Pusan National University
Computer Science and Engineering
Technical Report 2024-10

생성형 모델을 활용한 AI 헤어스타일러

201823132 박시형

201924609 한지훈

202155626 홍진욱

지도교수 전상률

목 차

1. 서론

1.1 연구 배경

현대 사회에서 개인의 외모는 자신감과 사회적 인상을 형성하는 데 중요한 역할을 한다. 그중에서도 헤어스타일은 개인의 외모 변화에 크게 기여하는 요소로, 적합한 헤어스타일을 선택하는 것은 이미지를 긍정적으로 변화시킬 수 있는 중요한 방법이다. 그러나 많은 사람이 실제로 헤어스타일을 변경하기 전에 시뮬레이션할 수 있는 효과적인 도구가 부족해, 예상치 못한 결과 때문에 미용실에서의 실패를 경험하는 경우가 흔하다.

기존의 헤어스타일 시뮬레이션 서비스는 제한된 스타일 선택지를 제공하고, 단순한 이미지 위에 헤어스타일을 덧씌우는 방식으로 실용성이 떨어진다. 또한, 이러한 서비스는 라이브 스트리밍을 지원하지 않으며, 텍스처나 스타일의 세밀한 부분이 재현되지 않아 사용자 경험이 제한적이다. 이 문제를 해결하기 위해 실시간으로 다양한 헤어스타일을 시도해 볼 수 있는 생성형 AI 모델이 필요하게 되었다.

처음에는 헤어스타일 변환 모델을 선정하여 스트리밍 형태의 실시간 서비스를 개발하는 것을 목표로 했으나, 연구 과정에서 발견된 기술적 및 시간적 한계로 인해 경량화가 어려운 문제에 직면했다. 특히 GAN(Generative Adversarial Network) 기반 헤어스타일 변환 모델은 단일 모델이 아니라 얼굴 인식, 정렬, 회전, 헤어 추출 등의 여러 작업이 결합한 복잡한 구조로 되어 있어, 모든 단계에서의 경량화가 어려운 점이 큰 한계였다.

이에 따라 연구 방향을 수정하여, 모델을 API로 구현하고 FastAPI를 통해 모델 서빙 시간을 단축하는 방향으로 변경하였다. 이러한 수정된 방향은 사용자의 요청에 더 빠르게 대응할 수 있으며, 더 나아가 AI를 서비스에 도입할 때 비용 효율성을 극대화하는 방법에 관한 연구로 이어지게 되었다.

1.2 연구 목표

본 프로젝트의 주된 목표는 복잡한 GAN 기반 헤어스타일 변환 모델을 API 화하고, FastAPI와 같은 기술을 통해 모델의 서빙 시간을 최적화하는 것이다. 이를 통해 사용자가 다양한 헤어스타일을 빠르게 시도해 볼 수 있는 환경을 구축하며, 기존 모델을 단순히 로컬에서 사용할 때보다 신속한 서비스 서빙을 통한 사용자 경험을 향상시키는 것을 목표로 한다.

기존 목표였던 실시간 스트리밍 형태의 변환 모델 구현은 경량화의 한계로 인해 어려움이 있었으나, 수정된 목표에서는 모델을 API 형태로 변환하고, 서비스 서빙 시간을 줄이는 방법을 적용하여 최적화하고자 한다. 이를 통해 이미지 업로드 및 전처리 과정을 최적화하고, 최종적으로 사용자가 더 빠르고 안정적인 결과를 제공하는 것을 목표로 설정하였다.

또한, 연구 과정에서 추가로 모델의 경량화 기법을 적용하고, AI 서비스를 도입하는 데 있어 비용 효율적인 방법을 연구하여, 더 나은 경제적 성과를 도출할 수 있도록 한다. 이를 통해 실제 서비스로 적용할 수 있는 수준의 기술적 완성도를 높이고, 사용자 친화적인 인터페이스를 제공하는 것을 최종 목표로 한다.

2. 연구 내용

2.1 개발 환경 및 언어

a. Host Machine:

- Host OS: Ubuntu 2.04
- Host CPU : AMD EPYC 7643 48-Core Processor
- Host GPU : A5000

b. Docker:

- 실험에 사용될 다양한 모델에 따라서 적합한 Docker image를 사용한다.
- Docker image: pytorch/pytorch 2.3.0-cuda11.8-cudnn8-devel)

c. Language:

- Programming Language : Python
- DL FrameWork : PyTorch

2.2 사용 기술

2.2.1 서비스 구현

a. Diffusion Model:

Diffusion Model은 이미지 생성 및 변환에서 매우 중요한 역할을 하는 생성형 모델(Generative Model)이다. 이 모델은 Markov Process를 기반으로 하여 점진적으로 데이터에 노이즈를 추가하고, 이를 역으로 복원하는 과정을 통해 데이터를 생성한다. 일반적으로 이미지 생성 과정에서, 처음에는 노이즈가 가득한 상태에서 시작해 이를 단계적으로 정제해 가며, 마지막에는 고품질의 이미지를 얻는 방식이다.

Diffusion Model의 핵심 원리는 "Forward Process"와 "Reverse Process"로 나뉜다.

Forward Process에서는 점차 이미지에 노이즈를 추가하며, 이는 데이터를 점점 더 불확실하게 만들어 결국 노이즈로만 구성된 상태로 변환된다. Reverse Process에서는 이 과정을 반대로 적용하여, 노이즈 상태에서 시작해 원래 이미지 또는 새롭게 생성된 이미지를 복원해 낸다. 이 방식은 이미지의 디테일을 세밀하게 처리하는 데 뛰어나며, 특히 고해상도의 이미지 생성에서 탁월한 성능을 발휘한다.

Diffusion Model은 다단계 과정에서 많은 계산량이 필요하므로 연산 시간이 길고, 실시간 처리에는 적합하지 않다. 하지만 그 대신 매우 자연스럽고 현실감 있는 이미지를 생성할 수 있기 때문에, 창의적인 이미지 생성이나 예술적인 작업에서 많이 사용된다.

본 연구에서는 헤어스타일을 변환한 사람의 새로운 얼굴 이미지를 생성하므로, Diffusion Model을 후보 모델 군으로 선정하되, 추론 시간을 고려하여 최종 모델을 선정한다.

b. GAN(Generative Adversarial Networks):

GAN(Generative Adversarial Networks)은 두 개의 신경망(Neural Networks)이 경쟁하면서 학습을 진행하는 구조로 이루어져 있다. 이 모델에서 하나는 Generator, 다른 하나는 Discriminator로 구성된다. Generator는 실제와 유사한 가짜 데이터를 생성하려고 하고, Discriminator는 입력된 데이터가 실제 데이터인지, 혹은 Generator가 생성한 가짜 데이터인지를 구별하는 역할을 한다.

GAN의 학습 과정에서 Generator는 랜덤한 노이즈로부터 데이터를 생성하여 Discriminator를 속이려 한다. 반면, Discriminator는 실제 데이터와 가짜 데이터를 구별하려고 하면서 두 네트워크 간에 경쟁이 발생한다. 이 과정에서 Generator는 점점 더 실제에 가까운 데이터를 생성하게 되며, Discriminator는 가짜 데이터를 더 정교하게 구별하게 된다. 이러한 경쟁 관계는 GAN이 학습을 통해 점점 더 뛰어난 성능을 발휘하게 하는 원동력이 된다.

GAN의 주요 장점은 고해상도 이미지 생성에 매우 효과적이며, 특히 Style Transfer, Super-Resolution, Image-to-Image Translation 등의 작업에 자주 사용된

다는 점이다. 그러나 GAN은 학습이 불안정할 수 있으며, Mode Collapse와 같은 문제가 발생할 수 있다.

다만, 헤어스타일 변환 서비스를 스트리밍 형태의 실시간 서비스 제공이 본 연구의 기존 목표라는 점을 고려하여, Diffusion Model과 비교했을 때, GAN의 연산 과정 및 추론 시간이 짧다는 점을 활용하고자 한다. 따라서, 본 연구에서 사용하는 생성형 모델은 GAN 기반의 모델을 선정하고, 기존에 제시된 다양한 모델 중에서 적절한 모델을 선정하여 사용한다.

c. Streamlit

Streamlit은 데이터 과학 및 머신러닝 애플리케이션을 구축하는 데 특화된 Python 기반의 오픈소스 프레임워크로, 간단한 Python 코드만으로도 웹 애플리케이션을 빠르게 만들 수 있다. 일반적인 웹 개발 도구들은 복잡한 설정이나 코드 구조를 요구하지만, Streamlit은 그와 달리 데이터 시각화 및 모델 인터페이스를 손쉽게 구축할 수 있다는 점에서 매우 혁신적이다. 이는 특히 데이터 분석가나 머신러닝 연구자들이 프로토타이핑이나 결과 시각화를 즉시 확인할 수 있는 환경을 제공한다.

Streamlit의 작동 방식은 매우 직관적이다. 파이썬 코드를 작성하고 애플리케이션이 그 코드에 따라 실시간으로 업데이트되는 구조로 되어 있어, 사용자는 코드 변경 사항을 즉각적으로 반영할 수 있다. 예를 들어, 데이터셋을 로드하고 이를 시각화하거나, 사용자 입력받아 모델 예측 결과를 실시간으로 보여주는 등의 작업이 Streamlit에서는 매우 쉽게 구현된다.

Streamlit의 가장 큰 장점은 사용자가 별도의 웹 개발 지식 없이도 데이터 기반 애플리케이션을 구축할 수 있다는 점이다. 버튼 클릭, 슬라이더, 입력 상자 등과 같은 다양한 위젯을 통해 사용자와의 상호작용을 지원하며, 이를 통해 인터랙티브한 대화형 애플리케이션을 제작할 수 있다. Streamlit은 기본적으로 실시간 처리가 가능하므로 인공지능 모델의 결과를 시각적으로 확인하고 피드백을 받을 수 있는 환경을 제공하는 데 최적화되어 있다.

따라서 본 연구에서는 Streamlit을 사용한 실시간 처리를 통해 생성형 모델을 이용

한 서비스 서빙 시간 단축을 추구한다.

d. FastAPI

FastAPI는 Python 기반의 고성능 웹 프레임워크로, 비동기 프로그래밍(Asynchronous Programming)을 지원하며, 매우 빠르고 효율적으로 API를 개발할 수 있도록 설계되었다. FastAPI는 RESTful API를 구축하는 데 최적화되어 있으며, 특히 비동기 I/O 처리로 인해 많은 요청을 효율적으로 처리해야 하는 대규모 시스템에서 높은 성능을 발휘한다.

FastAPI의 가장 큰 특징 중 하나는 Python Type Hint를 적극적으로 활용하여 코드의 가독성을 높이고, 자동으로 데이터 검증과 문서화를 지원한다는 점이다. 개발자는 명시적으로 데이터 타입을 정의하고, FastAPI는 이를 바탕으로 자동으로 입력 데이터의 유효성을 검증한다. 또한 FastAPI는 OpenAPI 및 JSON Schema와 호환되어, 자동으로 API 문서를 생성해주는 기능을 제공하여 개발자가 API를 더 쉽게 관리하고 테스트할 수 있다.

FastAPI는 비동기 요청을 처리하는 데 매우 유리하며, 특히 실시간 데이터 처리, 대규모 트래픽을 다루는 API 서버, 인공지능 모델 서빙 등의 작업에서 매우 효과적이다. 따라서 본 연구의 GAN 기반의 모델 서빙 과정에서 기존 모델을 API로 호출하여 로컬에서의 모델 추론 시간보다, WEB에서 사용자가 서비스를 사용할 때 서빙 시간을 단축하여 사용자 경험을 개선한다.

