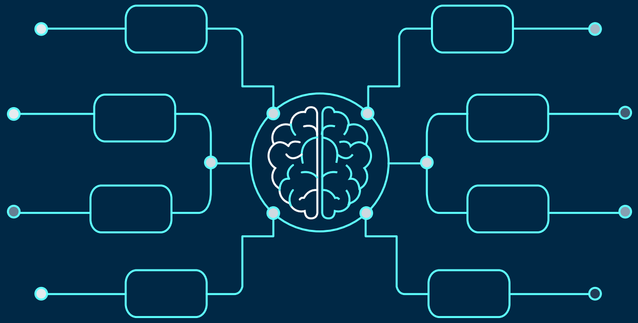
Podo Luca □ Aragona Dario

DEEP LEARNING PROJECT



DEEPFAKES

GOAL

Our goal is to investigate the most important deepfake **creation** and **detection** methods.

Starting from real videos, we **create** their deepfake version and then we try to build a model to **detect** real and fake ones.

In particular, we focus on **faceswap** algorithms, an important and discussed topic in recent years.

01:
DEEPFAKE
CREATION

02:
DEEPFAKE
DETECTION

01

DEEPFAKE CREATION

Index

1. **What is a deepfake?**

2. **Faceswap software**

   a. Autoencoders
   b. Reconstruction train
   c. Face transformation
   d. Swapping

3. **Faceswap GAN software**

   a. Definition
   b. Architecture
   c. SAGAN
   d. Loss functions
   e. Reconstruction train
   f. Face transformation
   g. Swapping

4. **Dataset generation**

5. **Results**

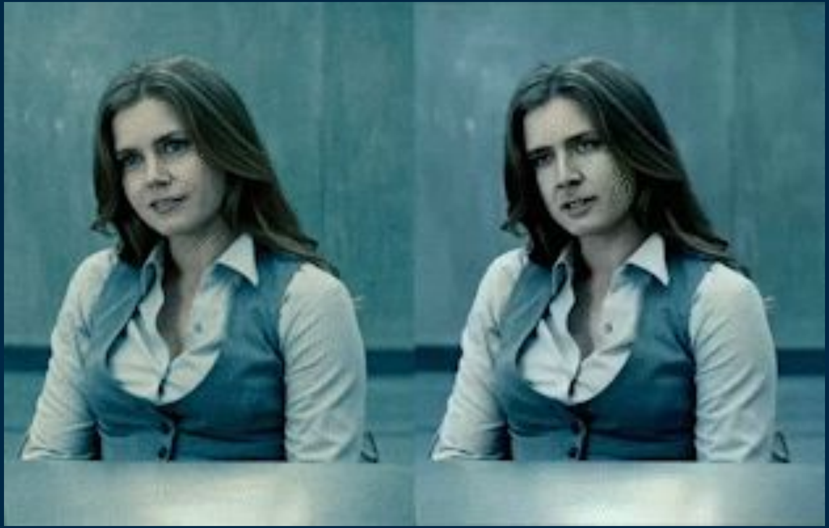
6. **Project Limitations**

1. What is a deepfake?

In general, **facial manipulation** by means of Deep Learning can be categorized in the following categories:

- Face synthesis
- Facial attributes and expression
- **Faceswap**

Deepfakes are synthetic media in which a person in an existing image or video is **replaced** with someone else's likeness



But, how are deepfakes created?

Man of Steel. Modification done by Reddit user "derpfakes"

2. Faceswap software

Faceswap is one of the leading free and Open Source multi-platform Deepfakes software.

It is powered by Tensorflow, Keras and Python.

It puts together lot of disciplines like Deep Learning, computer vision and image processing.

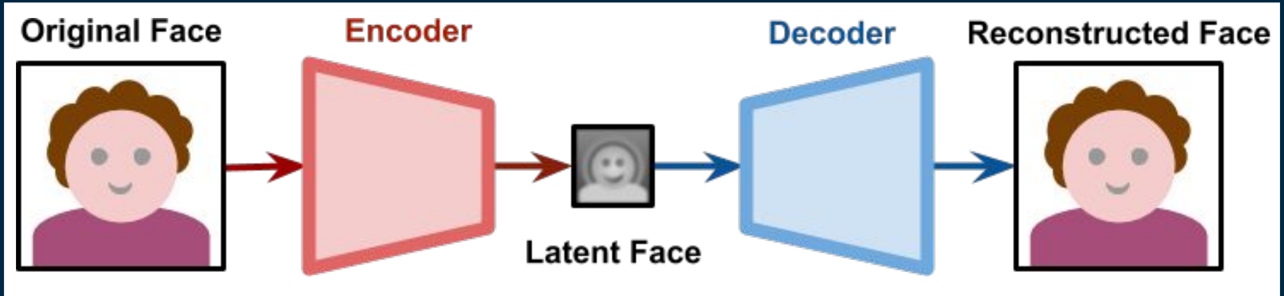
How does it work?

Autoencoders!

https://github.com/deepfakes/faceswap/
https://faceswap.dev/

2. Faceswap // Autoencoders

Everything is built over the concept of **autoencoder**



Encoder: given an input data, its result is a **lower dimensional representation** of that input in a **latent space**.

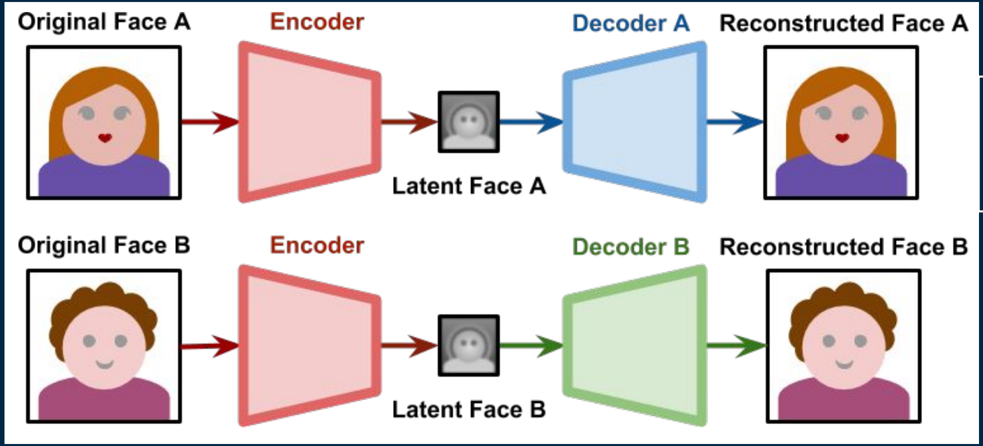
Decoder: given a latent representation, the original input data is **reconstructed**.

Autoencoders are **lossy**: reconstructed input is unlikely to have the same level of **detail** that was originally present.

An Introduction to DeepFakes

2. Faceswap // Reconstruction train

The faceswap approach is based on training **two autoencoders** that:

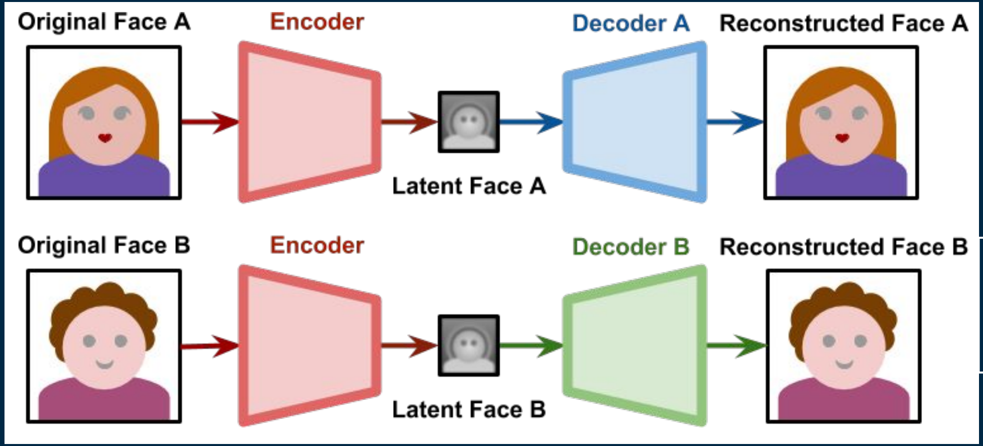


- **share a single encoder** (to create a common, meaningful latent space)
- **have different decoders** (one for each face)

This means that the **encoder** itself has to identify **common features** in both faces: because all faces share a similar structure, the encoder learns the concept of "**face**" itself.

2. Faceswap // Reconstruction train

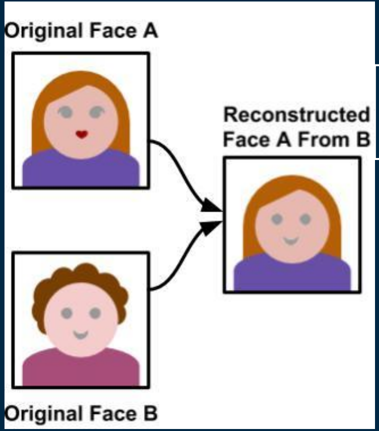
The faceswap approach is based on training **two autoencoders** that:



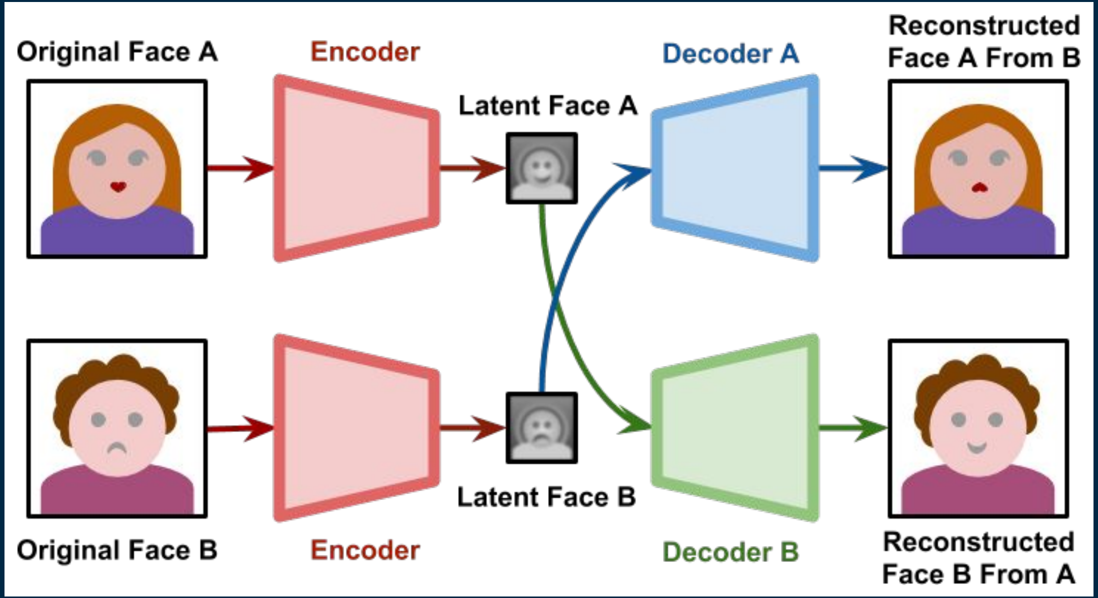
- **share a single encoder** (to create a common, meaningful latent space)
- **have different decoders** (one for each face)

Intuitively, the **decoder** detects face angle, skin tone, **facial expression**, lighting and other **context information** that is important to reconstruct the face on which it have trained on.

2. Faceswap // Face transformation



For example, let's try to put subject B's face on subject A.

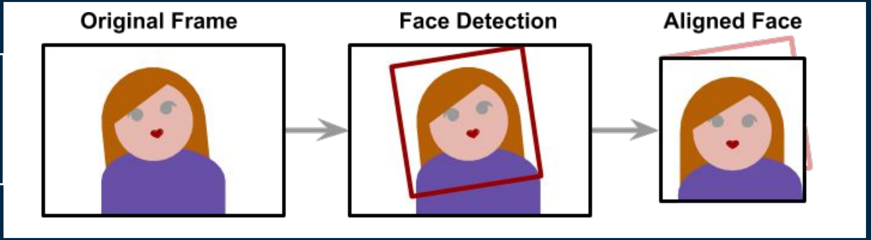


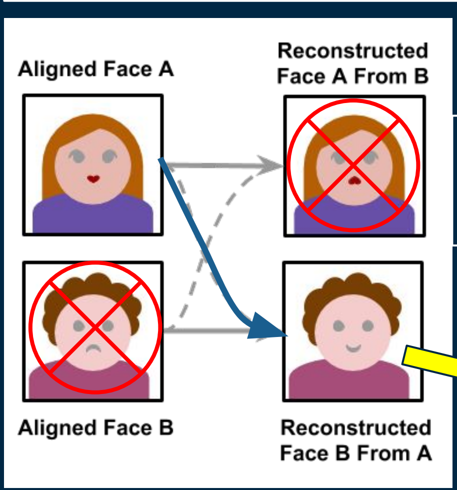
We can pass a **latent face** generated from Subject **A** to the Decoder **B**

Decoder **B** will try to reconstruct Subject **B**, from the **facial information** (for example, expression) relative to Subject **A**.

2. Faceswap // Swapping

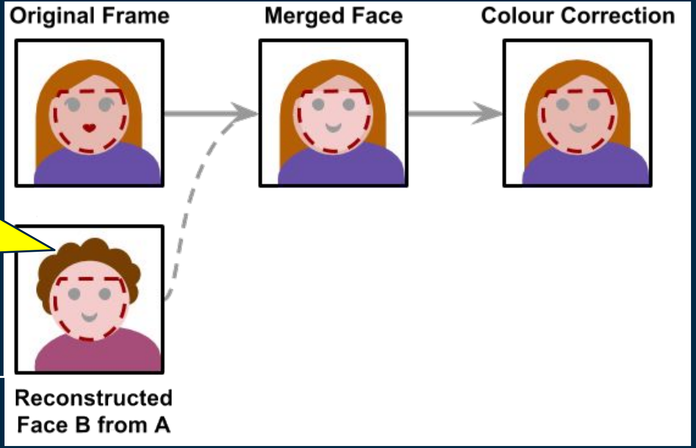
1. **Extraction**: A face detector is used to crop and to align the face A





2. **Transformation**: the trained Encoder and **Decoder B** are applied to the face A (to create face B with same expression of A)

3. **Swapping**: Transformed face B is blended with face A using **masks**



3. Faceswap GAN software

Open source **GAN**-based approach, developed from the original autoencoder-based deepfake faceswap.

New features are the addition of adversarial loss and perceptual loss (VGGface pretrained model).

