Deep Learning Report

Zhian Chen, Jianru Xu, Annan Ouyang, Junling Tu
12211829, 12211830, 12211831, 12213010

1 Introduction

Aiming to learn, explore and make use of the model of StyleGAN, we build a program in this project, which can be used to detect faces in the camera, stylize one's face and map the stylized faces into the real face in the camera.

In order to implement all the functions above, we are supposed to use the following methods and tools. Facial detection algorithm, the flask tool for the website and the styleGAN for stylizing the faces.

2 Pipeline

This section describes the pipeline of our program. Our program has two functions.

- Map the fine-grained faces into the face in the camera.

  For the first function, we need to get the images from the camera, **1. detect the face** in the video stream. By setting a seed, we can get **2. a generated fake face by StyleGAN**, and are able to **3. adjust the features gradually**, like the hair color, beard and the extent of smile of the generated face. The modified fake face can be **4. mapped to the face** in the camera.

- Generate the stylized face from the face in the camera.

  For the second function, we need to get the images from the camera, **1. detect the face** in the video stream. Then the face would be sent to the server to **2. solve the latent vector of the face** which is needed in the stylizing process. By getting the latent vector from the server, we can use StyleGAN to **3. mixing the features by changing the specific parameters** to get the mixing-style face. The stylized face of the people in the camera can be **4. mapped to the face** in the camera.

3 Models & Theory

3.1 StyleGan & StyleGAN2

The StyleGAN can modify the fine-grained features of human faces and stylize the type of the face by the specified type.

3.1.1 Baseline

StyleGAN is mainly combined by two important parts. The first one is the mapping network, and the second one is the synthesis network.

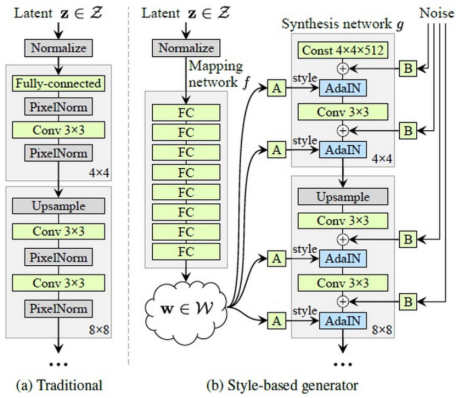

Figure 1: The architecture of the traditional network and styleGAN

For the traditional network, it uses the random noise directly and the CNN.

For StyleGAN, **the mapping network** aims to disentangle the feature vector, which helps to fine-grained control. **The synthesis network** aims to generate the image with the same style.

The Mapping Network: The mapping network in StyleGAN is a fully connected neural network that transforms the initial latent vector z into a latent space w. This transformation can be represented as: $w = f(z)$ where f represents the mapping function. The w space tends to have better disentangled properties than the z space.

To ensure diversity in generated samples, StyleGAN also applies a truncation trick where it blends each w_i with the average value $\overline{w}$: $w_i' = \psi w_i + (1-\psi)\overline{w}$ where ψ is a parameter that controls the degree of blending, and $0 \leq \psi \leq 1$.

The Synthesis Network: For generating the actual image from the transformed latent vector w, the synthesis network employs a series of convolutional layers, starting from a constant 4x4 input tensor.

Each layer's activations are modulated by the w vector through a process called adaptive instance normalization (**AdaIN**):

$$AdaIN(x_i, y) = \sigma(y)(\frac{x_i - \mu(x_i)}{\sigma(x_i)}) + \mu(y)$$

where

- x_i is the activation of the i-th layer,
- y is the style vector w,
- $\mu(x_i)$ and $\sigma(x_i)$ are the mean and standard deviation of x_i, respectively,
- $\mu(y)$ and $\sigma(y)$ are the mean and standard deviation derived from the style vector y.

Additionally, noise is injected at various layers to add stochastic details: $y = y + \mathcal{N}(0, I) \cdot \eta$ where $\mathcal{N}(0, I)$ denotes a noise map sampled from a normal distribution, and η is a learned per-channel scaling factor.

3.1.2 Improvement

There are several problems in the practice of StyleGAN, and the main one is the water drop problem.

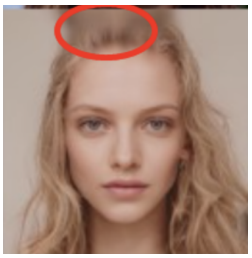

Figure 2: The water drop problem

StyleGAN2 introduces key improvements over StyleGAN, addressing issues like the "water drop" problem and enhancing image quality.

