GAN
&
AUTOENCODERS

NIKITA – A023119820034

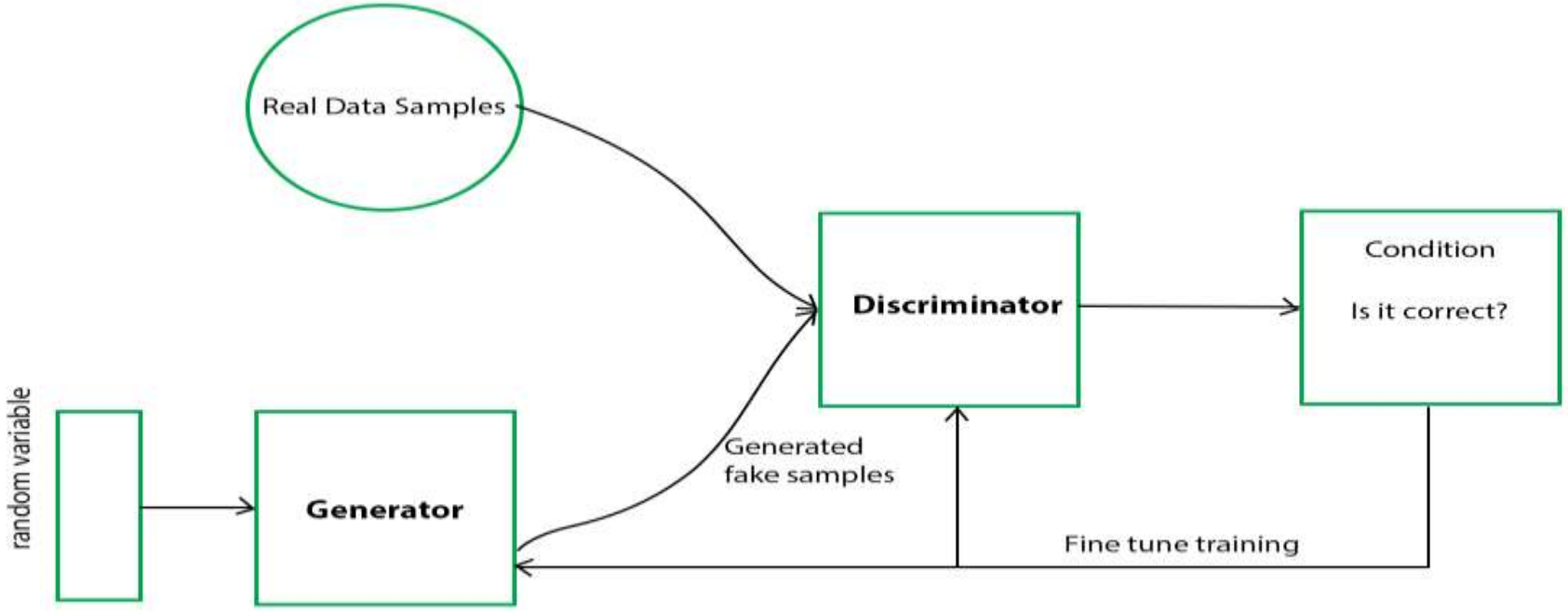
HITESH – A023119820027

UNDER THE GUIDANCE OF

Prof. (Dr) Archana Singh
H.O.D
Department of Artificial Intelligence

Generative Adversarial Network (GAN)

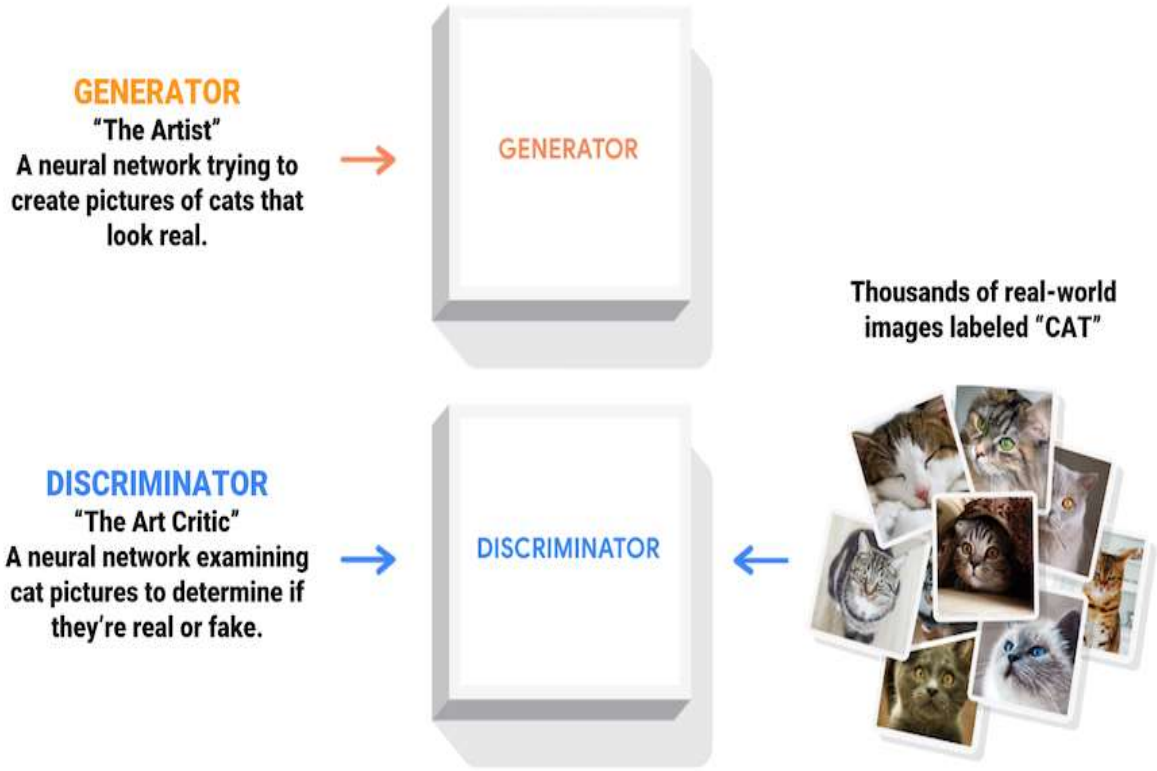
It is a type of artificial neural network architecture used in machine learning and deep learning. It is used for various tasks in the field of computer vision, image generation, and generative modeling.

