Generative Adversarial Networks: GANs

Andres Carrillo

Outline:

- Definition of GANs
- GAN Structure:
  - Generator
  - Discriminator
    - Coding example of both.
- GAN pitfalls and Solutions.
- Review of specialized GANs
  - Describe how GANs can be combined to create larger programs.
    - Deep fakes and puppeteering

Intro:

GANS:

- First described in 2014 with the paper “Generative Adversarial Nets” by Ian J. Goodfellow et al.
    - Inspired by the idea of a forger and critic working against each other.
    - Similar to MiniMax
- Since 2014 the numbers of GANs and its applications have exploded.
    - Spawning research by Nvidia, medical industries, marketing companies, animation studios, and more.

GAN Structure

GANs are a type of machine learning model unlike most other models which simply try to minimize a loss function, GANs pit two models against each other. One is the Generator and the other is the Discriminator.

- Loss: $L = E_x(\log [D(x)]) + E_z(\log [D(G(z))])$

Training Routine:

1. Train Discriminator using real images.
2. Pass noise through Generator to create fake images.
3. Pass output from Generator into the Discriminator mixed with real images.
4. Update the Generator based on the loss of the Discriminator.
5. Repeat until the Discriminator is unable to tell the difference between a real and fake image.

GAN Structure:

A GAN consists of two parts:

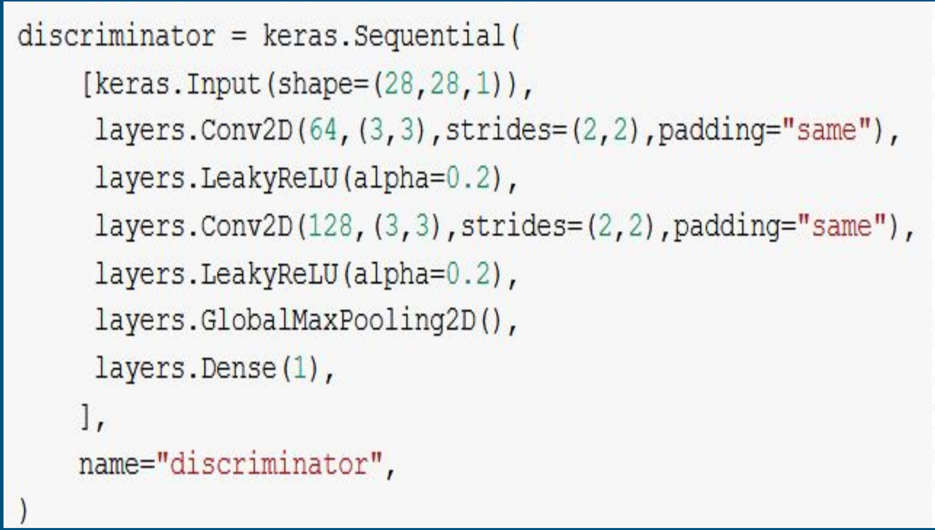
- Generator:
  - Learns to create fake data that mimics the training data.
  - Never sees training data, instead the generator is trained solely based on feedback from the loss of the discriminator.
- Discriminator:
  - Classifier that attempts to tell fake input from real input, typically ends in single sigmoid layer.
  - Convolution based in order to learn underlying patterns within data.
  - Updates the Generator on its ability to fool the discriminator.

Discriminator:

