A Anexo

A.1 Trabajo de otros artistas.

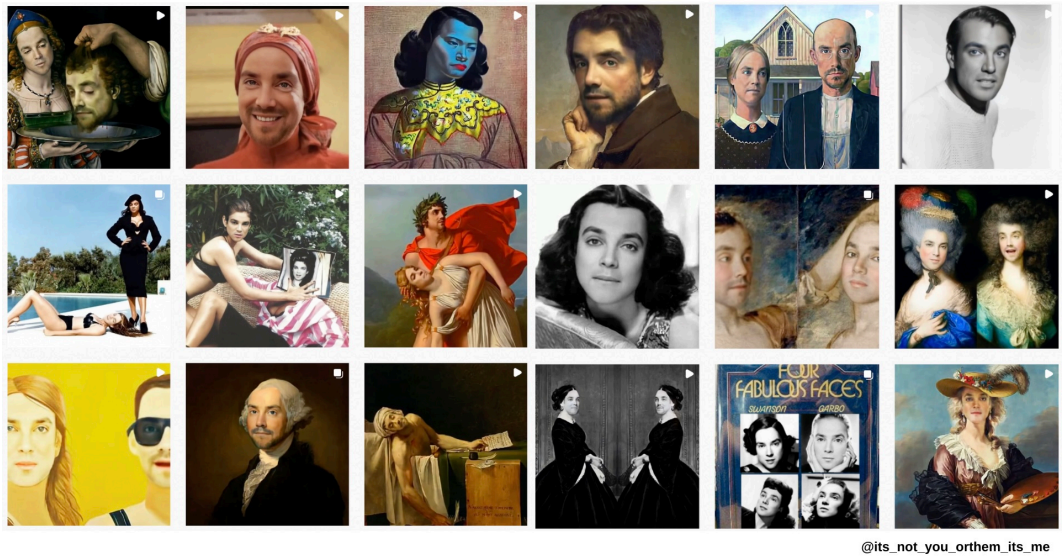

Figura A.1: *Imágenes del trabajo de 'Jaymayokey' en su cuenta de Instagram.*

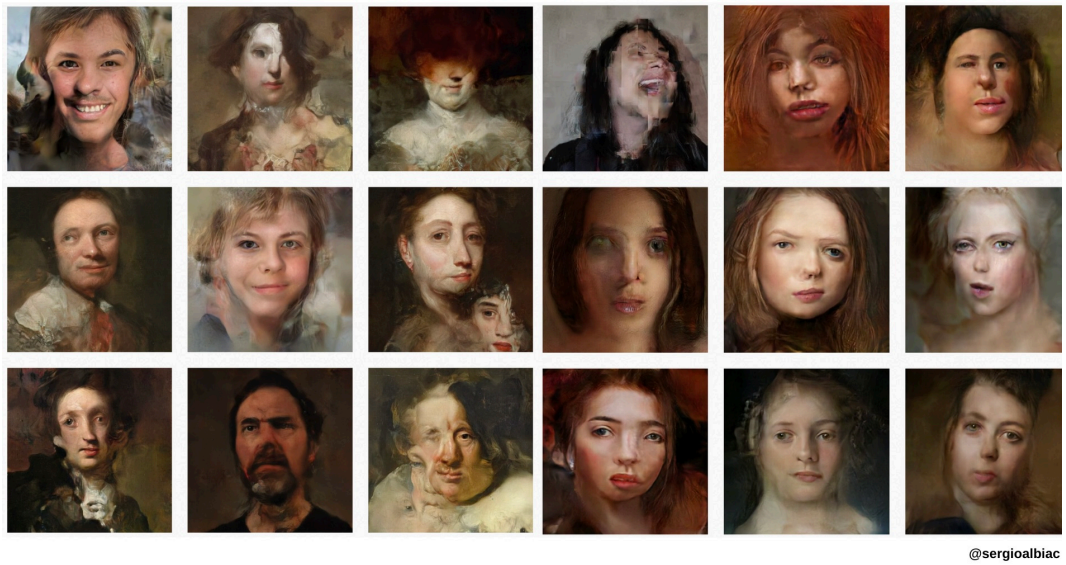

Figura A.2: Imágenes del trabajo de Sergio Albaic en su cuenta de *Instagram* @sergioalbiac.