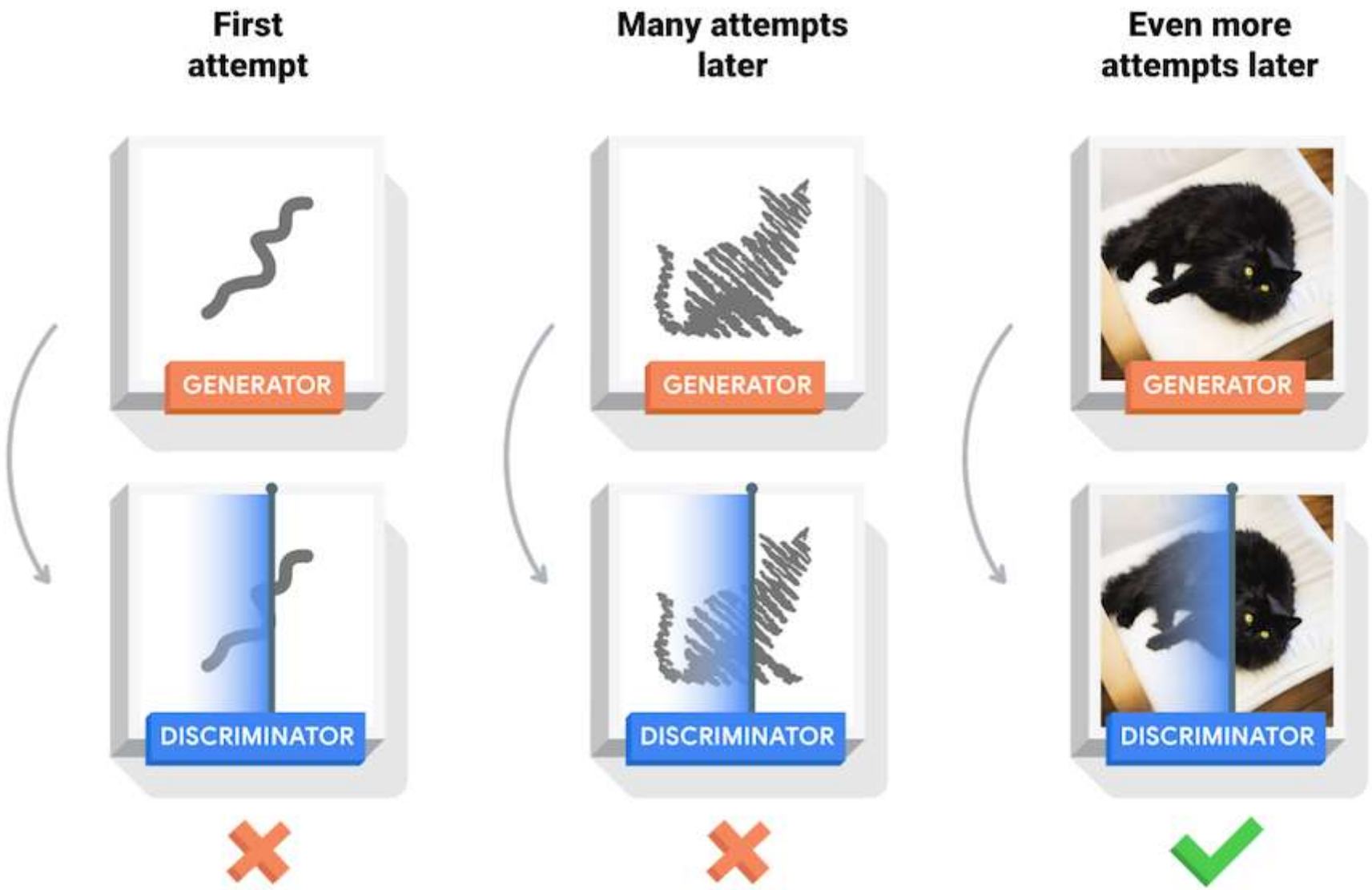


A GAN consists of two neural networks:

- **Generator:** The generator network takes random noise as input and generates data, such as images, audio, or text. Its goal is to produce data that is indistinguishable from real data.

- **Discriminator**: The discriminator network, on the other hand, evaluates the data it receives and tries to distinguish between real data and data generated by the generator. Its goal is to correctly classify data as real or fake.




First attempt
Many attempts later
Even more attempts later
GENERATOR
GENERATOR
GENERATOR
DISCRIMINATOR
DISCRIMINATOR
DISCRIMINATOR

APPLICATIONS

Image Generation: GANs can create realistic-looking images, even of objects or scenes that don't exist in the real world.

Super-Resolution: They can enhance the resolution and quality of images, making them useful for tasks like upscaling images.

Style Transfer: GANs can be used for artistic style transfer, where the style of one image is applied to another.

Data Augmentation: GANs can generate additional training data for machine learning models, especially in cases where data is limited.

Anomaly Detection: They can help identify anomalies in datasets by learning the normal patterns and flagging deviations from them.

Face Aging and Morphing: GANs can be used to age or morph faces in images, which has applications in entertainment and forensics.

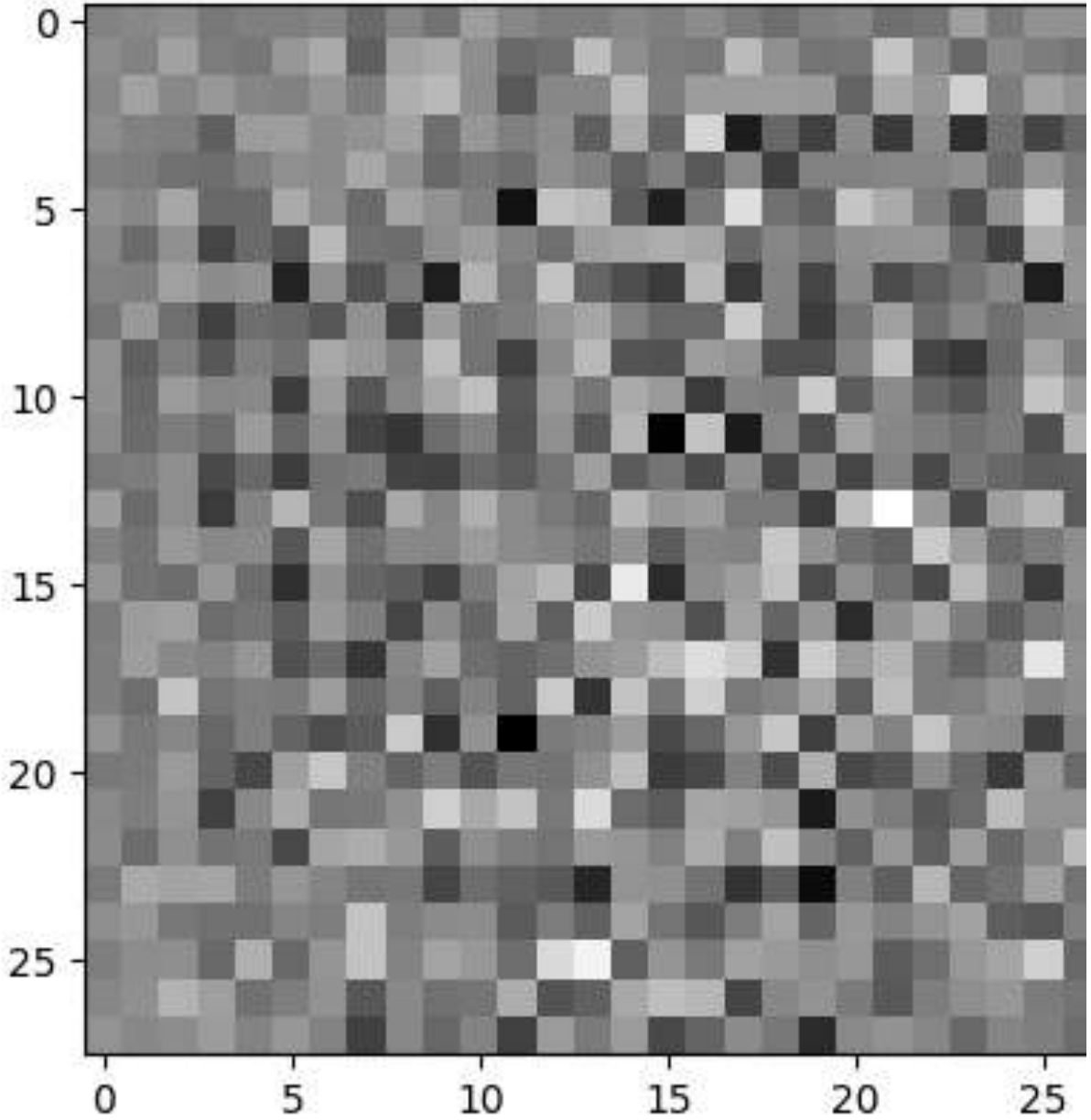
Drug Discovery: GANs can be used to generate molecular structures for drug discovery and design.

