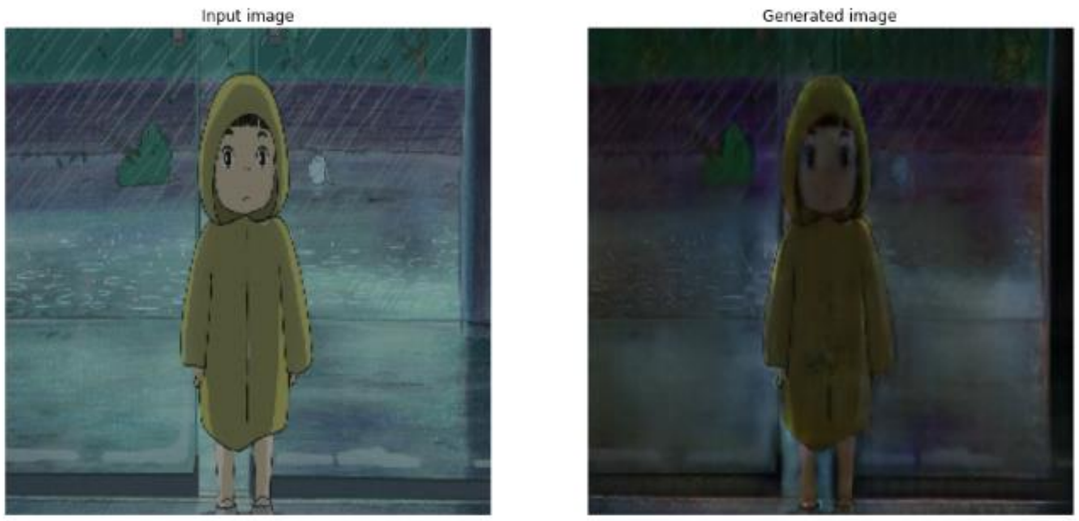


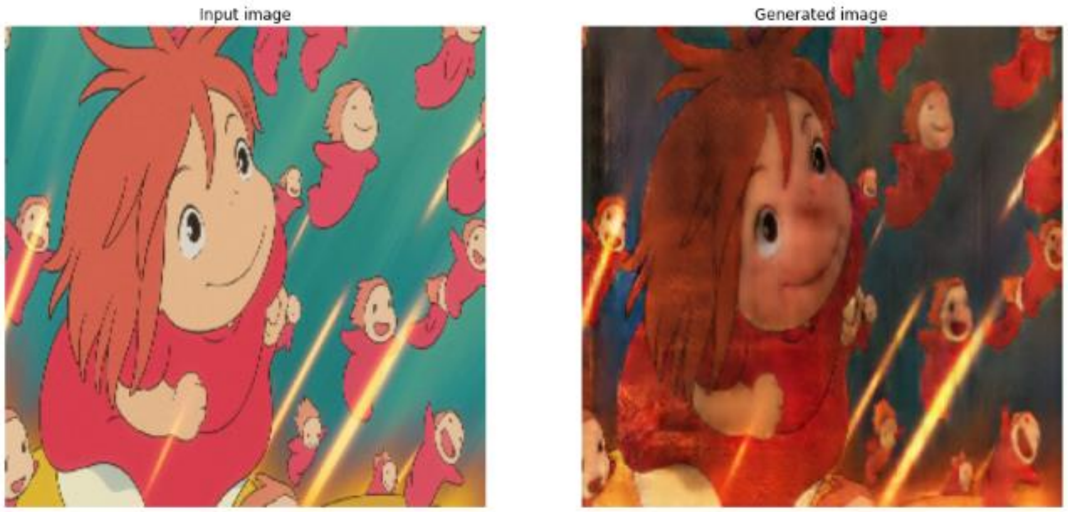
Generated image

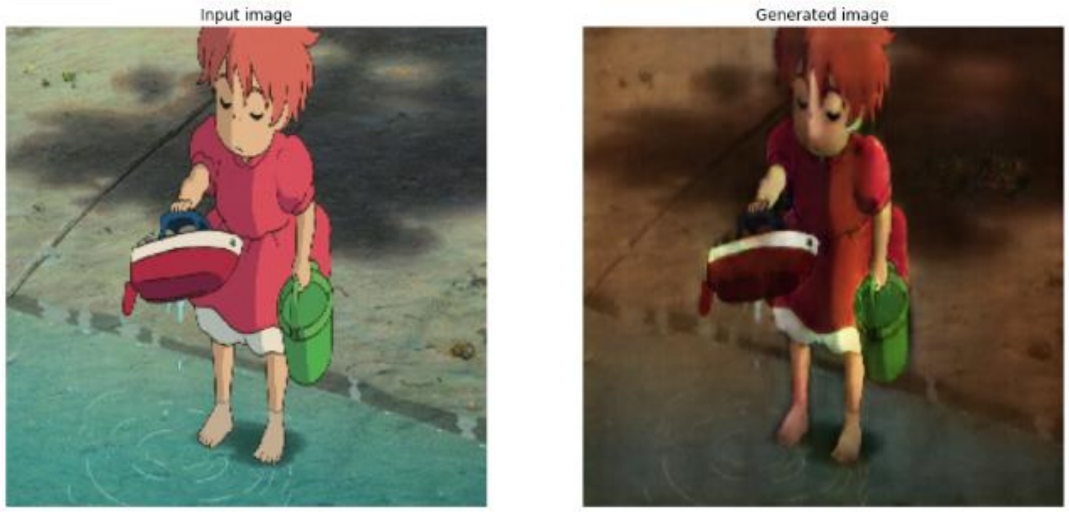