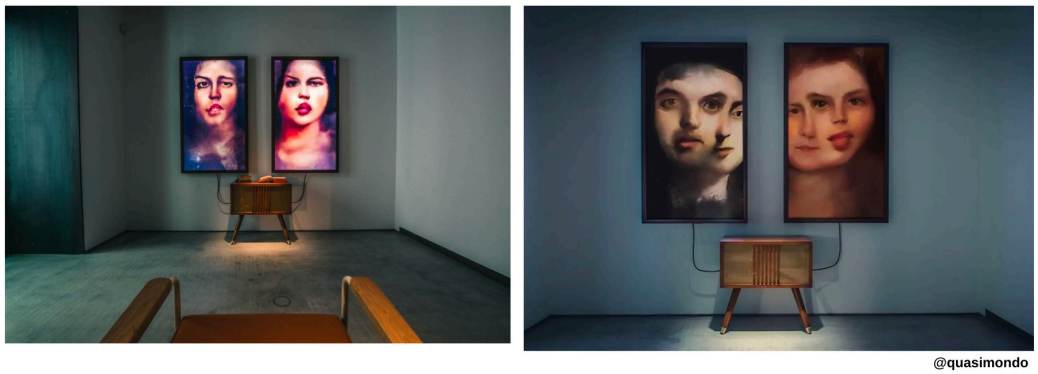

Figura A.3: *Imágenes del trabajo de Mario Klingemann en su cuenta de Instagram @quasimondo.*

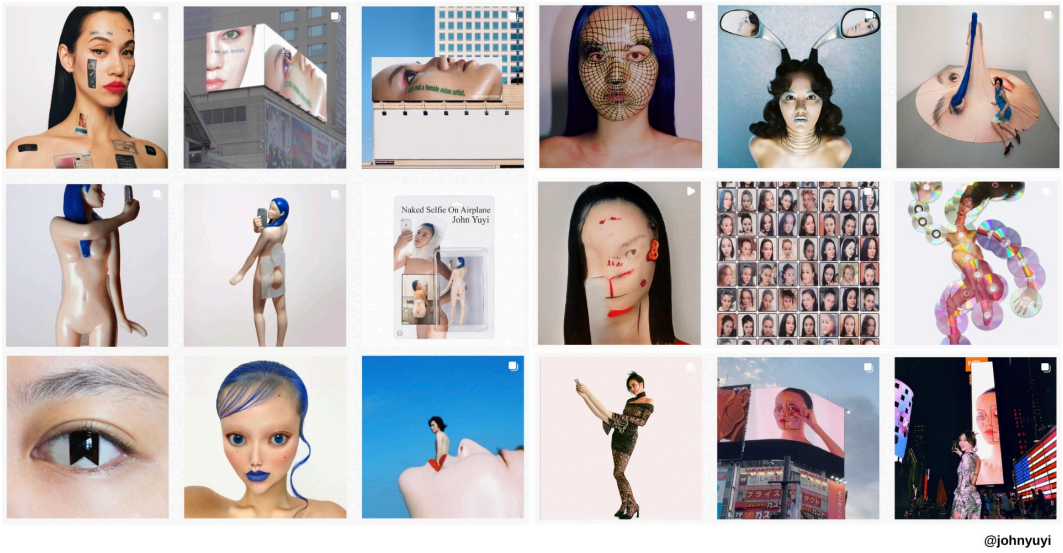

Figura A.4: *Imágenes del trabajo de John Yuyi en su cuenta de Instagram @johnyuyi.*

Figura A.5: *Imágenes del trabajo de Heather Dewey-Hagborg en su cuenta de Instagram @hdeweyh.*

Figura A.6: *Imágenes del trabajo de Carmen de la Roca en su cuenta de Instagram @carmenrocaigual.*

A.2 Retratos adicionales generados por GAN.