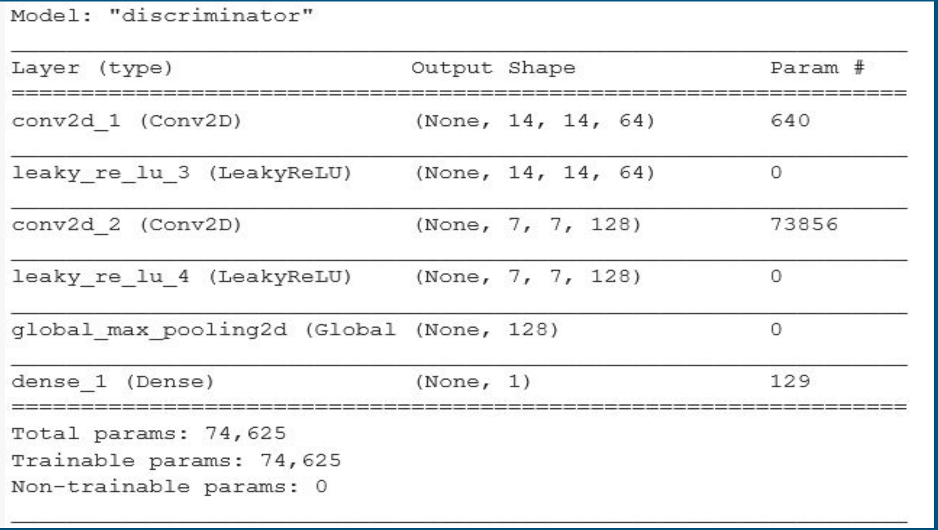
Binary classifier:

- If the image is real returns 1, else if the image is fake returns 0
- Internal structure can be any image clarifier, i.e Dense or Convolutional Networks.
- Patterns learned by Discriminator are both translationally invariant, and are spatial hierarchies of patterns.
    - Convolution based classifier.
    - Convnets can be seen as stacks of feature maps each of which capture some pattern within the input.
    - Each feature map consists of set of smaller filters from each layer.

Discriminator Ex:



```
discriminator = keras.Sequential(
    [keras.Input(shape=(28,28,1)),
     layers.Conv2D(64,(3,3),strides=(2,2),padding="same"),
     layers.LeakyReLU(alpha=0.2),
     layers.Conv2D(128,(3,3),strides=(2,2),padding="same"),
     layers.LeakyReLU(alpha=0.2),
     layers.GlobalMaxPooling2D(),
     layers.Dense(1),
    ],
    name="discriminator",
)
```

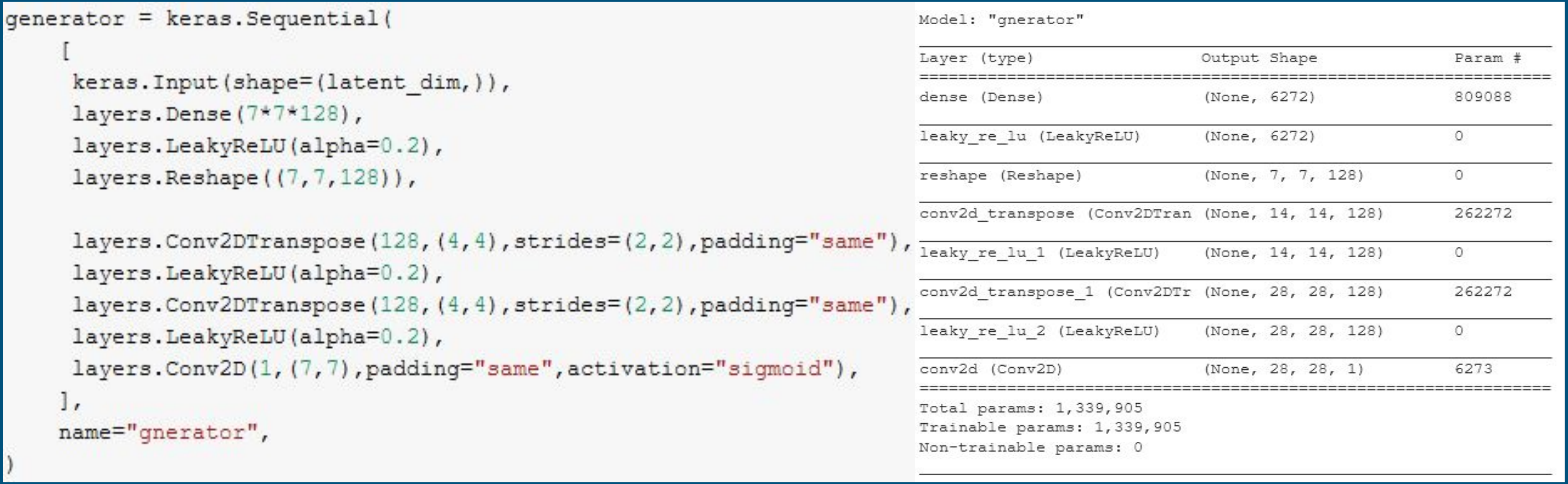


```
Model: "discriminator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 14, 14, 64)        640
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 7, 7, 128)         73856
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 7, 7, 128)         0
_________________________________________________________________
global_max_pooling2d (Global (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 129
=================================================================
Total params: 74,625
Trainable params: 74,625
Non-trainable params: 0
_________________________________________________________________
```