The following picture shows the architecture of StyleGAN2.

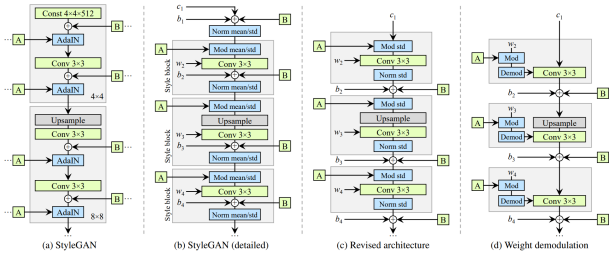

Figure 3: The architecture of the StyleGAN2

Removal of AdaIN Mean: In StyleGAN2, the mean shift in the AdaIN operation is removed. The original AdaIN used:

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

StyleGAN2 simplifies this to:

$$\text{AdaIN}(x_i, y) = \sigma(y)\frac{x_i}{\sigma(x_i)} + \mu(y)$$

This removes artifacts and sharpens the generated images.

Demodulation: A new demodulation step stabilizes training and improves image quality. In StyleGAN, scaling was controlled directly by $\sigma(y)$. In StyleGAN2, the modulation is followed by demodulation:

$$\text{Modulate}(x_i, w) = \sigma(w) \cdot \frac{x_i - \mu(x_i)}{\sigma(x_i)}$$

$$\text{Demodulate}(x_i, w) = \frac{x_i - \mu(x_i)}{\sigma(x_i)} \cdot \eta$$

where η is a learned scaling factor.

Noise Injection Position: In StyleGAN2, noise is injected earlier in the synthesis network, improving fine-grained details and reducing artifacts. The noise injection is:

$$y = y + \mathcal{N}(0, I) \cdot \eta$$

These changes lead to higher-quality, more stable image generation.

3.1.3 Project Process of Face

We can use a random seed to generate a random noise, which can get the random latent vector, but what if we want to get the latent vector of one's face, or project the face into the latent space. How can we achieve that?

Actually, we can define it as an optimization problem, that is, we can train a latent vector to get closer to the real latent vector of the face wanted.

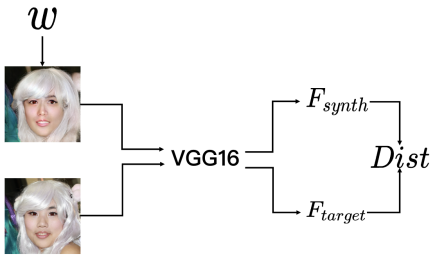

Figure 4: The process of project

- 1. Generate a image from the initial latent vector.
- 2. Use **VGG16** to extract the features of the generated face and the target face
- 3. Set the distance of the two faces as the loss function

$$\text{dist} = \sum (F_{\text{target}} - F_{\text{synth}})^2$$

- 4. Do the gradient descent to get a better latent vector. By training for many epochs, we can get a great latent vector of the face in the latent space.

3.2 Facial Detection

Our project requires a facial detection system, mainly for the following reasons:

- The real-world face should be replaced by the images generated by the Style Gan, so we need to get the location of people's faces.
- To achieve the latent inversion, Style Gan first needs to get the face image to be inverted. But the face image is not always in the center of the image. To remove the useless information like the background, we need to crop the face image.

We tried many facial detection models, including 'NVIDIA Jetson-Inference(TAO FaceDetect, facenet-120)', 'YOLO(v5 and v11)', 'YuNet'. We finially choose the 'YuNet' model, which is a lightweight facial detection model, developed by the team of professor 'Shiqi Yu'.

It stands out due to its exceptional balance between accuracy and efficiency. Unlike heavier models that necessitate GPU acceleration, YuNet's lightweight architecture allows it to run efficiently on CPUs. This was a crucial factor in our project, given the need to reserve GPU resources for the StyleGAN model.

YuNet achieves its high performance and efficiency through a carefully designed architecture that combines the following key elements:

- Lightweight Convolutional Neural Networks (CNNs): YuNet employs specially structured,

computationally efficient CNNs. These networks are designed with a reduced number of parameters and operations, making them faster to execute without sacrificing accuracy. The architecture focuses on capturing essential facial features while minimizing redundancy.

- Feature Pyramid Networks (FPN): YuNet also incorporates a Feature Pyramid Network, which enables it to detect faces of different sizes. This allows the model to capture multi-scale features, which is essential for facial detection where the faces can vary significantly in size within the input image.

- Context Aggregation: YuNet leverages a mechanism to aggregate global context, improving its performance in challenging scenarios where faces might be partially occluded or in non-ideal conditions. This context understanding helps enhance the accuracy of face localization.

