<u>FYP Final Report Template</u>

VIRTUAL SHIRT TRY-ON

Final Year Project Report

By

HASSAN KHALID BUTT

HISSAAN ALI SHAH

HASSAN JAVAID

In Partial Fulfillment
Of the Requirements for the degree
Bachelors of Engineering in Software Engineering (BESE)

School of Electrical Engineering and Computer Science National University of Sciences and Technology

Islamabad, Pakistan (2021)

DECLARATION

We hereby declare that this project report entitled **“Virtual Shirt Try-on”** submitted to the “SEECS”, is a record of an original work done by us under the guidance of Supervisor **“Dr. Zuhair Zafar”** and that no part has been plagiarized without citations. Also, this project work is submitted in the partial fulfillment of the requirements for the degree of Bachelor of Software Engineering.

Team Members:	**Signature**
Hassan Khalid Butt	hassan
Hissaan Ali Shah	hissaan
Hassan Javaid	hassan
Supervisor:	**Signature**
Dr. Zuhair Zafar	

Date: 23 May, 2022
Place: SEECS, NUST-H12, Islamabad

DEDICATION

To Allah the Almighty and Exalted

&

To our Family, Mentors and Faculty

“There is no god but He: That is the witness of Allah, His angels, and those endued with knowledge, standing firm on justice. There is no god but He, the Exalted in Power, the Wise” [Quran, 3:18].

ACKNOWLEDGEMENTS

This project is completed with the mentorship of our respected advisor and co-advisor, **"Dr. Zuhair Zafar" and "Dr. Maajid Maqbool"**. We express our indebted gratitude and special thanks to them who guided us throughout the project lifecycle, despite their busy schedules. We would like to express deepest appreciation to our university, **"National University of Sciences and Technology (NUST)"**, and school, **"School of Electrical Engineering and Computer Science (SEECS)"** and department, **"Department of Computing (DOC)"** , in honing us to be globally competitive by equipping us with the knowledge and skills to make a lasting positive impact in the world.

A humble "Thank you" to **NUST** and the aforementioned faculty.

"To accomplish great things, we must not only act, but also dream; not only plan, but also believe." - Anatole France

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

With e-commerce fashion sales set to double in 2022, soon tech-savvy consumers won’t view digital solutions as a luxury, but a right. Deep learning technologies are on the rise today making their way into all forms of life from health care and education to entertainment. The provision of more data and recent advancements in Computer Vision and Deep Learning algorithms has made it even easier to develop cutting edge technologies. Virtual try-On has been going on as one of the hot topics in the deep learning community. Great amount of research has been put into it and it continues to grow.

L’oreal Paris and Nike have already released their technology for virtual facial makeup and virtual shoes try-on respectively. They have discovered that with the development of their own try-on system, their product return rate has decreased by 250% and sales increased by 300%. While the world enriches from the cutting edge technologies we are still left behind. There is no try-on system available in Pakistan for virtual clothes try-on. To leverage this gap and Inspired by the idea and its unimaginable potential, we have developed Pakistan's first virtual try-on system for shirts. Given a target person and a target shirt, our system would allow you to seamlessly apply the style of target garment on the target person in a seamless fashion.

Our system offers both a web application and a REST API which could be easily integrated with popular ecommerce sites like Daraz, Gul Ahmad and Eden Robe. Our business model is software as a service with painless API integration. In just a matter of 5 lines, you can have it up and running on your online store. The style model is divided into 3 main milestones namely Human Parsing, Garment parsing and Style Generation. Harnessing the power of deep convolutional neural networks and Generative Adversarial Networks, Together they make up Our Virtual try-on system which would allow you to seamlessly try out clothes virtually

Chapter 1

INTRODUCTION

Many multinational companies like Nike, Loreal are launching virtual try-on technologies to make the online shopping experience better. Despite having a billion-dollar e-commerce market, Pakistan still does not have a single website with try-on technology. This pandemic further exposed our outdated websites when a large number of orders were being returned due to size, color, or similar issues. Therefore, we present a solution to this problem by creating a first of its kind virtual shirt try-on technology in Pakistan. Now, customers can try a shirt on their body online from the comfort of their home, without going to the physical store.

This project is divided into two main parts:

1) API

Since we want to provide our product as a service, we have created an API to cater this need. An API (application programming interface) is a connection between computers or between computer programs. It is a type of software interface, offering a service to other pieces of software. In simple words, any existing platform like website, android application or IOS application can integrate our virtual shirt try-on technology without any complexity. In fact, we have integrated this API in a website just to demonstrate how easy of a task this is.

This API requires an image of the person and an image of the shirt he/she wants to try on. Then, these images are passed to our try-on model and an output image is returned to the user, with the new shirt on his/her body.

2) Try-on model

We are using different libraries of Python (programming language) for our try-on technology. These libraries are used to create a complex model that generates the desired output image. This model is very complex and cannot be explained in simple terms, without using jargons. All the details of this model are explained in the following chapters of this report.

In this report, we will explore the related work already done in this area of research. Then, we will see our model's architecture and how it works. Then, we will analyze our model's and API's performance under different testing. We will also have a look at the implementation and results. Finally, we will present our conclusion and our future goals for related to this project.

Chapter 2

LITERATURE REVIEW

Virtual Try-on methods are divided into 2D based [1,2,3,4,5] or 3D based methods [6, 7,8,9,10]. All the 3D based methods require 3D human body models and 3D datasets, which are very difficult and expensive to produce.

2D image based virtual try-ons can be divided into two broad categories, parser-based and parser free methods. The major distinction is if a human parser is required for inference. For warping parameter estimation, parser-based approaches use a human segmentation map to mask the garment region in the input person image. The masked human image is combined with the distorted clothing image, which is then sent into a generator to create target try-on images. Most techniques [3, 4, 11, 13, 14, 15] parse the person image into many pre-defined semantic regions, such as the head, top, and pants, using a human pre-trained parser [12]. [11] also adjusts the segmentation map to fit the target clothing for better try-on image production. Final try-on image production is based on the altered parsing result, as well as the twisted clothing and masked human image. Because these methods rely on a parser, they are susceptible to poor human parsing outcomes [16, 17], resulting in incorrect warping and try-on results.

In the inference stage, parser-free methods [16, 17] just take the human picture and the garment image as inputs. They were created expressly to counteract the detrimental consequences of poor parsing results. Those methods usually start with a parser-based teacher model and work their way down to a parser-free student model. [17] developed a pipeline that uses paired triplets to condense the garment warping module and try-on generating network. [16] improved [17] by using cycle-consistency for improved distillation.