Generator:

- Takes random noise as input, which is expected to be a Gaussian or normal Distribution.
- Passes noise through a series of convolutional transpose layers and regularization layers.
  - Weights of each transpose is updated using the loss output by the Discriminator.
  - As noise passes through the generator the goal is to maximize the activation of each layer.
    - This acts like a filter through which the noise is shaped.
    - Generator can be thought of as a classifier in reverse.

Generator Ex:



```
generator = keras.Sequential(
    [
     keras.Input(shape=(latent_dim,)),
     layers.Dense(7*7*128),
     layers.LeakyReLU(alpha=0.2),
     layers.Reshape((7,7,128)),

     layers.Conv2DTranspose(128,(4,4),strides=(2,2),padding="same"),
     layers.LeakyReLU(alpha=0.2),
     layers.Conv2DTranspose(128,(4,4),strides=(2,2),padding="same"),
     layers.LeakyReLU(alpha=0.2),
     layers.Conv2D(1,(7,7),padding="same",activation="sigmoid"),
    ],
    name="gnerator",
)
```

```
Model: "gnerator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 6272)              809088
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 6272)              0
_________________________________________________________________
reshape (Reshape)            (None, 7, 7, 128)         0
_________________________________________________________________
conv2d_transpose (Conv2DTran (None, 14, 14, 128)       262272
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 14, 14, 128)       0
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 28, 28, 128)       262272
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 28, 28, 128)       0
_________________________________________________________________
conv2d (Conv2D)              (None, 28, 28, 1)         6273
=================================================================
Total params: 1,339,905
Trainable params: 1,339,905
Non-trainable params: 0
_________________________________________________________________
```

GAN Pitfalls:

- Large memory and computational requirements.
  - A typical GAN consists of millions of parameters, and require hundreds of thousands of training samples and training iterations.
- Vanishing Gradient:
  - The discriminator becomes to good, failing to return enough of an error for the generator to continue to train.
- Mode Collapse:
  - The Generator finds some niche output that can fool the discriminator, causing it to constantly output images that lack variation.

Pitfall Solutions:

- Wasserstein Loss:
  - Used to combat both vanishing Gradient and mode collapse.
    - Discriminator loss: D(x) - D(G(z))
    - Generator loss: D(G(z))
- Schocastity:
  - Injecting noise/randomness between layers helps prevent models from failing to converge.
- Mini-batch Discrimination:
  - Measures similarities of images across batches, in order to drop images that appear to similar on mass.
  - Usually when trying to avoid mode collapse.

StyleGAN:

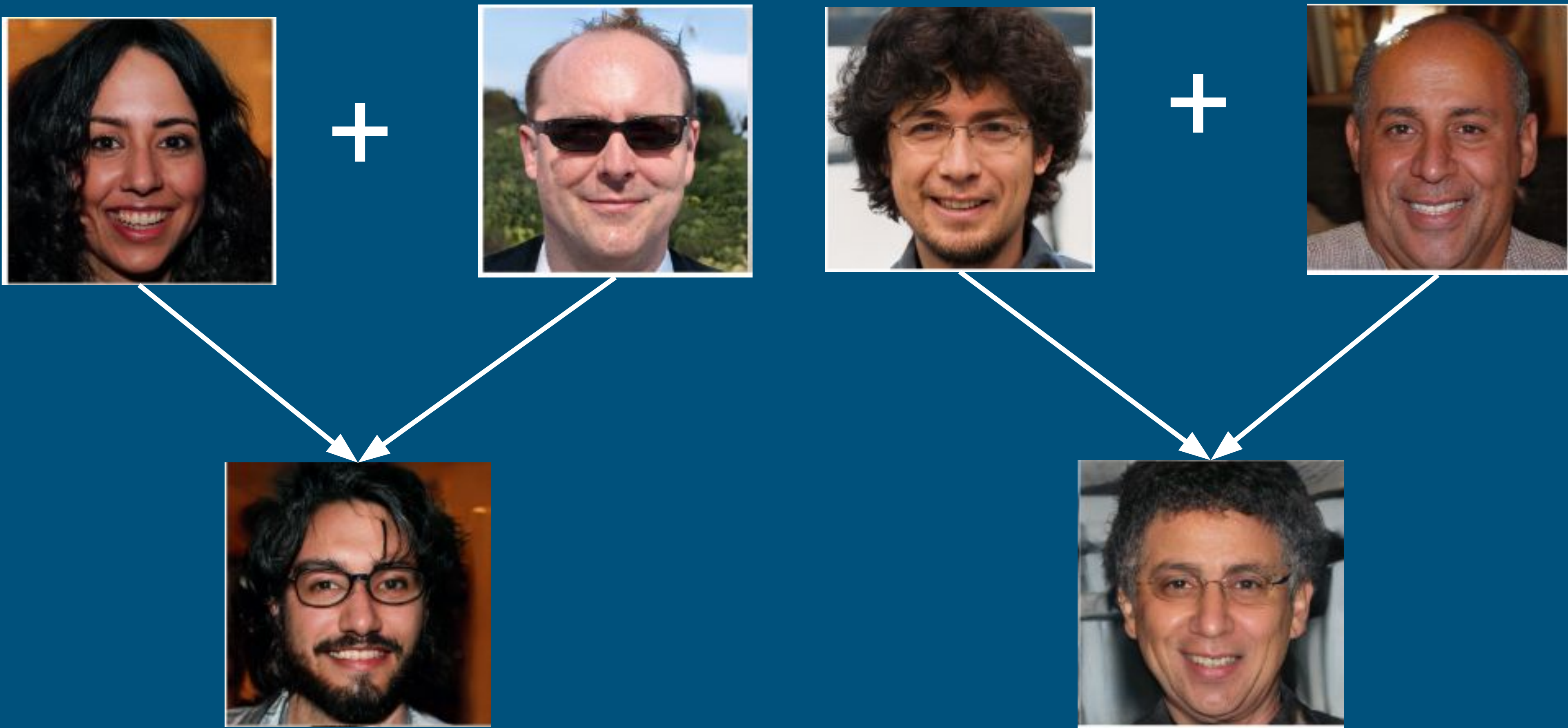
- Created by Nvidia researchers in 2018.
- Updated to StyleGAn v2 in December 2019.
- Returns high quality human faces.
  - Thispersondoesnotexist.com
- Originally trained using the CelebA-HQ
- Injects noise between output layers in order to add randomness.
  - This helps add things like freckles, hair strands, wrinkles etc.

StyleGAN:

StyleGAN Features:

- Dataset: FFHQ, flickr-face-HQ
  - Provides better quality photos with which to train the Discriminator
- Input: Rather than beginning with noise input is some known constant.
- Regularization: Between each convolution uses Adaptive instance normalization (AdaIN) layer that aligns the mean and variance of the content features with those of the style features. Which helps the generator generalize more.
- Style Mixing: Creates two "source" images then mixes features from both to create final image.

Style Mixing Example:



MoCoGAN:

Motion and Content decomposed Generative Adversarial Network.

- Used to generate video content.
- Splits video content into two parts:
  - Content: Represents face, person or object generated by GAN.
  - Motion: Represents dynamics of content.
- Learns Content and Motion separately, using content to make a subspace representing the possible movements, the GAN then samples from this new predictive space to predict the next frames.
- Allows you to keep content constant, while changing motion, in order to create video of one object in motion.

