ML-FINAL REPORT

Detect edited face image and real image and interpret multiple latent features simultaneously

Group Name - Titans.
Group Members - Daramalla Nishitha(11940320),
Bandela Santaz Sahithi(11940230).

Abstract:

Despite the recent advance of Generative Adversarial Networks (GANs) in high-fidelity image synthesis, there lacks enough understanding of how GANs are able to map a latent code sampled from a random distribution to a photorealistic image. Previous work assumes the latent space learned by GANs follows a distributed representation but observes the vector arithmetic phenomenon. In this work, we propose a novel framework, called InterFaceGAN, for semantic face editing by interpreting the latent semantics learned by GANs. We find that the latent code for well-trained generative models, such as PGGAN and StyleGAN, actually learns a disentangled representation after some linear transformations. Based on our analysis, we propose a simple and general technique, called **InterFaceGAN**, for semantic face editing in latent space. We manage to control the pose as well as other facial attributes, such as gender, age, eyeglasses. More importantly, we are able to correct the artifacts made by GANs.

Introduction:

The popularity of sharing selfies and portrait photos online motivates the rapid development of face editing tools. Facial attribute manipulation is especially attractive with the functions of adding/removing face accessories, such as facial hair and eyeglasses, and/or changing intrinsic face properties, such as age and gender. Facial attribute manipulation has attracted great interest, because of the great chance it brings to research and real-world application. Early work focuses on specific attributes of facial hair generation, expression change, beautification/de-beautification, aging, etc. Recently, with the development of deep neural networks, especially generative adversarial networks, several general face attribute manipulation frameworks were proposed. These approaches take facial attribute editing as an unpaired learning task, and thus are capable of handling different attributes by only changing

the data. Our method can be categorized into this group, which aims to provide a general solution for different facial attributes.

Problem Definition:

- Given an image(containing face) we need to edit the facial attributes like age, gender, smile etc. by interpreting the latent space of GANs.
- **Upgradation part is** we have to build a fake face detector and interpret multiple latent features simultaneously

Objective:

The main objective of our project is to generate the images and edit their facial attributes.
We have to train five independent linear SVMs on pose, smile, age, gender, eyeglasses, and then evaluate them on the validation set (6K samples with high confidence level on attribute scores) as well as the entire set (480K random samples). We also try to visualize some samples by ranking them with the distance to the decision boundary.

Upgrading our objective is to build a fake face detector to check if the images that we obtained after manipulating the facial expressions, if they are real or not.
And to interpret different latent features simultaneously.

Technology used:

- python 3.7
- pytorch 1.1.0
- tensorflow 1.12.2
- sklearn 0.21.2
- Google collab
- Google drive
- Github
- mtcnn
- keras

Problems faced:

- As the given models are already pre-trained and were saved as pytorch files, we didn't face many problems for collecting the data.
- The GitHub Repository and the Readme that we used for reference is badly documented and hence we faced issues in understanding the code and in it's implementation.

- The readme file didn't mention anything about how to train the model so it was very tough for us to understand how to train the model.

Datasets:

1.CelebA-HQ

The **CelebA-HQ** dataset is a high-quality version of CelebA that consists of 30,000 images with 1024×1024 resolution. To meaningfully demonstrate our results at high output resolutions, we need a sufficiently varied high-quality dataset. However, virtually all publicly available datasets previously used in GAN literature are limited to relatively low resolutions ranging from 322 to 4802. To this end, we created a high-quality version of the CELEBA dataset consisting of 30000 of the images at 1024 × 1024 resolution.

1024 × 1024 images generated using the CELEBA-HQ dataset.

2.FFHQ-(Flickr-Faced-HQ)

Flickr-Faces-HQ (FFHQ) is a high-quality image dataset of human faces, originally created as a benchmark for generative adversarial networks (GAN). FFHQ consists of 70,000 high-quality PNG images at 1024×1024 resolution and contains considerable variation in terms of age, ethnicity and image background. It also has good coverage of accessories such as eyeglasses, sunglasses, hats, etc. The images were crawled from Flickr, thus inheriting all the biases of that website, and automatically aligned and cropped using dlib. Only images under permissive licenses were collected. Various automatic filters were used to prune the set, and finally Amazon Mechanical Turk was used to remove the occasional statues, paintings, or photos.