Text-to-Image Synthesis: GANs can convert textual descriptions into corresponding images.

SIMPLIFIED CODE

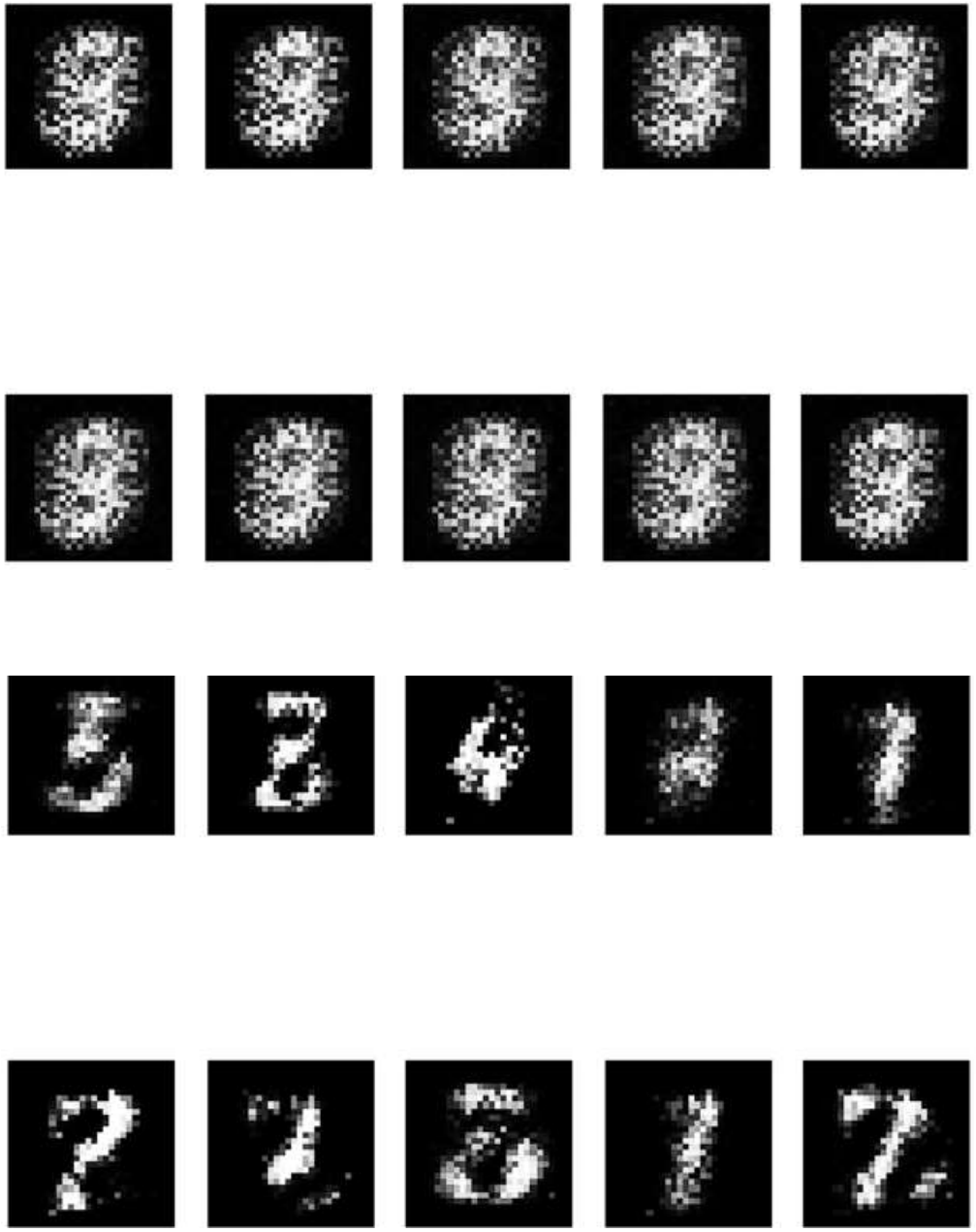
0s

```
def make_generator_model():
    model = tf.keras.Sequential()
    model.add(layers.Dense(7*7*256, use_bias=False, input_shape=(100,)))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Reshape((7, 7, 256)))
    assert model.output_shape == (None, 7, 7, 256)  # Note: None is the batch size

    model.add(layers.Conv2DTranspose(128, (5, 5), strides=(1, 1), padding='same', use_bias=False))
    assert model.output_shape == (None, 7, 7, 128)
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Conv2DTranspose(64, (5, 5), strides=(2, 2), padding='same', use_bias=False))
    assert model.output_shape == (None, 14, 14, 64)
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Conv2DTranspose(1, (5, 5), strides=(2, 2), padding='same', use_bias=False, activation='tanh'))
    assert model.output_shape == (None, 28, 28, 1)

    return model
```


0
5
10
15
20
25
0
5
10
15
20
25

0s

```
def make_discriminator_model():
    model = tf.keras.Sequential()
    model.add(layers.Conv2D(64, (5, 5), strides=(2, 2), padding='same',
                            input_shape=[28, 28, 1]))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))

    model.add(layers.Conv2D(128, (5, 5), strides=(2, 2), padding='same'))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))

    model.add(layers.Flatten())
    model.add(layers.Dense(1))

    return model
```



Objective:

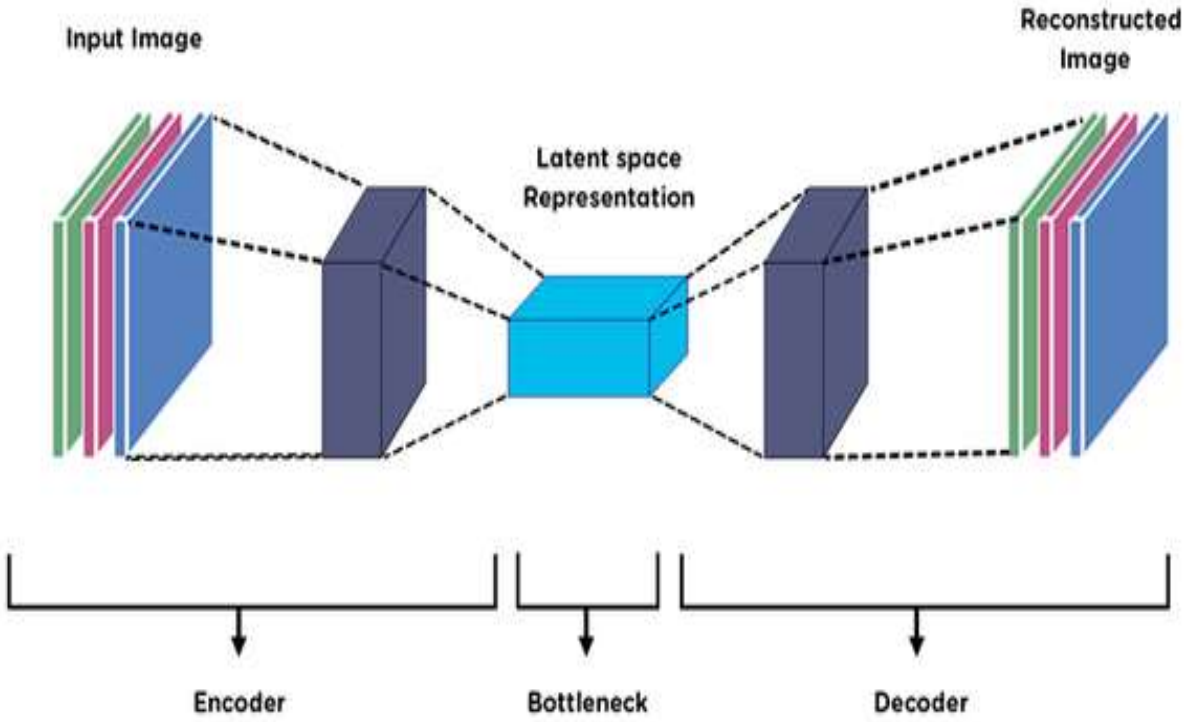
To get color images from black -n- white.

- The proposed method uses the DeOldify python library. DeOldify is a Black and White image Colorizing library created by Jason Antic.

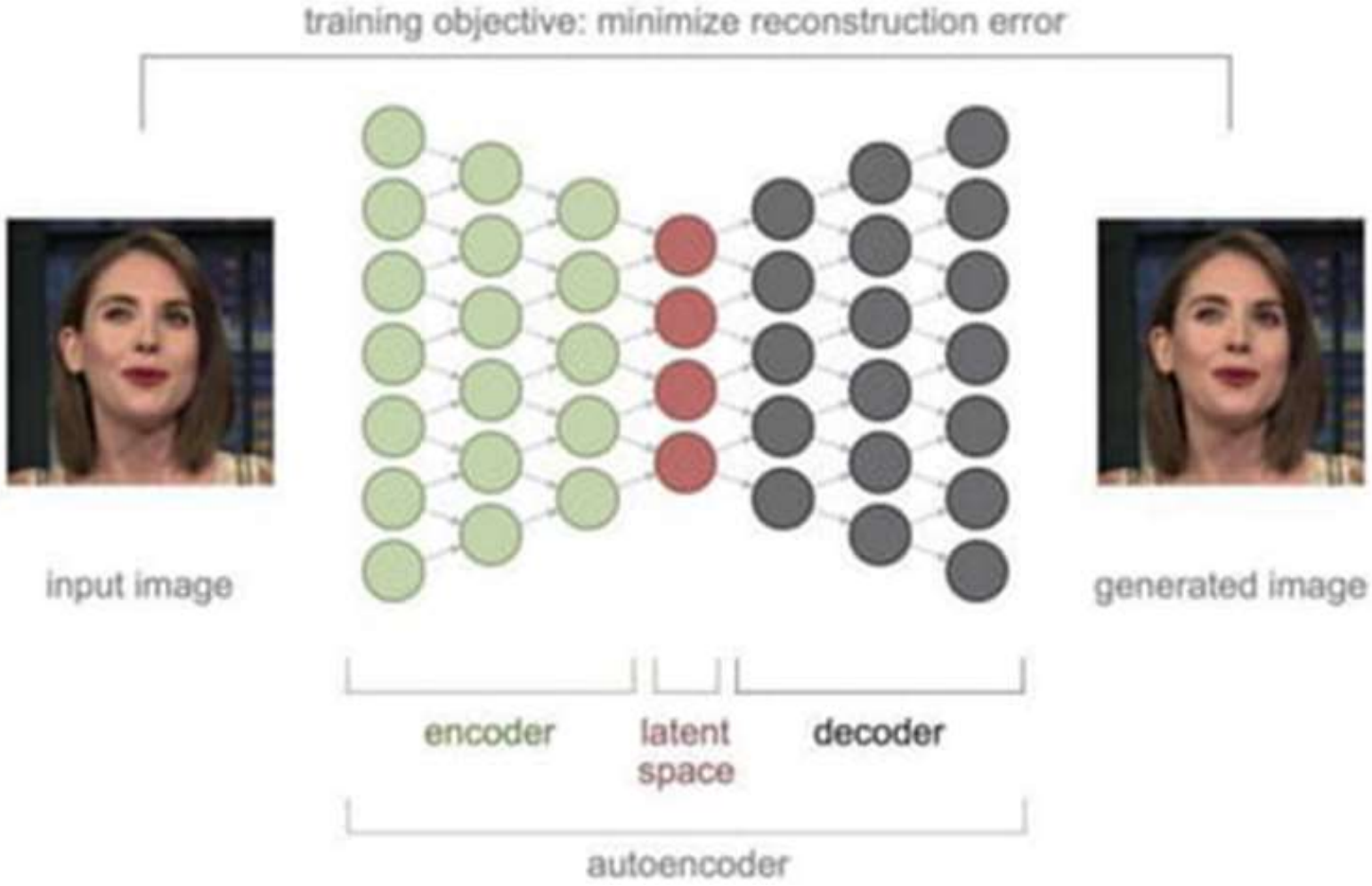
```
!git clone https://github.com/jantic/DeOldify.git DeOldify
cd DeOldify

from deoldify import device
from deoldify.device_id import DeviceId
#choices:  CPU, GPU0...GPU7
device.set(device=DeviceId.GPU0)

import torch

if not torch.cuda.is_available():
  print('GPU not available.')

!pip install -r requirements-colab.txt

import fastai
from deoldify.visualize import *

import warnings
warnings.filterwarnings("ignore", category=UserWarning, message=".*?Your .*? set is empty.*?")


!mkdir 'models'
!wget https://data.deepai.org/deoldify/ColorizeArtistic_gen.pth -O ./models/ColorizeArtistic_gen.pth

colorizer = get_image_colorizer(artistic=True)

source_url = '/content/image.jpg' #@param {type:"string"}
render_factor = 35  #@param {type: "slider", min: 7, max: 40}
watermarked = False #@param {type:"boolean"}

if source_url is not None and source_url !='':
  image_path = colorizer.plot_transformed_image(source_url, render_factor=render_factor, compare=True, waterma
rked=watermarked)
  show_image_in_notebook(image_path)
else:
  print('Provide an image url and try again.')

for i in range(10,40,2):
  colorizer.plot_transformed_image('/content/299316051_189557226842208_1325272231175630894_n.jpg', rende
r_factor=i, display_render_factor=True, figsize=(8,8))
```