We decided to use this kind of approach to build our deepfake dataset

https://github.com/shaoanlu/faceswap-GAN

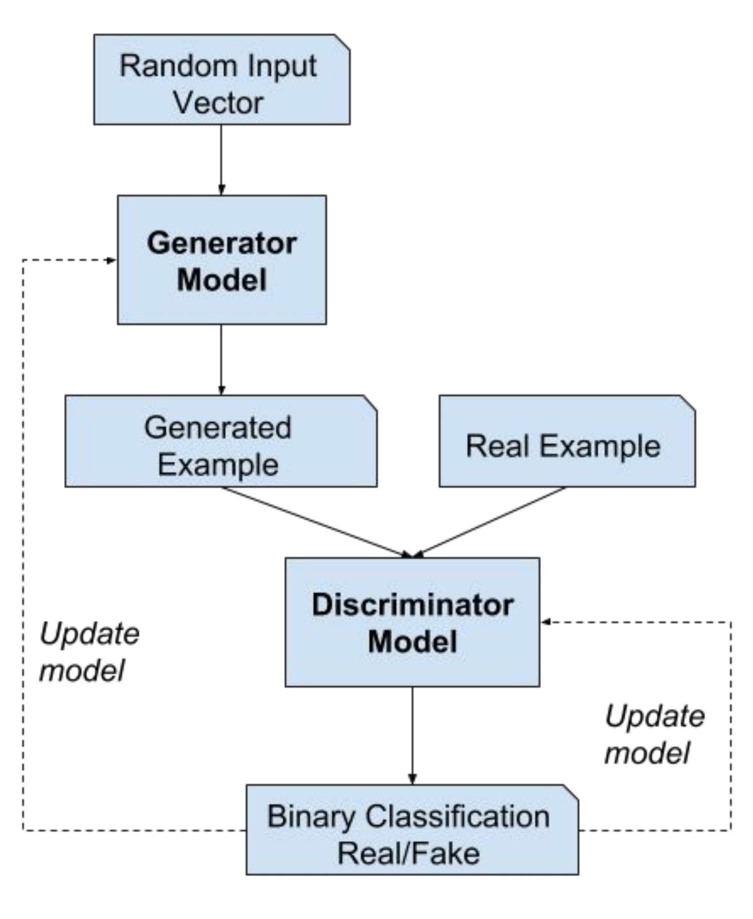
3. Faceswap GAN // Definition

What is a **Generative Adversarial Network**?

GANs are **generative** models: they create new data instances that resemble your training data.

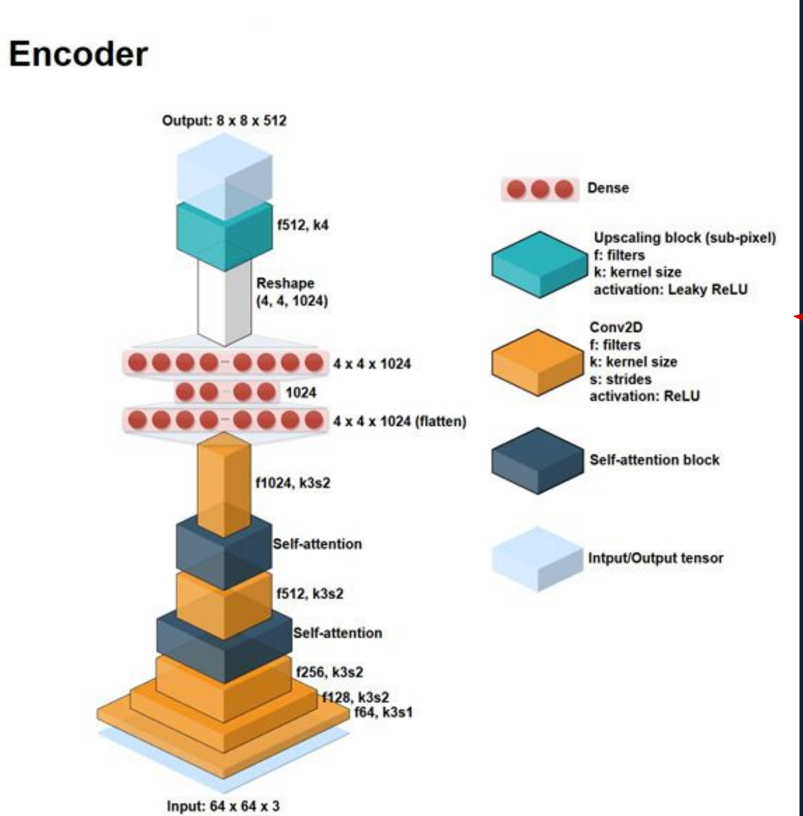
They achieve this level of realism by pairing a **generator**, which learns to produce the target output, with a **discriminator**, which learns to distinguish true data from the output of the generator.

The generator tries to **fool** the discriminator, and the discriminator tries to **keep** from being fooled.

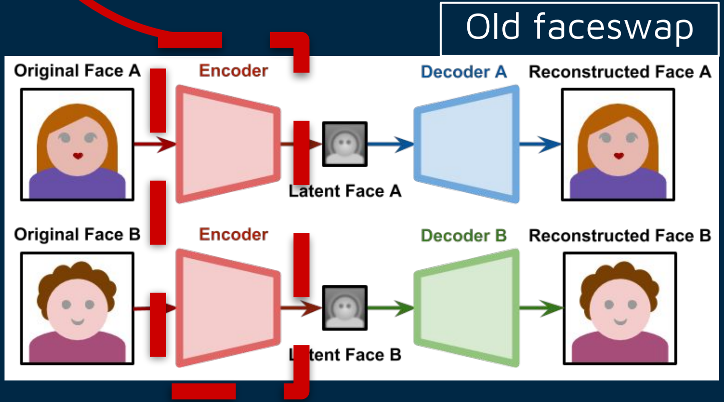


A Gentle Introduction to Generative Adversarial Networks (GANs)

3. Faceswap GAN // Architecture



One common encoder for both faces, like in faceswap

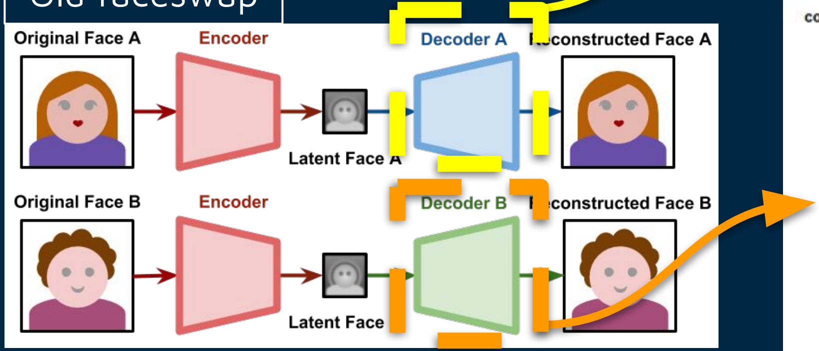


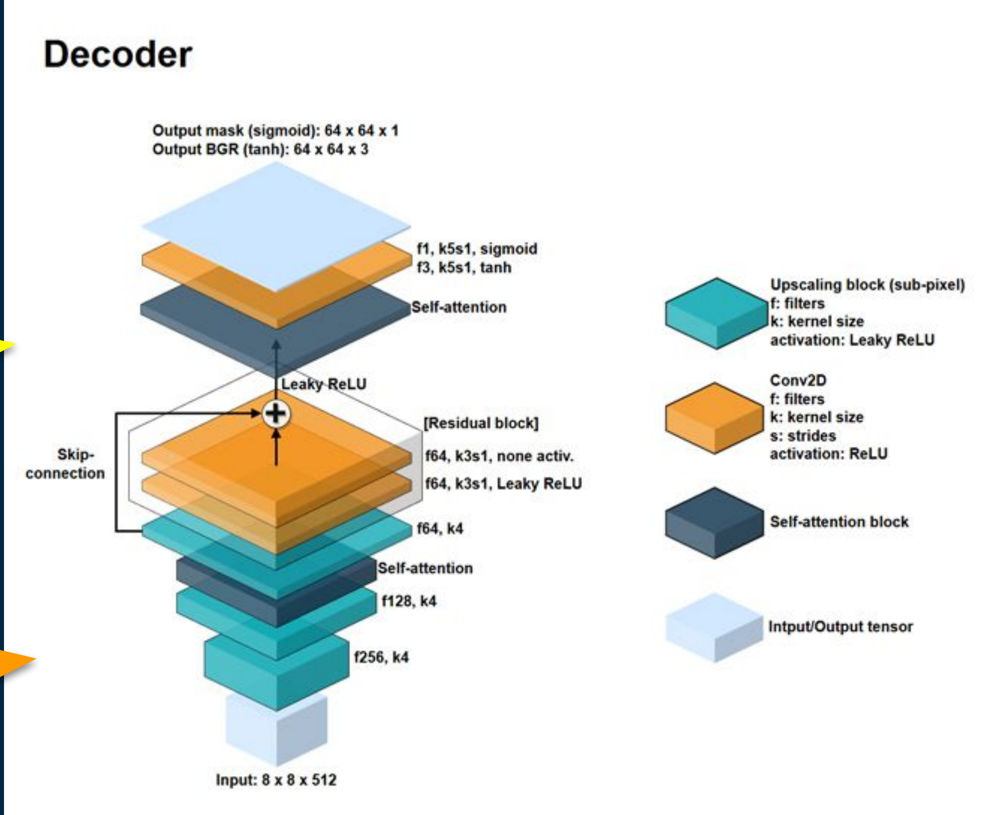
3. Faceswap GAN // Architecture

Two decoders, one for face A and one for face B, like in faceswap

Encoder and **decoder** together build the **GAN Generator**

Old faceswap



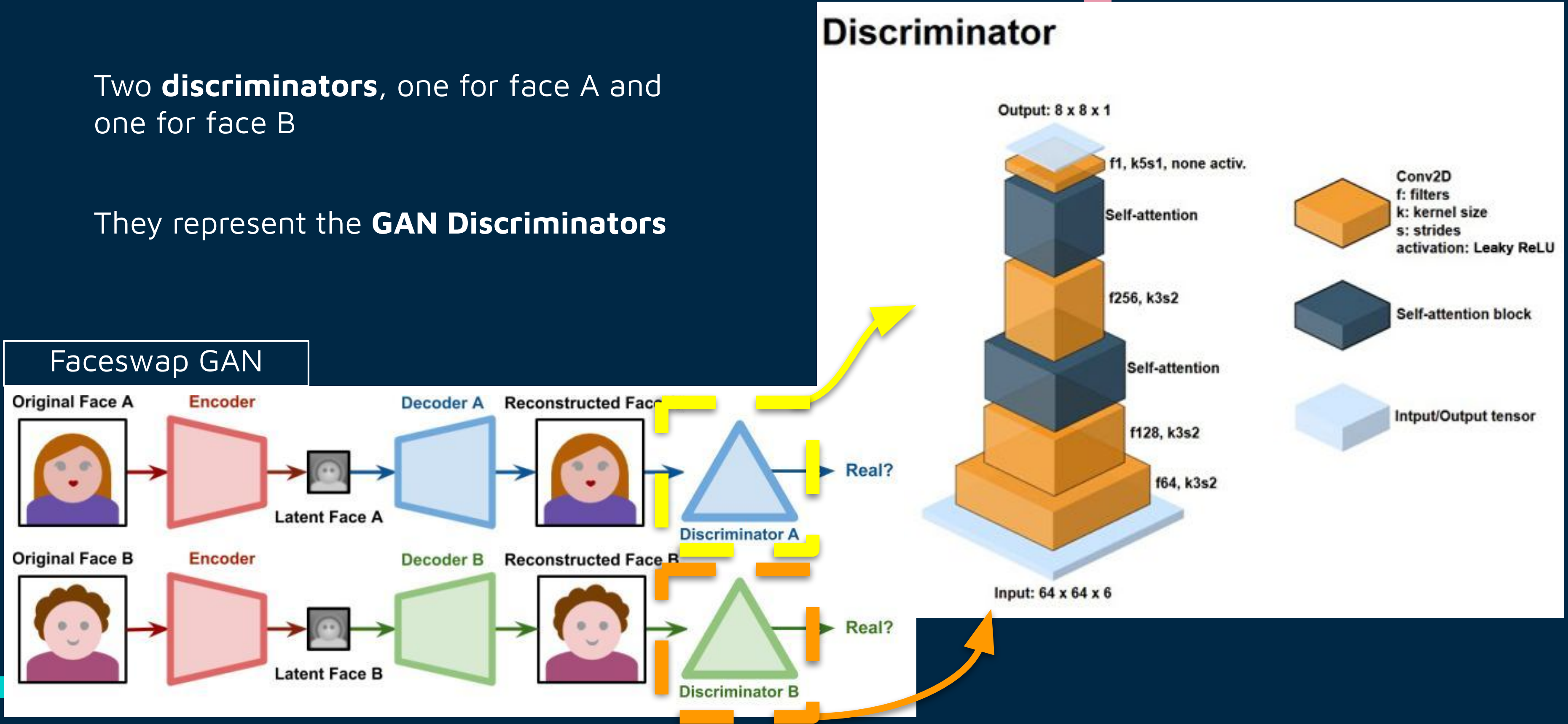


3. Faceswap GAN // Architecture

Two **discriminators**, one for face A and one for face B

They represent the **GAN Discriminators**

Faceswap GAN

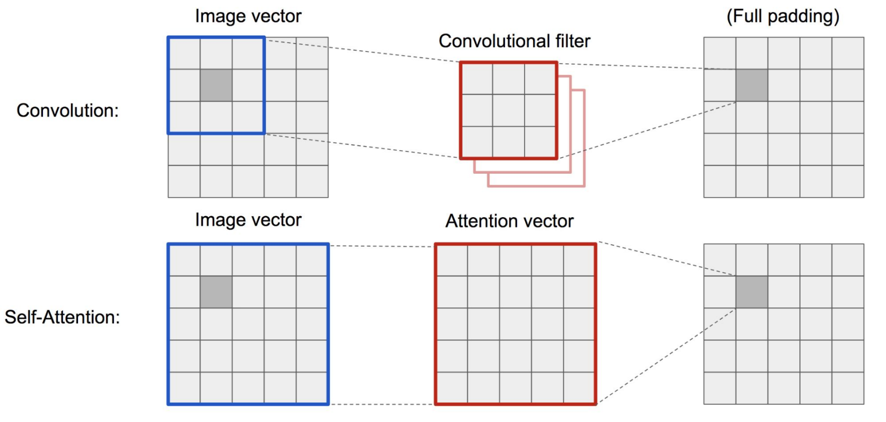


3. Faceswap GAN // SAGAN

Convolution processes the information in a **local** neighborhood, but is **inefficient** for modeling **global** long-range **dependencies** across image regions

The **self-attention** module is complementary to convolutions and helps with modeling these dependencies

Self-attention in the vision context is designed to explicitly learn the relationship between **one pixel** and all **other positions**, even regions **far apart**. It can easily capture global dependencies.