Our method is also a parser free method because parser-free methods [16,17] have produced far better results than parser-based methods [3, 4, 11, 13, 14, 15]

Chapter 3

PROBLEM DEFINITION

According to one estimate, the ecommerce market is worth $10,361 billion. From 2022 to 2027, it will expand at a CAGR of 14%. It has also played a significant role in Pakistan. Pakistan has surpassed India as the 46th largest ecommerce market. The market is worth $4 billion dollars in total. Around 19 percent of Pakistanis have purchased at least one thing in their lifetime. From 2022 to 2025, it will also expand at a CAGR of 7%. The fashion sector, which accounts for 70% of Pakistan's revenue, is the main segment that has performed well. Approximately 20% of fashion products purchased online are returned to the shops, according to our market study.

This clearly tells us that even in this era of technology, Pakistan is lagging behind in state of the art technological solutions. The major returns in the fashion industry are due to size issues and one major concern that the customer is not able to try it on. They are unaware that whether they will look good in that piece of garment or not. They only assume these things but what if we provide a solution that helps them to get clarity about the garments they are buying.

Now a days using technological solutions can helps us to gain our customer trust by letting them try virtually and also decreases the returns of our ecommerce stotres. It helps the bussiness in both gaining profits and increasing their worth. Therefore, we aim to provide our vitual shirt try-on technology as a service to Pakistani e-commerce store.

Many multinational companies like Nike, Loreal are launching virtual try-on technologies to make the online shopping experience better. Despite having a billion-dollar e-commerce market, Pakistan still does not have a single website with try-on technology. This pandemic further exposed our outdated websites when many orders were being returned due to size, color, or similar issues. Therefore, we present a solution to this problem by creating a first of its kind virtual shirt try-on technology in Pakistan. Now, customers can try a shirt on their body online from the comfort of their home, without going to the physical store.

Chapter 4

METHODOLOGY

Virtual shirt try on is one of the hot topics these days. Everyone in the computer vision community is talking about the prospect of using AI and Generative Machine Learning to try out clothes without actually having to visit the store.

The Virtual Shirt Try On system that we developed was divided into 4main sections namely:

1. Human Parsing Module
2. Garment Parsing Module
3. Warping Module
4. Generative Module

Let's look at each section in detail.

Part 1: Human Parsing Module

Human parsing is the first section of the project where the processing starts. It receives 2 inputs. The first one is the Image of the target person and The second one is Image of the target garment. The target garment is meant to be applied to the target person. When the Human Parsing Module (HPM) receives the input, it uses a state of the art Deep Learning Algorithm to parse the person in the image. The results contain different segmentation for head, body, arms and legs. This is an important step as the results of the parsing module are used by upcoming modules to synthesise styles onto the target garment. Here's how the results look like after parsing:

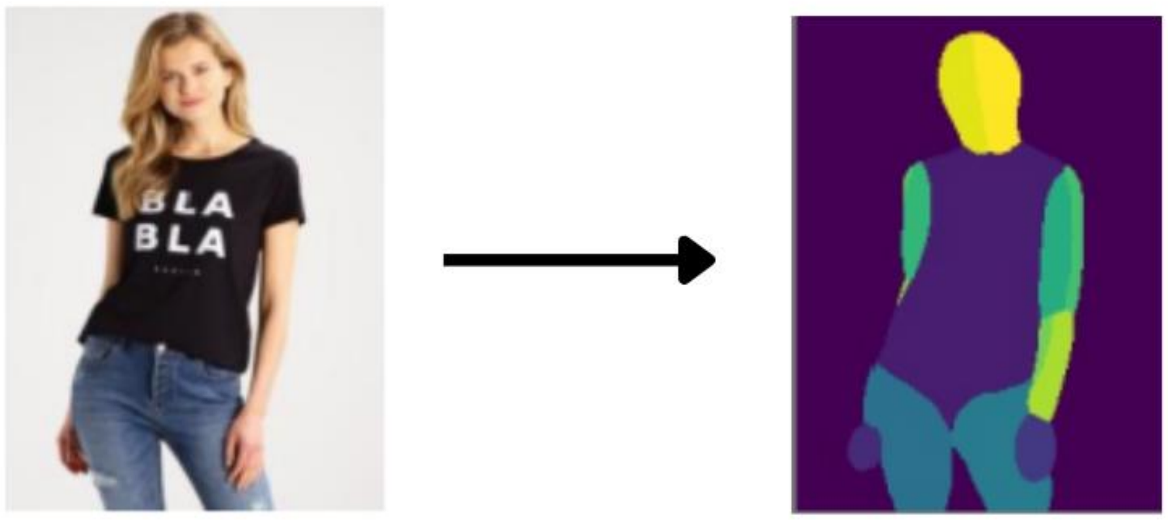


Figure 1: Human body segmentation

The human parsing is carried out by self correlation parsing that uses out of the box human parsing extractor for downstream applications. We used the publicly available self-correlation-parsing module on github (REF). It's important to note hair this gives a rough overall segmentation of the person's body. Features like hairs, occlusions and shirt patterns are not part of this section. Those details would be handled in the subsequent modules step by step.

Tools and technologies:

Pytorch and OpenCV were used to carry out the processing. OpenCV is an image processing library which comes pre-built with lots of handy image processing and computer vision algorithms which makes working with images really easy. Pytorch is a deep learning framework built by facebook to build, train and test deep neural networks of any kind. The image was red and converted to (192, 256) size as this is the size our model expects during training and inference time. The underlying technology of self correlation is based on detectron2 offered by facebook [27], detectron2 is a pytorch based deep learning architecture for image segmentation.

Algorithms used:

A number of algorithms were used in this section. Detector2's deep convolutional neural network was used to segment different regions of human body. Then self correlation implemented in python and pytorch was used to use those segmentation to give the parsed output as the end result.