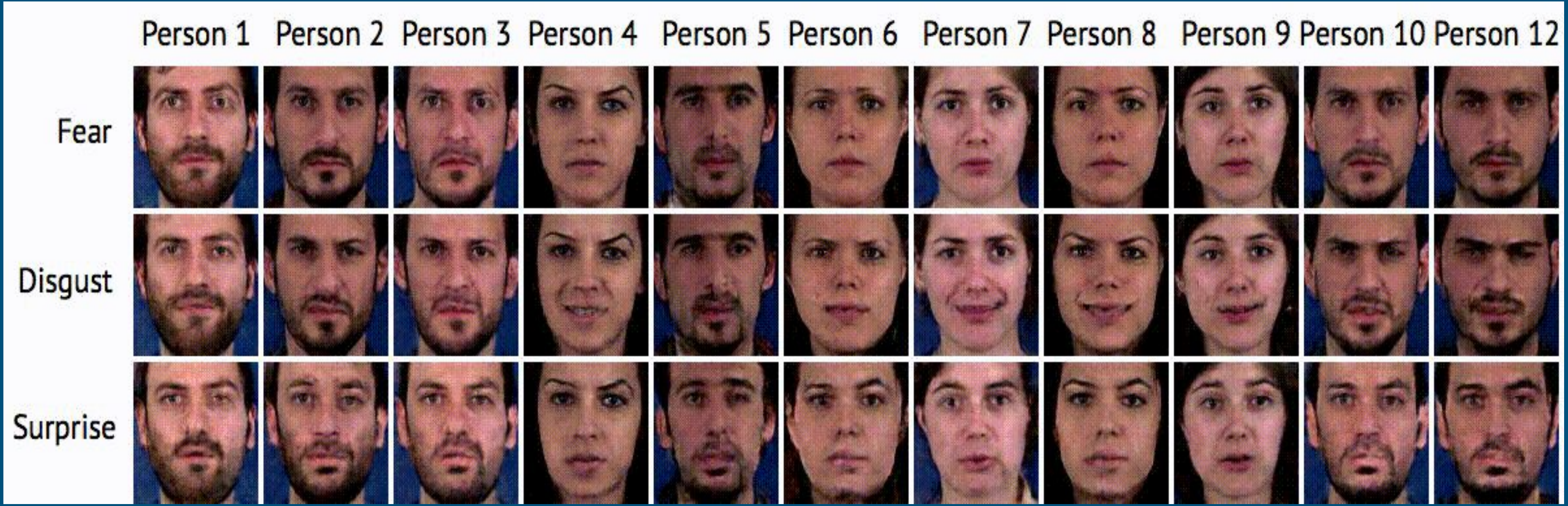
MoCoGAN:

Structure:

- Discriminator:
  - D_I = Image Discriminator
  - D_V = Video Discriminator,
    - Uses Spatio-Temporal CNN, to map time as well as image content.
    - Feeds loss directly to R_M
- Generator:
  - R_M = Recurrent Neural Network to generate motion/video clips.
  - G_c = Content generator.

MoCoGAN Example:

Each row represents an expression, while each column represents
The same person.



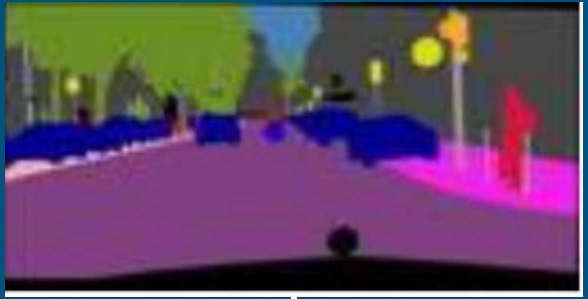
MoCoGAN Example:

Trained using 4k Tai Chi videos gathered from youtube

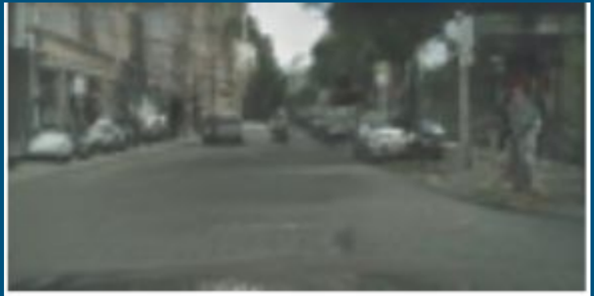
Cycle-Consistent GAN:

- Cycle-Consistent:
  - Given some translation from G: X → Y and another translation F: Y→ X, then G and F are inverses.
  - Model seeks to learn both G and F at the same time.
  - Combines Cycle-Consistent loss function with Adversarial loss.
- Used for image-to-image translation.
  - Transfer art style
  - Image to label
  - Edge map to photograph.
- Uses unlabeled input image pairs.
  - For image translation treats images as different filters on the same scene. Then attempts to learn the underlying relationship between the two images.