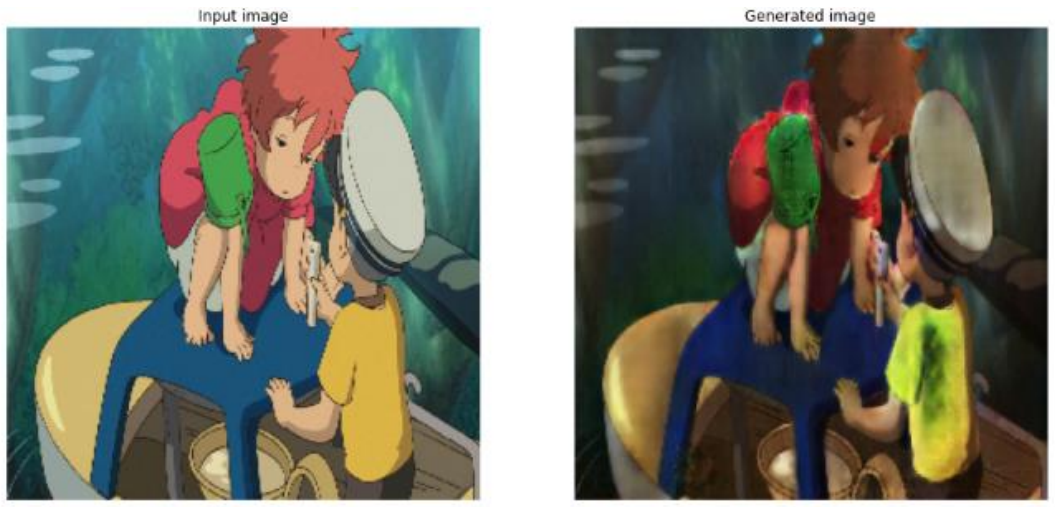


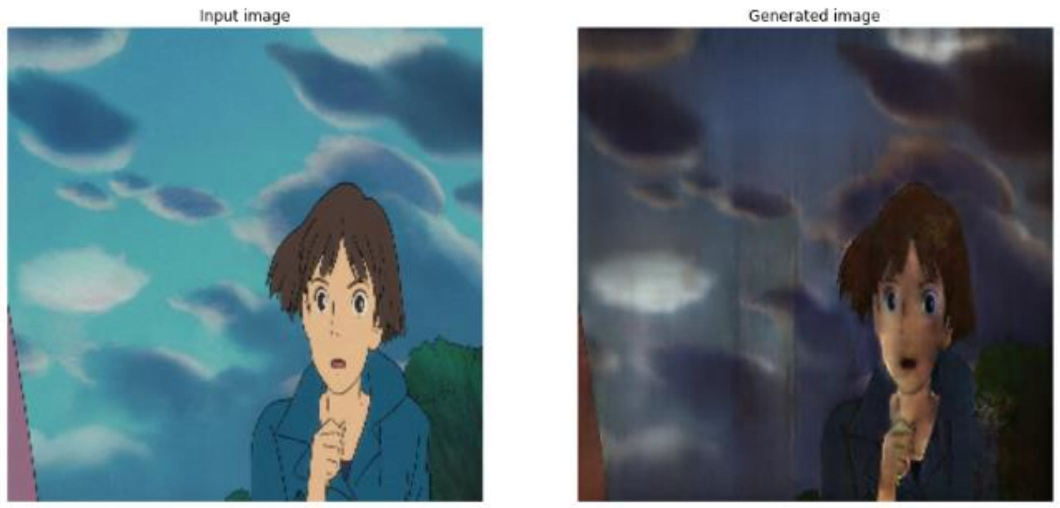
Ponyo (Ghibli 2D) to Toy Story 3 (Pixar 3D) style:

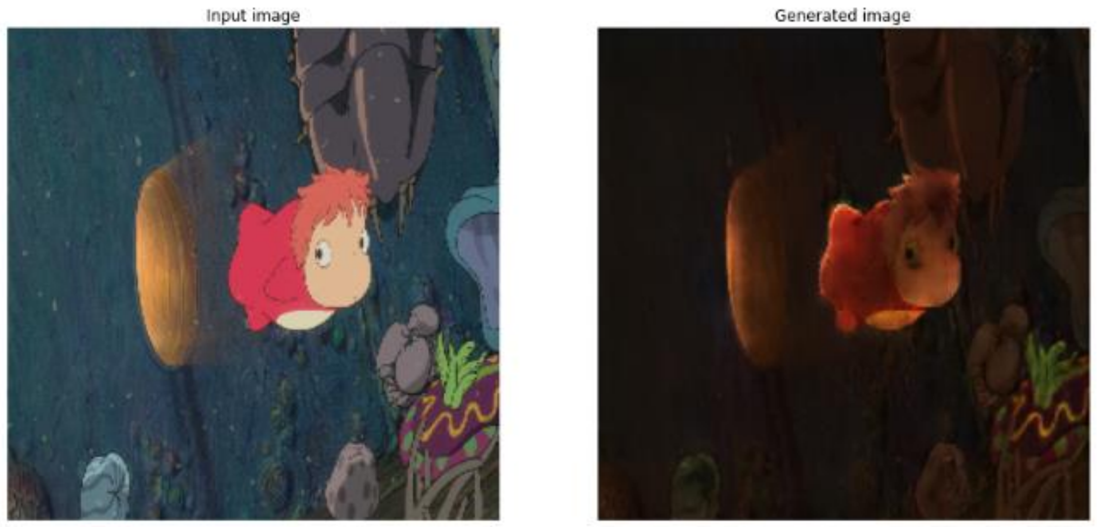

Input image
Generated image


Input image
Generated image


Input image
Generated image

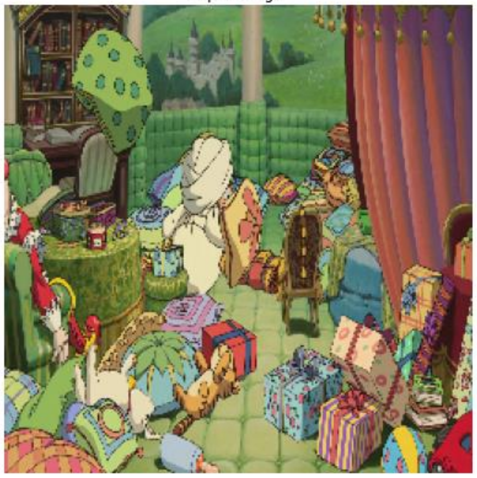




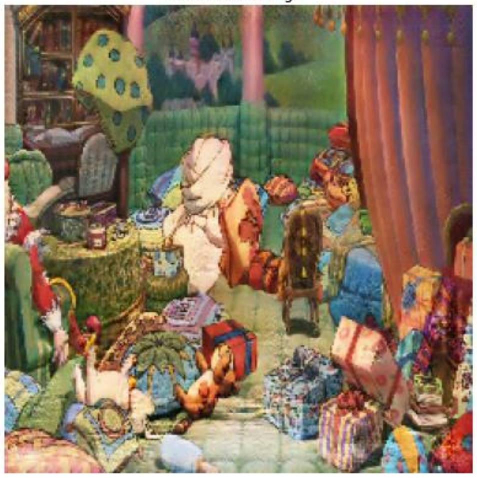


Spirited Away (Ghibli 2D) to Luca (Pixar 3D) style:

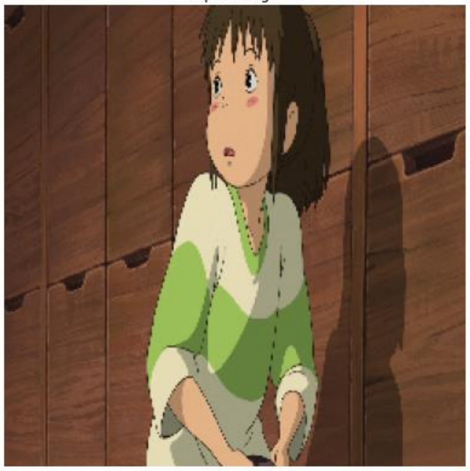
Input image



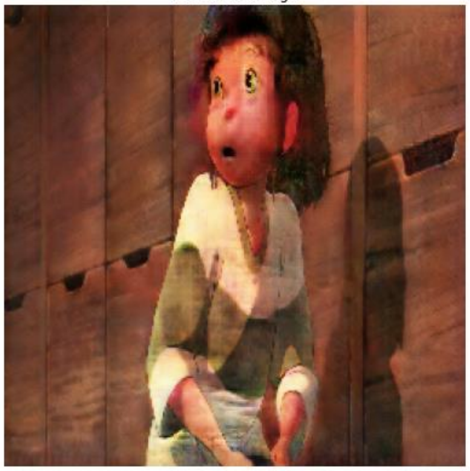
Generated image



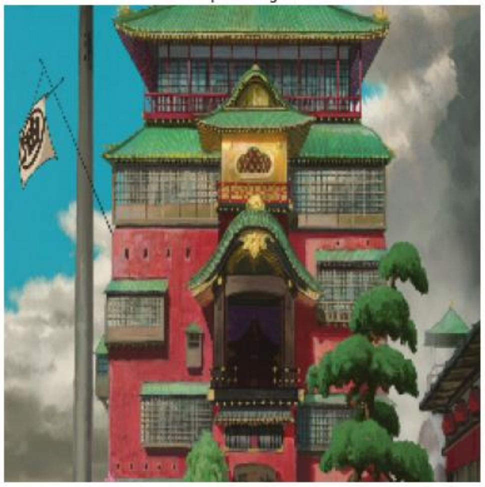
Input image



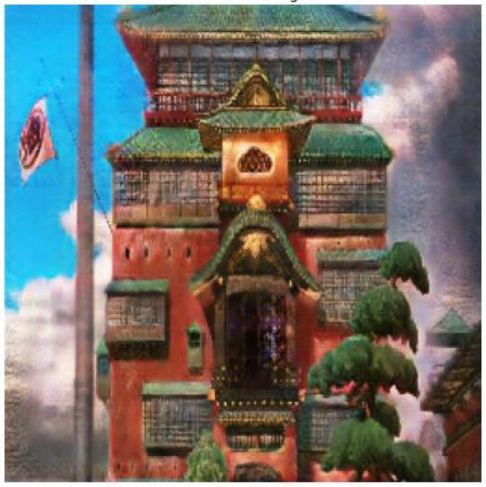