2.2.2 AI 모델 경량화

a. ONNX(Open Neural Network Exchange):

ONNX는 모델의 이식성과 확장성을 높이기 위한 표준 포맷으로, 이를 사용하면 AI 모델이 특정 하드웨어 또는 플랫폼에 종속되지 않고 여러 환경에서 실행될 수 있다. 이는 AI 서비스 운영에 있어 하드웨어 업그레이드나 환경 변화에 대응하는 유연성을 제공한다. 특히 클라우드 환경에서 ONNX는 다양한 인프라 자원(CPU, GPU, TPU 등)에 최적화된 형태로 모델을 실행할 수 있게 하여, 특정 환경에서의 성능 병목을 피할 수 있다. 또한, ONNX는 다양한 프레임워크에서 모델을 쉽게 교환할 수 있도록 해, 개발과 배포 과정에서의 비용을 절감하고, 유지보수에 드는 시간과 노력을 최소화할 수 있다.

또한, ONNX는 모델 최적화 도구와 결합할 수 있다. ONNX Runtime은 실행 성능을 극대화하기 위한 런타임 엔진으로, 다양한 최적화 기법(모델 그래프 최적화, Fusion 등)을 사용하여 추론 속도를 높이고 메모리 사용량을 줄인다. 이러한 최적화 덕분에 ONNX는 성능 요구 사항이 높은 AI 서비스에서도 안정적인 추론 성능을 제공할 수 있다. 이러한 최적화 작업은 클라우드 인프라 비용을 줄이고, 더 많은 사용자를 수용하면서도 운영 비용을 줄이는 데 중요한 역할을 한다.

b. Quantization:

대표적인 인공지능 경량화 기법의 하나인 Quantization의 핵심은 모델의 연산을 비교적 낮은 정밀도로 수행하여 계산 효율성을 극대화하는 것이다. Quantization은 모델이 실행되는 환경에 따라 다양한 방식으로 적용될 수 있다. 정수(8-bit integer) 기반의 Quantization은 일반적으로 32-bit floating point에 비해 연산 비용을 크게 줄일 수 있으며, 연산속도 향상과 메모리 사용량 절감이라는 두 가지 주요 이점을 제공한다. 이러한 정수 기반 Quantization은 모바일 및 엣지 디바이스에서 인공지능 서비스를 실행할 때 매우 중요한 역할을 한다. 예를 들어, 대규모 데이터 처리 또는 실시간 응답이 중요한 서비스에서는 Quantization을 통해 서비스 성능을 크게 향상시킬 수 있다.

또한 Quantization을 사용하면 모델의 전력 및 비용 효율성 측면에서 개선할 수 있다. 인공지능 모델을 8-bit 정수로 Quantization 하면, 연산 자체가 훨씬 적은 전력으로 가능해져 에너지 소모를 줄일 수 있다. 이에 따라 대규모 데이터 센터나 클라우드 기반의 인공지능 서비스에서 에너지 비용을 크게 절감할 수 있다. Quantization은 특히 지속 가능한 인공지능 서비스 운영을 목표로 하는 기업에서 중요한 기술로 자리 잡고 있으며, 이를 통해 탄소 배출을 줄이고, 친환경적인 인공지능 서비스를 구축할 수 있다.

c. PTQ(Post-Training Quantization):

PTQ는 인공지능 모델을 학습한 후 적용되는 최적화 방법으로, 추가적인 훈련 과정 없이 모델을 경량화할 수 있는 빠르고 효율적인 방법이다. PTQ는 모델의 정밀도를 낮추면서도 주요한 학습 결과를 유지할 수 있어, 모델 크기와 추론 속도를 동시에 개선할 수 있다. 특히 사전 훈련된 모델에 바로 적용할 수 있기 때문에, 인공지능 서비스 배포 시 즉각적인 성능 개선과 비용 절감 효과를 얻을 수 있다. 이를 통해 개발자는 서비스 배포 전에 복잡한 재훈련 과정을 거치지 않고도 모델을 최적화할 수 있어, 빠르게 대규모 인공지능 서비스를 확장할 수 있다.

PTQ는 하드웨어 가속기와 결합하여 더 큰 성능 향상을 이룰 수 있다. 예를 들어, TPU와 같은 특화된 하드웨어 가속기를 사용할 때, PTQ를 적용한 모델은 더 적은 연산 비용으로 높은 성능을 제공하며, 이는 클라우드 기반의 인공지능 서비스에서 특히 유용하다. 클라우드 서비스 제공자는 PTQ를 통해 더 많은 인공지능 모델을 한꺼번에 처리할 수 있으며, 이는 궁극적으로 비용 효율적인 클라우드 사용을 가능하게 한다.

d. QAT(Quantization-Aware Training):

QAT는 모델 훈련 중에 Quantization을 반영하여 모델을 최적화하는 방식으로, 모델이 8-bit 정수 연산에 익숙해지도록 학습하는 과정을 포함한다. 이로 인해 Quantization 후에도 모델 성능 저하가 거의 없으며, 고성능을 요구하는 인공지능 서비스에서 QAT는 매우 유용하다. 특히 실시간 성능이 중요한 서비스나 높은 정확

도가 요구되는 응용 프로그램에서 QAT를 사용하면, 비용 절감과 성능 유지의 두 가지 목표를 동시에 달성할 수 있다. 이는 모바일 기기나 엣지 컴퓨팅과 같은 제한된 자원 환경에서도 고성능 인공지능 기술을 실현하는 데 필수적인 기술이다.

QAT는 특히 딥러닝 모델의 복잡한 아키텍처에서도 유용하게 사용된다. Transformer, BERT와 같은 대규모 모델에서도 QAT를 적용하면 성능 저하 없이 모델의 효율성을 극대화할 수 있다. QAT는 모델이 Quantization 된 연산을 학습 과정에서 미리 경험하기 때문에, 모델은 실제로 Quantization이 적용된 환경에서 원래의 성능을 유지할 수 있다. 이는 대규모 언어 모델이나 이미지 생성 모델 등에서도 중요한 역할을 하며, 고성능 인공지능 모델을 실시간으로 제공할 때 필수적인 기술이다.

2.3 GAN 기반 생성형 모델

2.3.1 모델 선정

본 연구에서는 최근 성능이 개선된 Diffusion을 기반으로 하는 생성형 모델보다는 GAN을 채택하여 생성 시간 단축을 통해 추론 및 서비스 서빙 시간을 단축한다. 대표적인 헤어스타일 변환 GAN 기반 생성형 모델은 다음과 같다.

1) MichiGAN:

MichiGAN은 헤어의 형태, 구조, 외관, 배경을 포함한 주요 시각적 요소를 명확히 구분하고, 사용자 입력을 다양한 조건 모듈을 통해 이미지 생성 파이프라인에 통합하여 사용자가 직관적으로 헤어스타일을 조작할 수 있도록 설계되었다. MichiGAN은 복잡한 헤어의 기하학적 특성과 외관을 다루는 데 있어서 뛰어난 사용자 제어력과 성능을 제공한다.

2) Barbershop:

Barbershop은 GAN 기반 모델로, 다중 이미지의 특징을 하나의 일관된 이미지로 유연하게 결합할 수 있는 새로운 latent space과 GAN-embedding 알고리즘을 제공한다. Barbershop은 세부 사항을 보존하고 공간 정보를 인코딩하는 데 효과적이다.

3) InterfaceGAN:

InterFaceGAN은 GAN의 latent space에 인코딩된 다양한 의미를 해석하여 픽셀 단위로 얼굴 편집을 가능하게 한다. Linear transform 후에 분리된 표현을 학습하여, 다양한 얼굴 속성을 더욱 정밀하게 조절할 수 있도록 한다.

기존에 착수보고서에서 채택한 GAN 모델 후보군을 이용해서 추론 실험을 통해 연구 목표에 상응하는 생성된 이미지의 질감 및 자연스러운 정도 등과 같은 주관적 척도로 평가했을 때는 큰 차이가 없었지만, 위의 후보군 모델들에는 아래와 같은 기존의 연구 목표와 상충하는 문제가 있다.

a. pretrained_model의 제공 형태

초기 단계에서 선정한 GAN 헤어스타일 변환 모델 후보군의 pretrained_model의

제공 형태가 .npz 형태로, 이는 본 연구에서 주된 Task로 진행할 경량화 기법인 quantization과 ONNX 모델 변환과정에서 복잡하고 불필요하게 많은 사전 작업을 필요로 한다.

제공되는 .npz 파일의 pretrained model을 ONNX로 변환하기 위해서는, 기존 작업 환경인 PyTorch에서는 직접적인 툴이 제공되지 않아서, TensorFlow로 변환한 후에 경량화 작업을 진행하거나, 직접 .npz 파일의 weights 들을 읽어와서 새롭게 모델의 아키텍처에 삽입한 후에 변환과정을 진행하여야 했다.

위와 같은 불필요한 작업을 거치지 않기 위해서 헤어스타일 변환과 관련된 다른 GAN 모델 **(HairFastGAN)**을 선정하여 본 연구를 진행하기로 하였다. 위의 HairFastGAN 모델의 경우 pretrained model을 '.pt' 혹은 '.pth' 형태로 제공하기 때문에 위와 같은 불필요한 작업이 필요 없고, 동일한 추론 실험을 진행하였을 때도 좋은 성능을 보였기 때문에 해당 모델을 선정하였다.

b. 모델 추론 시간

초기 연구 설계 단계에서는 사용자의 얼굴을 웹캠 등을 통해 입력받아 모델을 통해 헤어스타일을 변환하고 이를 라이브 스트리밍 형태로 서비스를 제공하고자 하였다. 하지만, 기존 GAN 모델을 통해 실험해본 결과, 이미지 전처리 과정 없이 얼굴 부분만을 입력하고 통해 추론을 진행했을 때, 대략 2~3초 정도의 시간이 소요된다는 점을 확인하였다. 따라서 기존 계획과 같이 라이브 스트리밍 형태가 아닌, 캡처된 이미지를 이용해서 추론 및 서비스 서빙 시간을 최대한 단축하는 방향으로 연구의 주된 목표를 수정하였다.

이전의 모델 추론 실험은 아직 quantization이나 ONNX 모델 변환과 같은 경량화가 적용되지 않은 결과이기 때문에, 충분히 라이브 스트리밍을 위한 시간까지 추론 시간을 단축할 수 있지만, 단축된 시간은 아직 얼굴 부분만을 잘라내거나, 얼굴 정렬 등과 같은 여러 입력에 대한 전처리 과정에 걸리는 시간과 상쇄된다는 점을 고려해서 먼저 위와 같이 연구 목표를 변경하고, 추가로 서비스에서 인공지능 기술 및 모델을 사용하기 위한 비용 효율적인 방법에 관해 연구한다.