Part 2: Garment Parsing Module

Now that we have the parsed results for the target person, it's time to go ahead and do the same for Target garments. It's also an important step since the end results should have the styles of target garments applied suitably to give realistic results. In the first step we will extract the shirt off the image. We won't get the shirt segmented at the time of inference so we had to find a way to segment the shirt in the given image. This particular section utilises a UNet architecture based publicly available ImageSegmentation python library which segments the shirt in the given image. There is another reason to perform image segmentation which is to find the binary masks of shirts. Binary masks of shirts are important because it is required by the subsequence modules to seamlessly transfer styles on the target person. Binary mask is nothing but an image with every pixel black other than the concerned areas (which are colored white). Following is a sample of binary mask:

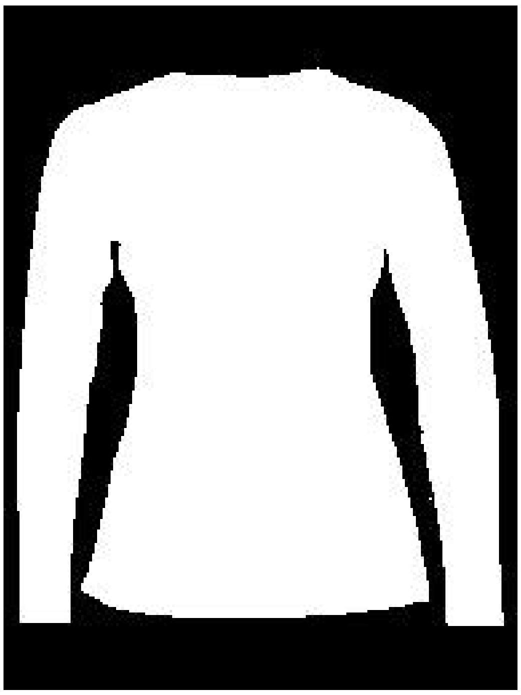


Figure 2: Binary mask

Once we have the binary masks, we would need the flow-maps so that the Generative Module can use those flow maps to apply styles. This part is tricky. It's not easier to find out flow maps. These are also called Appearance flows. Appearance flow refers to the 2D coordinate vectors which indicate which pixels in the source image can be used to synthesise the target image. The idea is to find out which image pixels matter for the target image. Appearance flow was first introduced by [28] to synthesise images of the same objects observed from a different angle or a different viewpoint. We used the publicly available code for Appearance Flows offered by [28]. Here's what the end results look like for appearance flows:

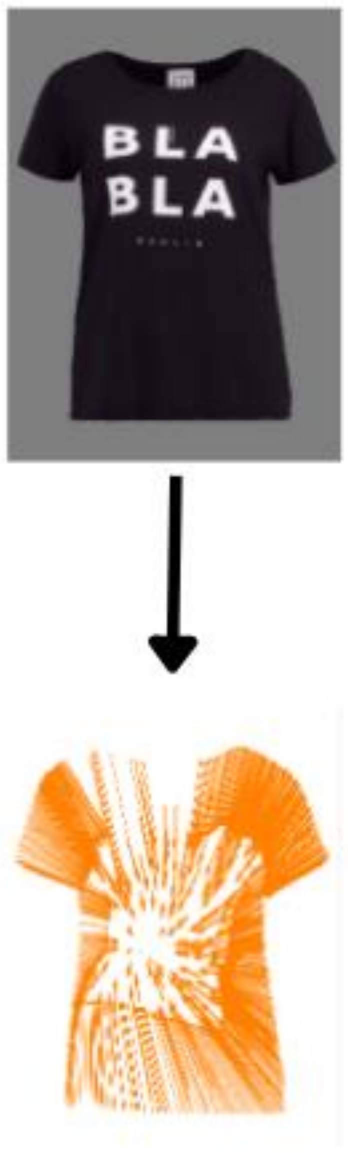


Figure 3: Appearance flow

Appearance flow vectors are significant to warp clothes on the target person, since the angle and position of clothes could change, appearance flows help us find which pixel in the target shirt should onto which position on the target person.

Tools and Technologies:

Pytorch was used to perform the deep learning operations and OpenCV was used for image manipulation tasks. OpenCV is an image processing library which comes pre-built with lots of handy image processing and computer vision algorithms which makes working with images really easy. Pytorch is a deep learning framework built by facebook to build, train and test deep neural networks of any kind. We used an open source code for cloth segmentation. The idea was inspired by U2Net architecture which uses UNet based architecture to segment the image. The particular model was trained on 45k clothing images and gives state of the art results.

Algorithms:

U2Net and UNet architecture were used for Image Segmentation. Once we had the segmented Image which was in colored form, we used OpenCV adaptive binary image thresholding to convert it into a binary mask. FlowNet neural network architecture was used to find the appearance flows for the upcoming warping module.

Part 3: Warping Module

Image warping is the process of transforming one static image into another static image. That’s a very general definition. In our case Image warping was required to transform the straight angled garment photo according to the dynamics of the target person. Image warping for simpler shapes can simply be carried out easily in OpenCV by supplying the affine transformation using the corner points. In our case, Garments are not ordinary shapes, so we couldn’t use them. Therefore we made use of Appearance flows so that our warping module could appropriately warp clothes as per the target person. It’s also important to note that the Appearance flows also make use of Feature Pyramid Networks (FPN). Feature Pyramid Networks are used to find the features in an image at different scales. It helps with better and more realistic warping.

The input to the Warping module is the Target shirt, Appearance flows, Feature pyramids, and the target Person. It uses the appearance flows and features pyramids on to the target shirt to give the warped results. Let’s look at the output of this warping module:

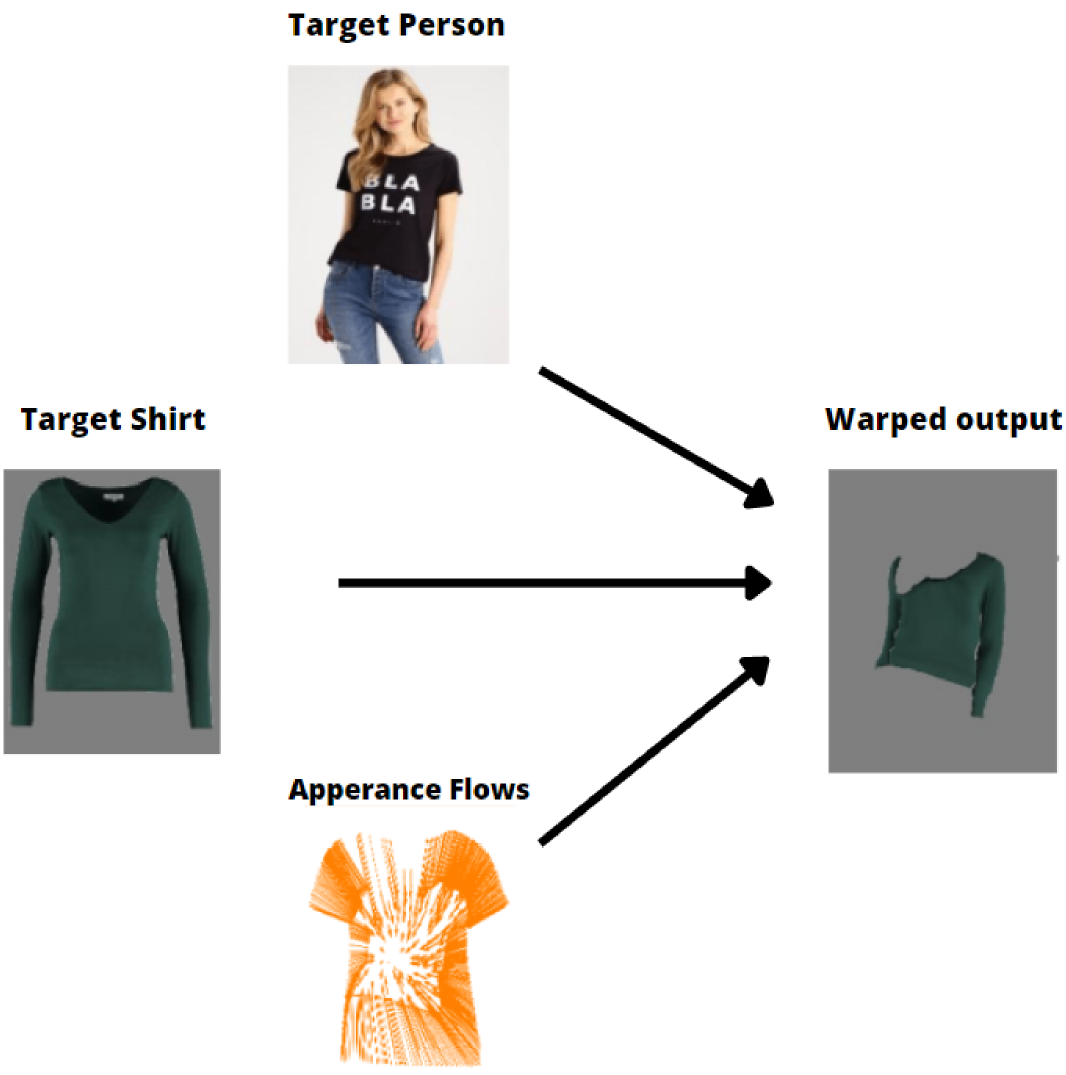


Figure 4: Warping module

Tools and technologies:

Pytorch and OpenCV were used to carry out the development. OpenCV is an image processing library which comes pre-built with lots of handy image processing and computer vision algorithms which makes working with images really easy. Pytorch is a deep learning framework built by facebook to build, train and test deep neural networks of any kind. The Appearance flows generated in the previous step were combined with Feature Pyramids Network. Feature Pyramid is a Convultional neural network that finds features at different scales. It improves the warping results. Another Convolutional neural network was created consisting of 8 Conv2d layers called **AFWM** , It was designed to create warped images using target image, target shirt and the appearance flows.

Algorithms:

Feature Pyramids Network (FPN) was used to get the feature pyramids for the given shirt image. A convolutional neural network (AFWM) was built in pycharm to apply the warping. OpenCV and Numpy were used to pass the image between the different networks.

Part 4: Generative Module

Now that we have all the data, and features at hand, it's time to combine those results and synthesise the target image onto the target person. Generative Models were used in this case to get the job done. Generative models are a class of Neural Network models where instead of classifying or detecting something, we generate something based on our criteria. Generative Modules have different types like variational auto encoders , Generative Adversarial Networks, etc…

We developed two neural networks, a Generator and a Discriminator. The job of the Generator is to generate samples that look like our end results and the job of the discriminator is to find out whether it's real one or fake/generated one. The two neural networks battle against each other and in doing so improve as well. Great amount of care needs to be ascertained so that the amount of learning is the same for both of the models. If any model overpowers the other with a big margin, then the learning diminishes and the model doesn't train well. Both the Generator and the Discriminator were Convolutional neural networks written in Pytorch. Initially the results are not good but with every epoch the model improves and the results look more realistic. Fridrich Inception Distance (FID) was used as the loss function to monitor the learning. FID is the standard loss function for the evaluation of the GAN models. The model was run for a total of 200 epochs on the AWS Cloud GPU. Following are the results after synthesising with GAN:

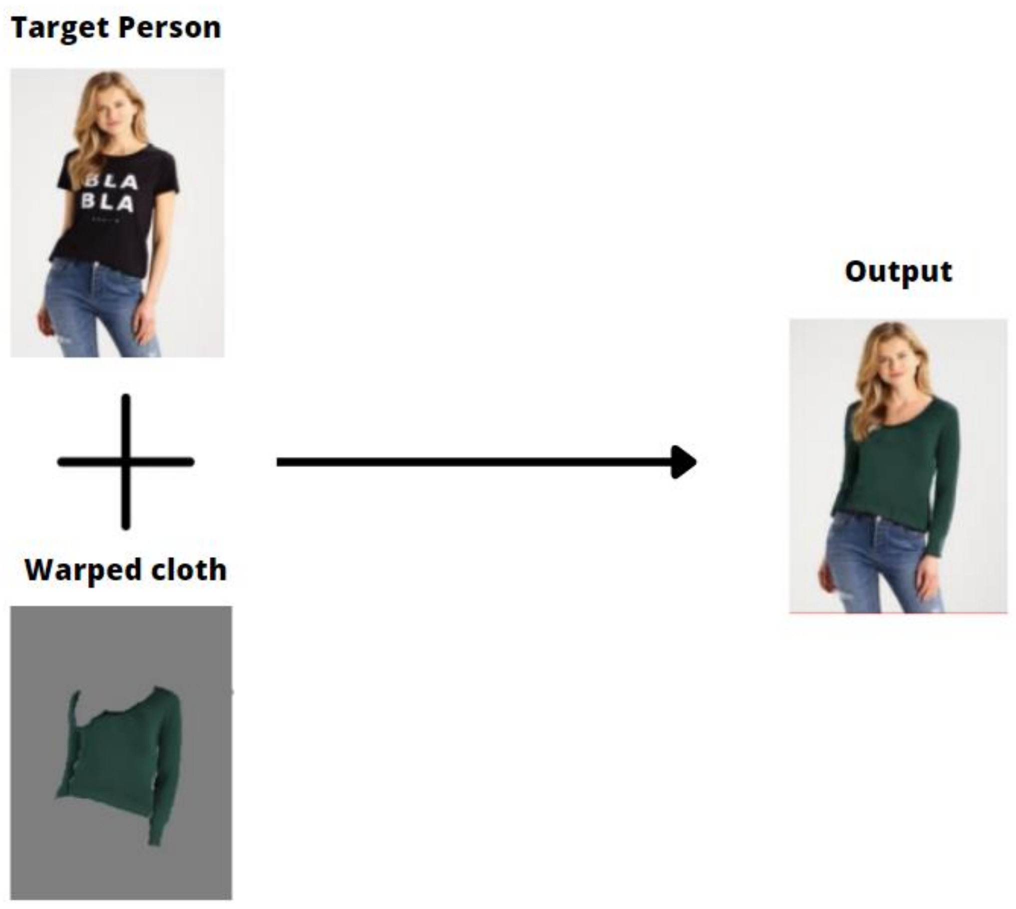


Figure 5: Synthesize with GANs

Tools and technologies

Pytorch and OpenCV were used to carry out the development. OpenCV is an image processing library which comes pre-built with lots of handy image processing and computer vision algorithms which makes working with images really easy. Pytorch is a deep learning framework built by facebook to build, train and test deep neural networks of any kind. GAN architecture was used for image synthesization, the style was inspired by human parsing based architectures which use human parsing to generate realistic results.

Algorithms

A generative model GAN (Generative Adversarial Network) was used for the seamless transfer of styles from the warped shirt onto the target person. Pytorch was used to develop the architecture of the GAN model. OpenCV was also used for reading images, carrying out some image processing tasks and then finally saving it.

Chapter 5

DETAILED DESIGN AND ARCHITECTURE

5.1 System Architecture

Our Virtual Try On System is divided into 4 main sections. Each carries out their own tasks. The 4 main sections are :

- Web application
- Image Segmentation Model
- Generative Style Model
- Dataset

The goal of the project is to make Pakistan's first virtual shirt try on system which is able to seamlessly transfer from a piece of a garment onto the target person. The system would take the input of the target person along with the image of the target garment to generate a realistic photo of the person wearing that shirt. It would have extensive applications in the field of ecommerce, online shopping as well as clothing retail outlets. People can completely avoid the hassle of trying out 10s of shirts one after another just to find out which suits them better. It's also worth noting that this is the first virtual try-on system in pakistan.

The project will be developed as a web application which will allow its users to upload their target shirt along with the target photo of theirs, the system would take those images as input and using our deep learning model deployed on the cloud, it would return the resulting photo with the style of the shirt applied. The project would also offer a rest API on subscription basis which would allow sites like Daraz, Gul Ahmed and ELO to allow users to try out their branded clothes on their website with easy integration.

Our project will make use of advanced deep learning techniques, along with computer vision techniques to seamlessly transfer style from garment on the target person. Based on the input of the target person and the target garment, the system would generate results. Our system heavily uses pytorch for the development of the neural network model. Pytorch offers a lot more customization compared to tensorflow which is why pytorch was chosen. Some of the open source libraries that we made use of were also written in pytorch which added to our choice. The project is available both as a website as well as a subscription based REST API. Let us look at the series of steps that we followed for the development of our Virtual Try On System:

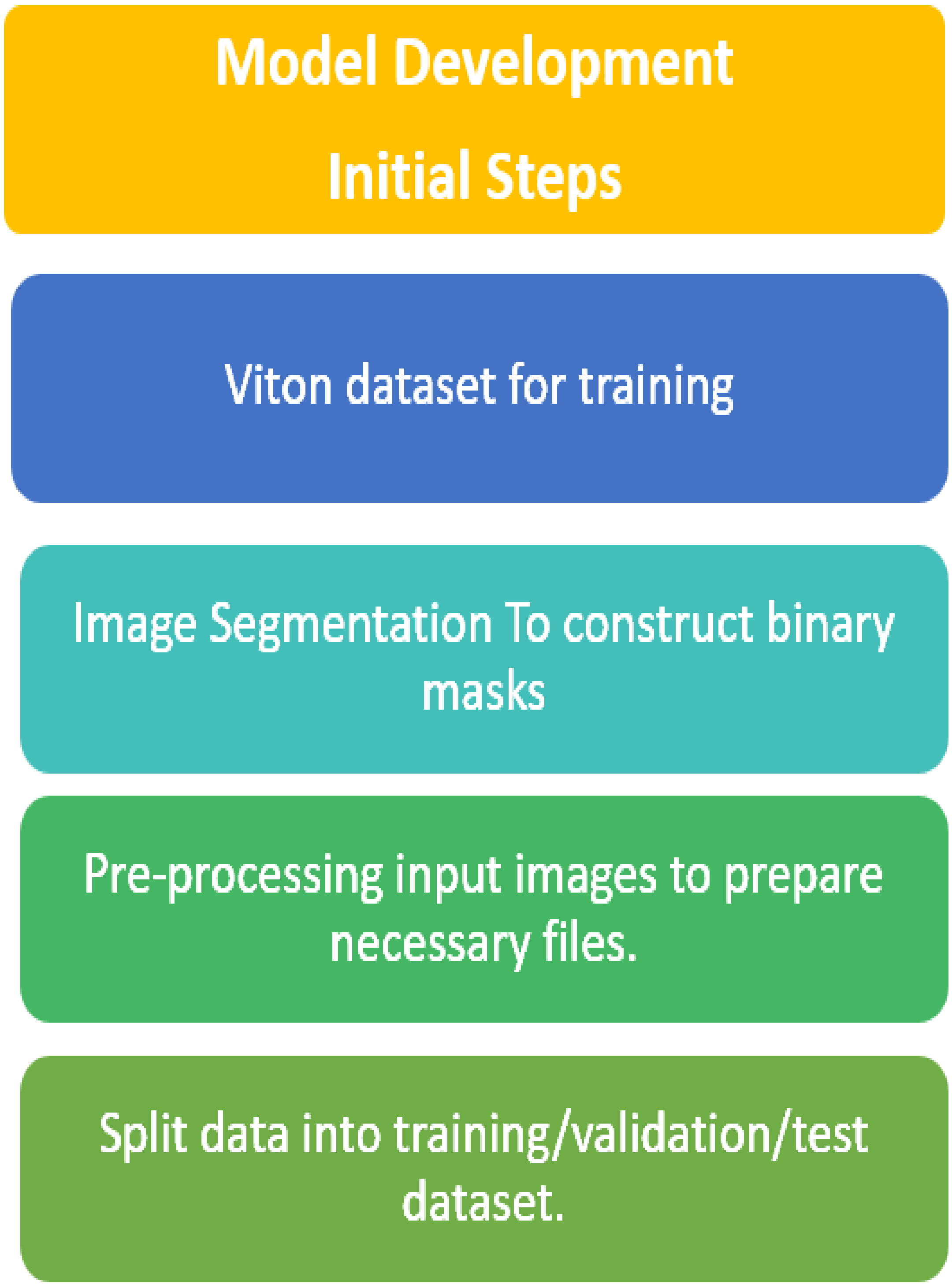

Model Development
Initial Steps
Viton dataset for training
Image Segmentation To construct binary masks
Pre-processing input images to prepare necessary files.
Split data into training/validation/test dataset.