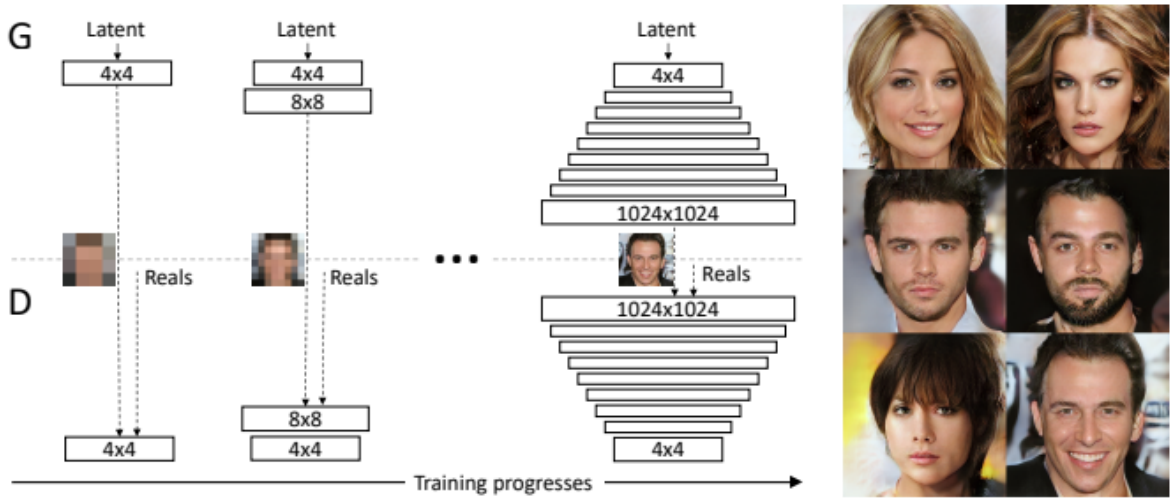
MODELS:

GAN**-** Generative Adversarial Networks
The generator model in the GAN architecture takes a point from the latent space as input and generates a new image. It basically maps the latent codes (commonly sampled from high-dimensional latent space, such as standard normal distribution) to photo-realistic images.

- Progressive GAN

Architecture of the PGGAN Model:

We describe a new training methodology for generative adversarial networks. The key idea is to grow both the generator and discriminator progressively: starting from a low resolution, we add new layers that model increasingly fine details as training progresses. This both speeds the training up and greatly stabilizes it, allowing us to produce images of unprecedented quality, e.g., CelebA images at 1024^2. We also propose a simple way to increase the variation in generated images, and achieve a record inception score of 8.80 in unsupervised CIFAR10. Additionally, we describe several implementation details that are important for discouraging unhealthy competition between the generator and discriminator. Finally, we suggest a new metric for evaluating GAN results, both in terms of image quality and variation. As an additional contribution, we construct a higher-quality version of the CelebA dataset.



Training progress in pggan

Our training starts with both the generator (G) and discriminator (D) having a low spatial resolution of 4×4 pixels. As the training advances, we incrementally add layers to G and D, thus increasing the spatial resolution of the generated images. All existing layers remain trainable throughout the process. Here N × N refers to convolutional layers operating on N × N spatial resolution. This allows stable synthesis in high resolutions and also speeds up training considerably. One the right we show six example images generated using progressive growing at 1024 × 1024.

- Style GAN (A Style-Based Generator Architecture for Generative Adversarial Networks)

Architecture of the StyleGAN

We propose an alternative generator architecture for generative adversarial networks, borrowing from style transfer literature. The new architecture leads to an automatically learned, unsupervised separation of high-level attributes (e.g., pose and identity when trained on human faces) and stochastic variation in the generated images (e.g., freckles, hair), and it enables intuitive, scale-specific control of the synthesis. The new generator improves the state-of-the-art in terms of traditional distribution quality metrics, leads to demonstrably better interpolation properties, and also better disentangles the latent factors of variation. To quantify interpolation quality and disentanglement, we propose two new, automated methods that are applicable to any generator architecture. Finally, we introduce a new, highly varied and high-quality dataset of human faces.

The main contributions of the StyleGAN architecture are:

1. The introduction of a mapping network.
2. The introduction of the Affine Transformation (A) and Adaptive Instance Normalization (AdaIN).
3. The addition of a noise vector (B).

Different from conventional GANs, StyleGAN proposed a style-based generator. Basically, StyleGAN learns to map the latent code from space Z to another high dimensional space W before feeding it into the generator. Latent space W shows much stronger disentanglement property than Z, since W is not restricted to any certain distribution and can better model the underlying character of real data. We did a similar analysis on both Z and W spaces of StyleGAN as did to PGGAN and found that W space indeed learns a more disentangled representation. Such disentanglement helps W space achieve strong superiority over Z space for attribute editing.