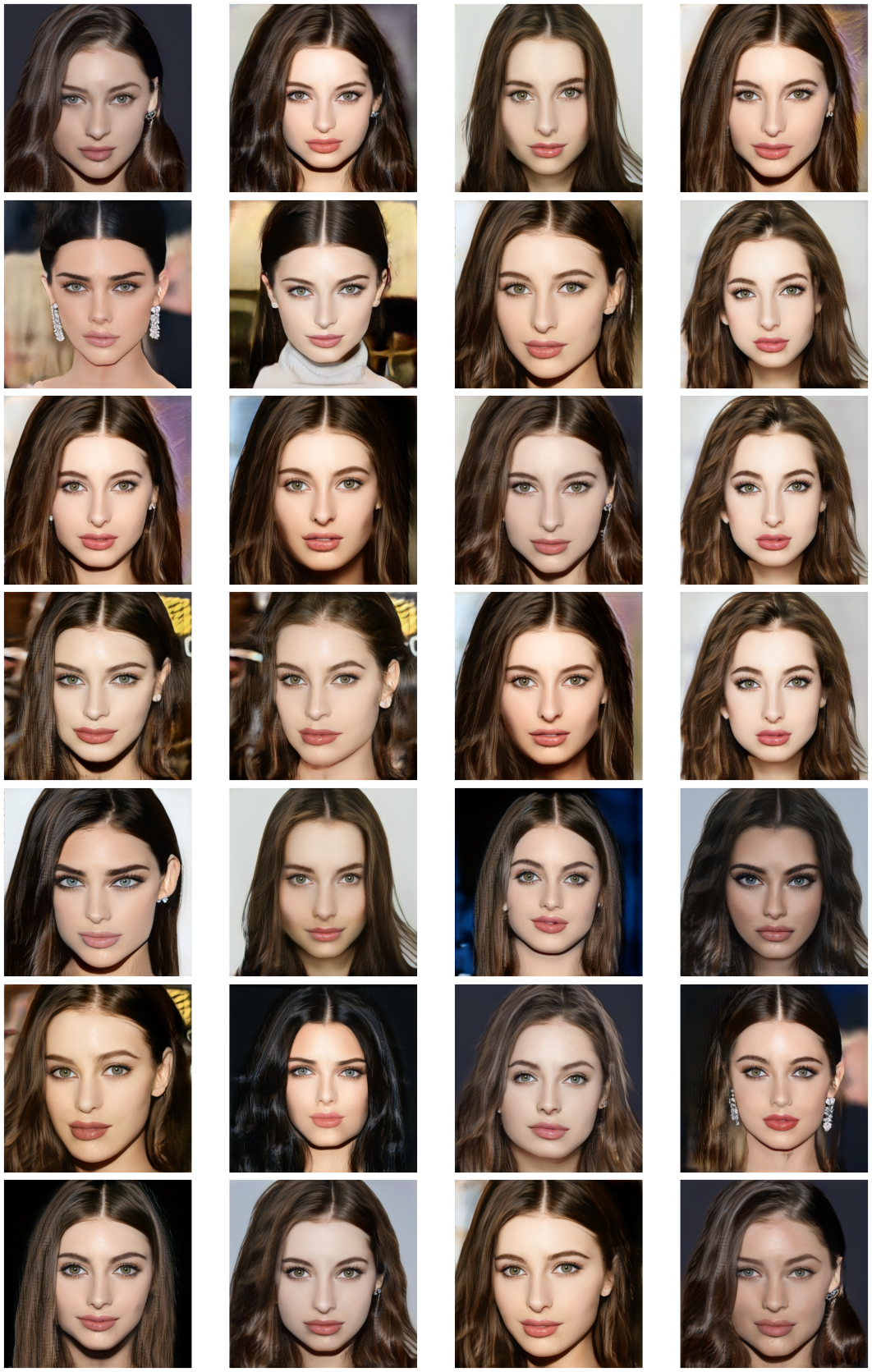

Figura A.7: Retratos modernos generados por GAN adicionales.

A.2.1. Rostros adicionale generados por GAN, como pinturas (truncation>1.5) e implementación de *Style Transfer*.

Figura A.8: GAN generó pinturas debido a los altos valores de *truncation*.

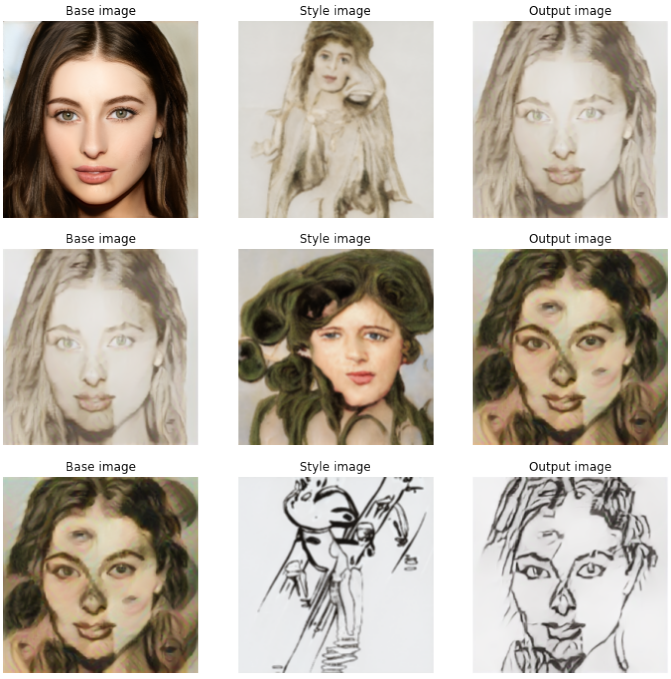

Figura A.9: CNN feed-forward style transfer.

A.2.2. *Entretain/me/nt for /me/n. Play Boy Series.*(2022).

Figura A.10: *Entretain/me/nt for /me/n. Play Boy Series.*(2022).

Figura A.11: *Entretain/me/nt for /me/n. Play Boy Series.*(2022).

Figura A.12: *Entretain/me/nt for /me/n. Play Boy Series.*(2022).

Figura A.13: *Entretain/me/nt for /me/n. Play Boy Series.*(2022).

Figura A.14: *Entretain/me/nt for /me/n. Play Boy Series.*(2022).

Figura A.15: *Entretain/me/nt for /me/n. Play Boy Series.*(2022).

Figura A.16: *Entretain/me/nt for /me/n. Play Boy Series.*(2022).

Figura A.17: *Entretain/me/nt for /me/n. Play Boy Series.*(2022).