Figure 6: Model development initial steps

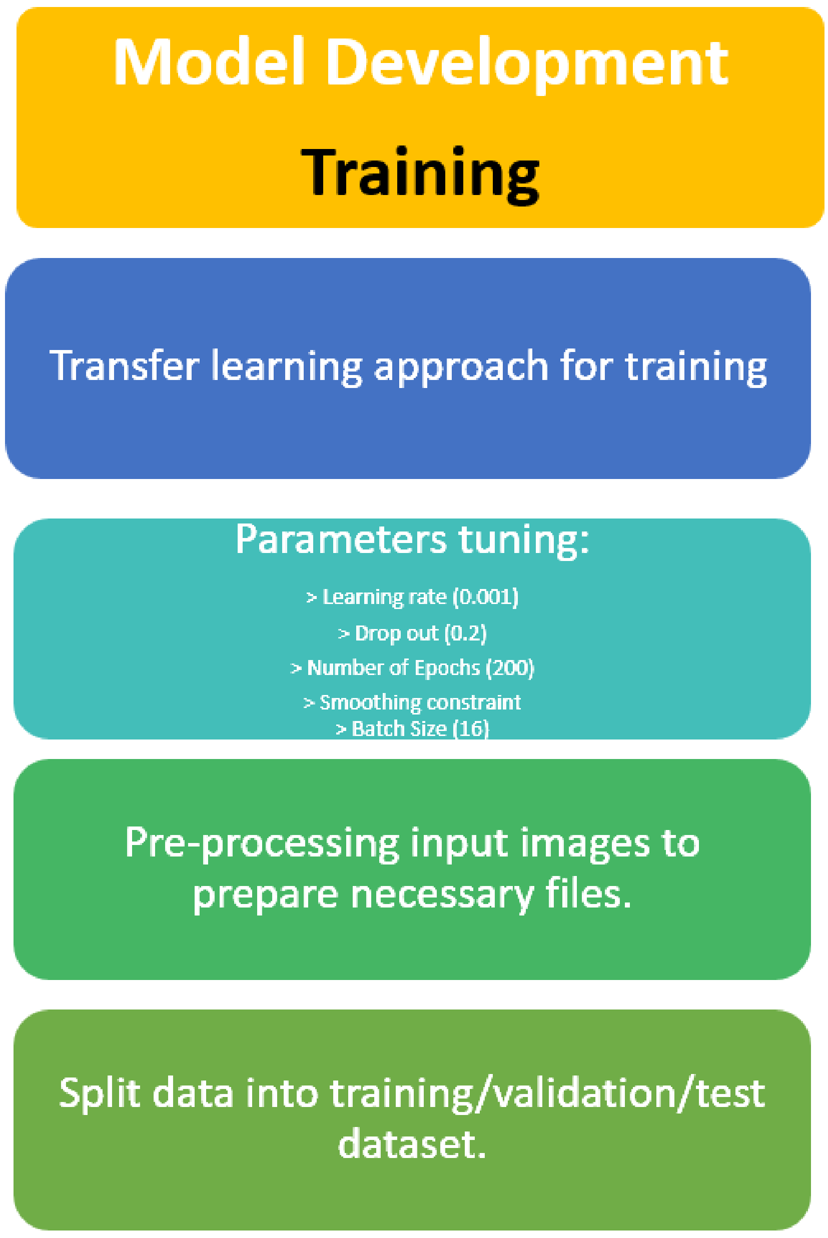


Figure 7: Model training steps

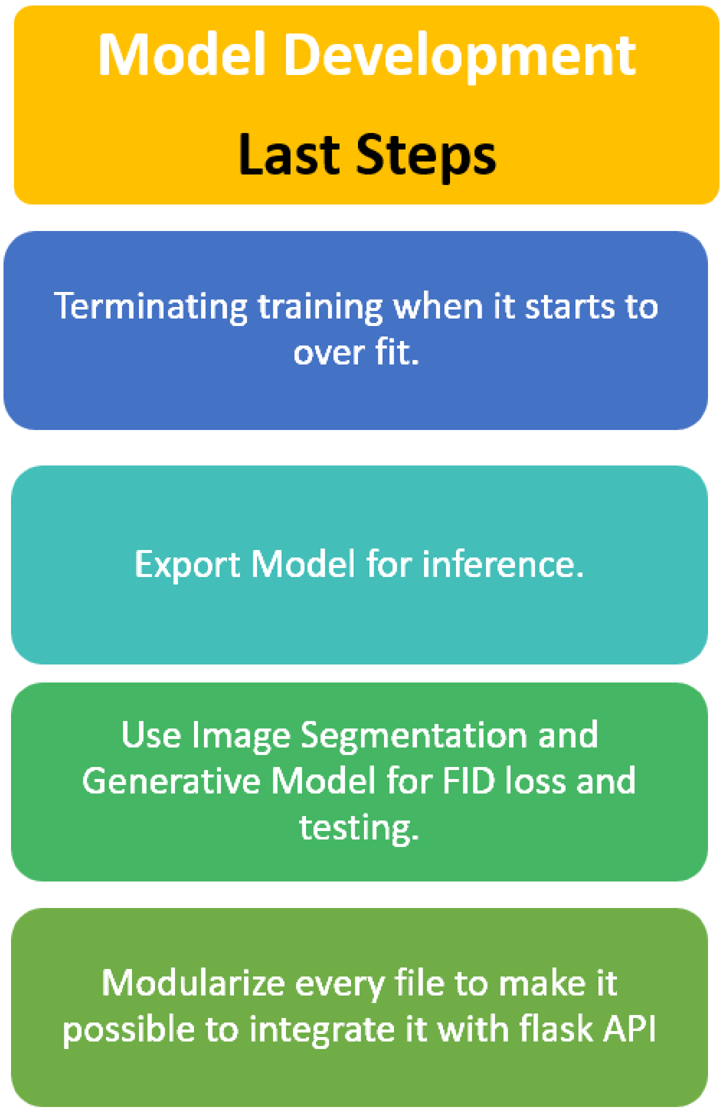

Model Development
Last Steps
Terminating training when it starts to over fit.
Export Model for inference.
Use Image Segmentation and Generative Model for FID loss and testing.
Modularize every file to make it possible to integrate it with flask API

Figure 8: Model Development last steps

5.1.1 Architecture Design Approach

All high level components and their interaction are explained below.