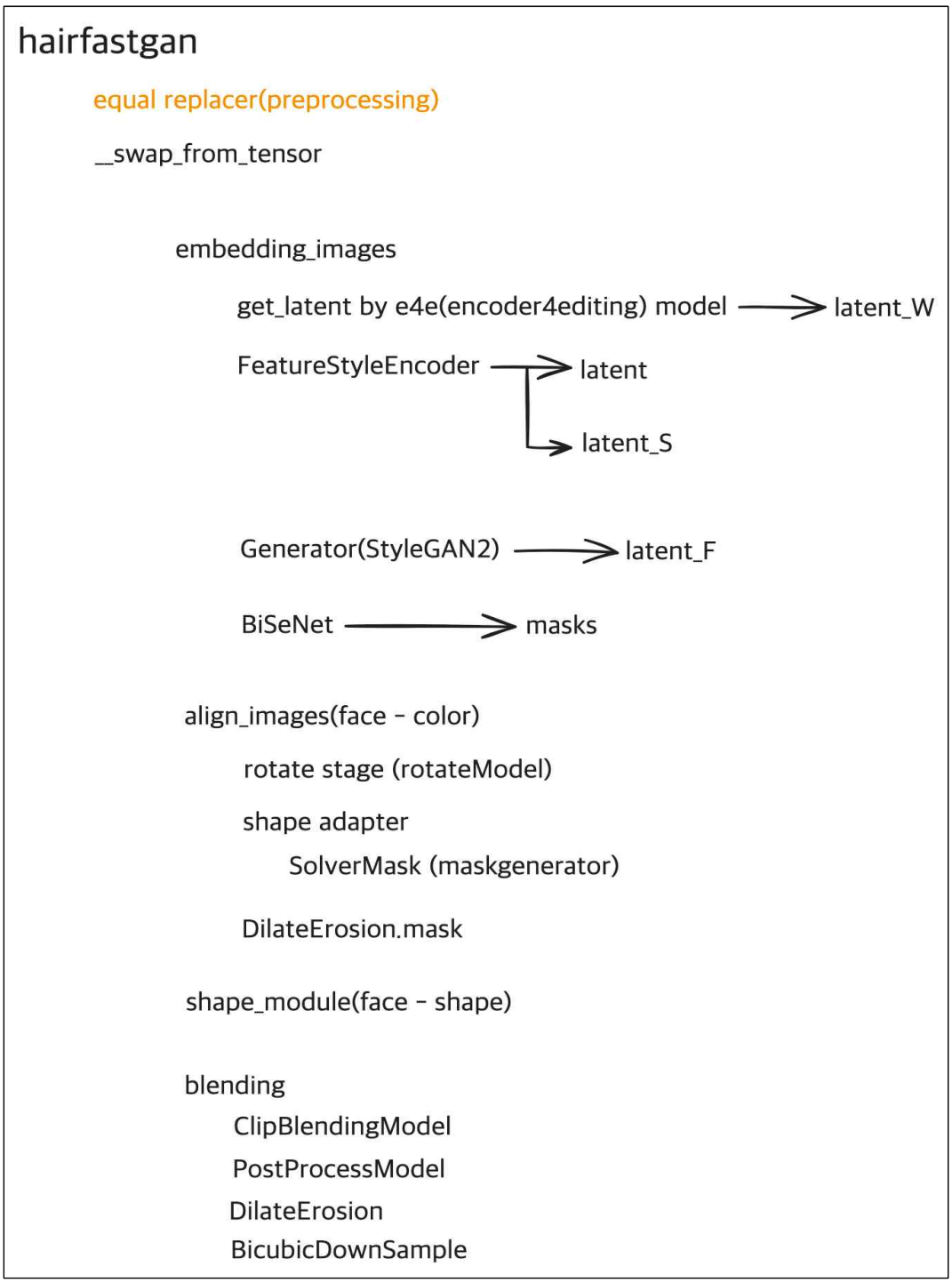
2.3.2 모델 경량화

기존 연구 목표 및 방향성은 헤어스타일 변환을 목적으로 개발된 GAN 기반으로 하는 생성형 모델을 선정하여, 다양한 인공지능 모델 경량화 기법을 사용하여 사용자가 이미지 형태가 아닌, 라이브 스트리밍 형태로 실시간으로 변환된 생성 결과를 확인할 수 있는 서비스 개발이다.

다만, 연구 진행 과정에서 분석한 결과 헤어스타일 변환 서비스는 단순히 하나의 GAN 모델을 사용하는 것이 아니라, 이미지에서 얼굴 부분을 감지하여 crop하고, 변환할 헤어스타일 이미지의 회전 각도와 위치에 알맞게 rotation & alignment의 전처리를 위한 별도의 모델이 사용된다. 자세한 모델 분석 결과 및 이에 따른 연구 방향성 수정에 관한 내용은 아래와 같다.

2.3.3 연구 방향성 수정

a. HairFastGAN 모델 분석 결과



[그림 1] HairFastGAN 모델 분석 결과

중간보고서에서 작성한 내용을 토대로 선정한 "HairFastGAN" 모델의 경량화 작업을 위해 모델 구조를 분석한 결과는 위와 같다. 헤어스타일 변환만을 위한 하나의 GAN 모델이 아니라, 입력된 각각의 3개의 이미지 (Input, HairStyle, HairColor)에 대해 전처리를 위한 다양한 모델을 추가로 사용하였고, 각각의 모델에서 일부 Layer의 결과만을 사용한다.

기존 연구 설계 단계에 따라 연구 목표를 달성하기 위해서는 선정한 HairFastGAN 모델에서 사용된 중간 단계 모델들 전부에 대해 ONNX 변환 및 Quantization 경량화 기법을 적용하고, 이 중에서 기존 모델에서 채택한 HairFastGAN 모델에서 사용하는 일부 Layer의 결과를 호출하는 부분을 전부 수정하여야 한다.

따라서, 연구 기간의 한계를 고려하여 목표와 방향성을 아래와 같이 수정한다.

b. 연구 방향성 수정

기존에 라이브 스트리밍 영상 형태의 실시간 변환 서비스를, 사용자 CAM을 통해 입력받은 이미지에 대해 원하는 헤어스타일 및 색상을 변환하는 서비스를 제공한다. 이 과정에서 로컬에서 CLI를 통해 모델을 호출하여 추론을 진행할 때보다, 모델 서빙 및 추론 시간 단축을 목표로 하는 서비스를 개발하는 방향으로 연구 목표를 수정한다.

다만, 기존 연구 목표에 해당하는 인공지능 모델의 경량화 기법을, 서비스 개발을 위해 적용하는 것이 아니라, 서비스에서 인공지능 기술 및 모델을 사용하기 위한 비용 효율적인 방법에 관해 연구하는 방향으로 수정하며, 이에 따라 대표적인 경량화 기법을 적용하여 기존의 모델과 성능을 비교한다.

2.4 모델 서빙 시간 단축

수정된 연구 목표 및 방향성에 따라서, 기존의 헤어스타일 변환 GAN 모델을 서비스로 서빙하는 과정의 시간을 단축하여 사용자의 경험을 개선하기 위해 아래와 같은 방안을 적용한다.

2.4.1 Streamlit

FastAPI를 통해 원격 GPU에 백엔드 서버를 구축하고, 사용자로부터 웹캠 혹은 업로드된 이미지를 입력받고, Streamlit에 업로드된 Shape & Color 후보군에서 원하는 헤어스타일을 사용자가 선택하면, 백엔드 서버에서 모델을 통해 변환된 결과 이미지를 웹 페이지에 출력한다.

사용자의 입력 이미지는 PIL을 통해 백엔드 서버에 byte 형태로 저장하지만, 사용자가 원하는 Shape 및 Color를 선택하는 과정에서, 사전에 얼굴 추출 및 정렬, 정규화 등의 전처리 과정을 수행한 후보군 이미지를 Streamlit에 미리 업로드하여, 선택한 이미지의 index만을 전달받아서 백엔드 서버에 사전에 저장된 이미지를 모델 추론 과정에서 사용하도록 한다. 이는 불필요하게 이미지를 전달받고 저장하는 오버헤드 및 시간을 단축하기 위함이다.

2.4.2 모델 API

백엔드 서버에 저장된 학습된 모델을 CLI를 통해 호출하여 추론을 진행할 경우, 'subprocess'를 통해 외부 프로세스를 생성 및 관리하고, 가상 환경 실행 등의 추가적인 시스템 리소스로 인한 불필요한 오버헤드가 발생한다. 또한, 디버깅이나 모델 실행 중에 발생하는 에러나 예외 처리를 백엔드에서 실시간으로 처리하기가 어렵다.

따라서 학습 및 경량화가 완료된 헤어스타일 변환 GAN 모델을 API 형태로 제작하여, 외부 프로세스 생성 없이 메모리 내에서 바로 모델을 호출하여 성능을 향상하고, 서비스 개발과정에서 디버깅 및 예외 처리를 용이하게 한다.

2.4.3 GPU CUDA

원격 GPU에 백엔드 서버를 구축하여 모델 추론을 실행하기 때문에, 기존 모델에서 'torch.load' 함수를 사용해서 모델을 로드할 때, 디폴트 설정에 따라 CPU에 모델이 로드된다. 딥러닝 모델의 추론 작업을 위해 모델 로딩을 GPU CUDA로 변경한다. 모델을 직접 GPU 메모리로 로드해서 추가적인 메모리 이동 없이 연산속도가 빠른 GPU를 통해 추론을 실행하여 추론 시간을 단축한다.

대규모 딥러닝 모델이나 실시간 처리가 필요한 애플리케이션에서 GPU 로딩은 필수적인 최적화 작업이라는 점에 착안해서, 이러한 GPU 로딩 방식을 적용하여 더 빠른 성능을 구현한다.

2.4.4 모델 사전 로딩

원격 GPU 서버에서 서비스에 사용되는 다수의 모델 파일을 로드하는 데에 오버헤드 및 지연 시간이 오래 걸리는 문제를 해결하기 위해서, 백엔드 서버 메모리에 모델을 미리 로드해두고, 서비스 요청이 들어올 때 바로 사용할 수 있도록 한다.

서버가 시작될 때, 서비스에 필요한 모든 모델을 메모리에 미리 로드해서, 요청마다 모델을 새로 로드하는 오버헤드를 최소화한다. 이를 통해 로드 시간을 제거하여 응답 시간을 크게 단축하고, 즉각적인 추론 처리를 통해 실시간 응답성 및 전체 서비스의 처리 시간을 향상시킨다.

2.5 서비스 구성 및 사용방안

수정된 연구 방향에 맞추어 아래와 같은 서비스를 개발한다.

2.5.1 서비스 구성

본 연구를 통해 개발하고자 하는 서비스의 구성은 다음과 같다.

사용자의 얼굴 이미지를 내장, 또는 외장 된 카메라를 통해 전달받고, 전달받은 얼굴 이미지에서 사용자의 얼굴과 헤어가 포함된 얼굴 부분을 crop 하여 백엔드 서버로 전송한다. 사용자는 다음과 같은 3가지 변환 기준을 선정한다. [헤어스타일(Shape) / 색상(Color) / 헤어스타일 및 색상(Both)] 중에서 사용자가 선택한 원하는 변환 내용에 따라, Streamlit에 저장된 사전에 수집된 이미지 데이터 셋 중에서 원하는 헤어스타일 (shape)와 색상 (color)을 선택한다.

Transfer 버튼을 누르면, 백엔드 서버에 저장된 HairFastGAN 모델의 API를 호출하여, 사용자의 얼굴 이미지로부터 선택된 shape와 color에 해당하는 스타일로 변환되어 생성된 이미지를 전달받는다.

2.5.2 서비스 사용방안

개발된 서비스를 사용할 수 있는 방안은 다음과 같다.

a. 사용자의 얼굴 인식

사용자의 얼굴 이미지를 이용하여 변환된 헤어스타일이 적용된 생성형 이미지를 얻기 위해서는 “Use webcam” 옵션을 선택한다. 해당 옵션을 선택할 경우, CAM을 이용하여 인식된 이미지에서 사용자 얼굴 부분만을 crop 하여 Streamlit에 저장하고, 해당 이미지를 백엔드 서버로 전송하여 생성형 모델에 적용한다.

b. 원하는 이미지

사용자가 본인의 얼굴 이미지가 아니라, 별도의 원하는 얼굴 이미지의 헤어스타일 변환이 적용된 생성형 결과를 얻기 위해서는 "Upload an image" 옵션을 선택한다. 해당 옵션을 선택할 경우, 업로드한 이미지를 Streamlit에 저장하고, 해당 이미지를 백엔드 서버로 전송하여 생성형 모델에 적용한다.

2.6 서비스(WEB) 개발

수정된 연구 방향 및 목표에 알맞게 WEB 서비스를 개발한다. WEB 서비스 개발과정에서의 프론트엔드 및 백엔드 소스코드에 대한 설명은 아래와 같다.