A.3 Generative Adverserial Network using pytorch Colab notebok

```
""" GANnoteBook.ipynb

Automatically generated by Colaboratory.

Implementation of a Generative Adverserial Network for VIRself by Stefana Magdalena Tirle
for TMF at UPV. The training Loop is robust and augments the weights of the "Wikiart"
StyleGAN2 network presented by Nvidia in (T. Kerras ,2019).
This is an impressive network which scales well for the generation of 1024x1024 images
(as this is the dimension on which it was developed). Additionally this architecture was improved
to prevent divergent training on small datasets, so it is ideal for our use.
Thank you to Jacques Wolmarans for his insight on computational theory.
"""

!nvidia-smi -L #we require a P100 or higher performance GPU else the training
operations may overflow (for 1024x1024 images)
from google.colab import drive #Mount to our google drive to store logs
drive.mount('/content/drive')
```

```
"""Now we need to add the libraries to our google drive. To do this we clone
the Repository on GitHub into our own drive."""

# Commented out IPython magic to ensure Python compatibility.
import os
!pip install gdown --upgrade

if os.path.isdir("/content/drive/MyDrive/colab-sg2-ada-pytorch"):
#     %cd "/content/drive/MyDrive/colab-sg2-ada-pytorch/stylegan2-ada-pytorch"
elif os.path.isdir("/content/drive/"):
    #install script
#     %cd "/content/drive/MyDrive/"
    !mkdir colab-sg2-ada-pytorch
#     %cd colab-sg2-ada-pytorch
    !git clone https://github.com/dvschultz/stylegan2-ada-pytorch
#     %cd stylegan2-ada-pytorch
    !mkdir downloads
    !mkdir datasets
    !mkdir pretrained
    !gdown --id 1-5xZkD8ajXw1DdopTkH_rAoCsD72LhKU -O
    /content/drive/MyDrive/colab-sg2-ada-pytorch/stylegan2-ada-
    pytorch/pretrained/wikiart.pkl
else:
    !git clone https://github.com/dvschultz/stylegan2-ada-pytorch
#     %cd stylegan2-ada-pytorch
    !mkdir downloads
    !mkdir datasets
    !mkdir pretrained
#     %cd pretrained
    !gdown --id 1-5xZkD8ajXw1DdopTkH_rAoCsD72LhKU #This is the 1024x1024 wikiart trained styleGAN2
#     %cd ../
#Install the necessary software on our GPU runtime.
!pip install ninja opensimplex torch==1.7.1 torchvision==0.8.2
#These are the pytorch versions they used, newer versions lead to errors.

"""Now we need to pre-process and place our training images in
the 'datasets' folder for the training loop to find it. For this
all images need to be square 1024x1024 and placed inside a
single .zip folder. The code below takes all images with the
file extension mentioned and saves them in a new folder with the
correct size."""

import os
import glob
import cv2
in_directory = "/content/gdrive/My Drive/uploaded"
out_directory = "/content/gdrive/My Drive/FinalDataset"
raw_dir = in_directory
save_dir = out_directory
ext = "jpg"
target_size = eval("(1024,1024)")

assert isinstance(target_size, tuple) and len(target_size) == 2, msg
fnames = glob.glob(os.path.join(raw_dir, "*.{}".format(ext)))
os.makedirs(save_dir, exist_ok=True)
print("{} files to resize from directory '{}' to target
size:{}".format(len(fnames), raw_dir, target_size))
```

```
for i, fname in enumerate(fnames):
        print(".", end="", flush=True)
        img = cv2.imread(fname)
        img_small = cv2.resize(img, target_size,interpolation =
        cv2.INTER_AREA) #prevent loss during resize
        new_fname = "{}.{}".format(str(i), ext)
        small_fname = os.path.join(save_dir, new_fname)
        cv2.imwrite(small_fname, img_small)
print(
        "\nDone resizing {} files.\nSaved to directory: `{}`".format(
            len(fnames), save_dir
        )
    )

"""Now it is time to Train or continue training the GAN.
resume_from is the starting point of the training iteration.
snapshot_count represents how many training iterations must run
before a backup of the network is created. It is easier to
update the variables first and detect errors rather than wait to
find out the training iteration failed."""

#required: definitely edit these - Update after each training iteration.
dataset_path = './datasets/FinalUploaded.zip'
#This was the latest network set
resume_from = '/content/drive/MyDrive/colab-sg2-ada-pytorch/styl
egan2-ada-pytorch/results/00023-FinalUploaded-mirror-11gb-gpu-ga
mma50-bg-resumecustom/network-snapshot-000060.pkl'
#This was the abstract network I made at 004 and 008.pkl
#resume_from = '/content/drive/MyDrive/colab-sg2-ada-pytorch/style
gan2-ada-pytorch/results/00037-FinalUploaded-mirror-11gb-gpu-gam
ma50-bg-resumecustom/network-snapshot-000008.pkl'
#I want to try go from the wikiart a very small distance to get
some crazy abstract traversals for style transfer.
#resume_from = '/content/drive/MyDrive/colab-sg2-ada-pytorch/sty
legan2-ada-pytorch/pretrained/wikiart.pkl'
aug_strength = 5.52
#This is a reference to how aggressive the layer weights should
be adjusted. start at 0 for the first training iteration.
train_count = 17
#keep track of how many training iterations you have done.
mirror_x = True #mirror the faces to "extend" the training set
mirror_y = False #not a good idea for faces... but could make some cool abstract stuff

#optional: listed in the readme of the GIT repoository
gamma_value = 50.0
augs = 'bg'
config = '11gb-gpu-complex' #This is GPU DEPENDANT !
snapshot_count = 4
#Training and network data is saved in log.txt, the network weights are stored in the.pkl file.

#The actual training loop (train.py from pytorch)
!python train.py --gpus=1 --cfg=$config --metrics=None
--outdir=./results --data=$dataset_path --snap=$snapshot_count
--resume=$resume_from --augpipe=$augs --initstrength=$aug_strength --gamma=$gamma_value
--mirror=$mirror_x --mirrory=False --nkimg=$train_count

"""Once the Network is trained we can Generate Single Images using Generate.py:
```

```
--network: Make sure the --network argument points to the .pkl you want to use.

--seeds: A kind of starting point in the network. StyleGAN2 input is a 512-dimensional array.