A Self-Attention GAN is a DCGAN that utilizes self-attention layers

brain-research/self-attention-gan

3. Faceswap GAN // Loss functions

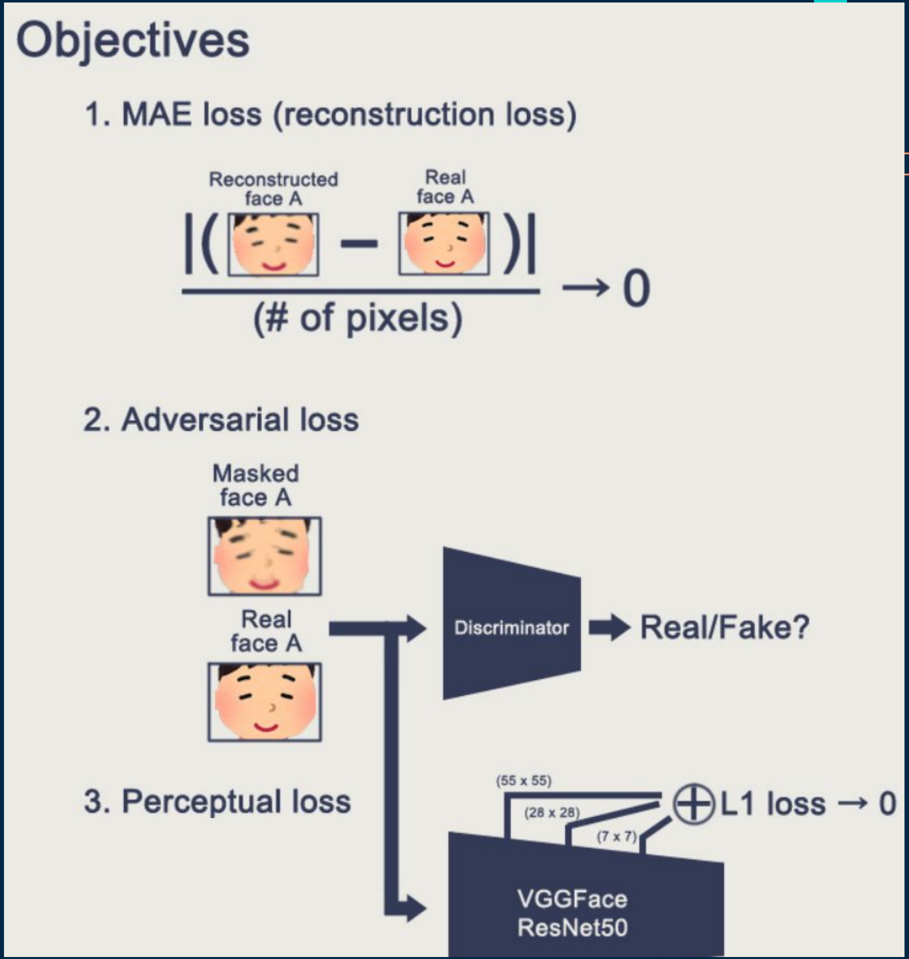
- **Reconstruction loss**: measures pixel-by-pixel differences

- **Perceptual loss**: measures differences between high-level features (extracted from pretrained model)

- **Minimax Loss**:

$$E_x[log(D(x))] + E_z[log(1 - D(G(z)))]$$

- **Edge loss:** have target image and the created image the same edge at the same location?