2.6.1 Front-end

front.py는 Streamlit을 기반으로 사용자가 헤어스타일 변환을 경험할 수 있는 웹 애플리케이션을 제공한다. 사용자는 자신의 이미지를 업로드하거나 웹캠을 통해 얼굴을 캡처할 수 있으며, 이를 서버로 전송하여 변환된 결과를 실시간으로 확인할 수 있다.

a. YOLO 모델 로드 및 사용

```
def model_load():
    return YOLO(model="model/yolov8n-face.onnx", task="detect", verbose=False)

DETECTOR = model_load()
```

[그림 2] YOLO 모델 로드 및 사용

YOLOv8 모델을 로드하여 얼굴 탐지를 수행하는 기능을 제공한다. YOLOv8 얼굴 탐지 모델은 ONNX 포맷으로 제공되며, DETECTOR 객체를 통해 얼굴 인식 및 전처리를 담당한다. 모델은 사용자가 업로드한 이미지나 웹캠에서 캡처한 이미지에서 얼굴 영역을 추출하고, 이를 변환 작업에 사용할 수 있도록 한다.

b. 이미지 전송을 위한 서버 요청

```python
def send_image_to_server(image):
    _, buffer = cv2.imencode('.jpg', image)
    response = requests.post("http://localhost:8002/upload/", files={"file": buffer.tobytes()})
    return response.json()
```

[그림 3] 이미지 전송을 위한 서버 요청

사용자가 업로드한 이미지 또는 웹캠을 통해 캡처된 이미지를 FastAPI 백엔드 서버에 전송하는 역할을 한다. 이미지를 JPEG 형식으로 인코딩한 후, HTTP POST 요청을 사용해 서버에 전송한다. 서버는 전송된 이미지를 처리한 후, 변환된 이미지를 반환하며, 해당 결과는 사용자의 화면에 표시된다.

c. UI 및 web-cam/image 업로드

```python
with left_column:
    upload_choice = st.radio("Choose an option:", ("Upload an image", "Use webcam"))
```

[그림 4] UI 및 web-cam/image 업로드

사용자가 이미지 업로드 또는 웹캠을 사용할 수 있는 선택지를 제공하는 인터페이스를 구현한다. Streamlit을 사용하여 간단한 라디오 버튼을 통해 사용자는 웹캠을 선택하거나 기존 이미지를 업로드할 수 있다.

d. web-cam 이미지 캡처 및 전처리

```python
elif upload_choice == "Use webcam":
    flip = st.checkbox("Flip", value=False)

    ctx = webrtc_streamer(
        key="example",
        video_processor_factory=MyVideoTransformer,
        media_stream_constraints={"video": True, "audio": False},
        async_processing=True
    )

    if ctx.video_transformer:
        ctx.video_transformer.flip = flip
        if st.button("Capture Image"):
            img = ctx.video_transformer.captured_image
            if img is not None:
                st.image(img, channels="BGR")
                img = Image.fromarray(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
                st.session_state["user_image"] = img
```

[그림 5] web-cam 이미지 캡처 및 전처리

웹캠을 통한 이미지 캡처와 Streamlit WebRTC 라이브러리를 사용해 실시간 비디

오 스트림을 처리한다. 사용자가 웹캠에서 이미지를 캡처하면 해당 이미지를 화면에 표시하고, 이를 서버로 전송할 준비를 한다. 사용자의 캡처 이미지가 세션 상태에 저장되어 서버에 전송될 수 있도록 관리된다.

e. 사용자 입력에 따른 전송 데이터 구성

```
if st.button("Transfer"):
    files = {}
    data = {}

    if "user_image" in st.session_state and st.session_state["user_image"] is not None:
        user_img = st.session_state["user_image"]
        if user_img.mode == "RGBA":
            user_img = user_img.convert("RGB")
        img_bytes = io.BytesIO()
        user_img.save(img_bytes, format="JPEG")
        img_bytes.seek(0)
        files["file"] = img_bytes  # User image file
```

[그림 6] 사용자 입력에 따른 전송 데이터 구성

사용자가 선택한 이미지를 서버로 전송하는 과정이다. 버튼을 클릭하면 세션에 저장된 사용자 이미지를 JPEG 포맷으로 인코딩하여 서버로 전송한다. 서버는 요청을 처리하고, 처리된 이미지를 반환해 사용자에게 표시한다.

2.6.2 Back-end

backend.py는 FastAPI를 사용하여 사용자로부터 전송된 이미지를 처리하고, 변환된 결과를 반환하는 서버 측 코드를 담당한다.

a. FastAPI 초기화 및 모델 로드

```
app = FastAPI()

# Use relative paths within the HairFastGAN directory
USER_IMAGE_PATH = "input/user_image.png"
RESULT_IMAGE_PATH = "output/user_result.png"
MODEL_PATH = "."

# Initialize the model arguments once
model_parser = get_parser()
model_args, _ = model_parser.parse_known_args()
```

[그림 7] FastAPI 초기화 및 모델 로드

FastAPI 서버를 초기화하고, 사용자 이미지를 저장할 경로와 결과 이미지를 저장할 경로를 정의한다. 또한, 헤어스타일 변환 모델의 파라미터를 설정하는 get_parser() 함수를 호출하여, 필요한 모델 설정을 로드한다. 모델은 GAN 기반의 변환 모델로, 사용자 요청에 맞추어 이미지를 변환하는 역할을 수행한다.

b. 모델 호출

```
# Call the process_image function
process_image(
    face_path=Path(USER_IMAGE_PATH),
    shape_path=shape_path,
    color_path=color_path,
    output_path=Path(RESULT_IMAGE_PATH),
    model_args=model_args,
    benchmark=False
)
```

[그림 8] 모델 호출

process_image 함수는 핵심적인 이미지 변환 작업을 처리한다. 입력된 사용자 이미지, 선택된 헤어스타일 및 색상을 바탕으로 변환 작업을 수행한다. 변환된 결과는 RESULT_IMAGE_PATH에 저장된다. GAN 기반 생성형 모델은 사용자 요구에 맞춰 헤어스타일을 변환하여 결과 이미지를 생성한다.

c. 생성 이미지 반환

```
if os.path.exists(RESULT_IMAGE_PATH):
    processed_image = Image.open(RESULT_IMAGE_PATH)
    buf = io.BytesIO()
    processed_image.save(buf, format="JPEG")
    buf.seek(0)
    return Response(content=buf.getvalue(), media_type="image/jpeg")
```

[그림 9] 생성 이미지 반환

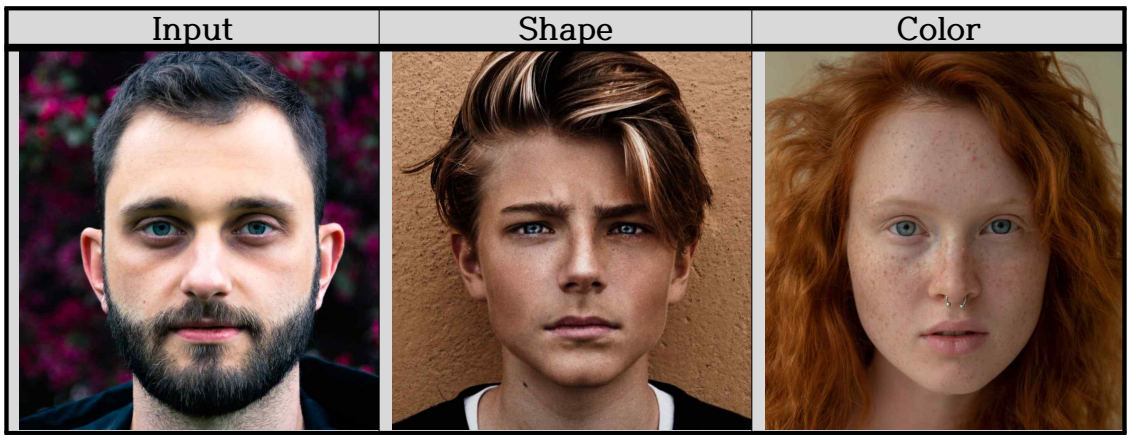
모델에서 생성된 이미지를 읽고 JPEG 형식으로 인코딩한 뒤 사용자에게 반환한다. 이 과정은 FastAPI가 비동기 방식으로 처리하며, 사용자가 요청한 이미지를 빠르게 반환할 수 있도록 한다.

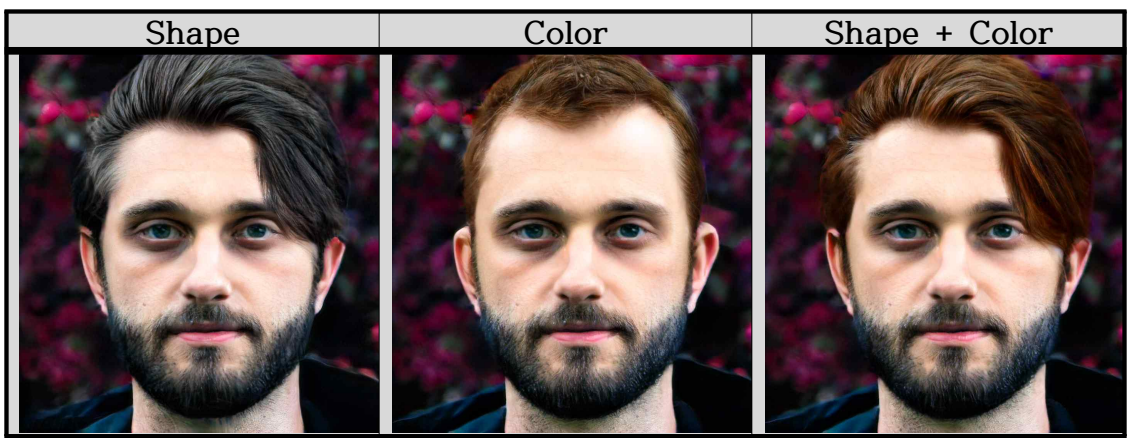
2.6.3 HairFastGAN

본 연구에서 채택한 GAN 기반의 헤어스타일 변환 모델인 HairFastGAN에서 사전에 수집된 이미지 데이터 셋에서 사용자 이미지를 임의의 input으로 지정하고, 각각 사용자가 원하는 스타일의 ***[Shape / Color / Shape + Color]*** 3가지 선택지를 선택하는 경우를 가정하여 모델의 추론 결과는 다음과 같다.

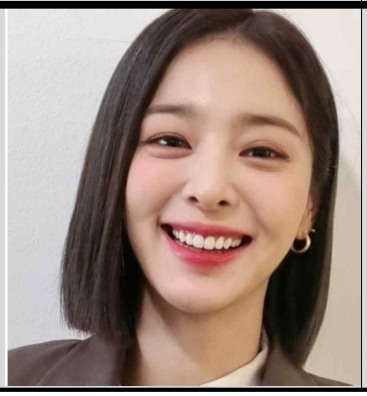
a. 전처리 작업이 없는 경우

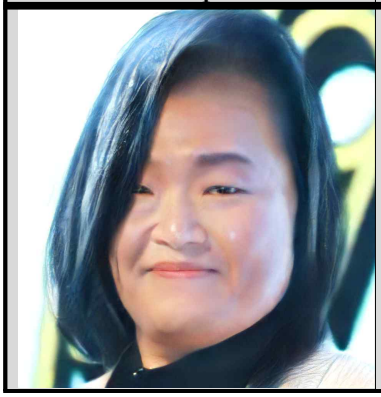
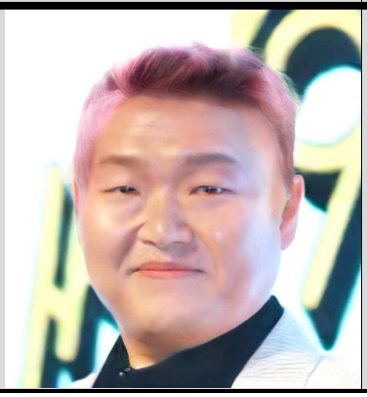
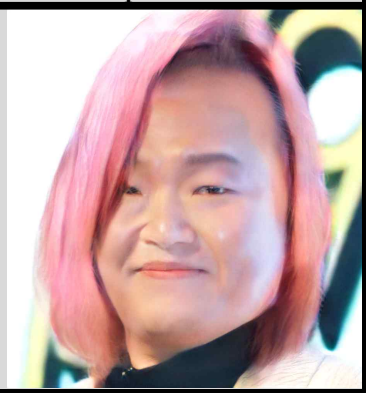
a-1. 추론 결과





위와 같은 결과는 동일한 크기의 입력 이미지와 얼굴 위치, 방향 등의 정렬과 관련된 전처리 작업이 없이도 좋은 성능을 확인할 수 있지만, 아래와 같은 데이터셋 이미지의 경우에는 일정하지 않은 크기, 방향 등으로 인해 왜곡된 이미지가 생성됨을 확인할 수 있다.