By using these features, YuNet delivers high accuracy at a fraction of the computational cost of many other face detection models, making it an ideal choice for resource-constrained environments and applications.

4 Training

This section describes the our approach of fine-tuning StyleGAN2 on FFHQ and CALFW.

4.1 Background

StyleGAN2 provides its pre-trained model on FFHQ datasets, named ffhq-res256-mirror-paper256-noaug. However, as its name suggests, it's trained on FFHQ datasets without setting aug to 'ada'. The core mechanism of ADA is to dynamically apply augmentation operations to the input data, thereby enhancing the discriminator's generalization ability while preventing the generator from relying on the discriminator's incorrect judgments of augmented samples. It is the key innovation of StyleGAN2-ADA.

Figure 5: Images randomly generated by the base model of 256x256 resolution.

Just as shown in the above image, the images generated by the base model are not realistic enough. They have following problems:

- **Distorted.** The generated face is not clear enough especially when there are multiple faces in the image.

- **Blurred.** The model performs poorly when it needs to generate hands.

- **Unrecognizable.** Sometimes the generated face is not a face at all, which is full of noise or lines.

To solve these problems, we need to fine-tune the model on FFHQ datasets with ADA.

4.2 Fine-tuning

First, we need to prepare the FFHQ dataset, which contains 70,000 high-quality images of human faces. The dataset is well-aligned and cropped, which is suitable for training StyleGAN2. We use the dataset_tool provided by StyleGAN2 to convert the dataset to the .zip format and 256x256 resolution.

The training consists of two phases. All the training samples are preprocessed by the ADA mechanism. The batch size is 128. The learning rate is 0.002 for stage1 and 0.001 for stage2, with default Adam Optimizer. The fmaps is set to 0.5, ema is set to 20, map is set to 8 and gamma is set to 0.8. The training takes about 25 hours for stage1 of 3000k images and 5 hours for stage2 of 500k images on a single NVIDIA A100 80GB PCIe.

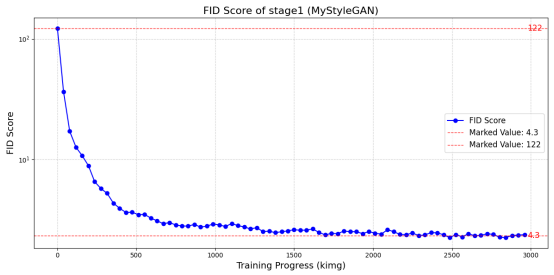

Figure 6: FID decreased from 122 at the start of training to around 4.2.

After stage2 training, the FID score finally reaches around 3.9, which is a significant improvement compared to the base model. The generated images are more realistic and clear. We also test our training script on CALFW datasets, which contains 80000 images of some historical figures. However, because of the low definition of the images, the FID score maintains around 10.0 and the generated images are not as clear as the FFHQ datasets.

5 Implementation

This section describes our program's implementation details and the reasoning and findings behind our decisions.

We implemented the project in Python, running both two models on the same machine at the same time. The StyleGAN2 model is nvidia's official pytorch implementation deployed on the GPU, while the YuNet model is deployed on the CPU. The web server is implemented using the Flask framework.

5.1 The layers modified in w

To change the specific features of the human face, we need to find the corresponding latent space to the features. By trying every combination of changing the continuous layers, we can see the difference of the various choices.

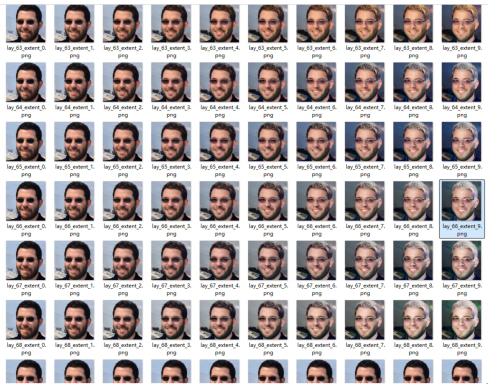

Figure 7: The various choices of layers

For example, the above figure shows several combination:

- choice 63: layer 6-9
- choice 64: layer 6-10
- choice 65: layer 6-11
- choice 66: layer 6-12
- choice 67: layer 6-13
- choice 68: layer 6-14

which layer 6-9 means modifying the layer 6-9 of the latent vector w of the target image in the 6-9 layer to the style image.

By doing this, we find several important choices to modify specific features.