Web Application

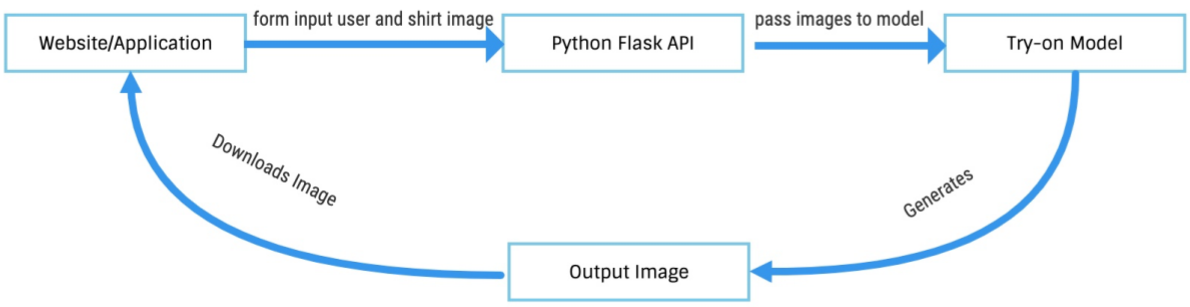


Figure 9: Flow of system

Flask Library is used for creating REST API, so that it can be called from any platform (like website, android, IOS). Our main goal is to make this Try-on service as easy as possible to integrate with existing websites and apps. Therefore, we have developed a very simple API that can be integrated in a very short time without any complexity.

Once the API is integrated, the user uploads his/her image and choose the shirt he/she wants to try on. Then, our API is called, and images are enclosed in a http multi-form post request. Those images are passed as a parameter to our try-on model in the backend and our model generates the output image, which is sent back to the user.

Since we are currently in development phase, we are using a temporary server called Ngrok. The API is hosted on the developer server "ngrok." Ngrok creates a dynamic link that is forwarded to the application through a developer server. Temporary public URLs for the development web server are generated using the ngrok utility. Ngrok is an impressive tool for that establishes a tunnel between a URL on the ngrok.io domain and a local computer model.

Image Segmentation Model

Our image segmentation model is composed of two main subsystems.

1. Image Segmentation UNet Model

This is an implementation of publicly available model that segments the Clothes in human body images. This is a generic model which segments the human into 3 sections, upper body clothing, lower body clothing and full body clothing. Since as a part of this project we were only concerned with upper body clothing so we ignored the rest of the segmentation results and kept only the upper body results which would correspond to shirt segmentation. Unet model was used in the implementation of this model. It comprised 18 Conv layers to segment the results. It accepts any image (be it jpg, png or bmp) and returns the results. We used the implementation of [] and their publicly available library to carry out the segmentation. Its significance in our project was to segment shirts off the human and use it to apply styles. Now segmenting shirts was not the end result we desired, it was just means to an end. We actually needed to find the binary masks of the target shirts. The output of this section was passed on to the next section for further processing.

2. Binary Mask Finder

This section was all about image processing. Segmentation results received from the previous models were passed on to the binary mask finder. OpenCV and its built-in algorithms were used to carry out the processing. We first of all converted the RGB image to grayscale. Then we applied OTSU's adaptive binary thresholding to segment the shirt from the background. The output of adaptive thresholding is a binary image or in other words a binary mask as we desired where all the shirt pixels are white and the rest of the background is black. Thresholding operations often leave out some extra artefacts which were then taken care of by smoothing the image using median thresholding. The result was a nice and decent binary mask with no extra artefacts.

Generative Style Model

Our generative module is composed of the following main subsystems:

1. Human parsing:

In order to apply styles on to the target person, we needed to take the appearance of the target person into consideration and this was all this section was about. This section would parse the human/person in the image into different regions like arms, head, clothes, legs etc… For the implementation and integration of this section we used a publicly available self correlation multi human parsing [29] which gave the state of the results.

2. Garment Parsing

Just like it’s important to take the dynamics of the person into consideration, it’s equally important to take the properties of shirt into consideration so that the style can be applied appropriately. This was the job of this section i.e parsing garments (which in our case is shirt). This section would make use of Feature Pyramids to extract the features of the shirt at multiple scales followed by the Appearance Flows using flow networks. Flow networks and appearance flows indicate which pixel in the source would make an impact on the pixel in the target image. This creates beautifully warped end results.

3. Image warping

The end results we meant to achieve was to warp (change) the target shirt according to the target person. This was the job of this section. It would take the feature pyramids, appearance flows and the inputs from the previous section and using a neural convolutional network it would warp the shirt according to the person. This result would then be applied by the generative model in the next section to get the final results.

4. Generative Model

Once we have the warped results, we would supply the warped results along with the input images to this model. It would use a generative neural network model To apply the styles from the target shirt on to the target person. Generative model is a supervised convolutional neural network model with FID loss. It’s job is to generate the end results with the styles applied. It would output a 192 x 256 dimension image.

Dataset

1. Data Collection

The first step in the development of any deep learning model is to gather and assemble the data since it’s very important for any supervised model. Luckily for us, try-on models are a hot topic in the field of computer vision. The Computer Vision and Pattern Recognition community has released a publicly available dataset consisting of 14000 training images and 4000 test images. The dataset is called as VITON and greatly helped us with the development of our model since we were able to find the labelled dataset .

2. Image pre-processing

Since the data was already cleaned by the VITON dataset owners, we didn’t have to run any pre-processing steps to pre process the dataset. The images were 192 x 256 dimension in .jpg format with standard RGB channels. The images were also centred with refined binary masks and labelled outputs.

3. Train/Test/Validation

We used the standard practice to split the data into training / validation and testing. The training and validation dataset were split at 80:20 ratio. While the test set was already there for us with 4000 samples.