Objectives

1. MAE loss (reconstruction loss)

2. Adversarial loss

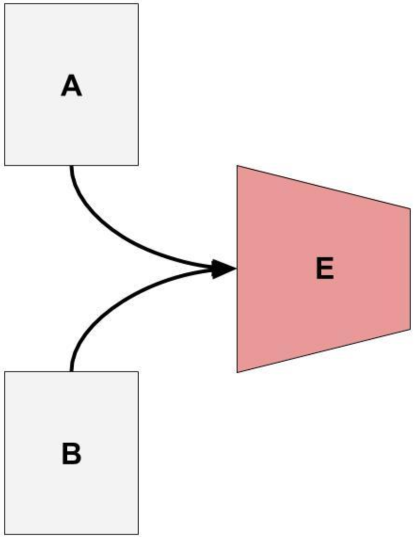
3. Perceptual loss

3. Faceswap GAN // Reconstruction train

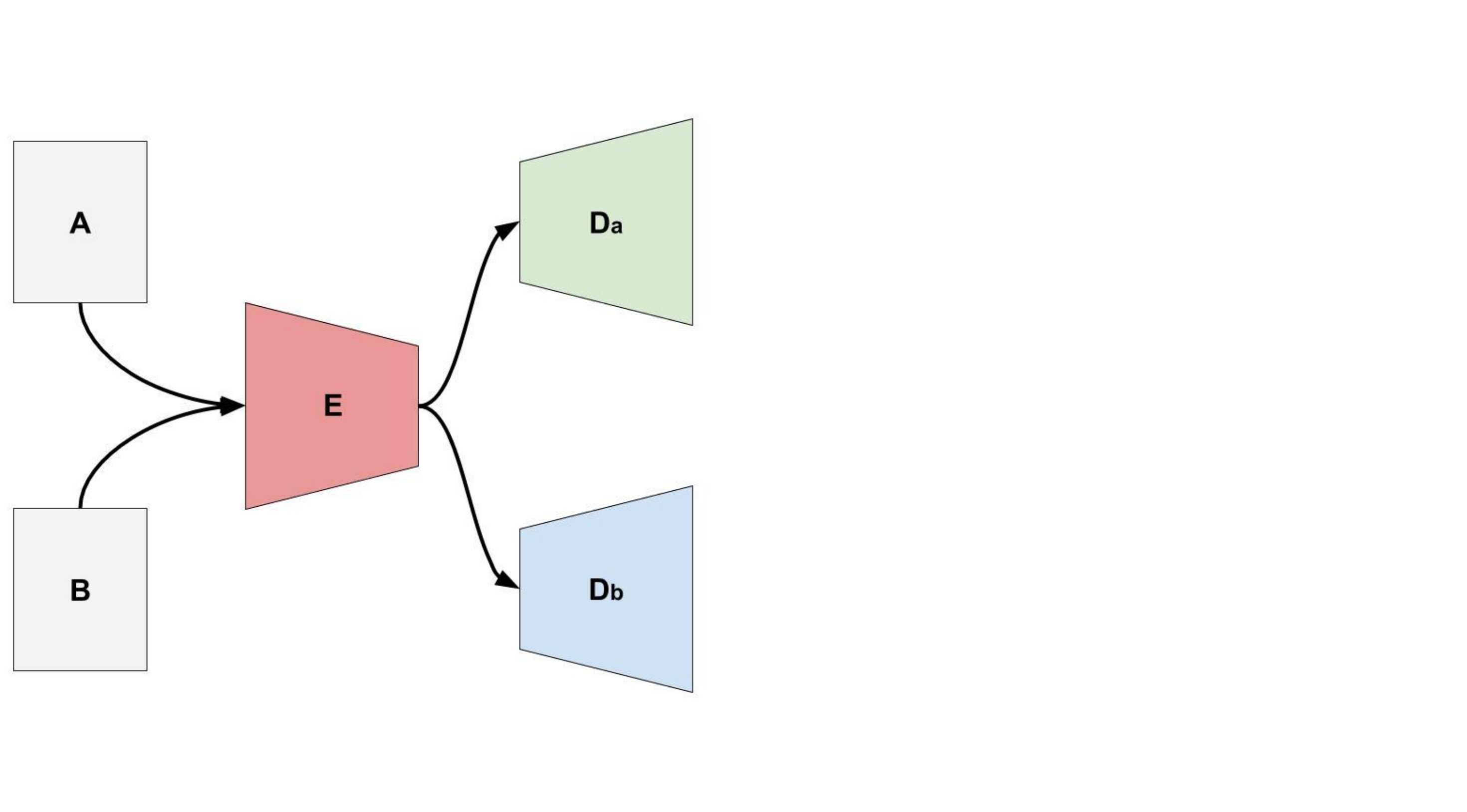
A

B

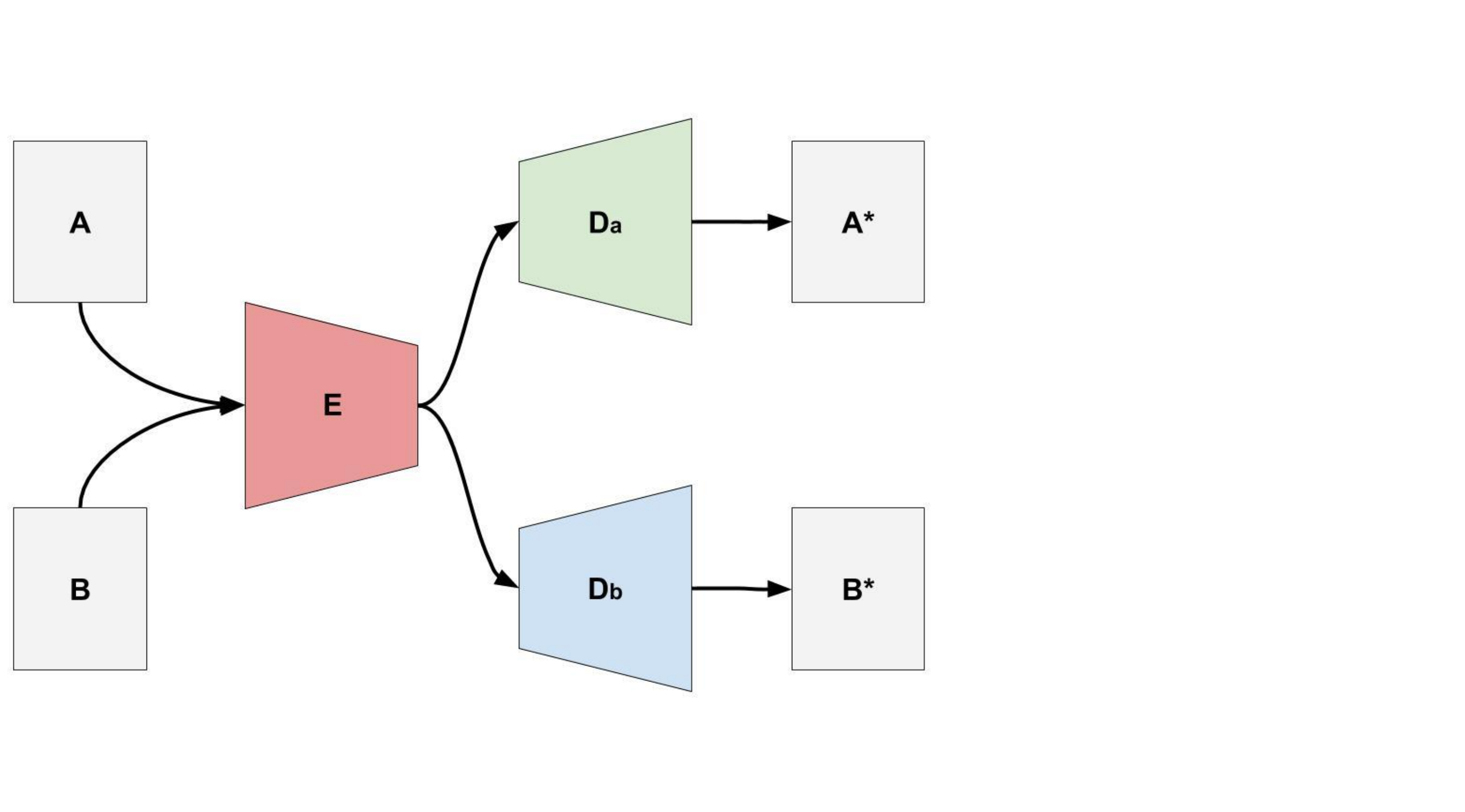
3. Faceswap GAN // Reconstruction train



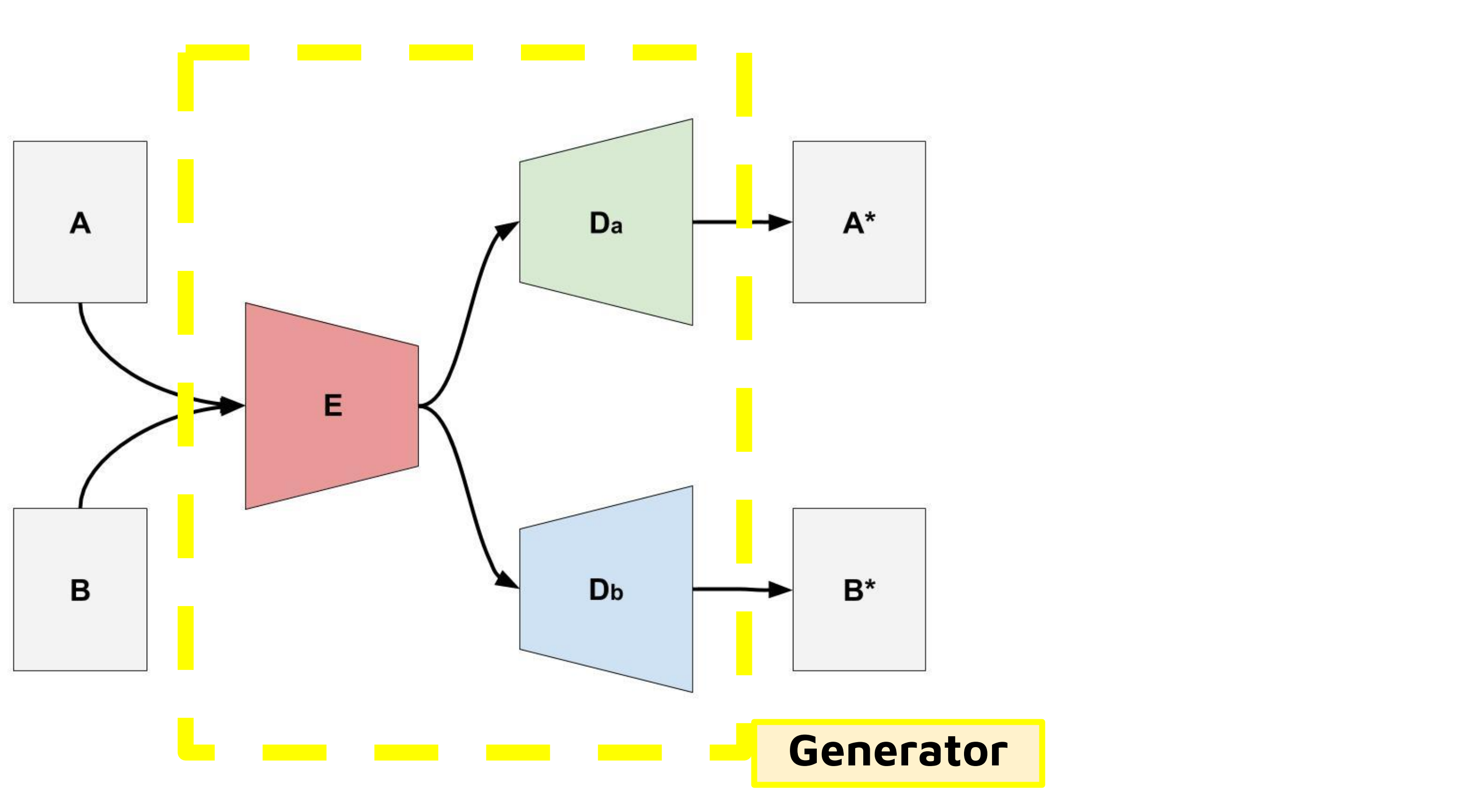
3. Faceswap GAN // Reconstruction train



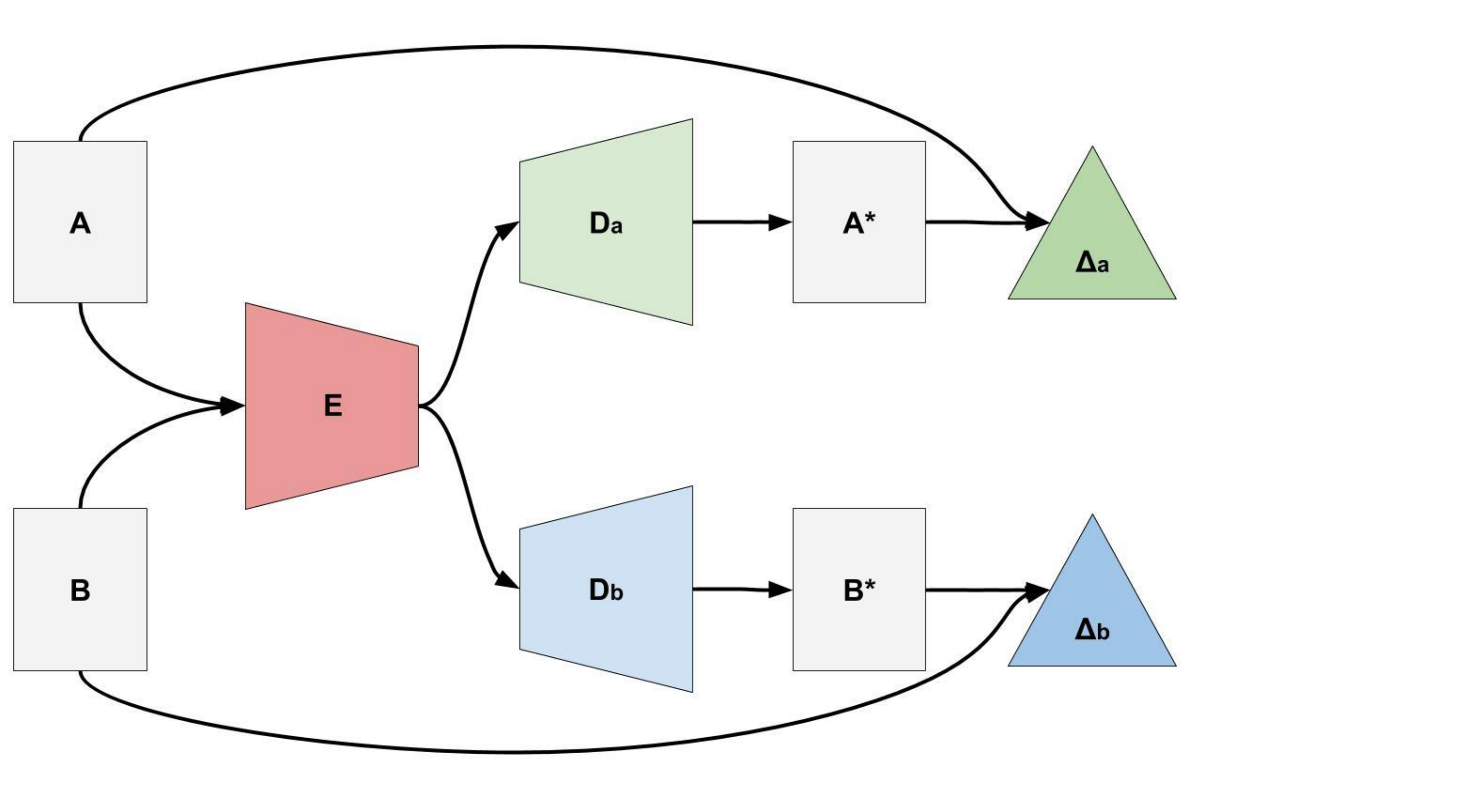
3. Faceswap GAN // Reconstruction train



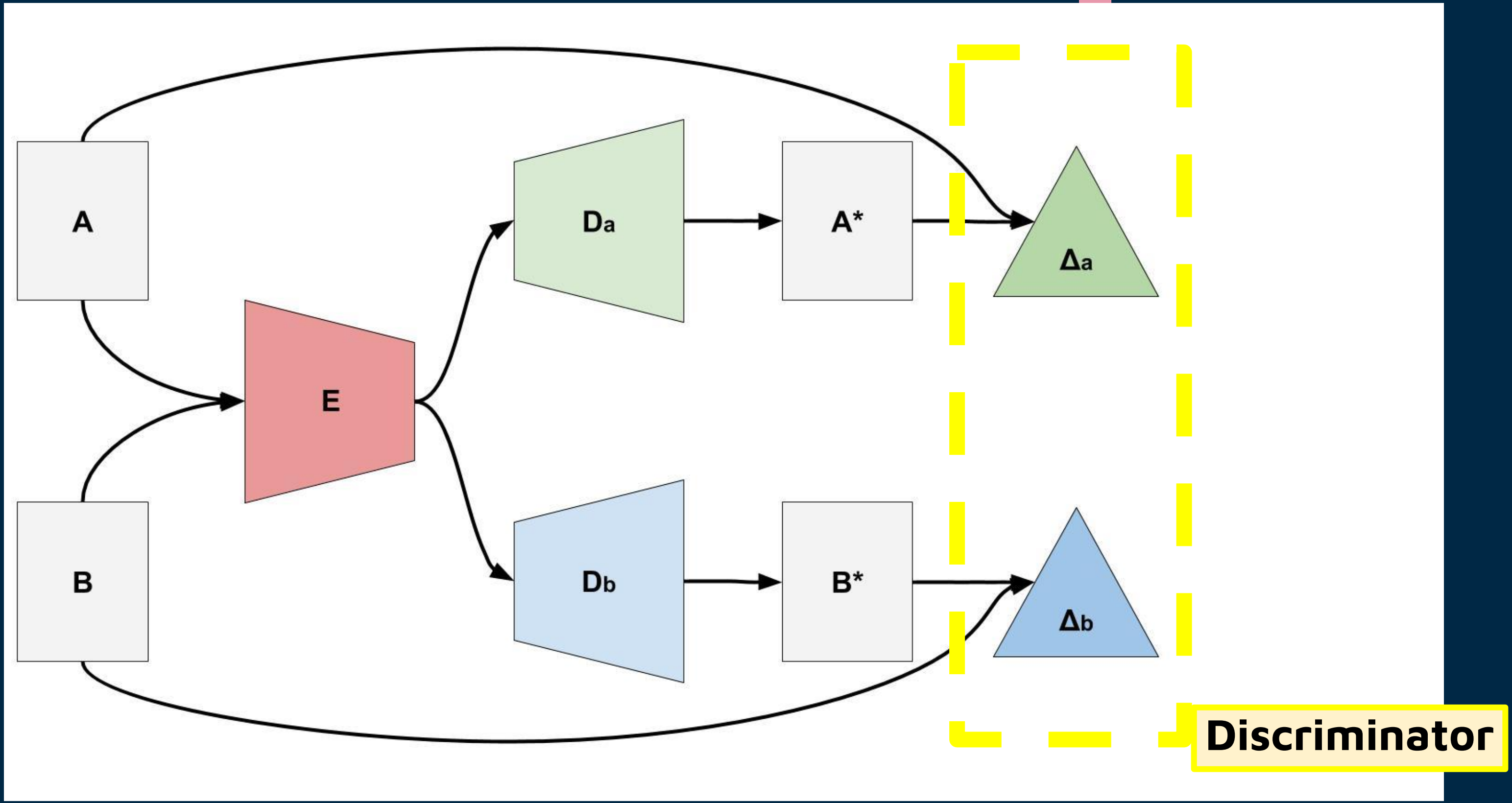
3. Faceswap GAN // Reconstruction train



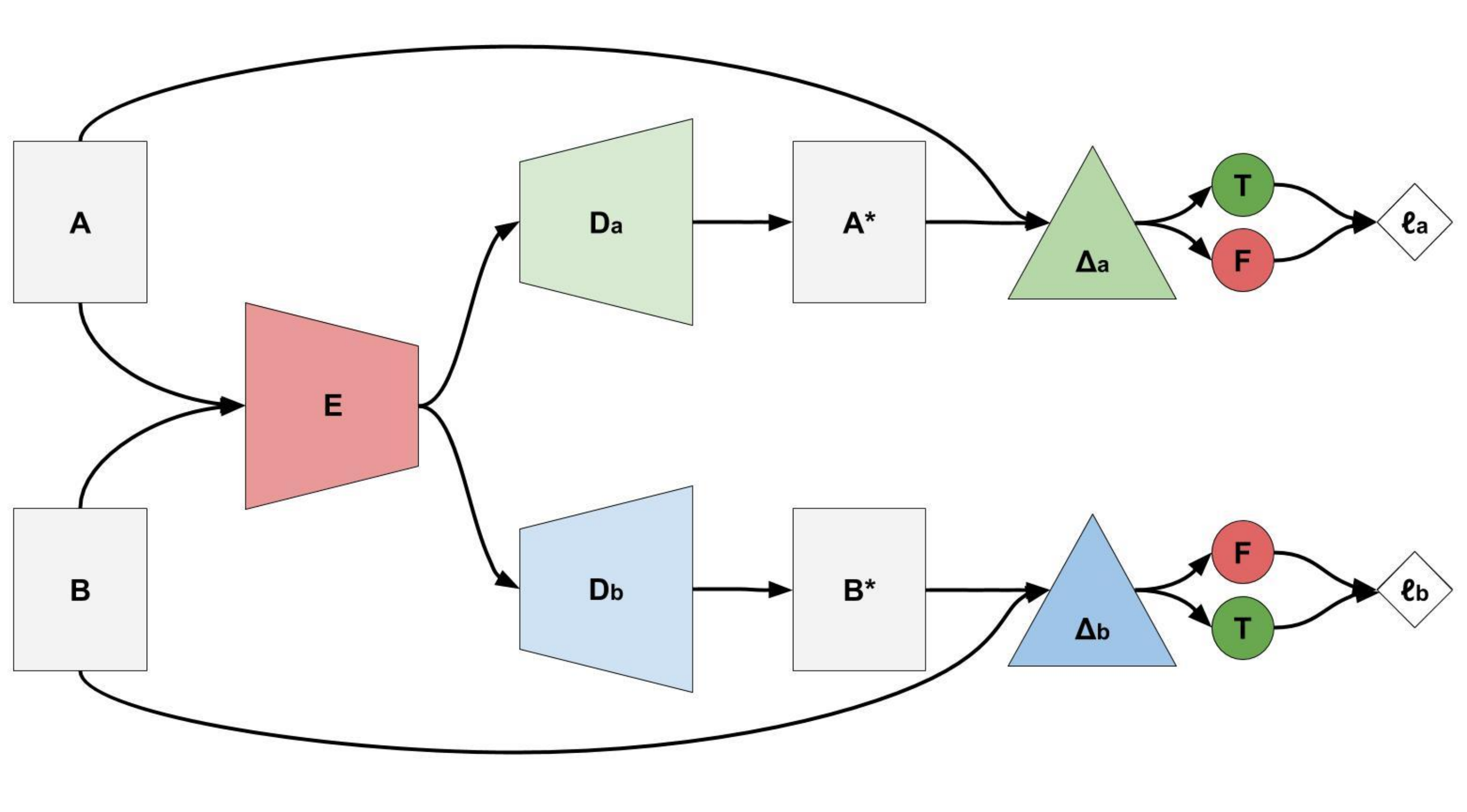
3. Faceswap GAN // Reconstruction train



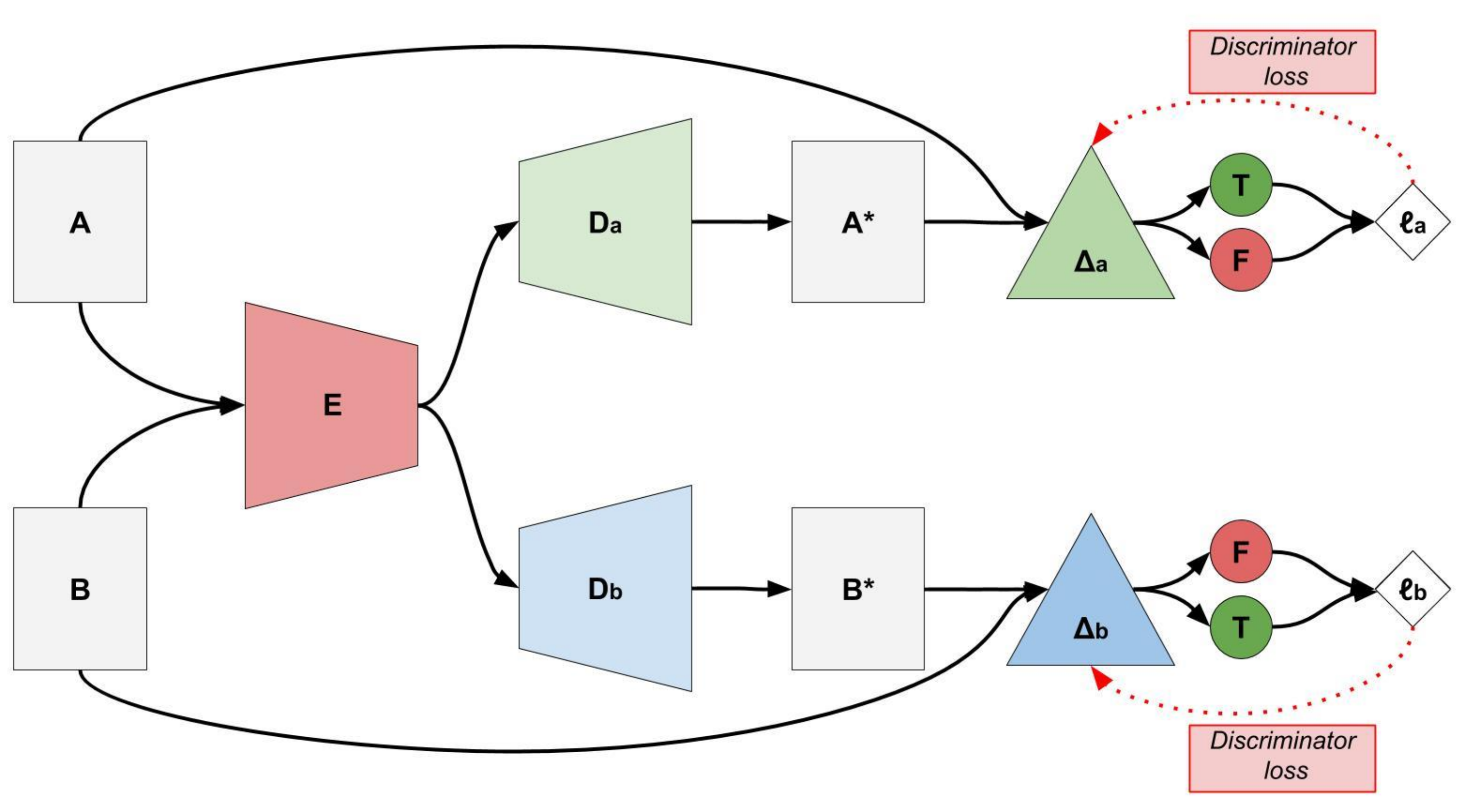
3. Faceswap GAN // Reconstruction train



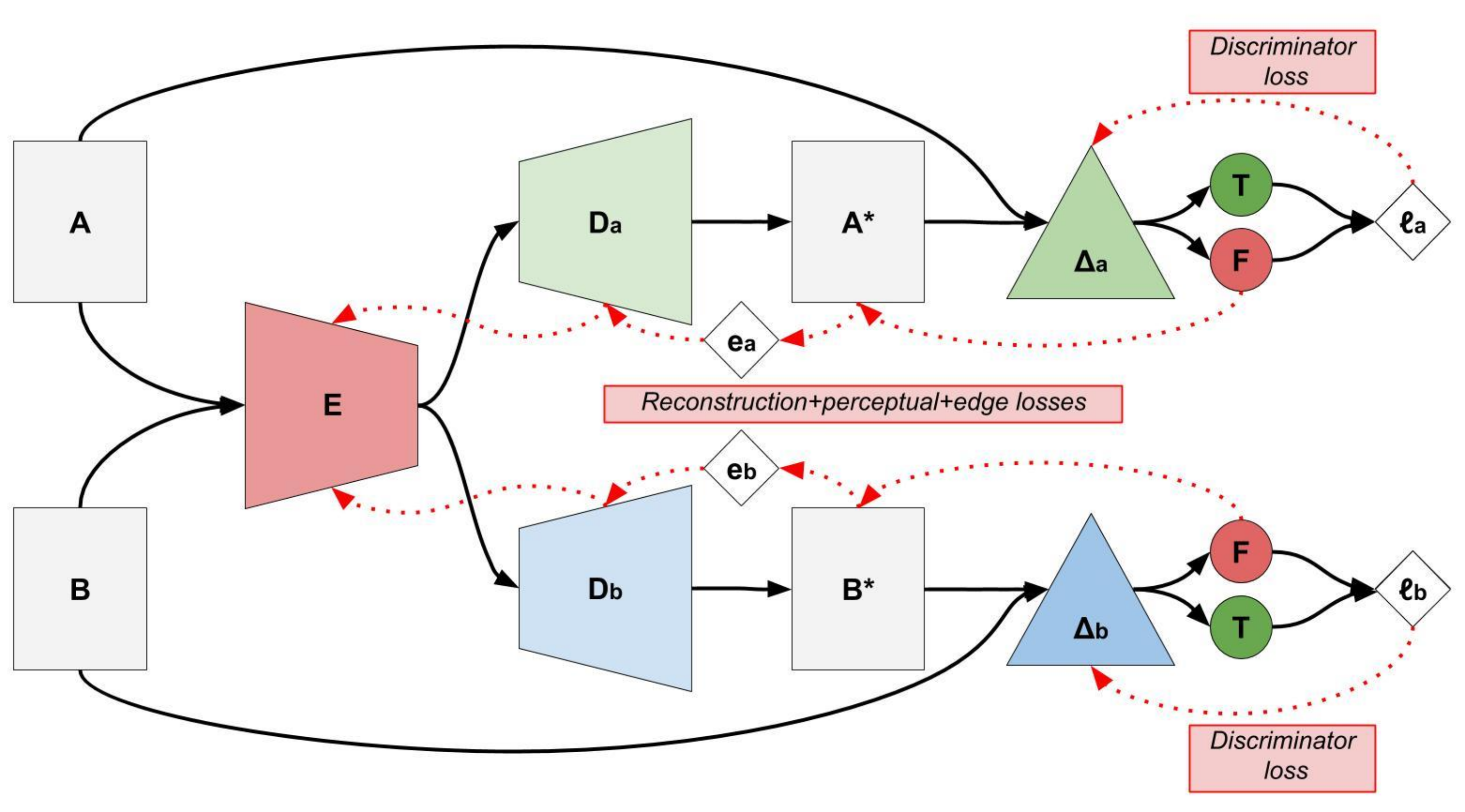
3. Faceswap GAN // Reconstruction train



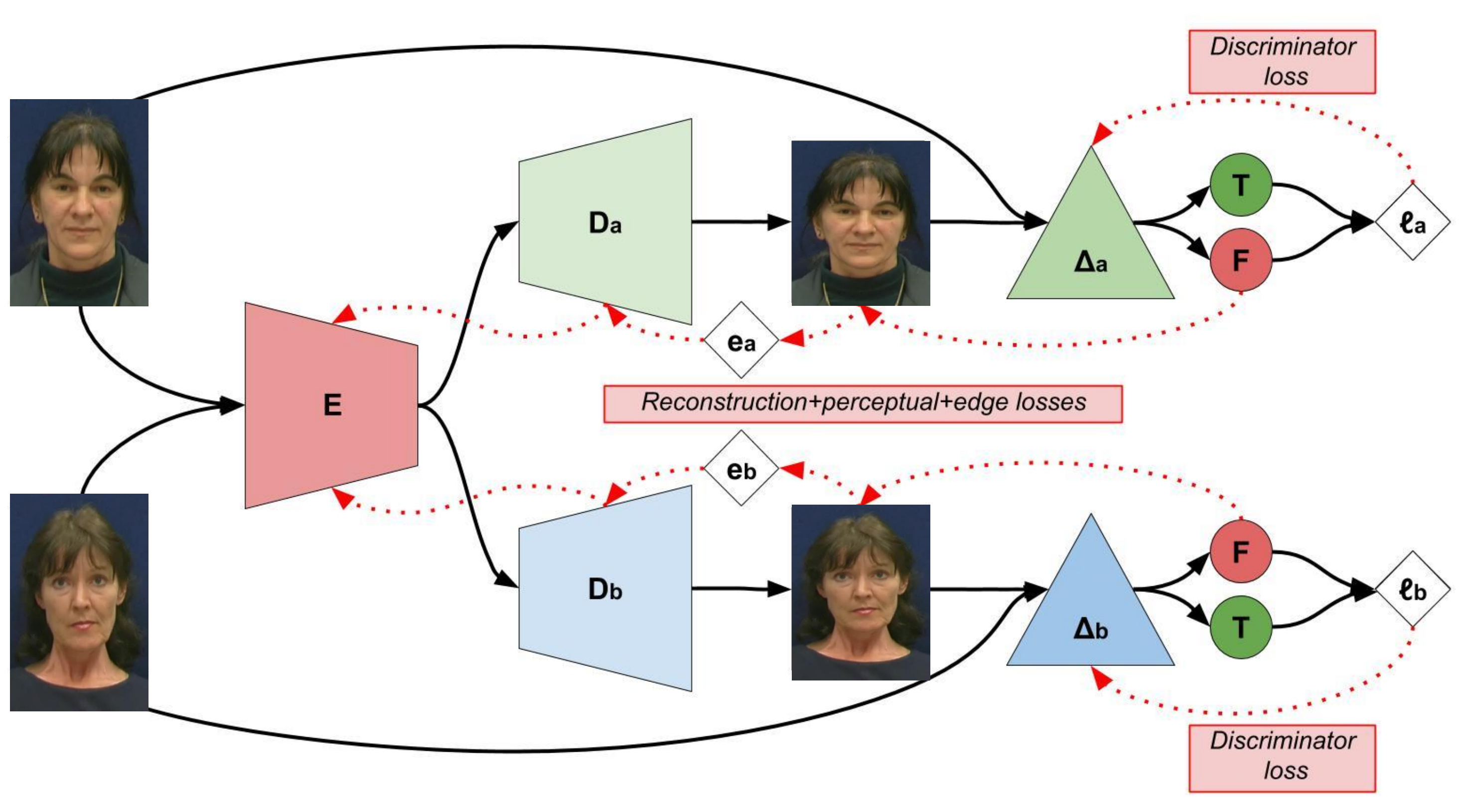
3. Faceswap GAN // Reconstruction train



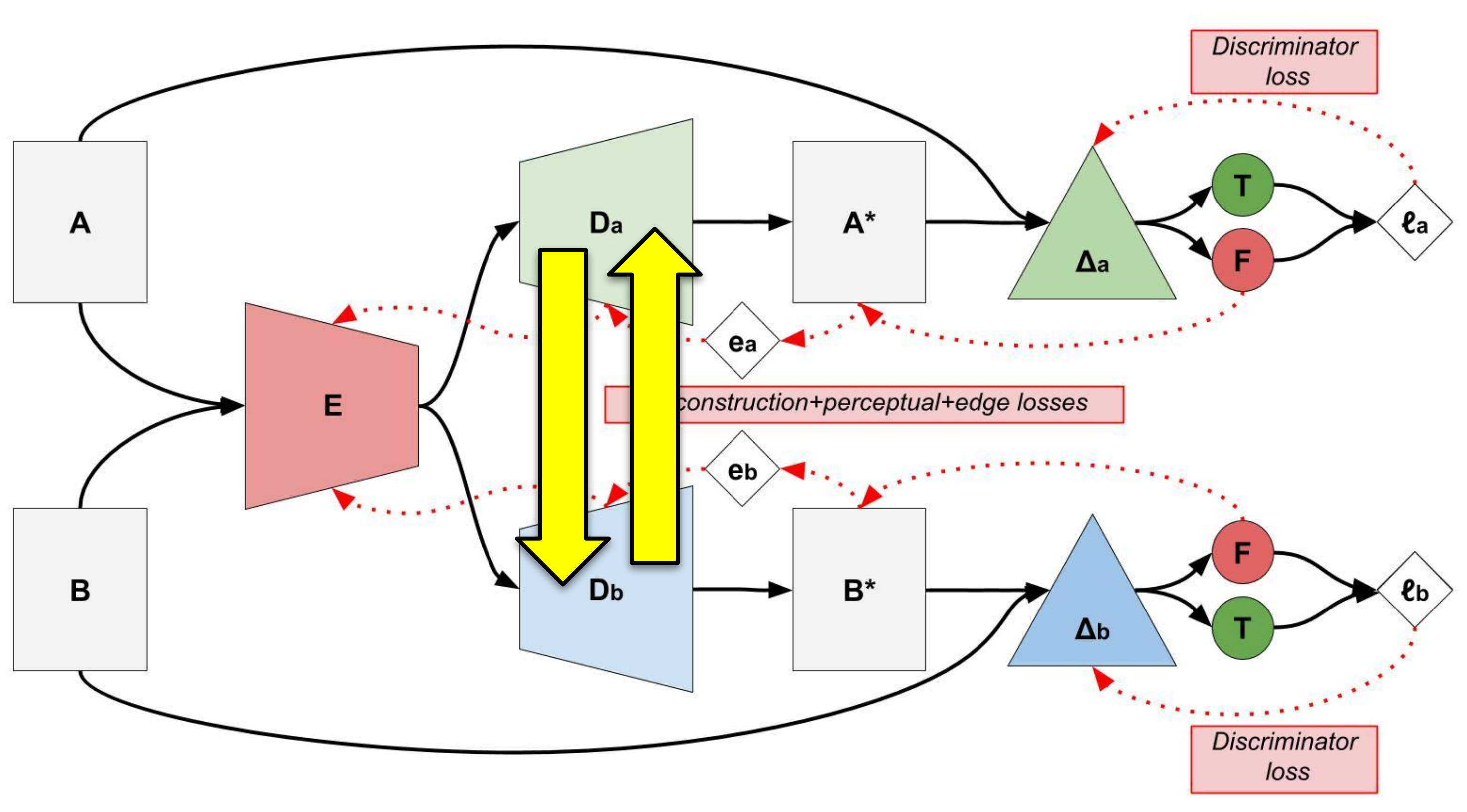
3. Faceswap GAN // Reconstruction train



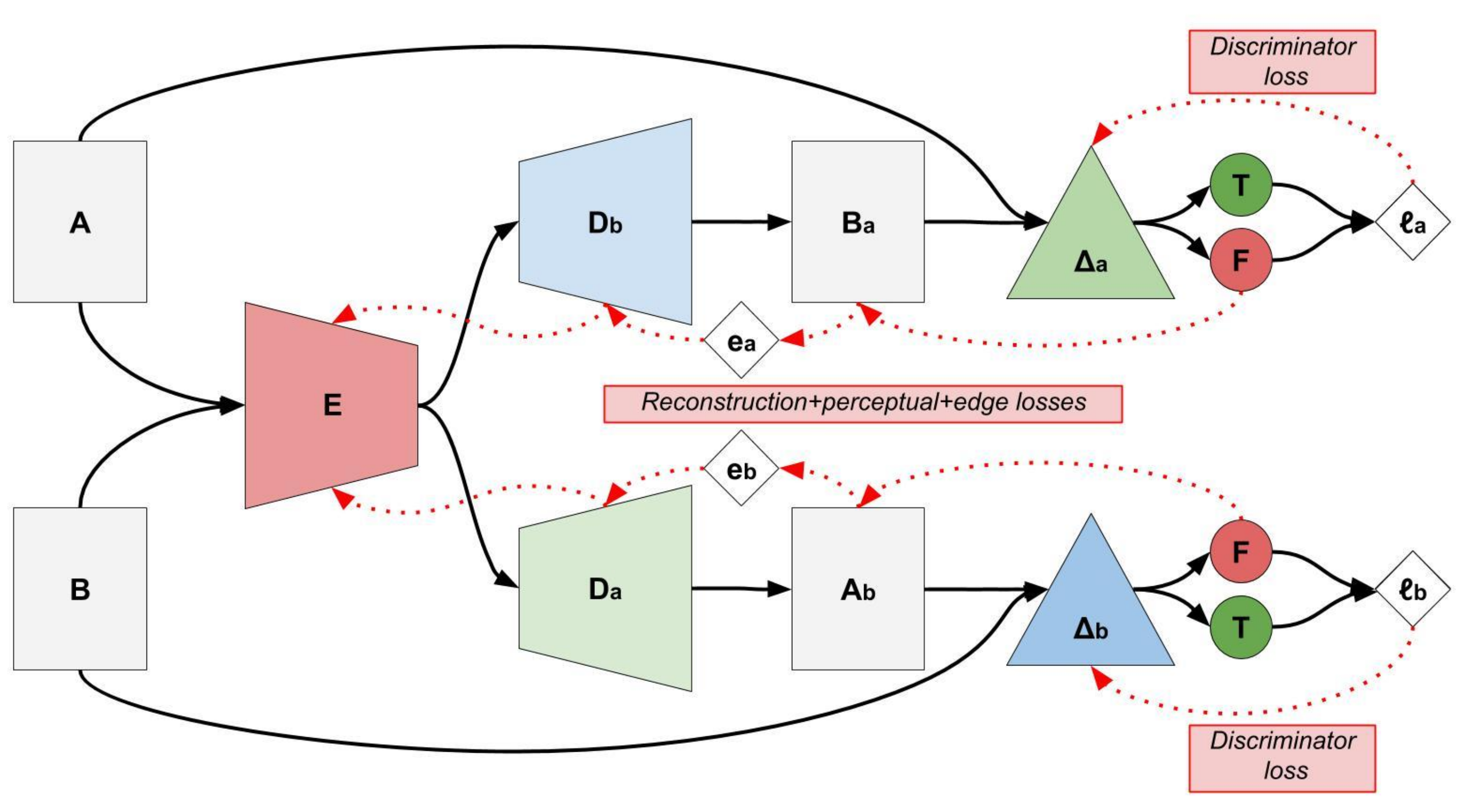
3. Faceswap GAN // Reconstruction train



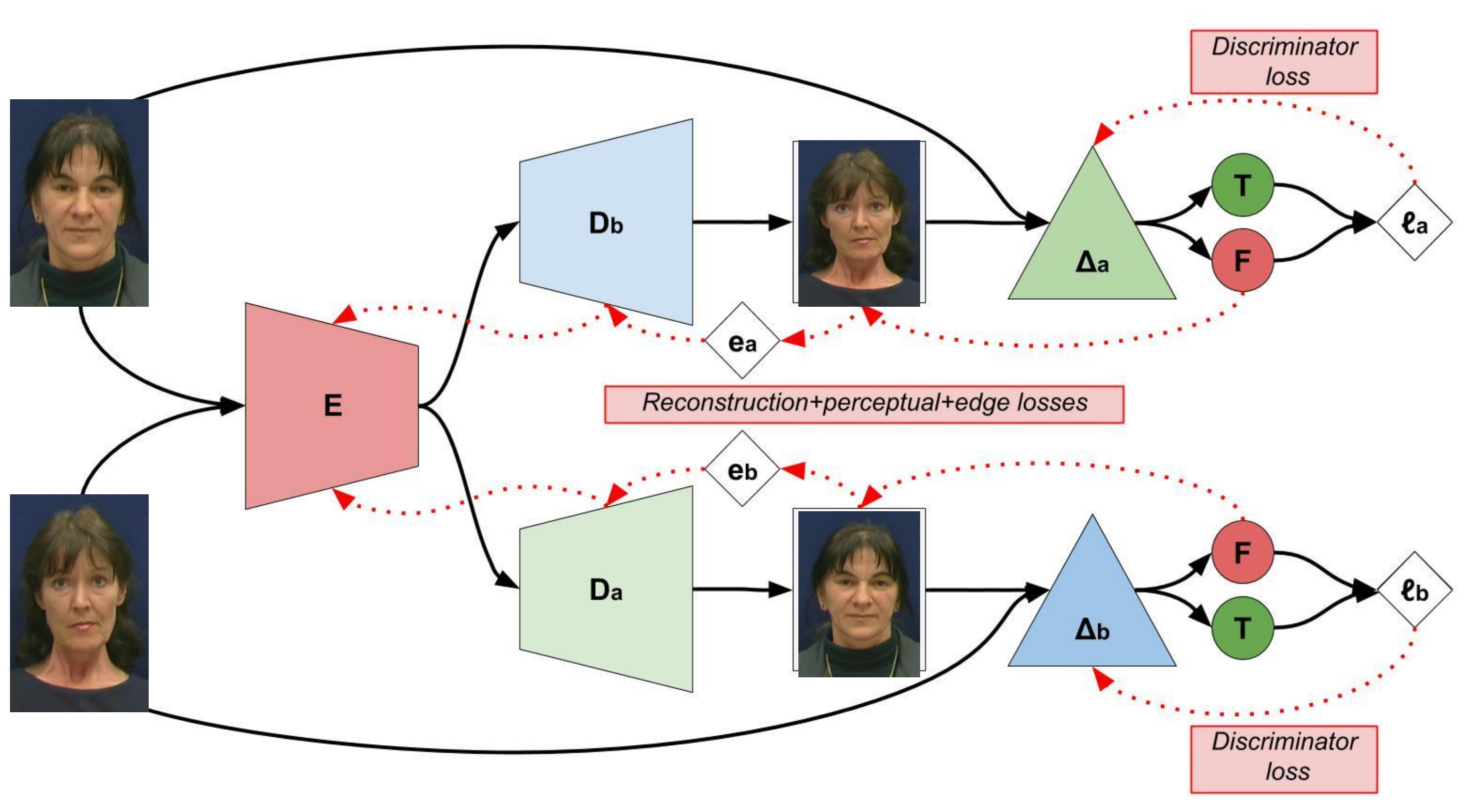
3. Faceswap GAN // Face transformation



3. Faceswap GAN // Face transformation