Each seed value generates a different, random array. The same
seed value will also always generates identical arrays if you wish to recreate a result.


--truncation: A measure of how much interpretation is done by
the network to complete images. Recommended between +- 0.5 and
+- 1.0, to balance diversity and training set distance, but
technically it's infinite. Very high values result in colour noise.
"""

!python generate.py --outdir=/content/out/images/ --trunc=0.4 --seeds=12-23
--network=/content/drive/MyDrive/colab-sg2-ada-pytorch/stylegan2-ada-pytorch/results
/00023-FinalUploaded-mirror-11gb-gpu-complex-gamma50-bg
-resumecustom/network-snapshot-000060.pkl

"""We can also scale the output buffer of the GAN to generate
images which aren't square by specifying a size argument.
Interpolation is as a result required, 'symm' works well for faces.
Options: pad, padside, symm, and symmside.
"""

!python generate.py --outdir=/content/out/images/ --trunc=0.7
--size=1920-1080 --scale-type=symm --seeds=0-499
--network=/content/drive/MyDrive/colab-sg2-ada-pytorch/stylegan2
-ada-pytorch/results/00037-FinalUploaded-mirror-11gb-gpu-complex
-gamma50-bg-resumecustom/network-snapshot-000008.pkl

"""Truncation Traversal - A way of observing how the network
interprets the possible variations of a single image. Here
select a single seed as well as truncation limits. A traversal
through '0' which would be the highest degree of certainty for
low output loss will show an image progfressing from abstract to realistic and back again.

The function accepts a seed value, a starting point, end point
and increment for the GAN we want to explore. The result is
saved as a video with png frames on the local GPU device.
You need to save to drive manually !
"""

!python generate.py --process="truncation"
--outdir=/content/out/trunc-trav-3/ --start=-2.0 --stop=2.0
--increment=0.02 --seeds=633
--network=/content/drive/MyDrive/colab-sg2-ada-pytorch/stylegan2
-ada-pytorch/results/00037-FinalUploaded-mirror-11gb-gpu-complex
-gamma50-bg-resumecustom/network-snapshot-000004.pkl

"""Interpolation through the network. Here we input several seed values to explore
the different images the GAN is able to create.
There are obviously several way to change this value, the most obvious being linearly.
However Noise and random circular loops create interesting animations and diverse frames quickly.
The truncation value is here used to determine
how much freedom the GAN has to create images along the path."""

#Linear interpolation along Z space - --frames for specifying how many frames to create
!python generate.py --outdir=/content/out/video1-w-0.25/
```

```
--space="z" --trunc=0.4 --process="interpolation"
--seeds=1,4000 --network=/content/drive/MyDrive/colab-sg2-ada-py
torch/stylegan2-ada-pytorch/results/00037-FinalUploaded-mirror-1
1gb-gpu-complex-gamma50-bg-resumecustom/network-snapshot-000008.pkl

#Linear Interpolation along the 'w' space - produces different
results: a different path inside the network.
!python generate.py --outdir=out/video1-w/ --space="w" --trunc=1
--process="interpolation" --seeds=85,265,297,849
--network=/content/stylegan2-ada-pytorch/pretrained/wikiart.pkl

#Noise Loop to create random frames, effectively using a random number generator to select seeds.
Diameter represents the range for these random values.
!python generate.py --outdir=out/video-noiseloop-8daccurate48/
--trunc=0.48 --process="interpolation"
--interpolation="noiseloop"
--diameter=5000 --random_seed=243
--network=/content/drive/MyDrive/colab-sg2-ada-pytorch/stylegan2
-ada-pytorch/results/00036-FinalUploaded-11gb-gpu-complex-gamma5
0-bg-resumecustom/network-snapshot-000009.pkl

#generates a random pattern of images which has the same
starting and end point - so we go in a circle path through the network.
!python generate.py --outdir=out/video-circularloopAccurate2/
--trunc=0.5 --process="interpolation"
--interpolation="circularloop" --diameter=500.00
--frames=300 --random_seed=1562 --network=/content/drive/MyDrive/colab-sg2-ada-pytorch/stylegan2
-ada-pytorch/results/00036-FinalUploaded-11gb-gpu-complex-gamma5
0-bg-resumecustom/network-snapshot-000009.pkl

"""Projection.py is another useful function in pytorch. Here the GAN attempts to recreate
a target image using it's network weights. Essentially we attach a loss function to the GAN output
to measure success, more iterations 'should' produce a better projection but
this depends on the target. In this case it is probably better to use a human face.
This is how the applications which convert images to paintings
actually work (They use the wikiart dataset). """

#Basic Projector
#Important the target has to be in the same format of the
training set ! steps is how many attempts the network will get.
!python projector.py --network= --outdir=/content/projector/
--target=/content/img005421_0.png --num-steps=200 --seed=0

#Peter Baylies Projector - a diferent loss function - usually better
!python /content/stylegan2-ada-pytorch/pbaylies_projector.py
--network=/content/ladiesblack.pkl --outdir=/content/projector-no-clip-006265-4-inv-3k/
--target-image=/content/img006265-4-inv.png --num-steps=3000
--use-clip=False --use-center=False --seed=99

"""Next is one of the best features of Pytorch, blending network weights to create a new network.
This is highly useful to create unique images. It is difficult to predict the outputs,
so is more of an experimentation. It takes the lower levels of
the one GAN and the higher layers of the other. """

!python blend_models.py --lower_res_pkl
/content/drive/MyDrive/colab-sg2-ada-pytorch/stylegan2
-ada-pytorch/pretrained/wikiart.pkl --split_res 64
--higher_res_pkl /content/drive/MyDrive/colab-sg2-ada-pytorch/st
```

```
ylegan2-ada-pytorch/results/00023-FinalUploaded-mirror
-11gb-gpu-gamma50-bg-resumecustom/network-snapshot-000060.pkl
--output_path /content/customCombination.pkl
```