AUTOENCODERS

Autoencoders are a class of artificial neural networks used for unsupervised learning and dimensionality reduction. They are primarily designed for feature learning and data compression tasks. The fundamental idea behind autoencoders is to encode the input data into a lower-dimensional representation and then decode it back to the original data, with the goal of minimizing the reconstruction error. The lower-dimensional representation in the middle is often referred to as the "latent space" or "encoding."



Autoencoder: a DNN architecture commonly used for generating deepfakes.



- **Anomaly detection:** autoencoders can identify data anomalies using a loss function that penalizes model complexity. It can be helpful for anomaly detection in financial markets, where you can use it to identify unusual activity and predict market trends.

- **Data denoising image and audio:** autoencoders can help clean up noisy pictures or audio files. You can also use them to remove noise from images or audio recordings.

- **Image inpainting:** autoencoders have been used to fill in gaps in images by learning how to reconstruct missing pixels based on surrounding pixels. For example, if you're trying to restore an old photograph that's missing part of its right side, the autoencoder could learn how to fill in the missing details based on what it knows about the rest of the photo.

- **Information retrieval:** autoencoders can be used as content-based image retrieval systems that allow users to search for images based on their content.

An Autoencoder consists of three layers:

- Encoder
- Code
- Decoder

- The Encoder layer compresses the input image into a latent space representation. It encodes the input image as a compressed representation in a reduced dimension.

- The compressed image is a distorted version of the original image.

- The Code layer represents the compressed input fed to the decoder layer.

- The decoder layer decodes the encoded image back to the original dimension. The decoded image is reconstructed from latent space representation, and it is reconstructed from the latent space representation and is a lossy reconstruction of the original image.

Training Autoencoders

- When you're building an autoencoder, there are a few things to keep in mind.

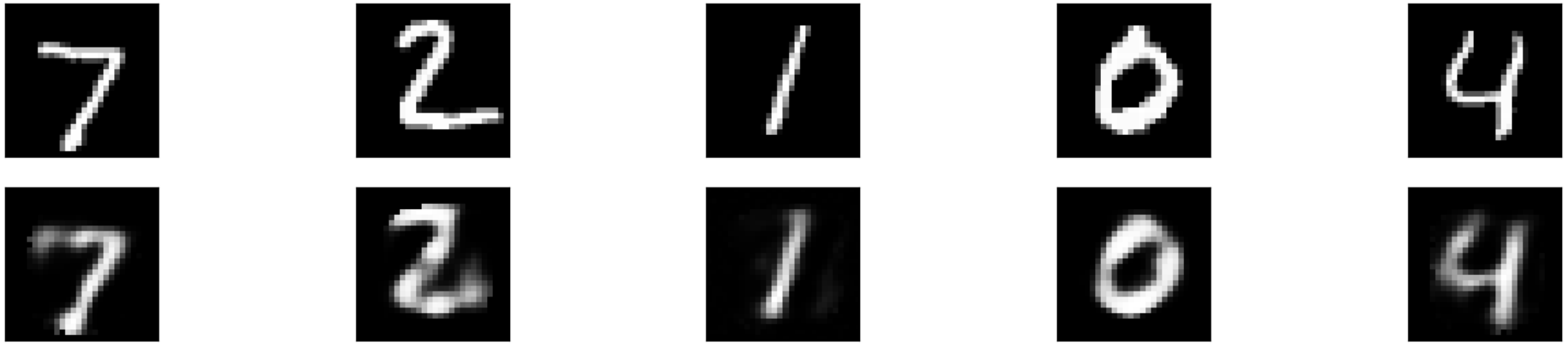
- First, the code or bottleneck size is the most critical hyperparameter to tune the autoencoder. It decides how much data has to be compressed. It can also act as a regularisation term.

- Secondly, it's important to remember that the number of layers is critical when tuning autoencoders. A higher depth increases model complexity, but a lower depth is faster to process.

- Thirdly, you should pay attention to how many nodes you use per layer. The number of nodes decreases with each subsequent layer in the autoencoder as the input to each layer becomes smaller across the layers.

- Finally, it's worth noting that there are two famous losses for reconstruction: MSE Loss and L1 Loss.

```python
import tensorflow as tf

# Define the encoder and decoder networks
class Encoder(tf.keras.Model):
    def __init__(self, latent_dim):
        super(Encoder, self).__init__()

        # Define the encoder layers
        self.dense1 = tf.keras.layers.Dense(128, activation='relu')
        self.dense2 = tf.keras.layers.Dense(64, activation='relu')
        self.latent_layer = tf.keras.layers.Dense(latent_dim, activation='relu')

    def call(self, inputs):
        # Encode the input
        x = self.dense1(inputs)
        x = self.dense2(x)
        latent_code = self.latent_layer(x)

        return latent_code

class Decoder(tf.keras.Model):
    def __init__(self, latent_dim):
        super(Decoder, self).__init__()

        # Define the decoder layers
        self.dense1 = tf.keras.layers.Dense(64, activation='relu')
        self.dense2 = tf.keras.layers.Dense(128, activation='relu')
        self.output_layer = tf.keras.layers.Dense(784, activation='sigmoid')

    def call(self, inputs):
        # Decode the input
        x = self.dense1(inputs)
        x = self.dense2(x)
        reconstructed_output = self.output_layer(x)

        return reconstructed_output

# Create the autoencoder model
encoder = Encoder(latent_dim=32)
decoder = Decoder(latent_dim=32)

autoencoder = tf.keras.Sequential([
    encoder,
    decoder
])
```



What is a Deepfake?

Deep Fakes are videos or images that replace someone's likeness with someone else.

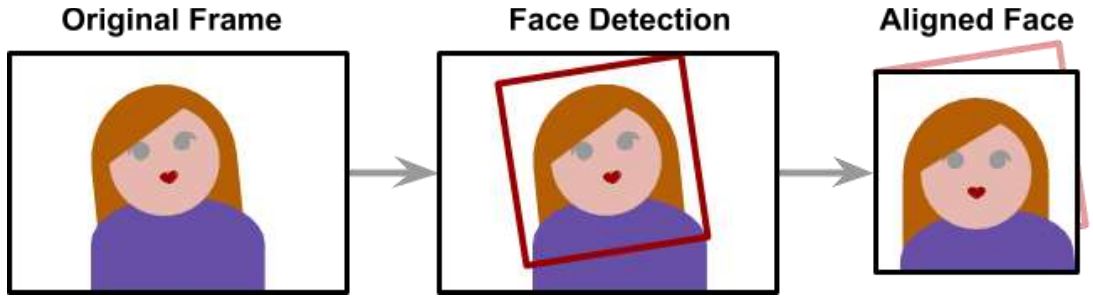
It is the next level of fake content creation that takes advantage of artificial intelligence (AI).