3. Faceswap GAN // Face transformation

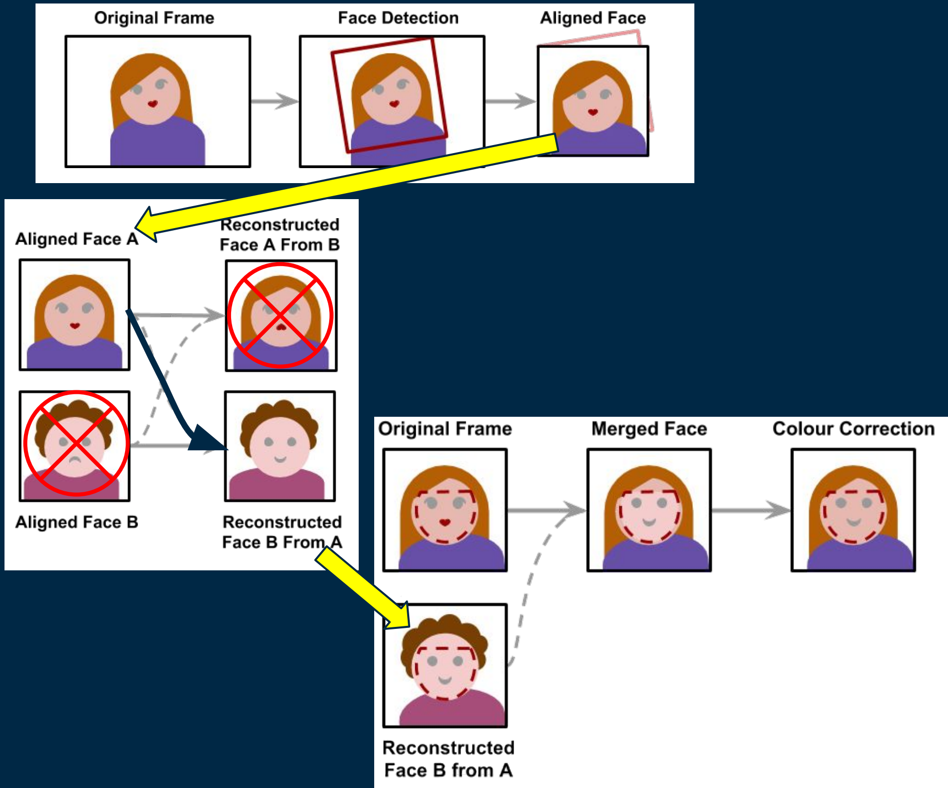


3. Faceswap GAN // Swapping

After trained the GAN with face **A** and **B** videos, it is ready to perform faceswap.

For each frame in the video A:

1. **Detect** face A
2. **Align** face A based on landmarks
3. Feed face A to **trained generator** that outputs the transformed face B and its mask
4. **Reverse** the previous alignment
5. **Merge** transformed face B over face A (post-processing)



4. Dataset generation

We got real faces from **VidTIMIT** dataset, comprised of video and audio recordings of 43 people, reciting short **sentences**.

Our dataset is made of:

- **Real instances**: we selected **18 subjects** and group **9** similar looking **pairs** of people.
- **Fake instances**: for each of the 9 pairs, we trained the model and created **18 fake videos** with faces swapped.

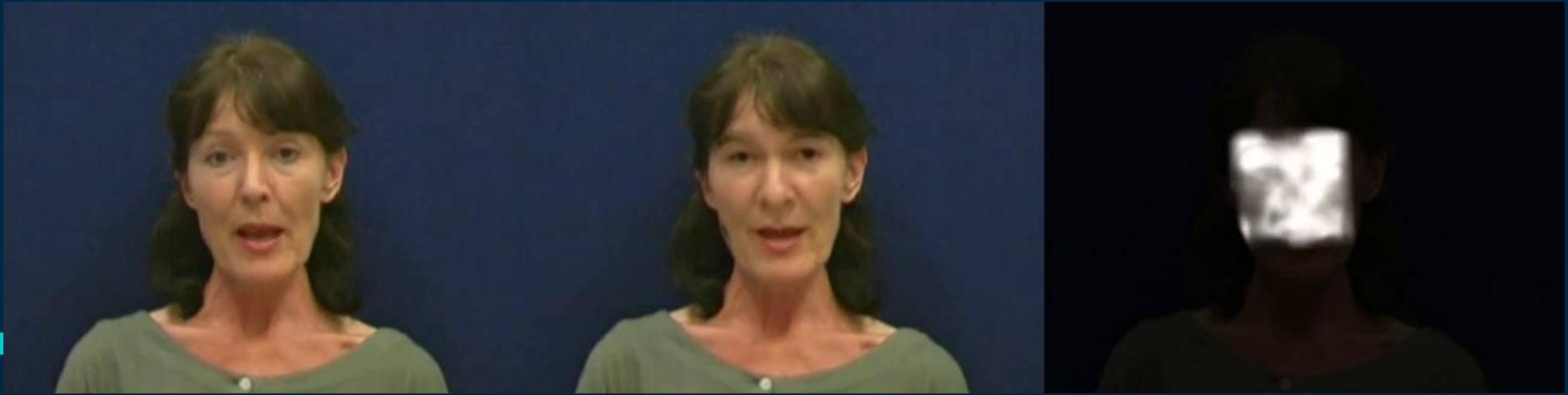
Final size of the dataset is **36 videos**.

VidTIMIT dataset

5. Results

Source

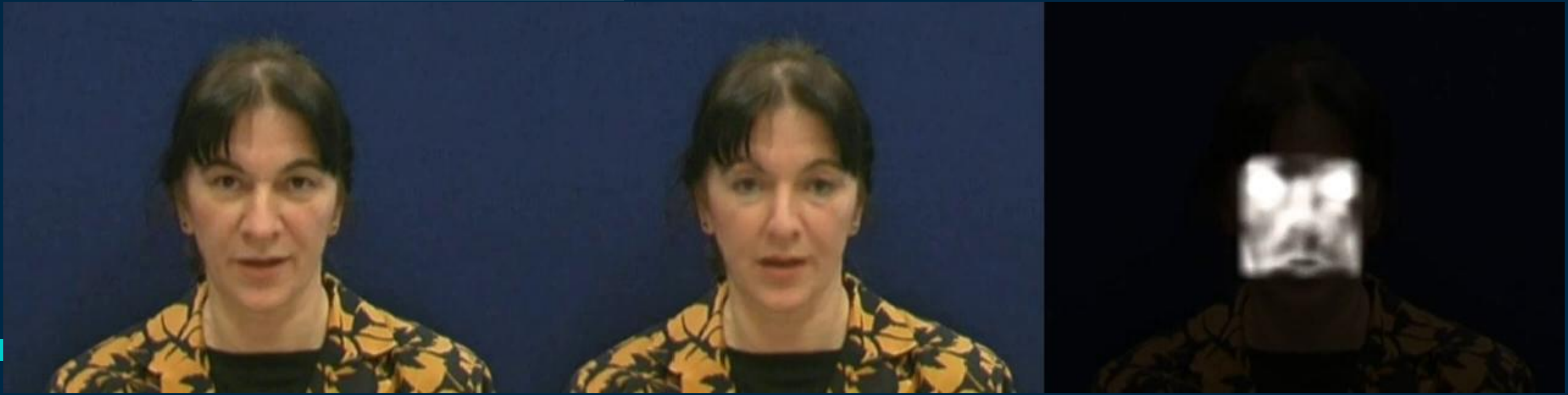
Target



5. Results

Source

Target



6. Project limitations

- **Google Colab** gives us 10-12 hrs of **GPU resources**, not enough to achieve good results