The introduction of a mapping network.

The mapping network consists of a multi-layer perceptron (MLP) with eight layers. Its role is to encode the input latent vector z into an intermediate latent space W. This input latent vector z must have the probability density of the training data, thus having a strong effect on how the various factors have represented the network. Unlike traditional architecture, where the latent vector is provided to the generator through an input layer, with StyleGAN, we start from a learned constant. The input layer is omitted in this architecture.The W vectors are then fed into the Affine Transformation module (A). The letter y represents the output styles of module (A). These outputs, along with the features of the previous convolutions x, are passed as input to the Adaptive Instance Normalization layers. Additionally, we have a constant tensor of size 4 x 4 x 512 feeding the synthesis network g, consisting of 18 layers.

The Affine Transformation (A) and Adaptive Instance Normalization (AdaIN)

Adaptive Instance Normalization (AdaIN) is a normalization technique that is derived from the Batch Normalization technique. Batch Normalization was first introduced to GANs to improve the discriminator training.They are also recently being used in generative image modeling.The batch normalization algorithm is shown below:

$$\text{AdaIN}(x, y) = \sigma(y)\left(\frac{x - \mu(x)}{\sigma(x)}\right) + \mu(y)$$

The noise vector (B)

A noise vector is introduced into the synthesis network through the affine transformation layer (B). They are added throughout the network, not just in the beginning as it was with GANs. It consists of single-channel images of un-correlated gaussian noise. It is added to the output of each corresponding 3 x 3 convolution layer.
The noise vector is introduced to induce stochastic details into images in the network. Thus, adding noise is ideal for controlling stochastic variations such as differently combed hair, skin pores, freckles, and beards.
All these stochastic variations can be adjusted without affecting the overall perception of the image. It leaves the overall composition of the image and the high-level aspects such as identity intact.

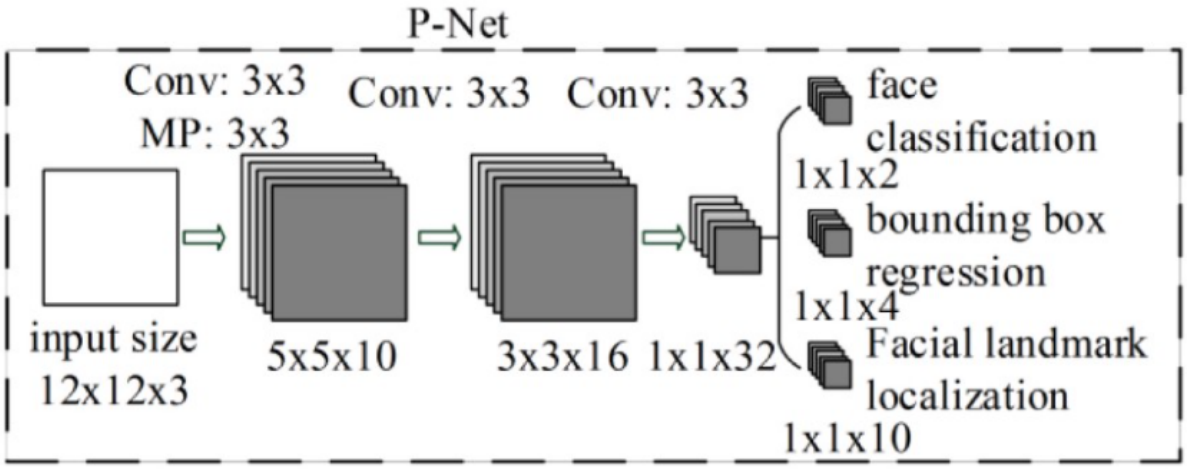
Architecture of MTCNN

MTCNN is the cascade structure of three networks; Proposal Network (P-Net), Refine Network (R-Net) and Output Network (O-Net). The process starts with re-scaling the image to a range of different sizes (referred as image pyramid) and passing them to the P-Net which proposes the facial regions in image, then R-Net filters the bounding boxes of the images and finally the O-Net proposes facial landmarks i.e. eyes, mouth etc.

The Three Stages of MTCNN:

The first step is to take the image and resize it to different scales in order to build an image pyramid, which is the input of the following three-staged cascaded network.