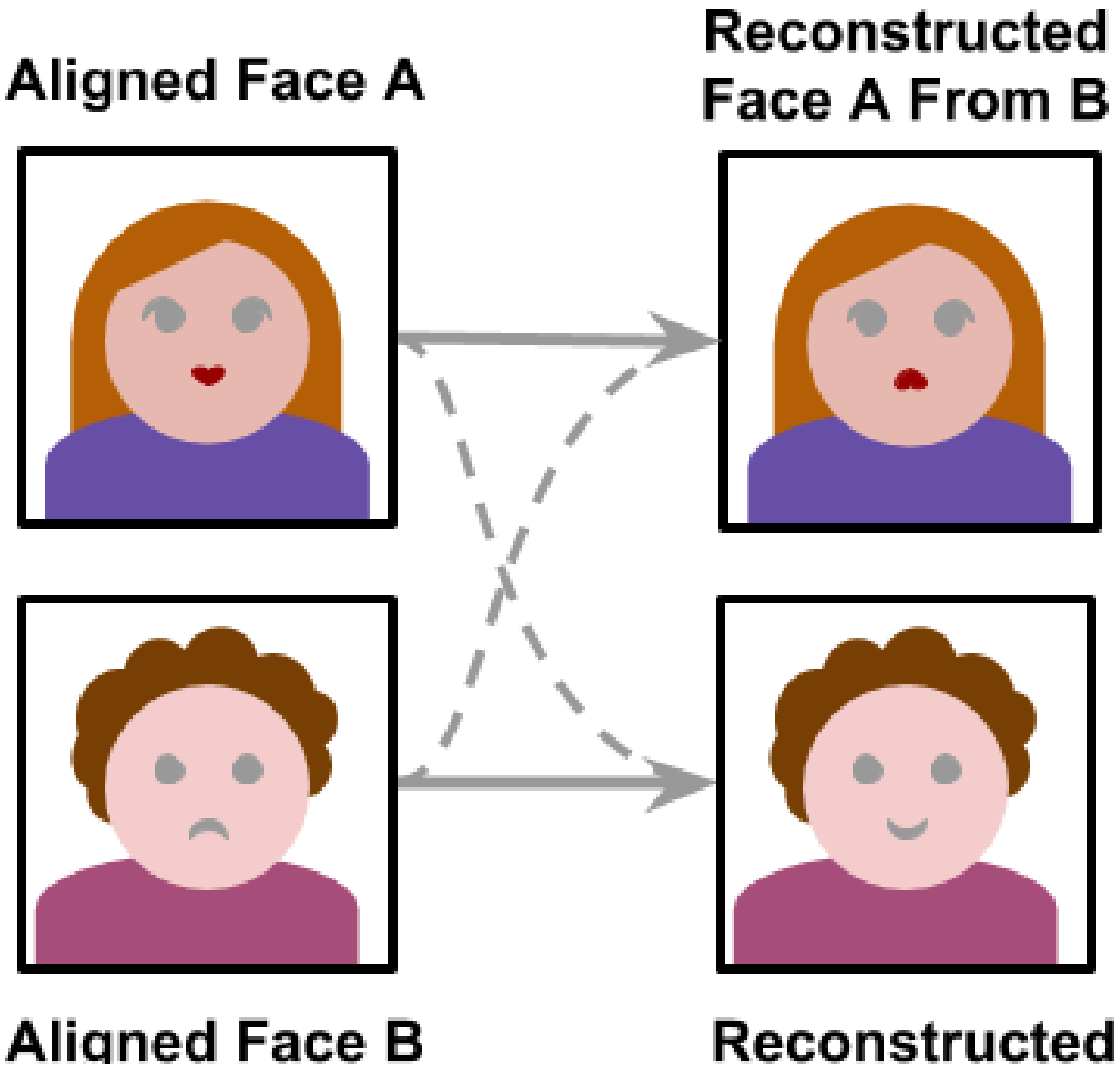
The phenomenon gained its name from a user of the Reddit platform, who went by the name "deepfakes" – (deep learning + fakes).

Extraction

Deepfakes leverage deep neural networks to transform faces and require large amounts of data (images) to make it all work smoothly and believable. The extraction process refers to the step of extracting all frames from video clips, identifying the faces and aligning them to optimize for performance.

The alignment is a critical step since the neural network that performs the faceswap requires all faces to have the same size (usually 256 x 256) and aligned features. Detecting and aligning faces is a problem that is considered mostly solved and is done by most applications very efficiently (face detection).



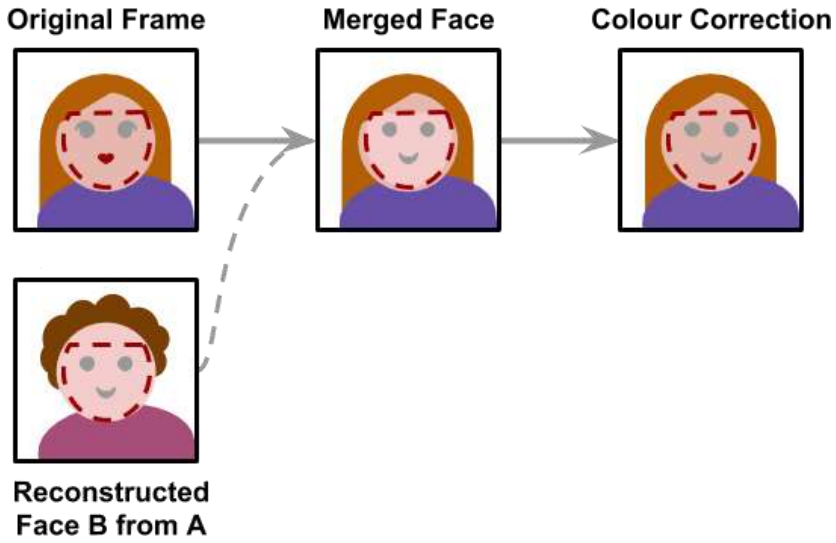


Training

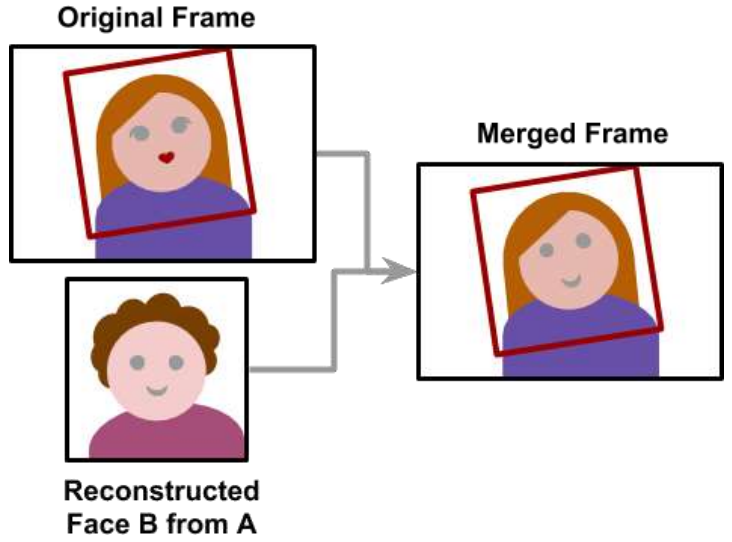
The training stage allows the neural network to convert a face into another. The training may take several hours or even days depending on the size of the training set and the device the model is trained on. Like training most other neural networks the training only needs to be completed once. Once the model is trained it would be able to convert a face from person A to person B.