  - Solution could be to save model weights and continue training when resources are available

- Even though GAN is powerful, it takes very **long** to train
  - We run 50000 iterations with low resolution, and results are not really good

- We have to train a **different** model for each different faces pair
  - There are some trials to build a one-for-all model

- Since we create faces independently across **frames**, we should expect the transition to be **less smooth** compared to a real video
  - This can be a start point in developing **deepfake detection tools**!

DEEPFAKE DETECTION

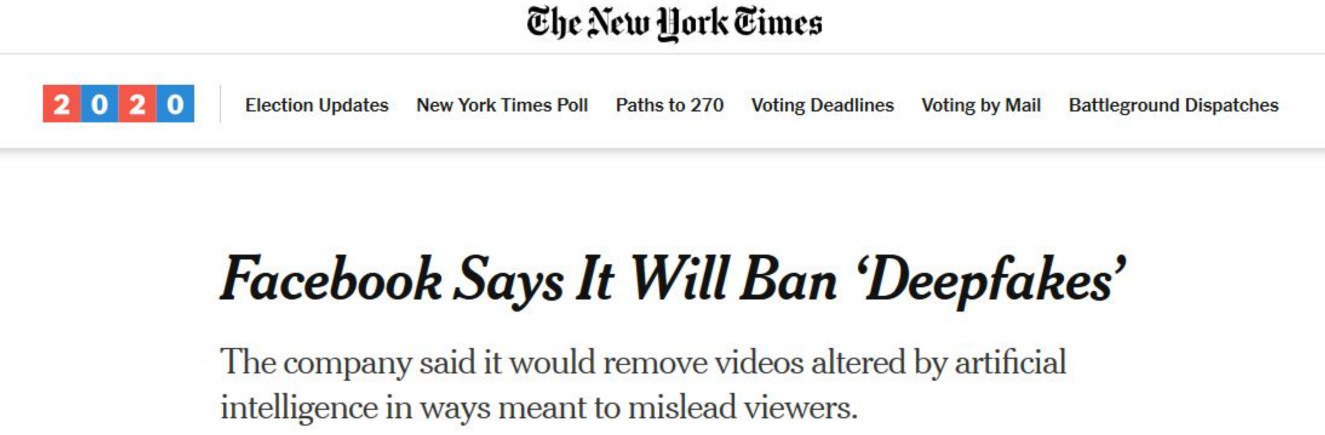
02

PROBLEMS

FAKE NEWS

PRIVACY

PROBLEMS



The New York Times

2020 | Election Updates | New York Times Poll | Paths to 270 | Voting Deadlines | Voting by Mail | Battleground Dispatches

Facebook Says It Will Ban 'Deepfakes'

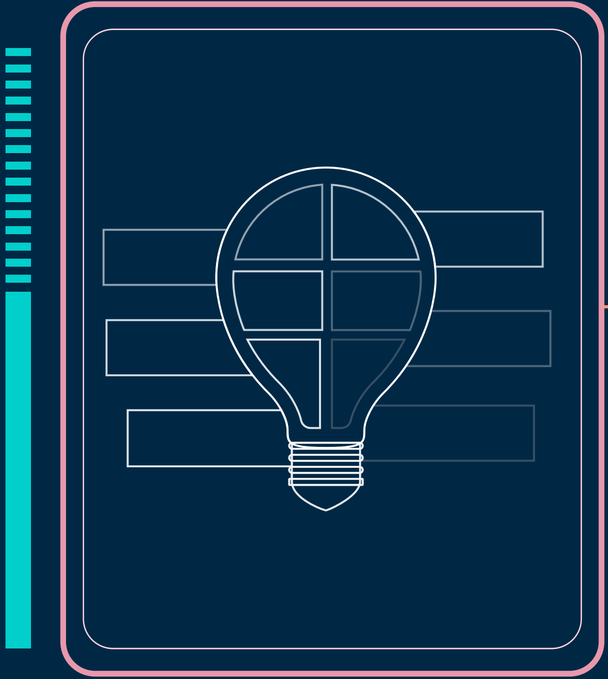
The company said it would remove videos altered by artificial intelligence in ways meant to mislead viewers.

HOW TO BAN THEM?

GOAL

Develop a pipeline to detect deepfakes using ConvLstm and pre-processing steps powered by OpenCv transformation.