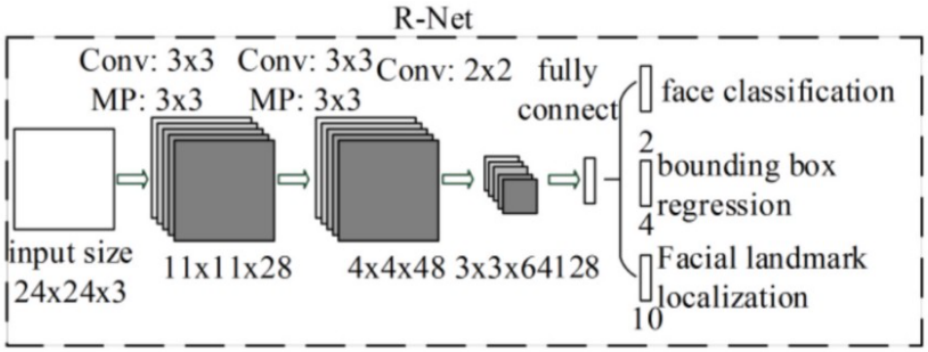
Stage 1: The Proposal Network (P-Net)



This first stage is a fully convolutional network (FCN). The difference between a CNN and a FCN is that a fully convolutional network does not use a dense layer as part of the architecture. This Proposal Network is used to obtain candidate windows and their bounding box regression vectors.
Bounding box regression is a popular technique to predict the localization of boxes when the goal is detecting an object of some predefined class, in this case faces. After obtaining the bounding box vectors, some refinement is done to combine overlapping regions. The final output of this stage is all candidate windows after refinement to downsize the volume of candidates.

Stage 2: The Refine Network (R-Net)

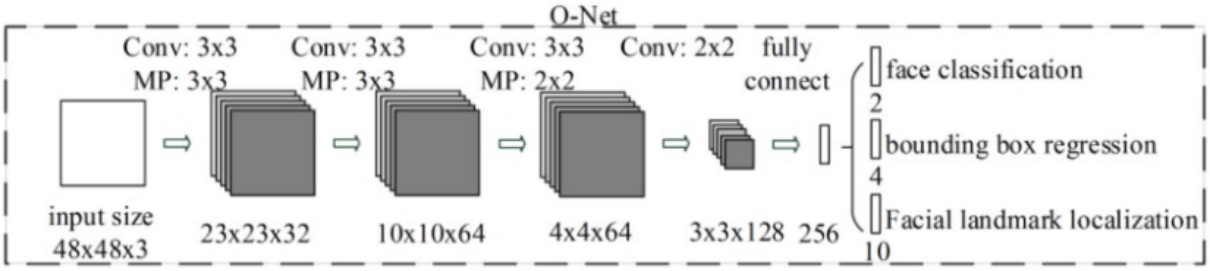
All candidates from the P-Net are fed into the Refine Network. Notice that this network is a CNN, not a FCN like the one before since there is a dense layer at the last stage of the network architecture. The R-Net further reduces the number of candidates, performs calibration with bounding box regression and employs non-maximum suppression (NMS) to merge overlapping candidates.



The R-Net outputs whether the input is a face or not, a 4 element vector which is the bounding box for the face, and a 10 element vector for facial landmark localization.

Stage 3: The Output Network (O-Net)

This stage is similar to the R-Net, but this Output Network aims to describe the face in more detail and output the five facial landmarks' positions for eyes, nose and mouth.



Implementation:

To find the semantic boundaries in the latent space, we use the pre-trained attribute prediction model to assign attribute scores for all 500K synthesized images. For each attribute, we sort the corresponding scores, and choose 10K samples with highest scores and 10K with lowest ones as candidates. The reason in doing so is that the prediction model is not absolutely accurate and may produce wrong predictions for ambiguous samples, e.g., middle-aged person for age attribute. We then randomly choose 70% samples from the candidates as the training set to learn a linear SVM, resulting in a decision boundary. Recall that, normal directions of all boundaries are normalized to unit vectors.Remaining 30% are used for verifying how the linear classifier behaves. Here, for SVM training, the inputs are the 512d latent codes, while the binary labels are assigned by the auxiliary attribute prediction model.
Note : We used the pretrained boundaries provided in github page

Multiple Latent Features

- Find all the boundaries for single attribute
- If the boundary for the a1 attribute is b1 and the boundary for the attribute a2 is b2
- The resulting plane or boundary we get by computing b1/2 + b2/2 is the boundary for changing both a1 and a2 attributes simultaneously.
- When the latent codes walk in the direction of normal to the above boundary the multiple latent features can be interpreted.

Fake Face Detection

For training the CNN classifier of our model in fake face detection. We just need to have the keras module of python . We can get it using pip install keras.

1) First of all import dependencies
2) Create model layers.