Creation

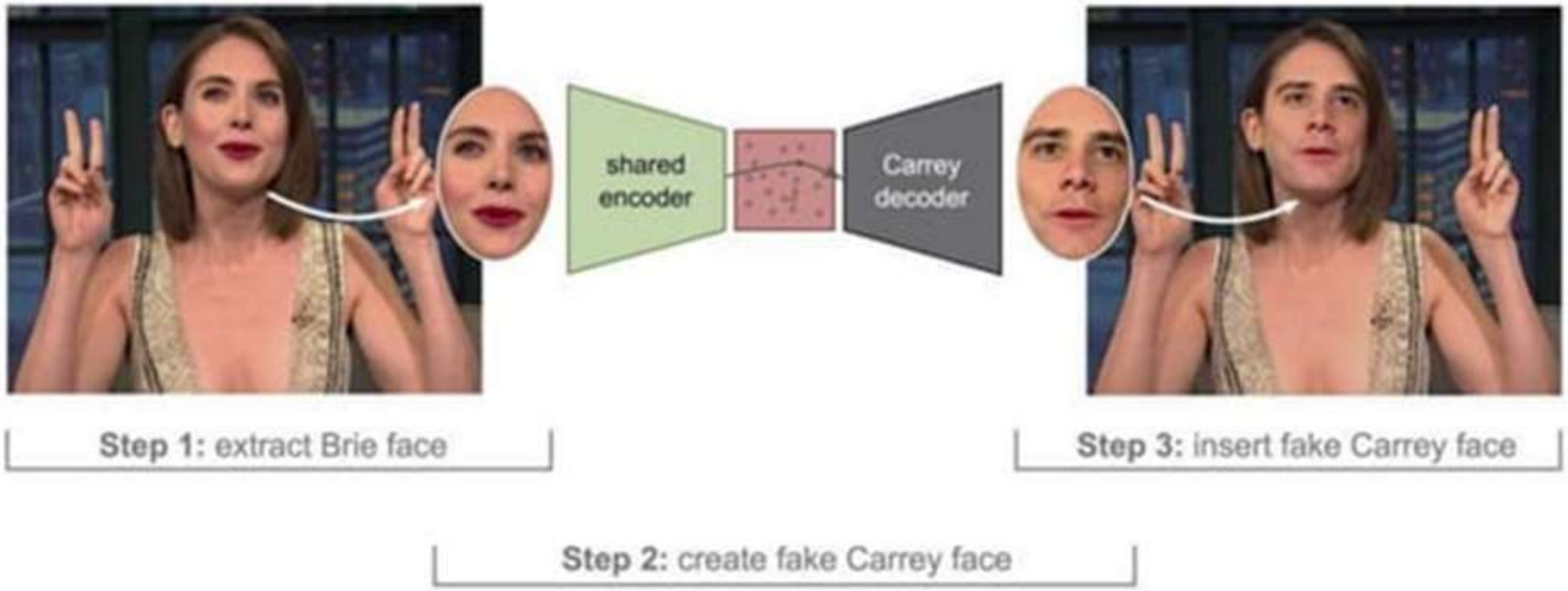
Once the model is trained, a deepfake may be created. Starting from a video, frames are extracted, and all faces are aligned. Each frame is then converted using the trained neural network. The final step is to merge the converted face back to the original frame.



The creation stage is the only one which does not make use of any machine learning algorithms. The process is to stich a face back onto an image is hard-coded and thus lacks the ability to detect mistakes.



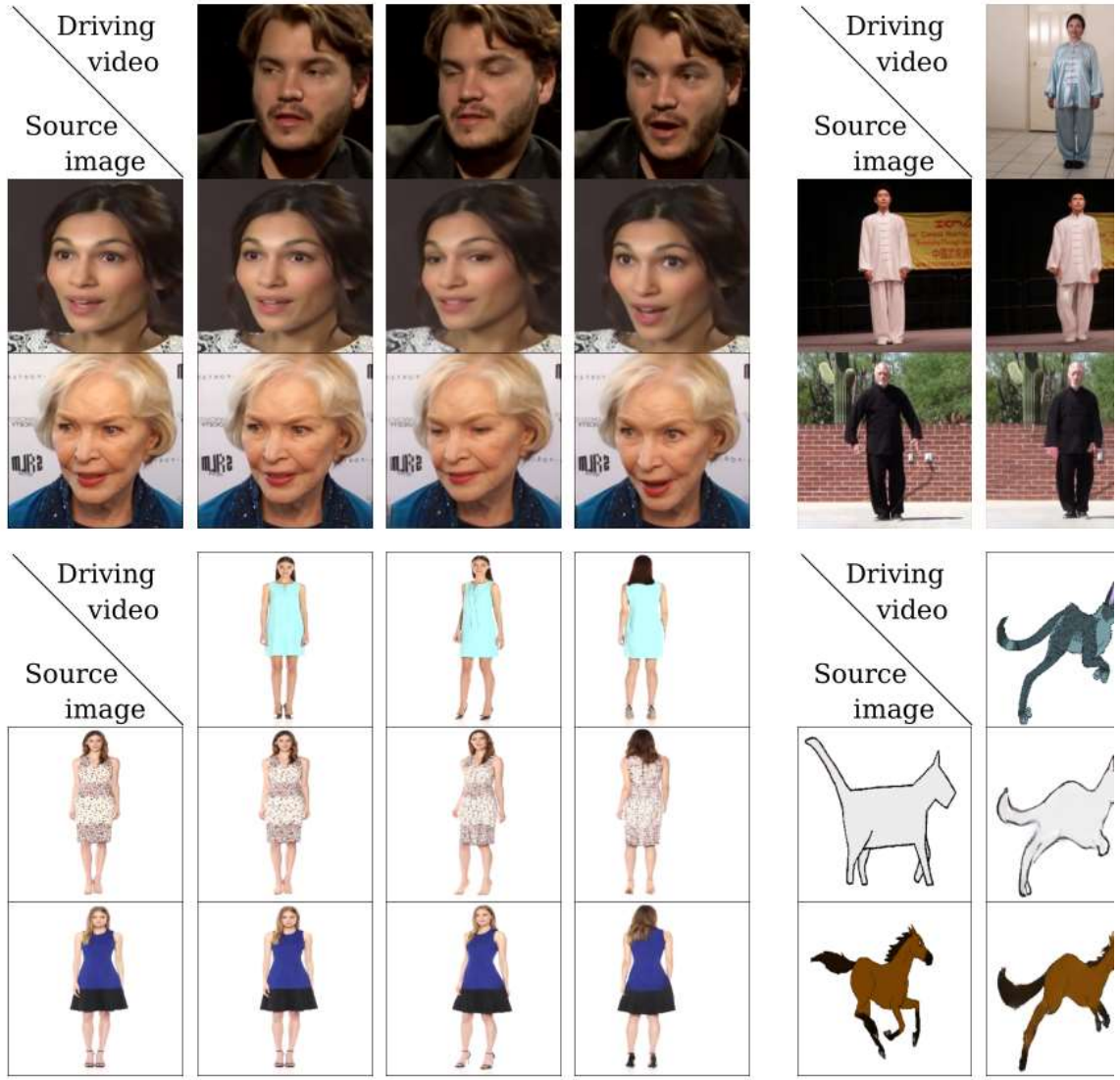
Each frame is also processed independitly which means there is no temporal correlation. A temporal correlation function is a function that gives the statistical correlation between random variable, contingent on the temporal (time) distance between those variables. The lack of this correlation results in the final video having some flickering.