Generated image

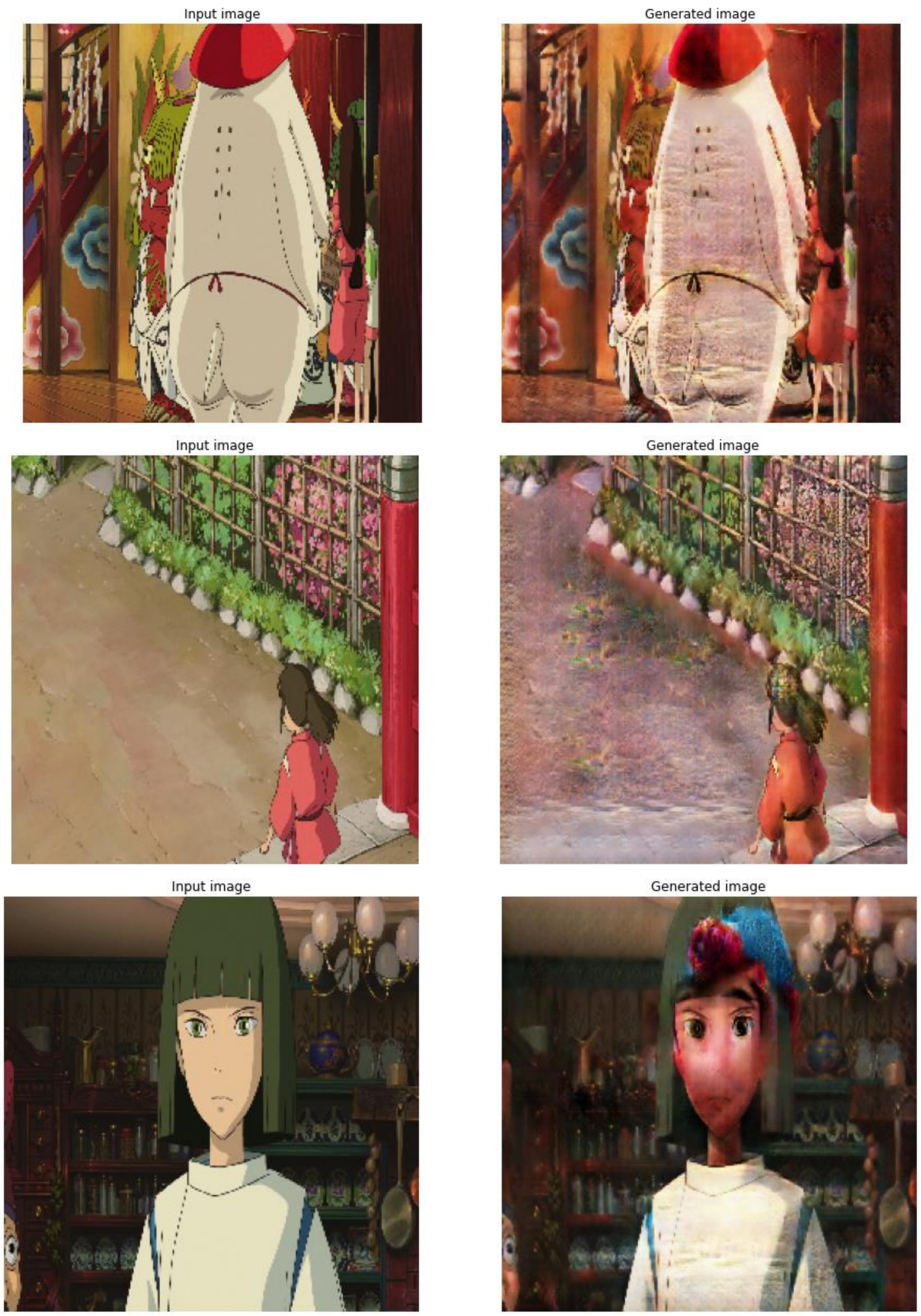


Input image



Generated image




Input image
Generated image
Input image
Generated image
Input image
Generated image