A.4 Style Transfer using Magenta CNN Colab notebook

```
"""ForAppendixStyleTransfer.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/...
"""

#Mount the google drive directory so we can access local image files.
from google.colab import drive
drive.mount('/content/gdrive')

"""The above is setup work.
Now we are going to pull in images and experiment with them:

This is where we add images to our database
either through URL or Gdrive file locations
"""

#@title
#we are going to load all the images we want to use into a dictionary.
#and "look them up" through call by reference to their name.
#Source_Image_urls = dict(
#  selfie1 ='https://i.pinimg.com/originals/ba/a6/8a/
baa68aa82a39b4bc35c17c98f03f355e.jpg',
#  selfie2 ='https://upload.wikimedia.org/wikipedia/commons/1/
11/Selena_Gomez_Beauty_Secrets_%28cropped%29.png',
#  mirrorselfie = 'https://i.pinimg.com/550x/91/42/11/
914211228f61ffa6832cefdb395c5d34.jpg',
  #It also supports local files in your Gdrive:
#  portrait1 = '/content/gdrive/My Drive/
VGG_transfer/worldsmostfamousphoto.jpg',
#  portrait2 = '/content/gdrive/My Drive/
VGG_transfer/photostyle.jpg'
 # )

 #To present results in context,
 these are results generated using the GAN we trained.

Source_Image_urls = dict(
  #local files in Gdrive:
  #Here I am focusing on using interesting outputs
  from GAN versions to create some new images of interest.

  GAN1 =  '/content/gdrive/MyDrive/frames/frame0041.png',
  GAN2 = '/content/gdrive/MyDrive/frames/frame0113.png',
  GAN3 = '/content/gdrive/MyDrive/frames/frame0002.png',
  GAN4 =  '/content/gdrive/MyDrive/frames/frame0178.png',
  GANA1 = '/content/gdrive/MyDrive/frames/frame0067.png',
```

```
  GANA2 = '/content/gdrive/MyDrive//frames/frame0197.png',
    ### More abstract art combinations
  GANR1 = '/content/gdrive/MyDrive/frames/frame0122.png',
  GANR11 = '/content/gdrive/MyDrive/frames/frame0233.png',
  GANR12 = '/content/gdrive/MyDrive/frames/frame0200.png',
  GANR13 = '/content/gdrive/MyDrive/frames/frame0109.png',
  GANR14 = '/content/gdrive/MyDrive/frames/frame0216.png',
  GANR15 = '/content/gdrive/MyDrive/frames/frame0018.png'
  )

#Now we are going to the load the images and convert them
into 4 dimensional tensors so that we can decompose and
reconstruct them on a tranched basis.
#While we can use any image sizes we will keep them
square for purposes of processing accuracy.

content_image_width = 256
content_image_height = 256
style_image_width = 256
style_image_height = 256

#The Magenta network was trained using 256x256 images and therefore,
it performs style extraction with reduced
content loss on images of this size.
content_images = {k: load_image(v,
(content_image_width, content_image_height))
for k, v in Source_Image_urls.items()}
style_images = {k: load_image(v, (style_image_width, style_image_height))
for k, v in Source_Image_urls.items()}
style_images = {k: tf.nn.avg_pool(style_image, ksize=[3,3], strides=[1,1],
padding='SAME') for k, style_image in style_images.items()}

#This code above simply duplicates the two dictionaries into two arrays
of Tensor versions resized to what we want.

#@title
#Now we import the Magenta CNN for arbitrary style transfer.
It consists of a NN
#which is weighted by the 'Style'
Source image and predicts what the pixels of the
# 'Content' image would look like
if it was part of the 'Style' image.
#It is well suited for our purpose as
it is not as specific as other neural nets in the space.

hub_handle = 'https://tfhub.dev/google/magenta
/arbitrary-image-stylization-v1-256/2'
hub_module = hub.load(hub_handle)

#Note that the NN was trained on 256x256 images,
so Style extraction should be more precise on images of that size.
#The output image will always take on the content image dimensions.

"""

The first Implementation here uses a standard two image approach.
You simply select the two images you wish to use and
load them using their name in the dictionary created earlier."""
```

```
#@title
#Single Image Generation
content_name = 'GAN3'
style_name = 'GANR1' #string
#This is the basic call format to the tensor flow magenta Neural Net
Output_image = hub_module(tf.constant(content_images[content_name]),
                          tf.constant(style_images[style_name]))[0]
#let us present some results:
display(tensor_to_image(Output_image))
#Display all three images alongside one another
show_n([content_images[content_name], style_images[style_name], Output_image],
       titles=['Base image', 'Style image', 'Output image'])
       #display using pyplot

#@title
#Single Image Generation
content_name = 'GAN2'
style_name = 'GANR11' #string
#This is the basic call format to the tensor flow magenta Neural Net
Output_image = hub_module(tf.constant(content_images[content_name]),
                          tf.constant(style_images[style_name]))[0]
#let us present some results:
display(tensor_to_image(Output_image))
#Display all three images alongside one another
show_n([content_images[content_name], style_images[style_name], Output_image],
       titles=['Base image', 'Style image', 'Output image'])
       #display using pyplot

##manual addition - apply a new style image
to this output shown above.
prevoutput = Output_image
style_name = 'GANR13' #string
#This is the basic call format to the tensor flow magenta Neural Net
Output_image = hub_module(tf.constant(prevoutput),
                          tf.constant(style_images[style_name]))[0]
display(tensor_to_image(Output_image))

#In order to experiment with the CNN we construct a feed forward loop
where the result of one style transfer into another.
# We can view the CNN as a dynamic matrix filter adapting
to the properties of the image at each step
in an attempt to transfer the styles.
#This loop iterates through style images applying
#each consecutive photo in the set to a 'base' image.
#We select some new artworks to apply sequentially,
you can experiment with the order to create different results.
Style_Image_urls = dict(
  # Note the sequence matters in the looping generators.
  GAN1 = '/content/gdrive/MyDrive/frames/frame0271.png',
  #Some more abstract artwork output from earlier iterations of the GAN
  GANR3 = '/content/gdrive/MyDrive/frames/frame0137.png',
  GANR15 = '/content/gdrive/MyDrive/frames/frame0067.png',
  GANR13 = '/content/gdrive/MyDrive/frames/frame0109.png'
  )
style_images = {k: load_image(v, (style_image_width, style_image_height))
for k, v in Style_Image_urls.items()}
style_images = {k: tf.nn.avg_pool(style_image, ksize=[3,3],
```

```
strides=[1,1], padding='SAME')
for k, style_image in style_images.items()}
#Things will likely get very abstract
content_loop_name = 'GANA2'
sequentialout = []
sequentialout.append(content_images[content_loop_name])
iterator = 0;
for k in style_images:
  Output_image = hub_module(tf.constant(sequentialout[iterator]),
  tf.constant(style_images[k]))[0]
  iterator = iterator+1;
  sequentialout.append(Output_image)
  #show the raw image added to the styling
  set and the current sum of results
  show_n([sequentialout[iterator - 1],
  style_images[k], Output_image],
       titles=['Base image', 'Style image', 'Output image'])
#Now let us view the final result.
#print("final Output ")
#display(tensor_to_image(Output_image))

#Now in this loop we augment the original
image by using the sequential output as the new style image
#i.e each time we stylize with previous output.
This will create less abstract images.
sequentialoutVaryingContent = []
#clear the intermediate output buffer
iterator = 0
######################################################

#This block of code will accomplish this.
#NOTE THESE NEED TO BE PNG BMG OR JPG WEBP not supported.
#Naming convention of GAN output: video-type of loop - diameter -
accurate/abstract - truncation value - frames - frame
Targeted_Content_Source_Img  = dict(
  #A few very good Accurate artifical Face samples
GANcontent1 =   '/content/gdrive/MyDrive/frames/frame0232.png',
GANcontent2 =   '/content/gdrive/MyDrive/frames/frame0239.png',
GANcontent18 =   '/content/gdrive/MyDrive/frames/frame0178.png'
)

#Some Good abstract art produced using the GAN
Targeted_Style_Source_Img  = dict(
  #A few abstract examples
GANstyle1 =   '/content/gdrive/MyDrive/frames/frame0238.png',
GANstyle2 =   '/content/gdrive/MyDrive/frames/frame0235.png',
GANstyle3 =   '/content/gdrive/MyDrive/frames/frame0233.png',
GANstyle4 =   '/content/gdrive/MyDrive/frames/frame0225.png',
GANstyle5 =   '/content/gdrive/MyDrive/frames/frame0222.png',
GANstyle6 =   '/content/gdrive/MyDrive/frames/frame0211.png',