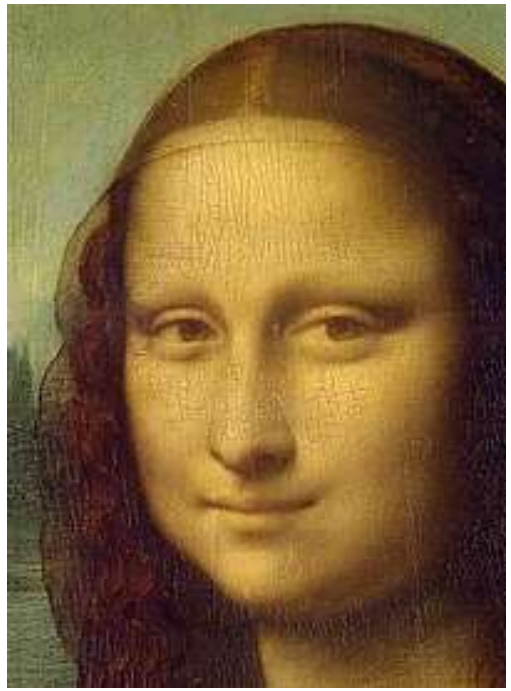


HOW DO DEEPFAKES WORK?

Dataset description

The model is trained over VoxCeleb dataset which is a face dataset. The 22496 videos in the VoxCeleb dataset are face-extracting videos from YouTube. In the first video frame, an initial bounding box has been extracted for pre-processing. This face is followed until it strays too far from its starting location. Then, the video frames are being cropped using the smallest crop that includes all the bounding boxes. Up until the end of the series, the process is repeated. Sequences with resolutions less than 256x256 are removed, and the remaining videos are scaled to 256x256 while maintaining the aspect ratio. It's noteworthy to notice that the employed Aliaksandr's first order model produces more realistic movies with faces moving freely within the bounding box when compared to X2Face.



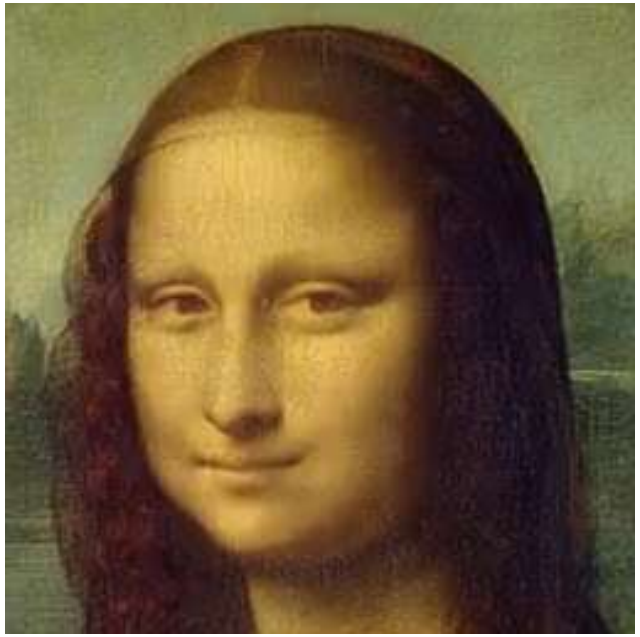


TARGET IMAGES

DRIVING VIDEO

OUTPUTS

Original



DeepFake

Original

DeepFake

Generative Adversarial Networks (GANs)

- Pros:

1. **High-Quality Generation:** GANs are known for their ability to generate high-quality and realistic data, such as images, text, and audio.
2. **Unsupervised Learning:** GANs can learn patterns and features in data without the need for explicit labels, making them suitable for unsupervised learning tasks.
3. **Variety in Output:** GANs can produce diverse outputs by sampling from the generator's latent space, which is useful for generating different variations of data.
4. **State-of-the-Art in Image Generation:** GANs have achieved remarkable success in generating realistic images and are widely used in computer vision applications.

- Cons:

1. **Training Instability:** GANs can be challenging to train and prone to instability, such as mode collapse, where the generator produces limited diversity in its output.
2. **Mode Collapse:** GANs can sometimes generate similar or repetitive samples, ignoring certain modes in the data distribution.
3. **Hyperparameter Sensitivity:** GANs require careful tuning of hyperparameters, and small changes can have a significant impact on training stability and results.
4. **Noisy Training:** Training GANs can be noisy and time-consuming, making it difficult to estimate convergence accurately.

Autoencoder

- Pros:

1. **Dimensionality Reduction:** Autoencoders are used for dimensionality reduction and feature learning, making them suitable for data compression and denoising.
2. **Anomaly Detection:** Autoencoders can be used for anomaly detection since they learn to reconstruct normal data and perform poorly on anomalies.
3. **Interpretable Latent Space:** The latent space of an autoencoder can be more interpretable, as each dimension can represent a learned feature or attribute.
4. **Stable Training:** Autoencoders are typically easier to train than GANs, and they converge more reliably.

- Cons:

1. **Limited for Data Generation:** Autoencoders are not primarily designed for data generation and may produce less realistic outputs compared to GANs.
2. **Lossy Compression:** When used for compression, autoencoders result in lossy compression, meaning some information is lost during the encoding-decoding process.
3. **Difficulty Capturing Complex Data Distributions:** Autoencoders may struggle to capture complex data distributions and generate high-quality data in some cases.
4. **Fixed Encoding-Decoder Architecture:** The architecture of an autoencoder is fixed once trained and may not adapt well to variations in data.

THANK YOU