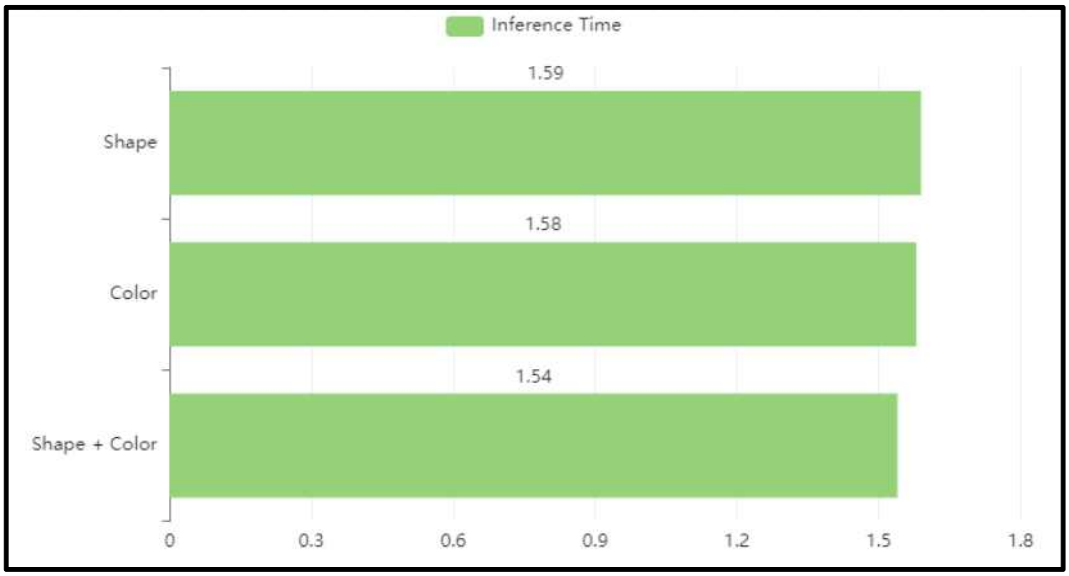
Input	Shape	Color
		

Shape	Color	Shape + Color
		

위와 같은 왜곡된 이미지 결과가 생성되는 이유로 연구 초기 단계에는 모델의 학습에 사용된 편향된 데이터 셋에 의해 한국인 이미지를 입력으로 사용하면 발생하는 왜곡된 결과로 추측하였다. 하지만, 다른 한국인 이미지를 입력할 때, 위의 경우와 다르게 얼굴의 방향이나 정렬이 어느 정도 이루어진 경우에 대해서는 자연스러운 이미지가 생성됨을 확인하고, 사용자의 입력 이미지에 대해 사전 처리가 필요함을 인지하였다.

a-2. 추론에 걸리는 시간

사용자 입력 이미지에 대해 [Shape + Color / Shape / Color] 세 가지 선택사항에 대해, 모델의 추론에 걸리는 시간은 다음과 같다.



[그림 10] 전처리 작업이 없는 경우 추론 시간

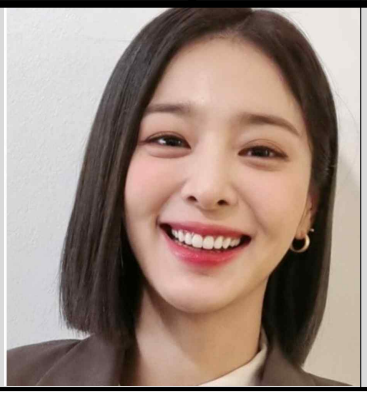
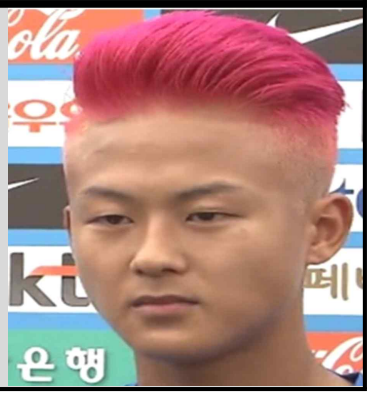
실제 GPU 서버에서 다음과 같은 추론을 진행하였을 때, 세 가지 선택사항에 대해서는 추론 시간이 크게 차이가 나지 않았고, 결과 이미지를 확인하기까지 시간이 오래 걸리는 부분은 학습된 모델의 checkpoint와 weight 들을 load 하는 시간이 오래 소요됨을 확인할 수 있었다.

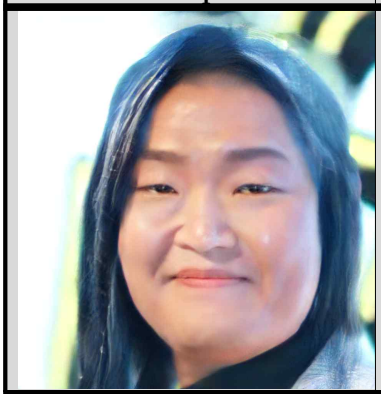
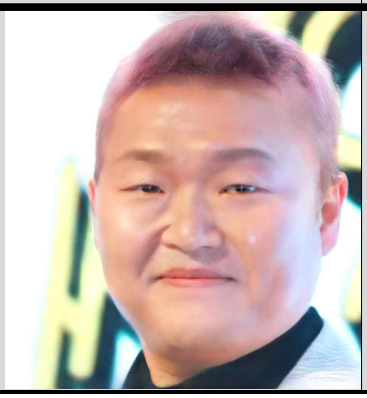
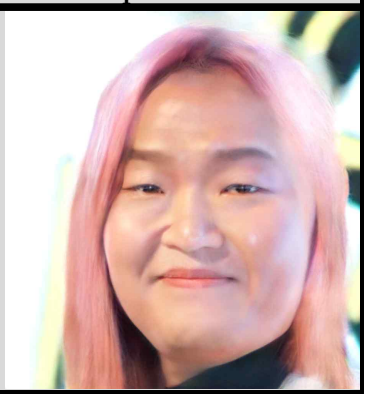
b. 성능 개선을 위한 전처리 작업

이전의 결과에서 확인한 것과 같이, 이미지 크기 조정, 얼굴 정렬 및 회전과 같은 성능 개선을 위한 이미지 전처리 과정뿐만 아니라, 사용자의 web-cam을 통해 입력받거나, 얼굴 부분이 아닌, 몸 전체가 나오는 이미지나 여러 사람의 얼굴이 인식되면 하나의 얼굴만을 crop 하여 입력으로 받기 위한 이미지 전처리 작업을 수행한다.

b-1. 얼굴 회전 및 정렬 (Face Alignment)

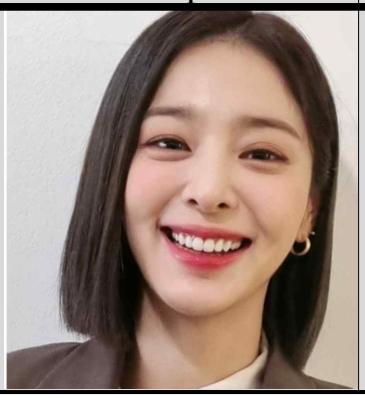
모델 입력에 사용될 사용자의 얼굴 이미지와 Color & Shape에 사용될 얼굴 이미지의 정렬을 통해 성능 개선을 위한 전처리 작업을 수행한 후에, 결과를 확인한다.

Input	Shape	Color
		

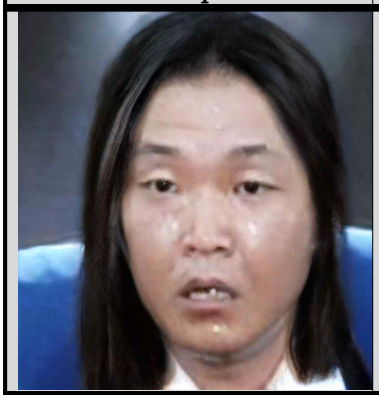
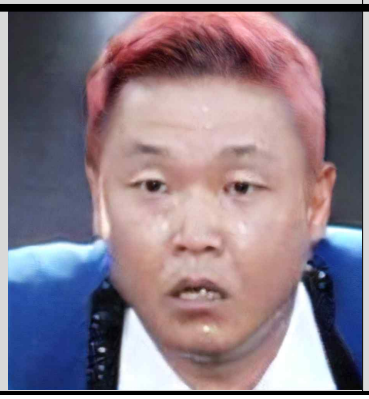
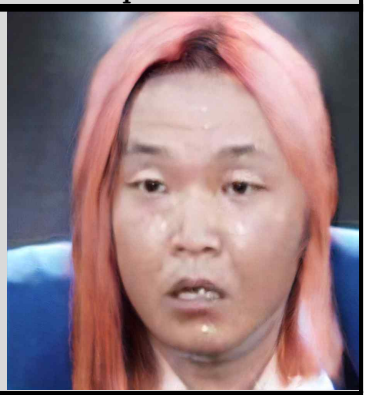
Shape	Color	Shape + Color
		

위의 전처리 과정은 Facial Landmark Detection을 통해 감지된 얼굴 이미지의 주요 위치들을 인식하고, 모델의 입력으로 사용하기 위해 고정된 위치로 회전 및 정렬하는 작업을 하였다. 이에 이전에 전처리를 거치지 않았을 때 비해 부자연스럽거나 왜곡된 부분이 없어지고, 비교적 주관적 성능이 개선됨을 확인할 수 있었다.

b-2. 얼굴 인식 및 추출 (Face Detection & Crop)

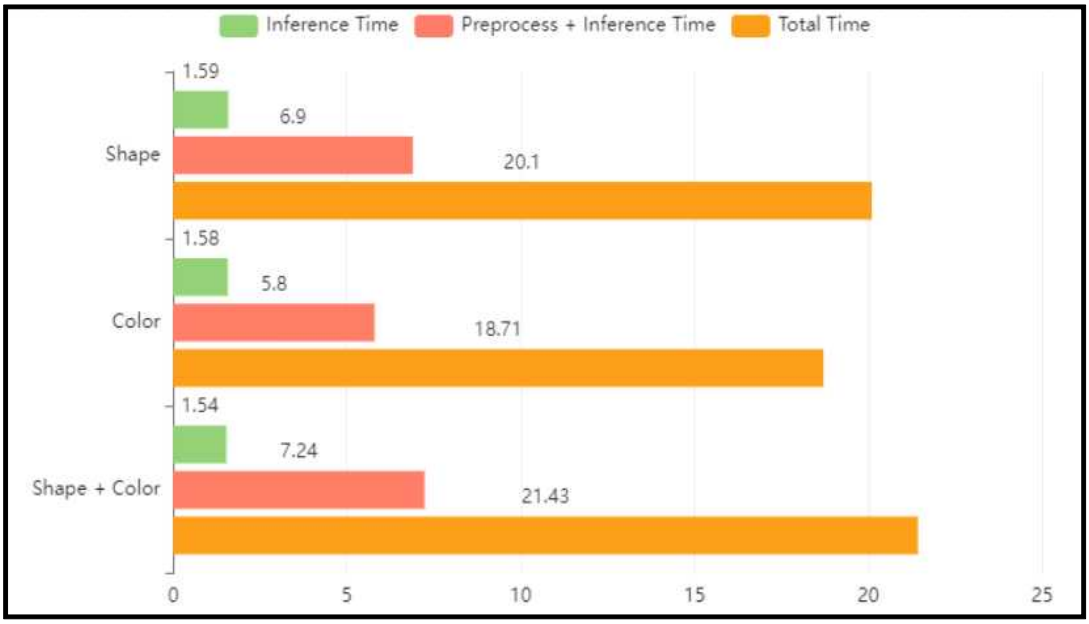
Input	Shape	Color
		

위와 같이 얼굴만이 나오는 입력 이미지가 아니라, 사용자가 웹캠 혹은 이미지를 업로드할 때 몸 전체가 나오는 사진이나, 여러 인물이 나올 때 따른 에러를 해결하기 위해 업로드된 이미지에 대해 얼굴 인식 (Face Detection)을 우선적으로 실시하여, 인식된 얼굴이 1개인 경우에만 이전의 전처리 작업을 거쳐 얼굴 정렬을 통해 모델의 입력으로 사용된다.