The dataset to train the model has been taken from Kaggle DeepFake Challenge



DATASET KAGGLE

470 Gb of full training set

50 smaller file of ~10 GB

Mp4 data format (30 fps, 10 seconds duration)

filename, **label** (REAL/FAKE), **original** and **split columns**

MAIN STEPS

01

DATASET EXPLORING

Balance the dataset and extract face from images

02

OPENCV PREPROCESSING

On the faces extracted we applied the Canny operator

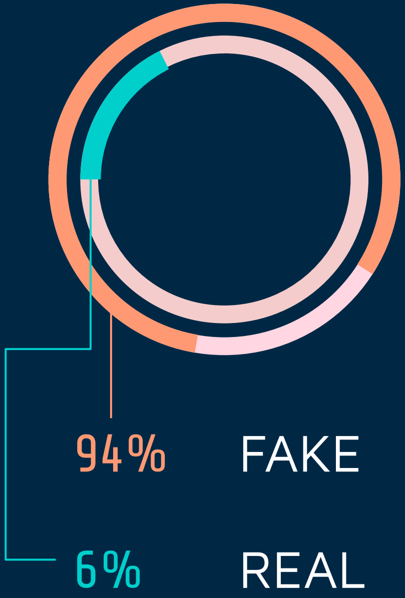
03

MODEL TRAINING

Train a model based on ConvLstm cells

DATASET

VIDEO DISTRIBUTION



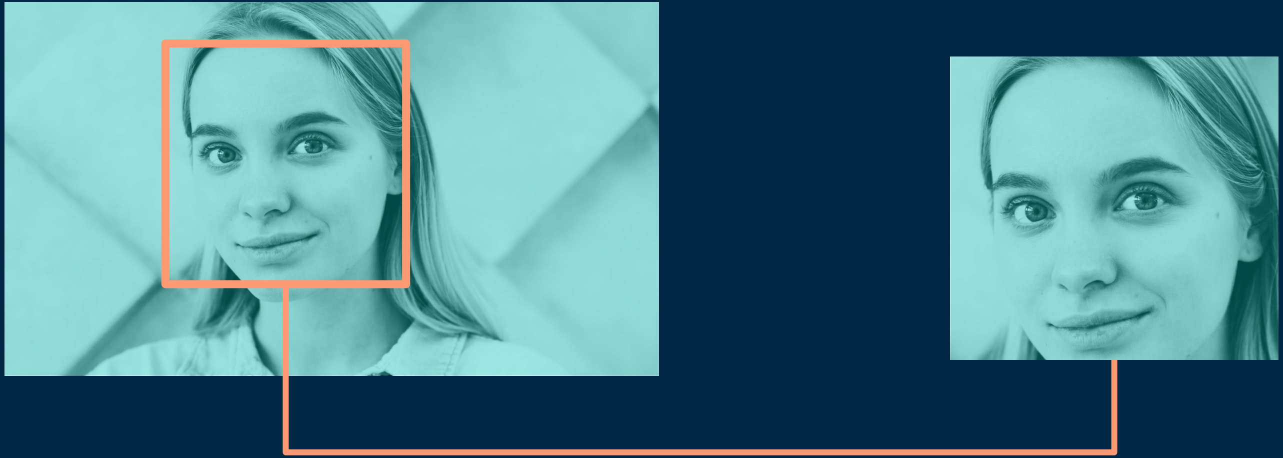
We used a small chunk of the entire dataset (~10Gb)

The dataset was unbalanced, with more FAKE video than REAL

The dataset contains different video of 10 seconds and 30 fps each one

1 // BALANCING THE DATASET

The function to balance read the video. Then Using OpenCv it try to detect frontal or profile face and cut the image. If there is no image detect it check the next frame.

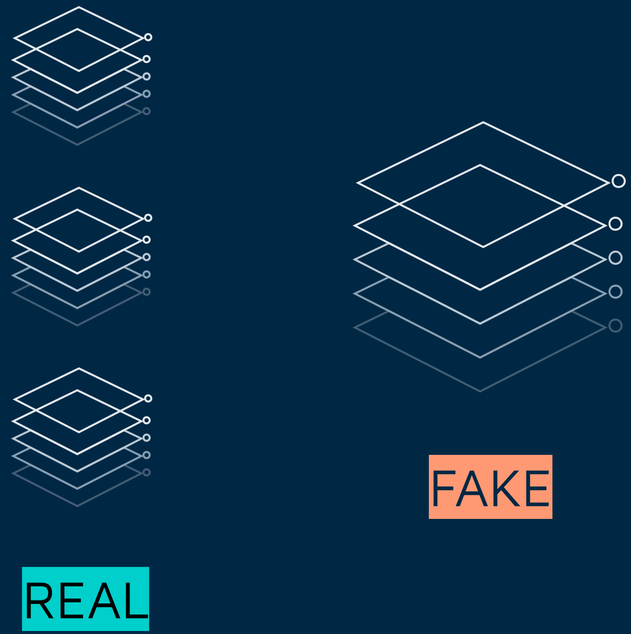


2 // BALANCING THE DATASET

Because each video was 30fps and for the model we want 5 frames per video, we decided to balance the system by taking just 5 frames from FAKE and more sequences of 5 frames from REAL, to make a data augmentation

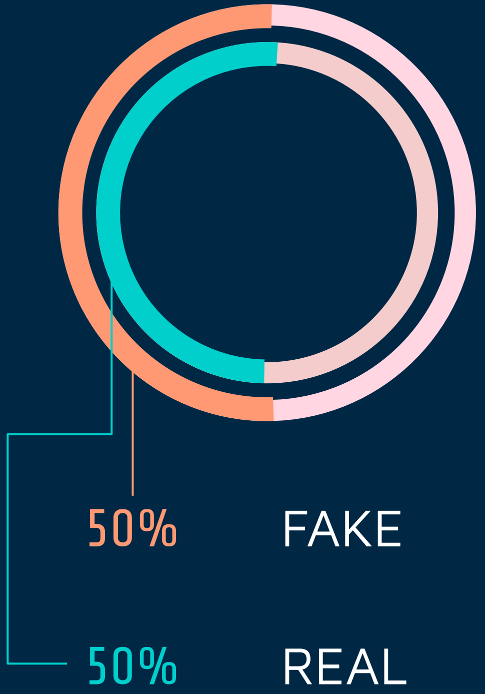
N_seq from real =

n_videos fake/n_videos real



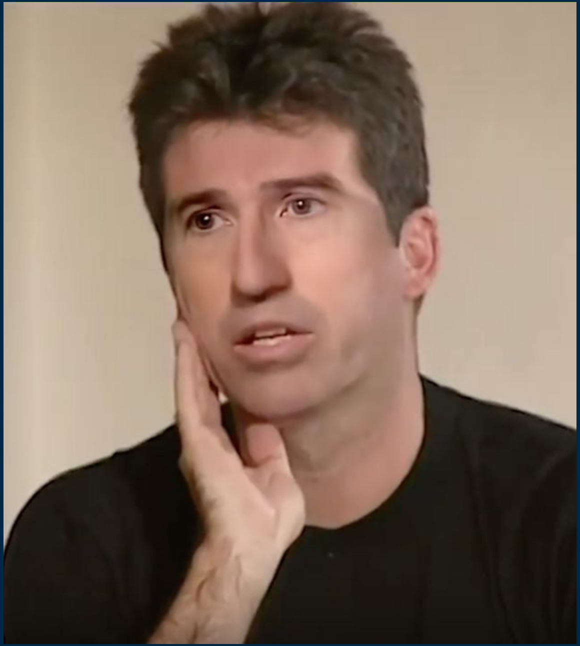
3 // BALANCING THE DATASET

RESULT
VIDEO DISTRIBUTION



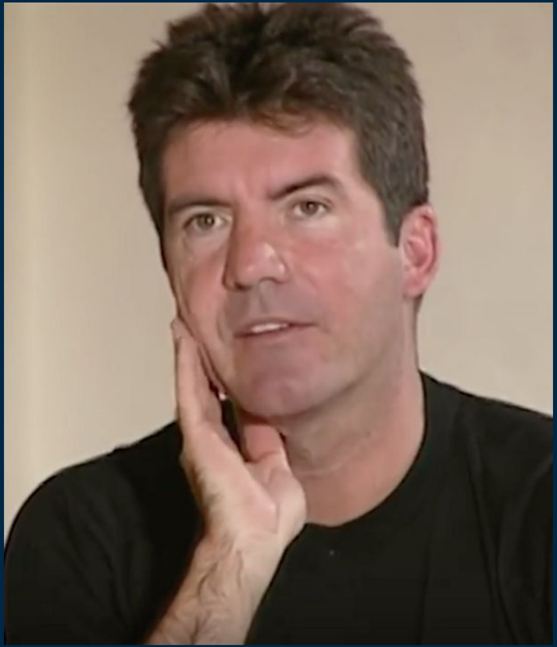
1 // FAKE OR REAL?

FAKE ?

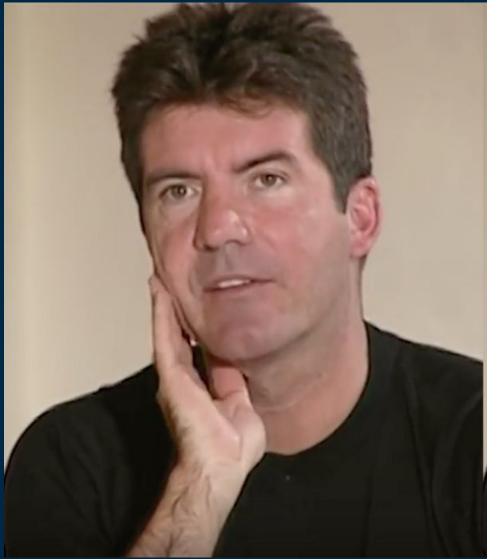


REAL ?

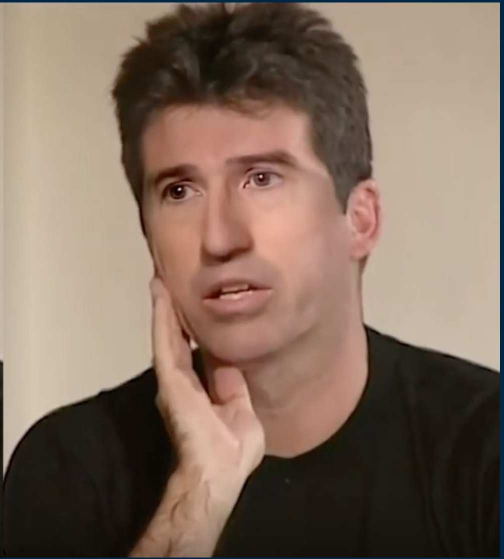
2 // FAKE OR REAL?



1 // OPENCV PREPROCESSING



REAL



FAKE

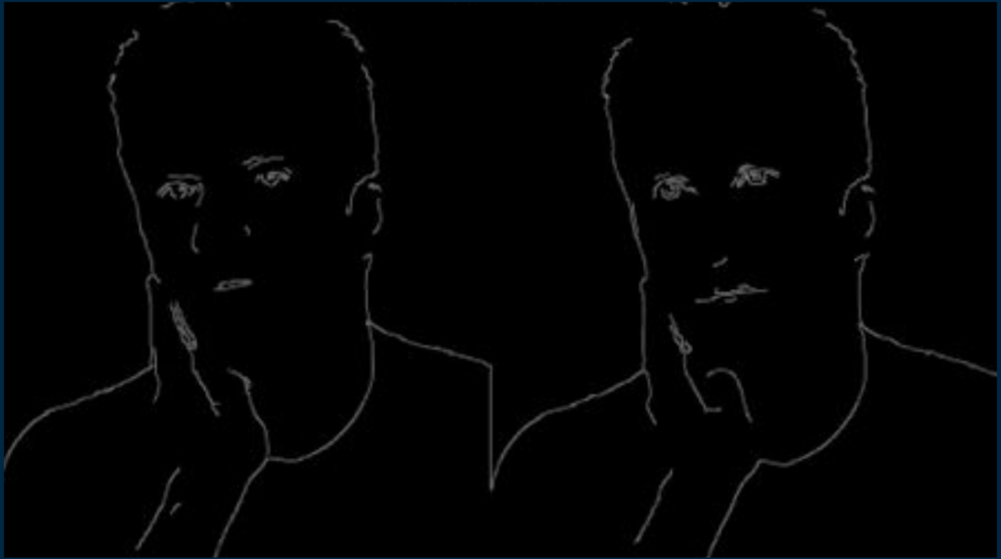
2 // OPENCV PREPROCESSING

Most of DeepFakes are created using GAN. This and other technique introduce visual defects or smooth more than the real images.

Exploiting the RNN we try to keep trace of the previous status of the frame, but before to train the model, we decided to apply EDGE detection method to capture just the most important details.

The algorithm we have used is Canny detection, tuning the threshold during multiple executions.

3 // OPENCV PREPROCESSING



REAL

FAKE

1 // MODEL

GAN generate DeepFakes by manipulating each frame of a video as a single image.

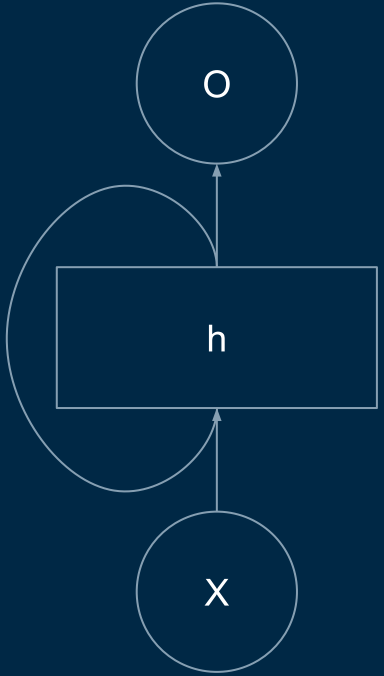
The model based on RNN instead analyze more frames together to try to extract a relationship between them.

Our assumption is that on real video the frame produce a more natural relationship rather than the fake one. So using RNN we want to detect variations due to GAN manipulation

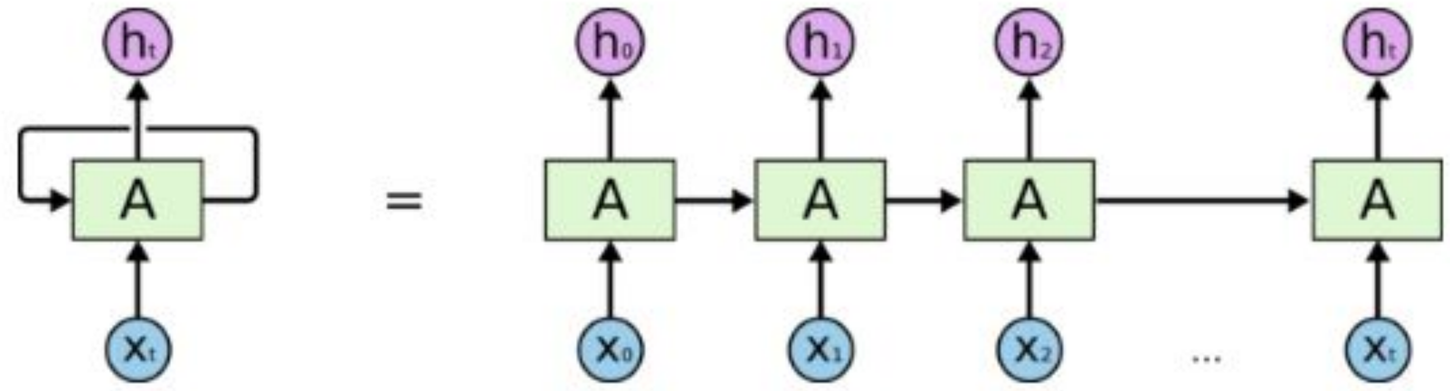
1 // MODEL

Recurrent Neural Network is a generalization of feedforward neural network that has an internal memory.