    The classifier will have 3 convolution layers , 3 maxpool layers , 1 flattening layer and finally an output layer with sigmoid activation.

3) Use an Image data generator for training the model.

Result and Performance

As a result of interpreting multiple latent features simultaneously the results we get are as follows

Original images

Smile and Age -ve extreme

Smile and Age +ve extreme

Pose and Gender -ve extreme

Pose and Gender +ve extreme

For fake face detection our model was able to predict 4 out of 5 faces used in testing purposes successfully. With an accuracy of about 80% on average in fake face recognition.

```
42/42 [==============================] - 16s 371ms/step - loss: 0.2313 - accuracy: 0.8970 - va
Epoch 17/50
42/42 [==============================] - 15s 370ms/step - loss: 0.2375 - accuracy: 0.9060 - va
Epoch 18/50
42/42 [==============================] - 15s 369ms/step - loss: 0.2310 - accuracy: 0.9015 - va
Epoch 19/50
42/42 [==============================] - 16s 369ms/step - loss: 0.1739 - accuracy: 0.9299 - va
Epoch 20/50
42/42 [==============================] - 15s 370ms/step - loss: 0.1414 - accuracy: 0.9455 - va
Epoch 21/50
42/42 [==============================] - 15s 369ms/step - loss: 0.1309 - accuracy: 0.9470 - va
Epoch 22/50
42/42 [==============================] - 16s 371ms/step - loss: 0.1206 - accuracy: 0.9515 - va
Epoch 23/50
42/42 [==============================] - 16s 374ms/step - loss: 0.1067 - accuracy: 0.9604 - va
Epoch 24/50
42/42 [==============================] - 16s 376ms/step - loss: 0.1118 - accuracy: 0.9530 - va
Epoch 25/50
42/42 [==============================] - 16s 374ms/step - loss: 0.0915 - accuracy: 0.9657 - va
Epoch 26/50
42/42 [==============================] - 16s 369ms/step - loss: 0.0809 - accuracy: 0.9724 - va
Epoch 27/50
42/42 [==============================] - 16s 375ms/step - loss: 0.0715 - accuracy: 0.9739 - va
Epoch 28/50
```

```
1 !python3 fake_face_testing.py
```

```
Fake Face
```


```
Epoch 19/50
42/42 [==============================] - 16s 375ms/step - loss: 0.1426 - accuracy: 0.9403 -
Epoch 20/50
42/42 [==============================] - 16s 375ms/step - loss: 0.1033 - accuracy: 0.9537 -
Epoch 21/50
42/42 [==============================] - 16s 376ms/step - loss: 0.0941 - accuracy: 0.9672 -
Epoch 22/50
42/42 [==============================] - 16s 374ms/step - loss: 0.0934 - accuracy: 0.9664 -
Epoch 23/50
42/42 [==============================] - 16s 376ms/step - loss: 0.0960 - accuracy: 0.9590 -
Epoch 24/50
42/42 [==============================] - 16s 377ms/step - loss: 0.0945 - accuracy: 0.9657 -
Epoch 25/50
42/42 [==============================] - 16s 381ms/step - loss: 0.0730 - accuracy: 0.9701 -
Epoch 26/50
42/42 [==============================] - 16s 379ms/step - loss: 0.0568 - accuracy: 0.9769 -
Epoch 27/50
42/42 [==============================] - 16s 379ms/step - loss: 0.0519 - accuracy: 0.9784 -
Epoch 28/50
```

```
[5]  1 !python3 fake_face_testing.py
```

```
Enter image path:face_pred/000618.jpg
Real Face
```

```
[ ]  1
```

Conclusions and Further work:

By leveraging the interpreted semantics as well as the proposed conditional manipulation technique, we are able to precisely control the facial attributes with any fixed GAN model, even turning unconditional GANs to controllable GANs. And we also built how to detect fake faces. We were even successful in changing multiple latent features(changing 2 facial attributes) simultaneously.

We want to improve the accuracy of fake detection model. And also there are some glitches while dealing with multiple latent features simultaneously which we are working on.

References:

1. https://medium.com/@iselagradilla94/multi-task-cascaded-convolutional-networks-mtcnn-for-fae-detection-and-facial-landmark-alignment-7c21e8007923
2. https://medium.com/analytics-vidhya/fake-face-image-classification-5a4151db9b8d
3. https://genforce.github.io/interfacegan/
4. https://arxiv.org/pdf/1907.10786.pdf
5. https://arxiv.org/pdf/2005.09635.pdf
6. https://www.kaggle.com/lamsimon/celebahq