GANstyle7 =   '/content/gdrive/MyDrive/frames/frame0206.png',
GANstyle8 =   '/content/gdrive/MyDrive/frames/frame0162.png',
GANstyle9 =   '/content/gdrive/MyDrive/frames/frame0153.png',
GANstyle10 =   '/content/gdrive/MyDrive/frames/frame0140.png'
)

Targeted_content_images = {k: load_image(v,
(content_image_width, content_image_height))
```

```
for k, v in Targeted_Content_Source_Img.items()}
Targeted_style_images = {k: load_image(v,
(style_image_width, style_image_height))
for k, v in Targeted_Style_Source_Img.items()}
Targeted_style_images =
{k: tf.nn.avg_pool(style_image, ksize=[3,3],
strides=[1,1], padding='SAME') for k,
style_image in Targeted_style_images.items()}

style_loop_name = 'GANstyle7' #This is the constant content image
sequentialoutVaryingContent.append(Targeted_style_images[style_loop_name])
#place the first style image in the intermediate output buffer.
for k in Targeted_content_images:
  Output_image = hub_module(tf.constant(Targeted_content_images[k]),
  tf.constant(sequentialoutVaryingContent[iterator]))[0]
  iterator = iterator+1;
  sequentialoutVaryingContent.append(Output_image)
  show_n([Targeted_content_images[k],
  sequentialoutVaryingContent[iterator - 1], Output_image],
       titles=['Base image', 'Style image', 'Output image'])

######################################################
#We now expand this loop to create extremely abstract
photos by each time using the result as a content image,
#and further transferring the style of a new
abstract artwork to it in a feed forward structure.
######################################################
#Some Good abstract art produced using the GAN
Targeted_Style_Source_Img  = dict(
  #A few abstract examples
GANstyle1 =   '/content/gdrive/frames/frame0238.png',
GANstyle2 =   '/content/gdrive/MyDrive/frames/frame0235.png',
GANstyle3 =   '/content/gdrive/MyDrive/frames/frame0233.png',
GANstyle4 =   '/content/gdrive/MyDrive/frames/frame0225.png',
GANstyle5 =   '/content/gdrive/MyDrive//frames/frame0222.png',
GANstyle6 =   '/content/gdrive/MyDrive/frames/frame0211.png',
GANstyle7 =   '/content/gdrive/MyDrive/frames/frame0206.png'

)
#we edit the dictionary as we want - experimentation is key!
Targeted_style_images = {k: load_image(v, (style_image_width,
style_image_height)) for k, v in Targeted_Style_Source_Img.items()}
Targeted_style_images = {k: tf.nn.avg_pool(style_image, ksize=[3,3],
strides=[1,1], padding='SAME')
for k, style_image in Targeted_style_images.items()}
iterator = 0 #reset the iterator for the loop
sequentialoutVaryingContent = [] #clear the intermediate buffer
sequentialoutVaryingContent.append(Targeted_content_images['GANcontent18'])
#This is the image you want to start with
#print('This is the base content image')
#display(tensor_to_image(Targeted_content_images['GANcontent18']))
for k in Targeted_style_images: #now step through the style
images each time using previous output as content image
  Output_image =
  hub_module(tf.constant(sequentialoutVaryingContent[iterator]),
  tf.constant(Targeted_style_images[k]))[0]
  iterator = iterator+1;
  sequentialoutVaryingContent.append(Output_image)
```

```
  show_n([sequentialoutVaryingContent[iterator - 1],
  Targeted_style_images[k], Output_image],
  titles=['Base image', 'Style image', 'Output image'])
  #display(tensor_to_image(Targeted_style_images[k]))
  #print('This is the current style image')
  #display(tensor_to_image(Output_image))
  #print('This is the current output image')

  #This code is easily experimented with to
  create abstract art with any input photograph or image.
  #After around 10 iterations the content
  loss in the original base image becomes substantial.

#This is a function which augments calls to show_n
to quickly display images in a grid for the report
def displaydict(pictures, columns):
  content_images = {k: load_image(v, (512, 512))
  for k, v in pictures.items()}
  #The image size here can be chosen and it will be
  processed with minimal loss interpolation names = []
  for index in content_images:
    names.append(index)
  for i in range(columns,len(pictures),columns):
    show_n([content_images[names[i-4]],content_images[names[i-3]],
    content_images[names[i-2]],
    content_images[names[i-1]] ], titles=['', '', '', ''])
displaydict(Targets, 4)
```