Shape	Color	Shape + Color
		

사용자 입력 이미지로부터 모델의 입력에 사용될 적절한 크기로 추출한다. 이러한 일련의 얼굴 인식 및 추출 과정과 성능 개선을 위한 전처리 과정을 거친 경우에 위와 같은 결과를 확인할 수 있다.

b-3. 추론에 걸리는 시간



[그림 11] 전처리 작업을 할 경우의 추론 시간

이전에 전처리 과정 없이 모델 추론에만 걸리는 시간은 세 가지 경우에 대해 1.5초 내외의 추론 시간이 소요되었다. 이에 비해 성능 개선 및 에러 없이 모델이 정상적으로 추론 과정을 수행하기 위해 여러 이미지 전처리 과정을 수행할 때 전체 과정이 6~7초가 소요된다는 것을 확인할 수 있다.

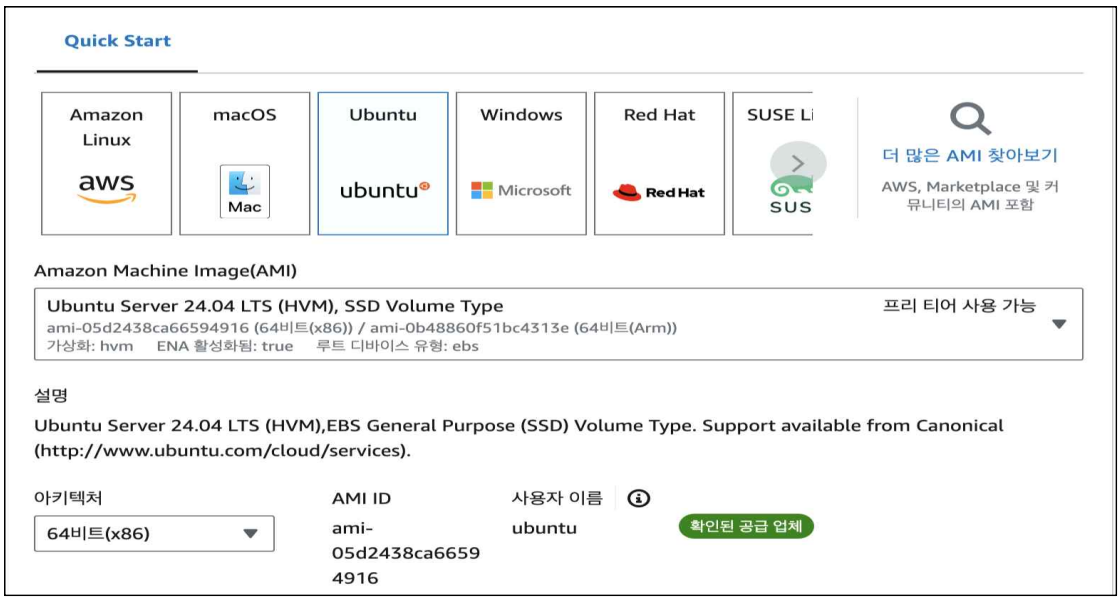
위의 시간은 GPU에 미리 로드 하지 않았으며, 학습된 모델을 로드하는 과정은 10초 내외의 시간이 추가로 소요되었다. 또한, 위의 결과는 경량화 등의 시간 단축을 위한 과정을 거치지 않은 결과이다. 따라서 전체적인 서비스 서빙 시간의 단축을 위해서는 학습된 모델을 백엔드 GPU 서버 메모리에 미리 로드하는 과정과 전처리 및 GAN 모델에 대한 경량화 과정이 필요하다.

2.7 AI 모델 경량화 및 비용 효율화 방안

수정된 연구 방향 및 목적에 따라 기존 인공지능 모델의 경량화 방법을 조사하고 직접 적용한다. 각각의 경량화 방법에 따른 성능과 비용적인 측면을 비교하고, 인공지능 기술 및 모델을 서비스에 활용할 때, 비용 효율적 측면의 방안에 관해 연구한 내용은 아래와 같다.

본 연구에서 위의 실험을 통해 경량화를 적용하기 위한 모델로 **"Resnet9" 모델**을 선정하여 연구를 진행하였다. 추가로 경량화를 적용하여 비율 효율화 방안에 관한 연구를 진행하는 과정에서의 실험 환경은 다음과 같다.

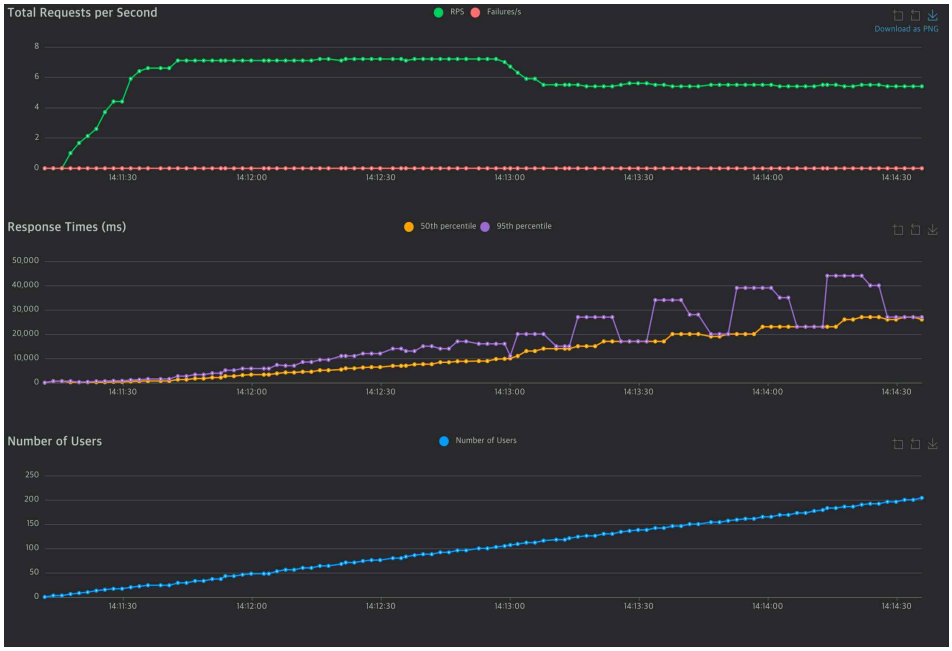
[그림 12] 실험 환경 – 인스턴스 유형



[그림 13] 실험 환경 – 사용 이미지

2.7.1 ONNX Model

실험을 진행한 "Resnet9" 모델에 대해, 기존의 [.pth] 모델과 모델의 경량화를 위해 [.onnx]로 변환한 모델의 성능 비교 결과는 다음과 같다.



[그림 14] .pth 모델 성능 실험 결과

[그림 15] .onnx 모델 성능 실험 결과

a. RPS(Request Per Second)

실험에서 사용한 metric인 "RPS"는 API가 1초에 처리할 수 있는 요청(request)의 수를 의미한다. 즉, RPS가 높을수록 시스템이 더 많은 요청을 동시에 처리할 수 있음을 나타낸다.

본 연구에서는 PyTorch 모델(pth)과 ONNX 모델의 RPS 성능을 비교하여 실시간 추론 성능을 평가하였다. 실험 결과, 두 모델 간의 성능 차이가 명확하게 나타났으며, ONNX 모델이 실시간 서비스에 더욱 적합한 성능을 보였다.

a-1. pth 모델의 RPS

.pth 모델의 경우, RPS는 사용자 수가 증가함에 따라, 사용자 수 단위 200에서 약 **6 RPS**로 saturation하는 결과가 측정되었다. 이는 초당 약 6개의 요청을 처리할 수 있음을 의미하며, 실시간 서비스에서의 처리 성능은 다소 낮음을 의미한다. pth 모델은 연구와 학습에 최적화되어 있어, 추론 단계에서 성능 최적화가 부족한 것으로 분석된다.

a-2. onnx 모델의 RPS

.onnx 모델은 경량화된 형태로 .pth 모델과 비교하였을 때 성능이 최적화되어, 사용자 수 단위 200에서 약 **32 RPS**로 saturation하는 결과가 측정되었다. 이는 pth 모델과 비교하면 5배 이상 성능을 보이며, 실시간 서비스에서 더욱 많은 요청을 동시에 처리할 수 있음을 의미한다. 이는 ONNX 모델이 다양한 하드웨어에서의 성능 최적화가 이루어졌기 때문에, 실시간 추론 작업에 적합한 모델로 판단된다.

b. Response Time

성능 비교 분석을 위해 사용한 metric인 "Response Time"은 초당 요청 처리(RPS)와는 다른 개념으로, 단일 요청이 처리되는 데 걸리는 시간을 의미하며, 사용자는 Response Time에 따라 빠르게 결과를 확인할 수 있다. 이를 통해 Response Time이 짧을수록 실시간 AI 서비스나 추론 시스템에서 서비스의 효율성이 증가하고, 시스템의 반응성에 대한 사용자 경험이 개선된다.

b-1. pth 모델의 Response Time

.pth 모델의 평균 Response Time은 약 **30,000ms**로 측정되었다. 이는 각 요청에 대해 약 30초의 응답 시간을 가지며, 연구와 학습 목적으로는 충분한 성능을 제공한다. 그러나 실시간 서비스 환경에서는 응답 시간이 다소 길어 사용자가 즉각적인 피드백을 받기에는 적절하지 않을 수 있다.

b-2. onnx 모델의 Response Time

.onnx 모델의 평균 Response Time은 **4,000ms**로 측정되었다. 이는 .pth 모델의 약 13.3%에 불과한 응답 시간을 보여주며, 실시간 서비스에서 더욱 빠른 응답을 제공할 수 있음을 의미한다. 따라서 onnx 모델은 경량화와 최적화를 통해 추론 속도가 개선되어, 실시간 애플리케이션에서 빠르고 원활한 사용자 경험을 보장함을 확인할 수 있다.

2.7.2 PTQ Model

DNN Model 크기가 커지는 속도는 현재 연산기 성능 향상 속도에 비에 빠른 편이다. 이렇게 빠른 속도로 model의 크기가 커지고, 연산 복잡도가 높아짐에 따라, Inference Time과 Training Cost 등의 시간 및 비용적 측면의 부담이 커지고 있다. 이를 해결하기 위해 다양한 모델 경량과 기법들을 사용하고 있다.