Features	Layers
The Overall style (hair & skin color)	layer 7-14
The hair color	layer 6-13
The beard	layer 4-14
The smile	layer 3-5

Table 1: Caption

There are some demos for modifying the features.

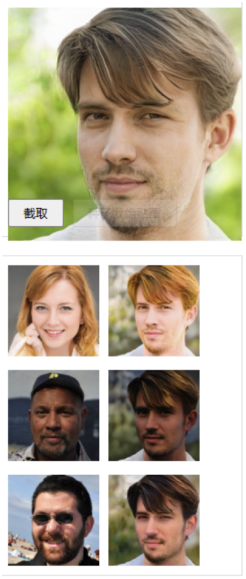

Figure 8: The Overall style

Figure 9: The combination of hair, beard and smile

5.2 High Performance And Resource Utilization

On Nano, running both two models and the web server at the same time, the facial detection's frame rate is still around 20 FPS, and the StyleGAN2 model's frame rate is around 3 FPS. The web server updates the generated image every face detection frame, so the experience is smooth and real-time. Both CPU and GPU are fully utilized reaching the limitation of the Nano. And the memory usage is approaching 4GB, which just avoids using the slow swap memory.

5.3 Why YuNet

As mentioned in the Introduction, we chose YuNet as our facial detection model due to its lightweight architecture and high performance even on CPUs. Actually, we initially tried other models like YOLO and NVIDIA Jetson-Inference and did most graphics processing like capturing and cropping images by using low-level cuda APIs provided by jetson-utils.

However, after deploying the StyleGAN2 model on the GPU, we found that the GPU was overloaded and even a small task like cropping images took a long time, making even the facial detection process drop to less than 1 FPS. This may be due to the scheduling mechanism of the GPU and its expensive context switching cost. Therefore, we have to move the facial detection process and all the graphics processing to the CPU.

Therefore, we need a lightweight facial detection model that can run efficiently on CPUs. YuNet is the best choice for us.

5.4 Smooth Experience

Our smooth experience can be roughly attributed to the following two tricky techniques.

5.4.1 Web Server

The main reasons for choosing web server instead of simply using cv2.imshow() are the heavy load of Ubuntu's GUI and inconvenience of using the screen. The StyleGAN2 model is heavy enough and we don't want to add more load to the GPU, especially the almighty, resource-gobbling Ubuntu GUI, GNOME. And the screen's numerous cluttered cables would ruin everything. By using the web server, we can easily work on the project and demonstrate it everywhere.

5.4.2 Multi-threading

There are four main parts in our program: web server, video capture, facial detection, and StyleGAN2. The last three parts each run in a separate thread. The video capture runs in a separate thread to avoid getting outdating frames led by the

producer-consumer architecture of gstreamer which heavily impacts the user experience. And we use many buffers and conditional variables to make sure efficient communication between threads and not to waste resources when current threads have nothing to do.

5.4.3 Update Image Each Face Detection

The facial detection runs much faster. So once a picture is processed by the facial detection, it will call the web server to update the generated image. Therefore, the web server can update the generated smooth image just like a video stream.

5.5 "Excellent" Community Support

ARM, CUDA, Jetpack, combine all these words, and you'll get a whirlwind of an unstoppable community—one so mighty that no amount of effort could ever hope to overcome it. Truly, a force of nature. Prepare yourself for the elite adventures of unearthing cryptic links from the shadowy depths of Dockerfiles and heroically taming otherworldly dependencies during manual compilation. A privilege reserved for the brave.

6 Future Work

Although we have achieved an outstanding performance and efficiency, there are still some optimizations and improvements that can be made in the future.

6.1 TensorRT

TensorRT is a high-performance deep learning inference library developed by NVIDIA. It optimizes the model for inference, reducing the model's size and increasing the inference speed. We can use TensorRT to optimize the StyleGAN2 model, which can further improve the performance and efficiency of the model. In this project, we have not used TensorRT due to the complexity of the model and the limited time.

6.2 Project Image to Latent Space

In this project, we have only deployed the project on another more powerful machine. This is because the projection process is too slow to run on the Nano. Even on the powerful machine, the projection process takes about 50 seconds for each image. We can further optimize the projection process to make it faster and more efficient and eventually deploy all the models on the Nano with no need for another machine.

6.3 Find Certain Traits

We have the function to adjust a certain trait of the generated image. This is done by adjusting the corresponding value in the latent space. However, the process finding the corresponding value is done by manually adjusting the value and observing the change in the generated image over and over again. And the process is not efficient and not accurate. We can use some optimization algorithms to find the corresponding value more efficiently and accurately.