After producing the output, it is copied and sent back into the recurrent network. For making a decision, it considers the current input and the output that it has learned from the previous input.



2 // MODEL



An unrolled recurrent neural network.

2 // MODEL

WHICH IS THE MAIN PROBLEM OF RNN ?

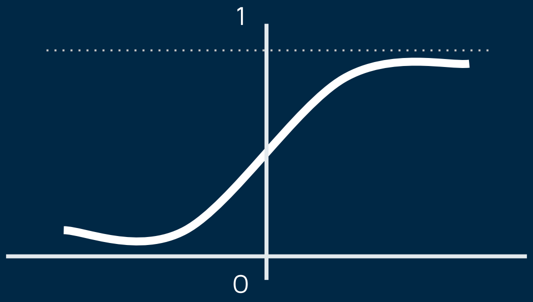
Gradient vanishing

2bis // MODEL

Vanishing gradient problem depends on the choice of the activation function.

Sigmoid maps the real number line onto a "small" range of [0, 1]. As a result, there are large regions of the input space which are mapped to an extremely small range.

With very small of value, the network can’t learn properly the parameters.

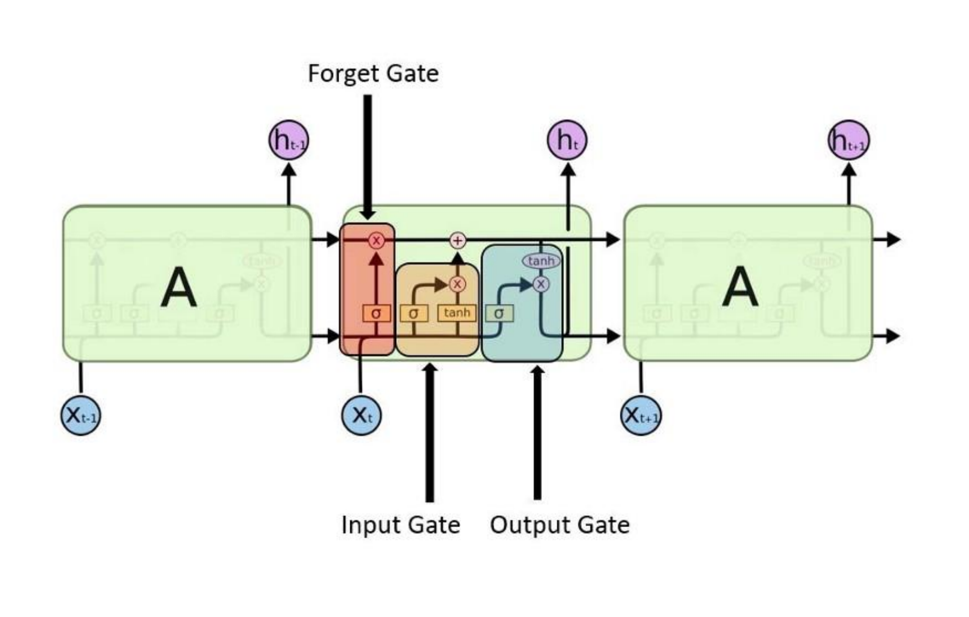


3 // MODEL

To solve this problem we can use LSTM.

The vanishing gradient problem of RNN is resolved here. LSTM is well-suited to classify, process and predict time series given time lags of unknown duration

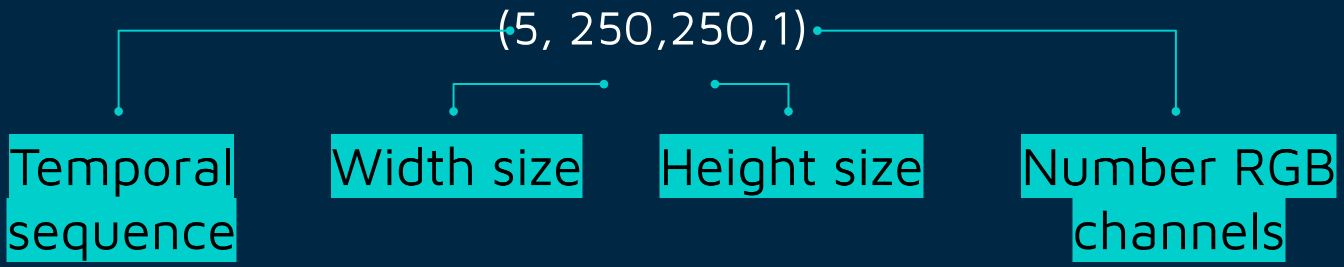
It trains the model by using back-propagation. In an LSTM network, three gates.



4 // MODEL

In our case we decided to use instead of a simple LSTM, a CONVLstm which is equals to the LSTM by the operation between matrix are done by convolution.

We defined a shape for each frame like:



(5, 250,250,1)

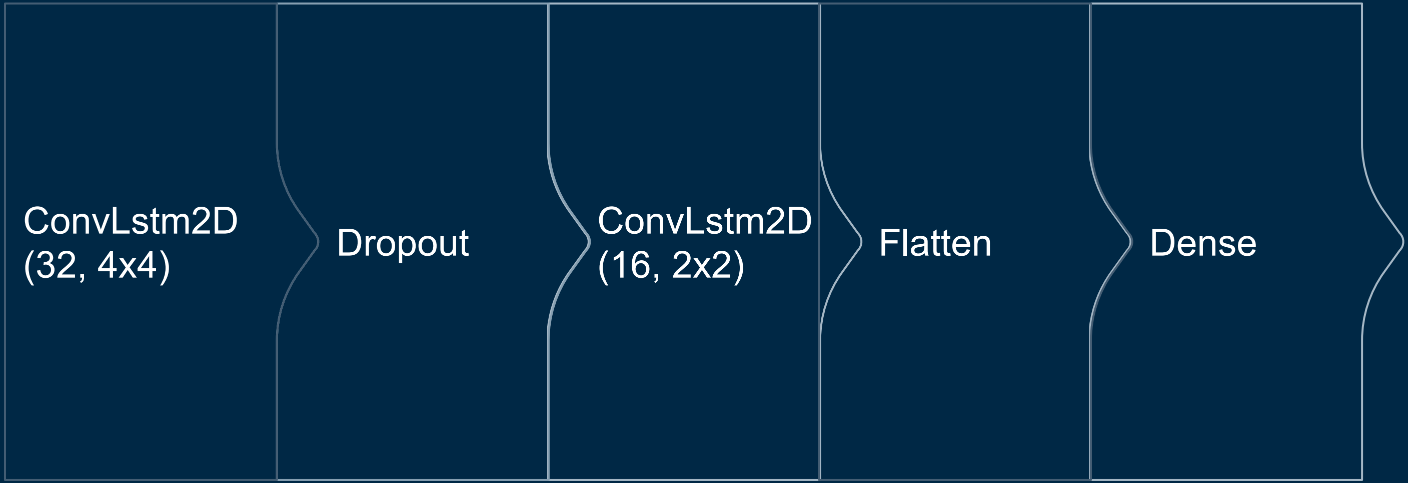
Temporal sequence

Width size

Height size

Number RGB channels

4 // MODEL



ConvLstm2D (32, 4x4)

Dropout

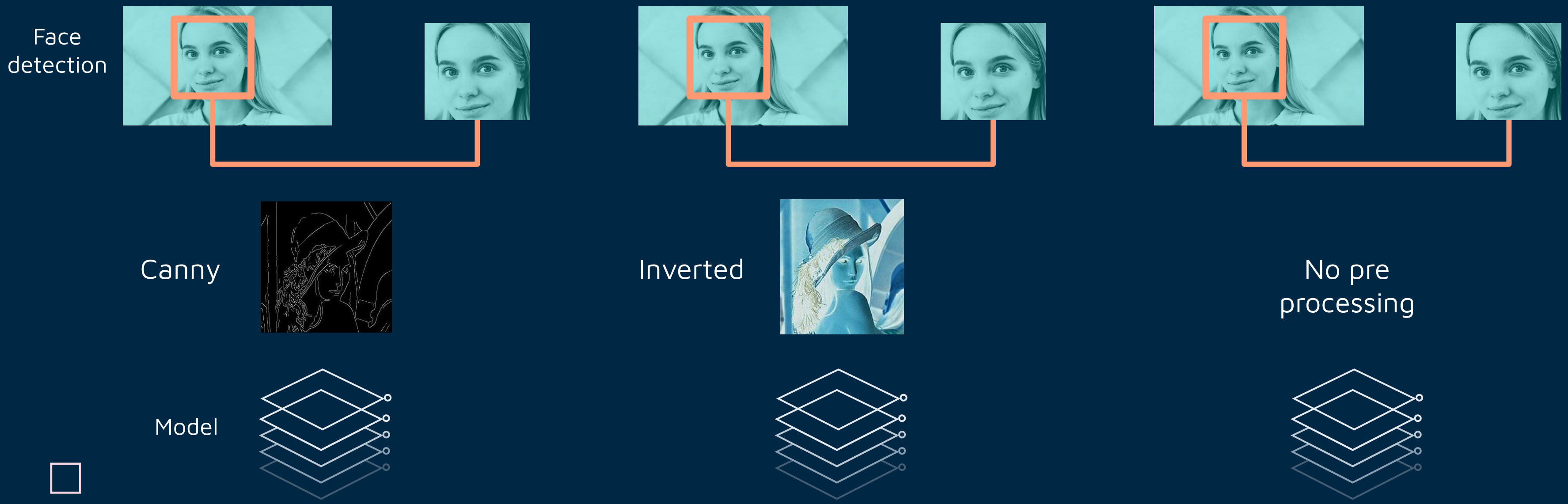
ConvLstm2D (16, 2x2)

Flatten

Dense

1 // EXPERIMENTS

To compare the results from the model described decided to create three differents experiments.



Face detection

Canny

Inverted

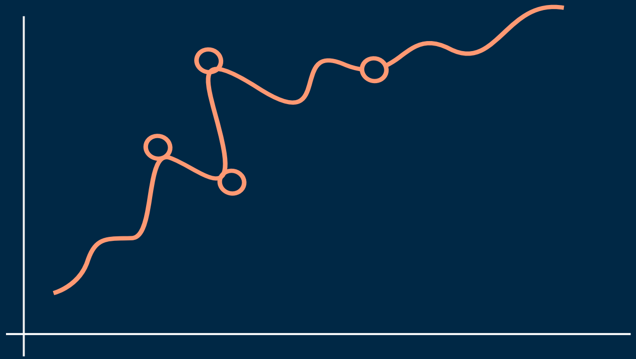
No pre processing

Model

1 // RESULTS

The results on the training set comes from a KFold of 5 fold and 40 epochs, with earlystop function

Kfold, because it generally results in a less biased or less optimistic estimate of the model skill



Earlystopping, to avoid overfitting stopping training at right time

2 // RESULTS

First experiment

```
------------------------------------------------------------------------
Score per fold
------------------------------------------------------------------------
> Fold 1 - Loss: 0.6267279386520386 - Accuracy: 73.69791865348816%
------------------------------------------------------------------------
> Fold 2 - Loss: 0.05866929516196251 - Accuracy: 98.17232489585876%
------------------------------------------------------------------------
> Fold 3 - Loss: 0.0008967018220573664 - Accuracy: 100.0%
------------------------------------------------------------------------
> Fold 4 - Loss: 0.0057458169758319855 - Accuracy: 99.7389018535614%
------------------------------------------------------------------------
> Fold 5 - Loss: 0.6369288563728333 - Accuracy: 81.20104670524597%
------------------------------------------------------------------------
```

2 // RESULTS

```
----------------------------------------------------------------------
Average scores for all folds:
> Accuracy: 90.56203842163086 (+- 10.983925554112066)
> Loss: 0.2657937217969447
----------------------------------------------------------------------
```

Accuracy on test (unseen examples)

```
----------------------------------------------------------------------
> Accuracy: 77
----------------------------------------------------------------------
```

2 // RESULTS

Second experiment

```
--------------------------------------------------------------------------------
Score per fold
--------------------------------------------------------------------------------
> Fold 1 - Loss: 0.6935967803001404 - Accuracy: 48.69791567325592%
--------------------------------------------------------------------------------
> Fold 2 - Loss: 0.6938835978507996 - Accuracy: 48.041775822639465%
--------------------------------------------------------------------------------
> Fold 3 - Loss: 0.6940265893936157 - Accuracy: 48.041775822639465%
--------------------------------------------------------------------------------
> Fold 4 - Loss: 0.6954705715179443 - Accuracy: 44.908615946769714%
--------------------------------------------------------------------------------
> Fold 5 - Loss: 0.6941540241241455 - Accuracy: 48.825064301490784%
--------------------------------------------------------------------------------
```

2 // RESULTS

```
----------------------------------------------------------------------
Average scores for all folds:
> Accuracy: 47.703029513335907 (+- 1.4343643778463617)
> Loss: 0.6942263126373291
----------------------------------------------------------------------
```

Accuracy on test (unseen examples)

```
----------------------------------------------------------------------
> Accuracy: 45
----------------------------------------------------------------------
```

2 // RESULTS

Third experiment

```
------------------------------------------------------------------------
Score per fold
------------------------------------------------------------------------
> Fold 1 - Loss: 0.6935833096504211 - Accuracy: 48.69791567325592%
------------------------------------------------------------------------
> Fold 2 - Loss: 0.6939404010772705 - Accuracy: 48.041775822639465%
------------------------------------------------------------------------
> Fold 3 - Loss: 0.6939529776573181 - Accuracy: 48.041775822639465%
------------------------------------------------------------------------
> Fold 4 - Loss: 0.6952157020568848 - Accuracy: 44.908615946769714%
------------------------------------------------------------------------
> Fold 5 - Loss: 0.6940943598747253 - Accuracy: 48.825064301490784%
------------------------------------------------------------------------
```

2 // RESULTS

```
----------------------------------------------------------------------
Average scores for all folds:
> Accuracy: 47.70302951335907 (+- 1.4343643778463617)
> Loss: 0.694157350063324
----------------------------------------------------------------------
```

Accuracy on test (unseen examples)

```
----------------------------------------------------------------------
> Accuracy: 44
----------------------------------------------------------------------
```

1 // EXAMPLE ON PIPELINE PRODUCED EXAMPLES

REAL IMAGE

CLASS PREDICTED

REAL

2 // EXAMPLE ON PIPELINE PRODUCED EXAMPLES

FAKE IMAGE

CLASS PREDICTED

FAKE

2 // EXAMPLE ON PIPELINE PRODUCED EXAMPLES

Using the dataset produced by previous steps, over 36 examples (18 REAL and 18 FAKE) the model was able to correctly classify detect 19 examples out of 36 (TP and TN)

1 // Limits

Working in this project produced some limits based on the assumption that we have made:

CONS

- The threshold in Canny detection is tuned manually
- The face detection algorithms of OpenV
- The size of dataset

THANKS