모델 경량화 기법은 크게 4가지로 볼 수 있다. Pruning, Quantization, Knowledge distillation, Low-rank factorization 중에서 본 연구에서는 양자화 경량화 기법(Quantization)을 실험 비교 기준 모델로 선정한 Resnet9에 적용하여 성능을 비교분석을 한다. 기존 모델의 Parameter를 FP32의 실수형 변수에서 INT 8의 정수형 변수로 변환하여, FP32 타입의 파라미터를 INT 8 형태로 변환한 다음에 inference 실험을 진행하였다.

첫 번째 방안은 PTQ(Post-Training Quantization)로 이미 학습이 끝난 모델을 양자화(Quantization)를 통해 경량화하는 방법이다. 일반적으로 모델의 출력값 범위가 고르게 분포되어 있지 않기 때문에, 단순히 출력을 기존 32bit, 16bit에서 8bit로 조절한다면, 모델의 성능이 많이 떨어질 수 있다. 따라서, 데이터를 통과시켜 나오는 출력값에 따라 scale을 조절하여, Quantization으로 인한 손실을 최소화하는 방법을 적용한다.

2.7.3 QAT Model

QAT(Quantization-Aware Training)은 학습 단계에서 미리 Inference 과정에 양자화(Quantization)를 적용하여, 최적의 가중치를 구하는 방식이다. 앞선 PTQ의 방법과 비교해서, 모델을 추가로 학습해야 하는 단점이 있지만 학습 과정부터 적용하다 보니 성능 하락을 최소화할 수 있다. 또한, 파라미터 사이즈가 크고, 정확도가 중요한 모델이 아니라면 QAT의 사용은 필연적이기 때문에 두 번째 방안으로 선택하여 실험을 진행한다.

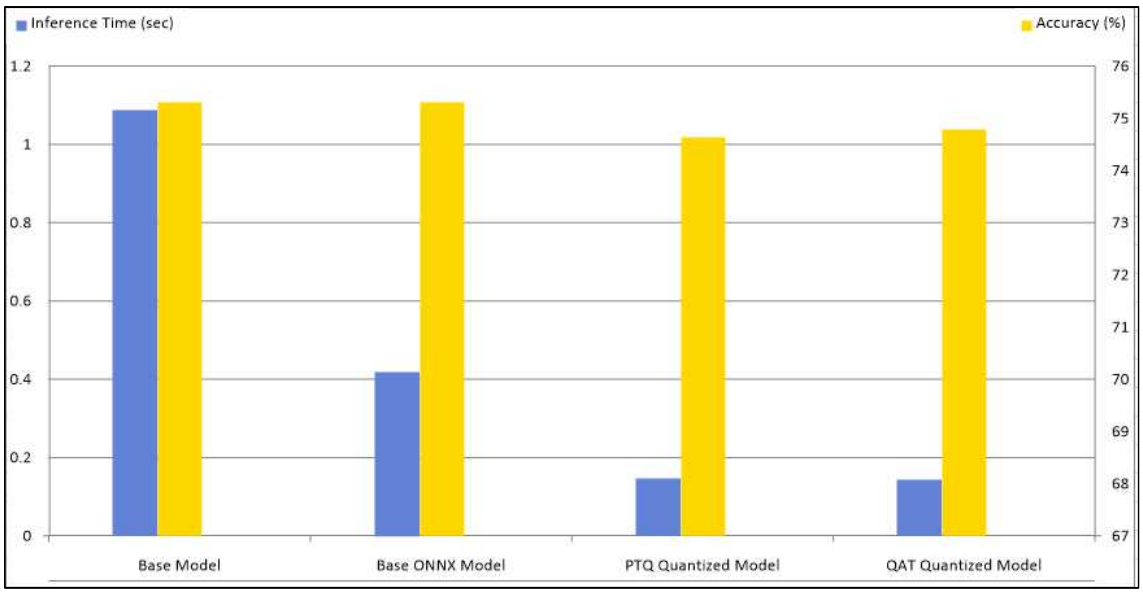
2.7.4 경량화 실험 결과

실험에 사용하는 모델은 Resnet9을 사용하였으며, 기존 모델을 ONNX 변환하고, 양자화(Quantization)를 적용하되, PTQ 방식과 QAT 방식을 적용하여 결과를 비교한다. 각각의 경량화 기법을 적용한 모델들의 성능 비교와 크기 비교에 사용된 metric들과 각각의 실험 결과는 아래와 같다.

a. Inference Time & Accuracy

```
(quantize) PS C:\Users\BSH\project\final_project> python .\compare_performance.py
Files already downloaded and verified
Files already downloaded and verified
----- Base Model -----
Infer Time : 1.086670sec | Accuracy 75.300%

----- Base Onnx Model -----
Infer Time : 0.419593sec | Accuracy 75.300%

----- PTQ Quantized Model -----
Infer Time : 0.147104sec | Accuracy 74.630%

----- QAT Quantized Model -----
Infer Time : 0.142816sec | Accuracy 74.780%
```

[그림 16] Inference Time 및 Accuracy 결과

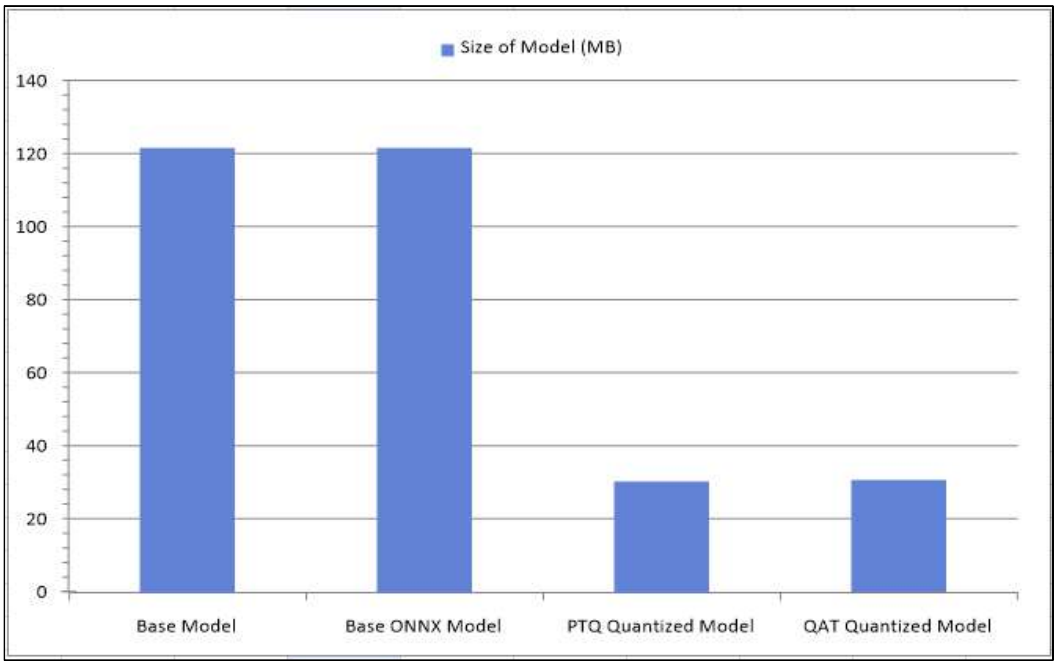


[그림 17] Inference Time 및 Accuracy 결과 그래프

b. Size of Model

```
----- Base Model -----
Size of Model(MB): 121.8823

----- Base Onnx Model -----
Size of Model(MB): 121.729567

----- PTQ Fused Model -----
Size of Model(MB): 121.782529

----- PTQ Quantized Model -----
Size of Model(MB): 30.481575

----- QAT Fused Model -----
Size of Model(MB): 121.898074

----- QAT Quantized Model -----
Size of Model(MB): 30.574047
```

[그림 18] Size of Model 비교 결과



[그림 19] Size of Model 비교 그래프

c. PTQ 및 QAT 모델의 ONNX 변환

인공지능 기술 및 모델을 이용한 서비스를 서빙 할 경우, 추론 시간 단축 및 비용 효율적인 방안에 관한 연구를 위해 기준 실험 모델로 선정한 Resnet9에 대해 ONNX 변환, 양자화(Quantization) 기법을 적용하는 PTQ 및 QAT 방식을 통한 모델들에 대해 Inference Time 및 Accuracy를 비교하고, 모델의 크기 변화 결과를 확인하였다.

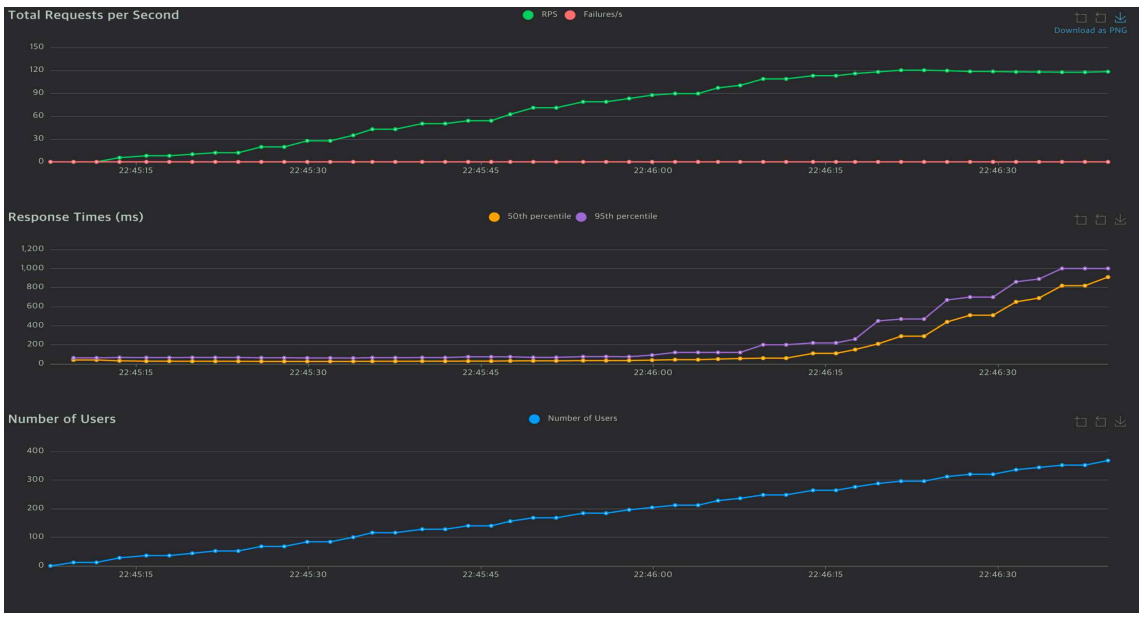
이를 토대로 최종적으로 인공지능 모델을 서비스 서빙을 위해 서버에 올려서 사용할 때, 사용자 수가 늘어남에 따라 RPS(Requests Per Second)와 Response Time 변화 결과를 확인한다. 이를 통해 최종적으로 인공지능 기술 및 모델을 이용한 서비스 서빙 과정에서의 비용 효율적인 방안에 대한 결론을 도출한다. 다만, 현재 양자화를 적용하여 PTQ 및 QAT 방식으로 변환된 모델에 대해 직접적으로 ONNX 형태로 변환하는 것은 불가능하다. 따라서, 모델을 서버에 올려서 실험을 진행하는 과정에서, 모델들을 onnx runtime API를 사용하여 경량화하여 결과를 비교하는 실험을 진행한다.