Edge Map Input:



GAN Reconstruction:

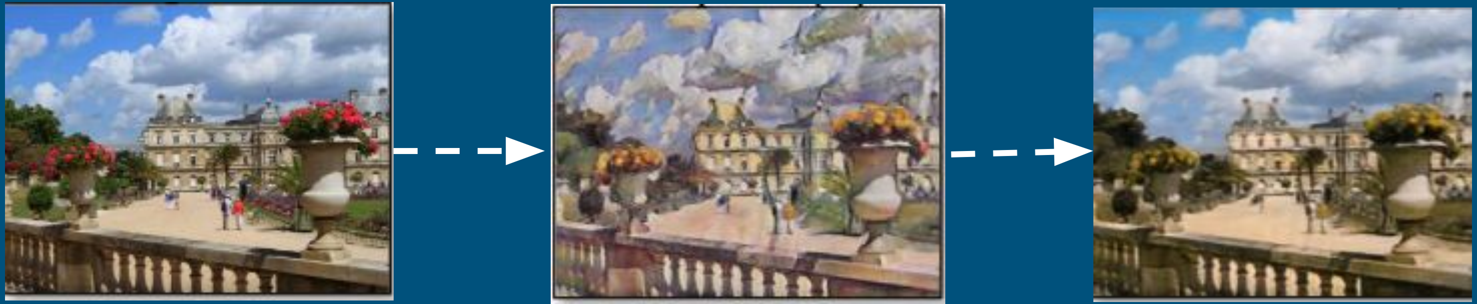


Cycle-Consistent GAN:

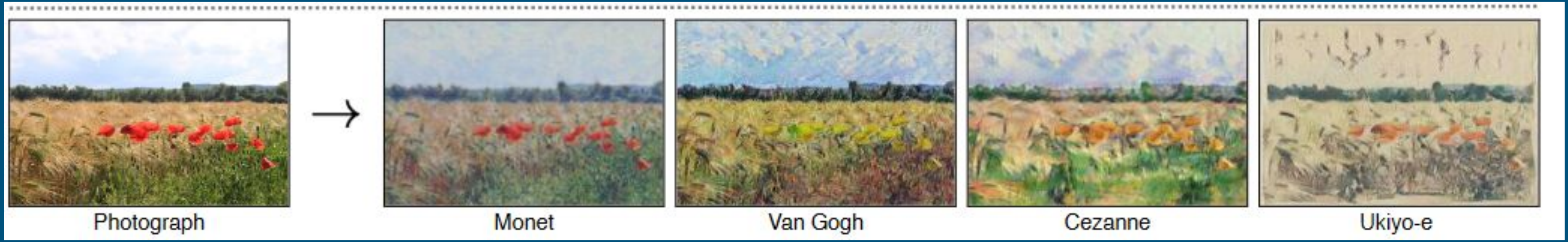
Structure:

- Discriminator:
    - D_x = Tries to distinguish between images.
    - D_y = Attempts to distinguish between Y and G(X)
- Generator:
    - G = learns mapping from $X \rightarrow Y$ using the loss returned from D_y
    - F = learns mapping from $Y \rightarrow X$ using loss from D_x

Input:

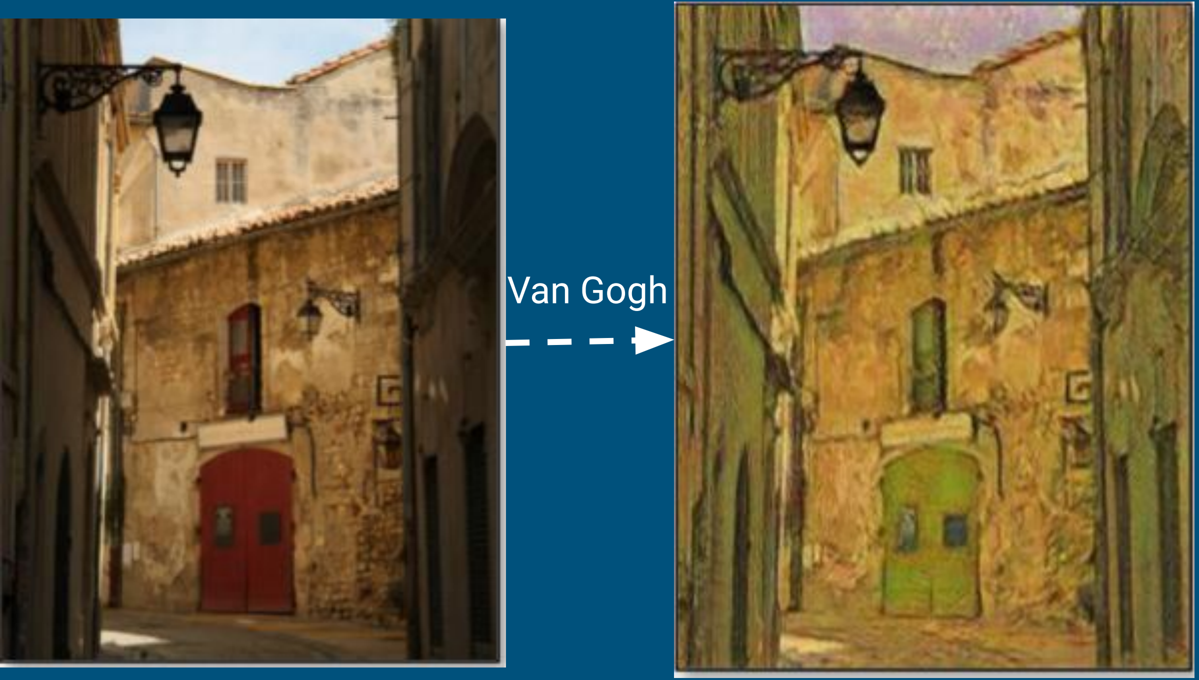


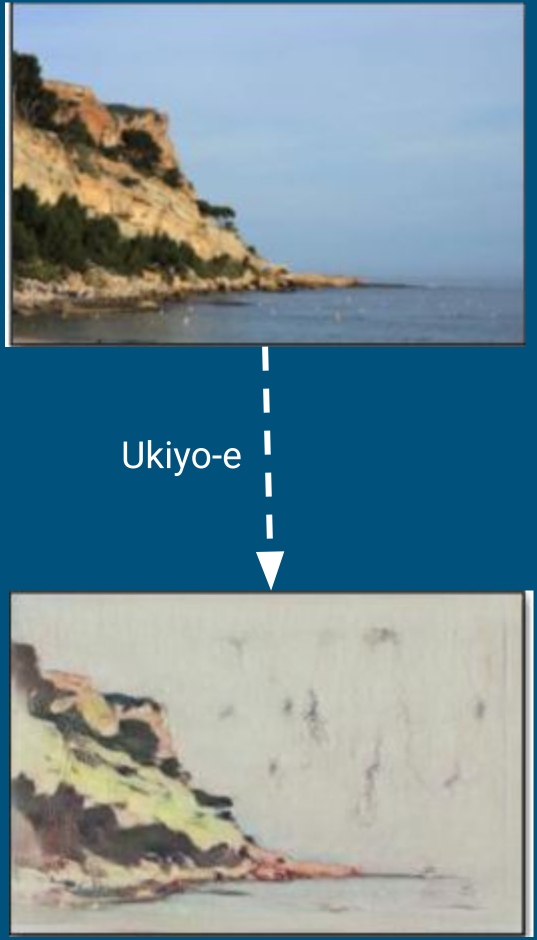
CycleGAN Examples:



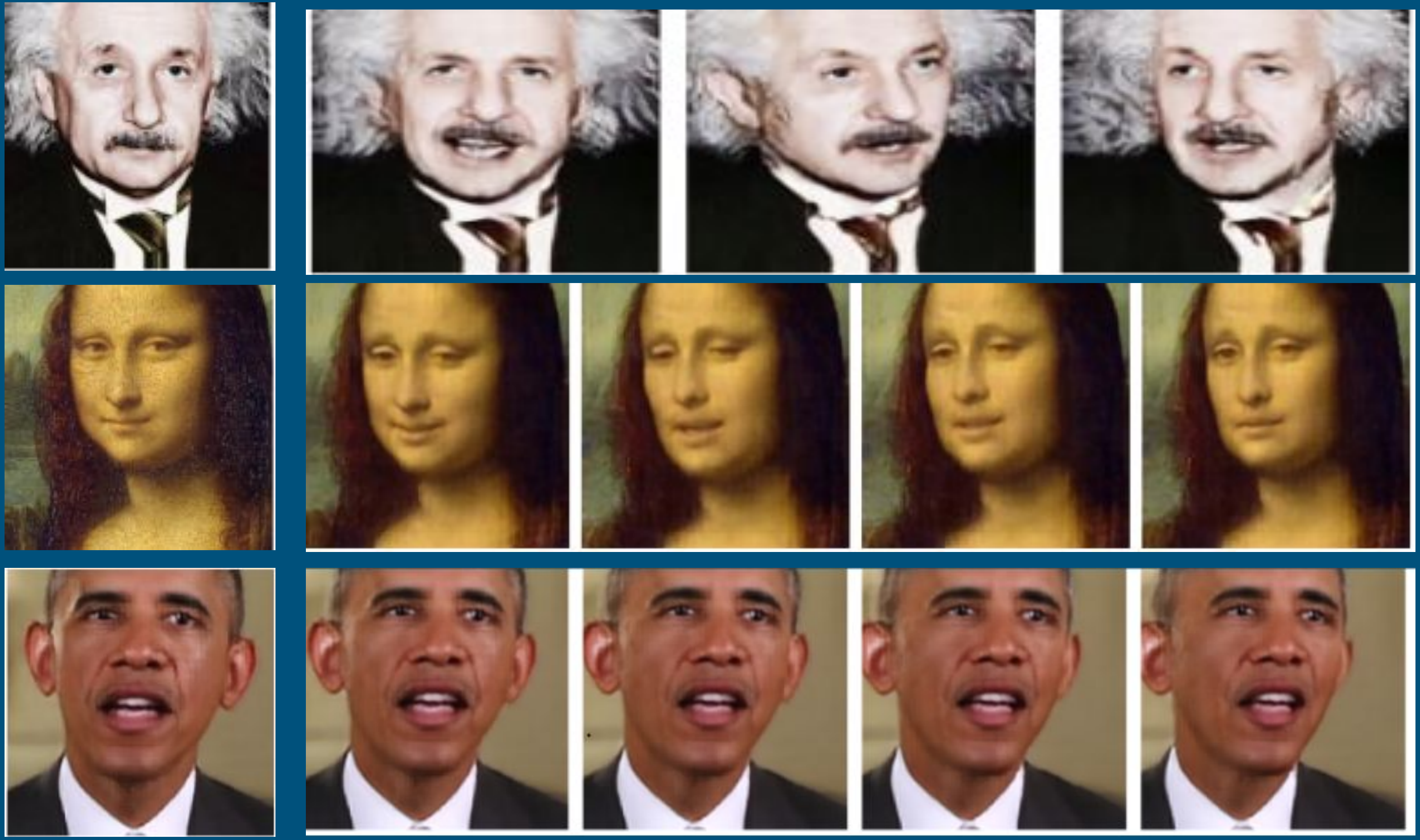
CycleGAN Examples:

Photo-to-Painting:





Deep Fakes and Puppeteering



Summary:

- Defined what a GAN is and its main components.
    - Generator.
    - Discriminator.
    - Training Routine.
    - Described applications:
- Looked at coding example for both a Generator and Discriminator.
- Discussed pitfalls and mitigation techniques.
- Reviewed specialty GANs:
    - StyleGAN: Creates high quality images of fake people.
    - MoCoGAN: Generates video content.
    - CycleGAN: Inspired by impressionist artist.
- Deep Fakes & Puppeteering:
    - Took a look at how our specialty GANs might be put together to create more complex programs.