추가로, 경량화를 적용하여 성능 비교 실험을 진행하는 점을 고려하여, 기존 실험 환경 (Amazon AWS EC2)는 같지만, 사용자 수가 기존 200명에서 400명까지 증가할 때의 성능 비교 실험을 진행하였다.

a. PTQ + ONNX Model

[그림 20] PTQ + ONNX Model 모델 성능 실험 결과

b. QAT + ONNX Model



[그림 21] QAT + ONNX Model 모델 성능 실험 결과

c. RPS (Requests per Second)

2가지 경량화 기법을 적용한 모델(PTQ + ONNX / QAT + ONNX)들에 대해 Amazon AWS EC2에서 사용자 수가 400명까지 증가함에 따른 성능 비교 결과는 다음과 같다.

첫 번째 metric인 RPS (Requests per Second)를 비교해 보았을 때, [PTQ+ONNX] 모델과 [QAT+ONNX] 모델 2가지 모두 **120 RPS**로 수렴함을 확인할 수 있다.

d. Response Time (ms)

두 번째 metric인 Response Time(ms)을 비교해 보았을 때는, 사용자 수가 200명까지 증가할 때는 차이가 없지만, 이후 400명까지 사용자 수가 증가함에 따라 2가지 모델의 성능에 소폭 차이가 있음을 확인할 수 있다.

[PTQ+ONNX] 모델의 경우에는 사용자 수가 200명일 때부터 Response Time이 증가하여, 400명이 될 때까지 약 **1,400ms** Response Time으로 수렴한다. 동일하게, [QAT+ONNX] 모델도 사용자 수가 200일 때까지의 Response Time은 같지만, 400명 이 될 때까지는 약 **1,000ms** Response Time으로 수렴한다. 따라서, 위 2가지 경우의 모델에서는 "QAT+ONNX" 모델의 케이스가 소폭 나은 성능을 보여줌을 확인할 수 있다.

3. 연구 결과 분석 및 평가

본 연구에서는 초기 목표였던 HairFastGAN 기반의 실시간 라이브 스트리밍 형태의 헤어스타일 변환 서비스 개발이 모델 경량화와 전처리 과정에서 발생하는 기술적 한계로 인해 불가능함을 확인하였다. 헤어스타일 변환 작업은 얼굴 인식, 얼굴 정렬, 회전, 헤어 추출 등의 복잡한 단계를 거치며, 이러한 과정에서 실시간 추론을 제공하기 위한 성능 최적화가 어려움을 겪었다. 따라서 연구 방향을 비실시간 이미지 변환 서비스로 변경하여, 추론 시간이 짧은 형태의 서비스 개발을 목표로 삼았다.

수정된 연구 방향에 따라, 모델 API, FastAPI, Streamlit을 활용해 서비스 서빙 시간을 최대한 단축하는 것을 목표로 하였다. 추가로, 이미지 로딩 및 저장, 전처리 과정에서 발생하는 오버헤드를 줄이기 위해 각 과정에서 최적화 작업을 수행하였으며, 이를 통해 사용자 경험을 향상했다. 이러한 최적화 작업은 전체 추론 시간 및 서비스 서빙 시간 단축에 큰 기여를 하였으며, 비실시간 서비스임에도 불구하고 상대적으로 추론 시간 및 서빙 시간을 단축할 수 있었다.

AI 서비스의 비용 효율성을 극대화하기 위한 연구 또한 병행되었다. AI 모델을 클라우드 기반으로 서빙하는 경우, 추론 속도와 비용 효율성이 매우 중요한 요소로 작용한다. 본 연구에서는 ONNX 모델 변환과 PTQ(Post-Training Quantization), QAT(Quantization-Aware Training) 경량화 기법을 적용하여 다양한 성능 지표를 비교 분석하였다. 실험 결과, ONNX 모델은 PyTorch 기반 pth 모델보다 높은 RPS(Requests Per Second) 성능을 보였으며, 추론 응답 시간(Response Time)도 크게 단축되었다. 이러한 성능 향상은 실시간 서비스 환경에서 많은 요청을 동시에 처리할 수 있도록 하여, 서비스의 확장성과 안정성을 보장할 수 있다.

마지막으로 PTQ와 QAT의 성능 비교 실험을 통해 두 경량화 기법 간의 차이를 분석하였다. PTQ 방식은 학습 후에 양자화를 적용하는 방식으로, 구현이 간단하고 추가 학습이 필요 없다는 장점이 있다. 그러나 QAT 방식은 학습 단계에서 양자화를 반영하여 최적의 가중치를 구하는 방식으로, 성능 하락을 최소화하면서도 더 높은 처리 성능을 제공한다는 특징이 있다. 실험 결과, QAT 모델이 PTQ 모델보다 더 우수한 성능을 보여주었으며, 특히 사용자 수가 증가할 때도 낮은 응답 시간을 유지하는 점에서 실시간 서비스에 더 적합한 것으로 평가되었다.

4. 결론 및 향후 연구 방향

본 연구에서는 HairFastGAN 기반의 실시간 헤어스타일 변환 서비스 개발을 목표로 하였으나, 기술적 제약으로 인해 비실시간 이미지 변환 서비스로 연구 방향을 수정하게 되었다. 헤어스타일 변환 작업의 복잡한 전처리 과정과 GAN 기반 모델의 추론 시간이 실시간 서비스를 제공하기에는 한계가 있었기 때문이다. 이러한 문제를 해결하기 위해 FastAPI와 Streamlit을 활용하여 API 기반의 서비스 서빙 시간을 단축하였으며, 여러 최적화 작업을 통해 사용자 경험을 개선하였다. 특히, 비동기 프로그래밍과 GPU 로딩 최적화 등의 기술을 통해 이미지 전처리 및 추론 과정에서 발생하는 지연 시간을 최소화하였고, 이를 통해 사용자가 빠른 결과를 얻을 수 있도록 하였다.

추가로, AI 모델을 서비스하는 과정에서 비용 효율성을 고려한 경량화 기법을 적용하여 실험을 진행하였다. ONNX 모델 변환을 통해 PyTorch 기반 모델보다 더 빠른 추론 속도를 제공할 수 있었으며, PTQ와 QAT 같은 경량화 기법을 적용하여 성능을 최적화하였다. PTQ 모델은 간단한 구현으로도 어느 정도의 성능 개선을 보여주었으나, QAT 모델은 특히 실시간 서비스에서 성능 저하 없이 더 많은 요청을 처리할 수 있는 장점을 보였다. 이러한 경량화 기법의 적용은 AI 서비스를 대규모 사용자에게 확장할 때 매우 중요한 요소로 작용하며, 향후 AI 서비스의 비용 절감과 성능 최적화에 중요한 기여를 할 수 있을 것으로 기대된다.

향후 연구에서는 QAT 기법을 더욱 발전시키고, ONNX와 같은 표준 모델 변환 기법을 통해 더 다양한 하드웨어 환경에서 성능 최적화를 연구할 필요가 있다. 또한, 서비스 사용자 수가 급격히 증가하는 상황에서도 성능 저하 없이 서비스를 제공할 수 있도록 더 효율적인 모델 경량화 기법을 도입하고, AI 모델의 추론 성능과 응답 시간을 더욱 향상시키기 위한 연구가 필요하다. 이를 통해 실시간 서비스뿐만 아니라, 다양한 AI 응용 프로그램에서 더 나은 사용자 경험을 제공할 수 있을 것이다.

결론적으로, 본 연구는 헤어스타일 변환 서비스의 경량화 및 성능 최적화에 기여하였으며, 경량화 기법을 통해 AI 서비스를 비용 효율적으로 운영하는 방안을 제시하였다. 향후 더 많은 사용자를 수용할 수 있는 AI 서비스 개발과 실시간 성능 최적화를 위한 연구를 지속하여, 더 나은 서비스 품질을 제공하는 데에 기여할 수 있을 것이다.

5. 구성원별 역할

a. 한지훈: 모델 실험 및 서비스 API 개발

- HairFastGAN 모델 실험 및 결과 분석
- 이미지 전처리 여부에 따른 성능 비교
- 백엔드 서버 구축 및 모델 사전 로딩, API 구현

b. 박시형: AI 모델 비용 효율화 방안 연구

- 기존 pth 모델과 onnx 변환 모델 성능 비교
- QAT 및 PTQ 방법으로 모델 경량화 적용
- 각 방안에 대해 성능 비교 분석

c. 홍진욱: AI 모델 경량화 및 분석

- Streamlit 개발 및 프론트엔드 구축
- Resnet9 모델 경량화 및 분석
- GPU/CPU 비용 비교 및 분석

6. 참고 문헌

〈Papers〉

[1] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. A. Bharath, "Generative adversarial networks: An overview," IEEE Signal Processing Magazine, Vol. 35, No. 1, pp. 53-65, Jan. 2018.

[2] A. Polino, R. Pascanu, and D. Alistarh, "Model compression via distillation and quantization," arXiv preprint arXiv:1802.05668, 2018.

[3] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," arXiv preprint arXiv:1511.06434, 2016.

[4] T. Karras, S. Laine, and T. Aila, "StyleGAN2: Analyzing and improving the image quality of StyleGAN," arXiv preprint arXiv:1912.04958, 2020.

[5] T. Ritschel, and T. Weyrich, "Real-time hair rendering on mobile devices," Journal of Computer Graphics Techniques, Vol. 4, No. 4, 2015.

[6] P. Zhu, R. Zhang, B. Zhang, D. Chen, R. Rong, P. Zhang, and L. Van Gool, "Barbershop: GAN-based image compositing using segmentation masks," arXiv preprint arXiv:2106.01505, 2021.

[7] Y. Shen, C. Yang, X. Tang, and B. Zhou, "Interpreting the latent space of GANs for semantic face editing," Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 9243-9252, 2020.

[8] Z. Tan, P. Zhu, X. Wu, L. Shen, S. Shi, and L. Van Gool, "Michigan: Multi-input-conditioned hair image generation for portrait editing," arXiv preprint arXiv:2010.16417, 2020.

〈Github Projects〉

[9] MichiGAN: Multi-Input-Conditioned Hair Image Generation for Portrait Editing (SIGGRAPH 2020), GitHub Repository, Available: https://github.com/tzt101/MichiGAN

[10] Hairstyle Transfer between Face Images, GitHub Repository, Available: https://github.com/subrtade662/hairstyle_transfer

[11] HeadNeRF, GitHub Repository, Available: https://github.com/CrisHY1995/headnerf

[12] Style Your Hair: Official Pytorch Implementation, GitHub Repository, Available: https://github.com/Taeu/Style-Your-Hair

[13] Barbershop: GAN-based Image Compositing using Segmentation Masks, GitHub Repository, Available: https://github.com/ZPdesu/Barbershop

[14] Interpreting the Latent Space of GANs for Semantic Face Editing, GitHub Repository, Available: https://github.com/genforce/interfacegan

[15] Hairstyle Transfer, GitHub Repository, Available: https://github.com/Azmarie/Hairstyle-Transfer

〈Commercialized Services Research〉

[16] Kittl, Available: https://www.kittl.com

[17] Playground AI, Available: https://playgroundai.com

〈Websites〉

[18] PyTorch Static Quantization Tutorial, Available: https://pytorch.org/tutorials/advanced/static_quantization_tutorial.html

[19] PyTorch Discussion on Fuse Modules, Available: https://discuss.pytorch.org/t/fuse-modules-how-to-fuse-modules-in-the-sequential-block/140423

[20] PyTorch Quantization Documentation, Available: https://pytorch.org/docs/stable/quantization.html

[21] Gaussian37. "PyTorch Quantization," Available: https://gaussian37.github.io/dl-pytorch-quantization/

[22] Amazon Web Services. "EC2 Spot Pricing," Available: https://aws.amazon.com/ko/ec2/spot/pricing/

[23] Amazon Web Services. "EC2 Instance Types," Available: https://aws.amazon.com/ko/ec2/instance-types/

[24] Google Cloud. "GPU Pricing," Available: https://cloud.google.com/compute/gpus-pricing?hl=en

[25] Amazon Web Services. "AWS Pricing Calculator," Available: https://calculator.aws/#/createCalculator/ec2-enhancement