5.1.2 Architecture Design

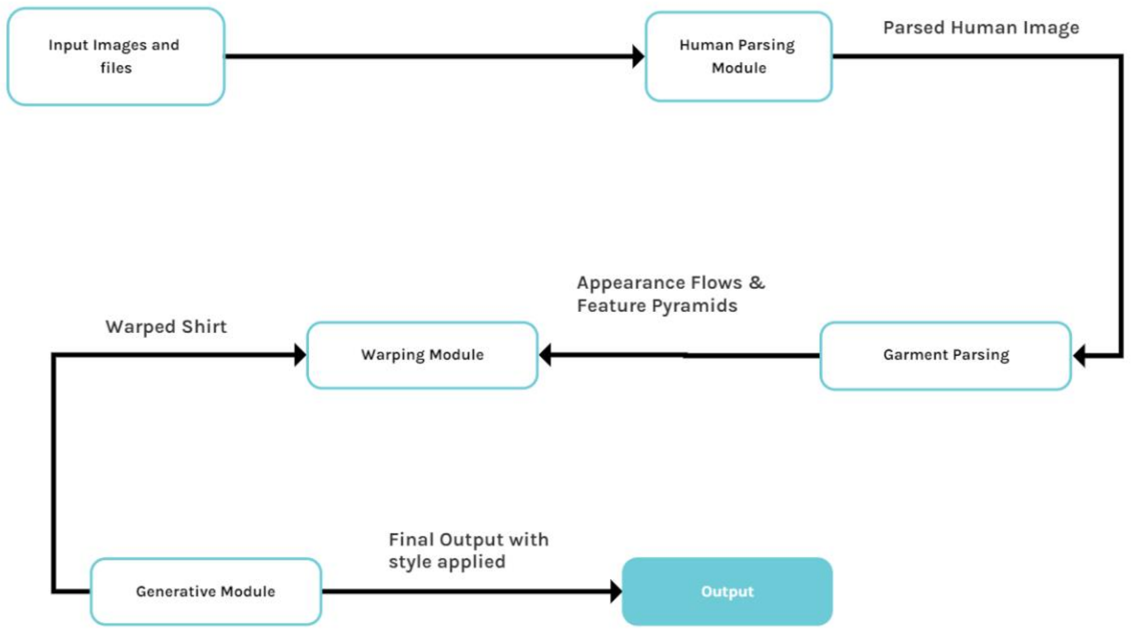


Figure 10: Architecture design

The figure depicts the overall architecture of our project. The system has 3 main modules namely Web application, Image Segmentation Model and the Generative Style model. Our project would make use of advanced machine learning and deep learning practices along with computer vision and image processing algorithms to deliver the results. The system would have both a web application as well a REST API which would be available to sites like DARAZ, ELO and GUL AHMAD on subscription basis. Users can simply upload the image of the target person and the target shirt to our system and our system would apply the styles and download the results back to the user's computer with the styles applied seamlessly.

5.1.3 Subsystem Architecture

Image Segmentation & Preprocessing Architecture

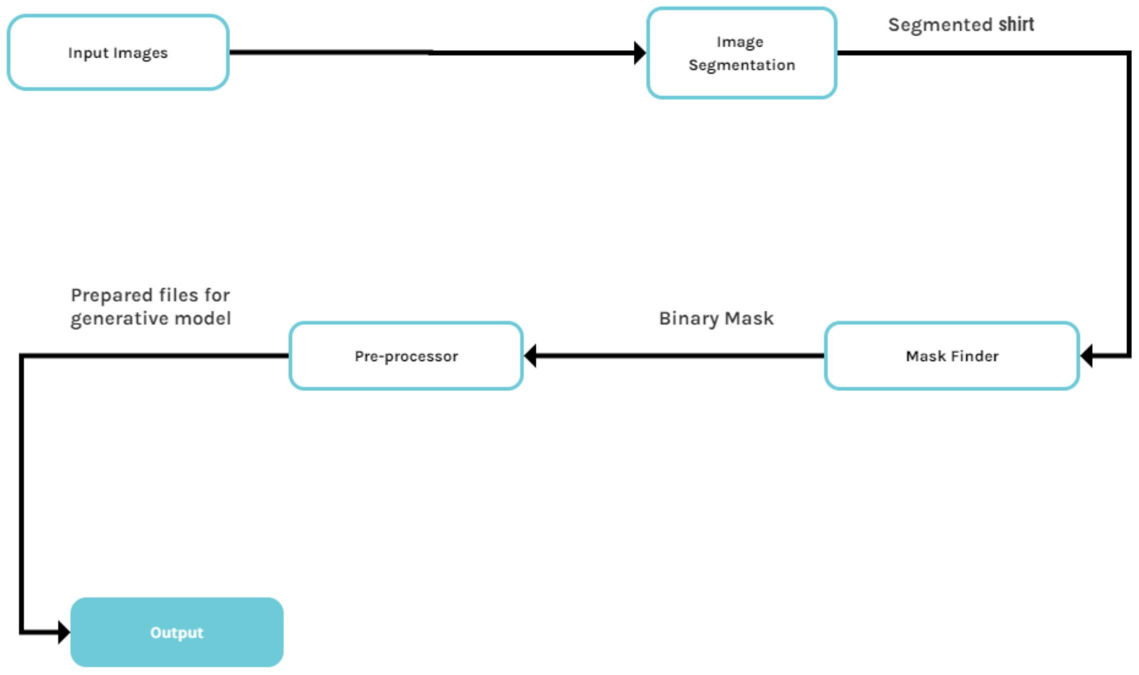


Figure 11: Image segmentation and preprocessing

This section would form the initial basis for the project. The job is this module is to appropriately create the necessary files which are required by the upcoming Style transfer module which would transfer style onto garments. The input to this section is the target person image along with the target garment image. Now it makes to have carefully tailored images during the training session but when it comes to inference or testing time, we don't have much control over the image. So it's important to make sure that the input files are converted into appropriate format for the generative module to apply the styles.

First of this module would take input images of the target person along with the target garments, it would then use image segmentation to segment off the entire image. The shirt shirt would work well for humans but doesn't make much sense to computers digitally, therefore the segmented shirt is passed on the mask finder which would construct a binary mask for the target shirt such that the pixel where there is a shirt is white and the rest of the image is black. The binary mask carries great importance and henceforth passed on to the next section. Then comes the pre-processing section which takes all the data and places it into appropriate files and folders since deep learning models need the data in a proper format and proper locations. The files are placed in 4 different folders.

- The target person image is placed in test_img folder
- The target garment image is placed in test_shirt_img folder
- The binary mask is placed in test_edge folder
- The cloth segmented image is placed in the test_clothes folder.

The deep learning model expects the images to be in those appropriate folders. Once the images are placed appropriately, we are done with this section.

Generative Model Architecture

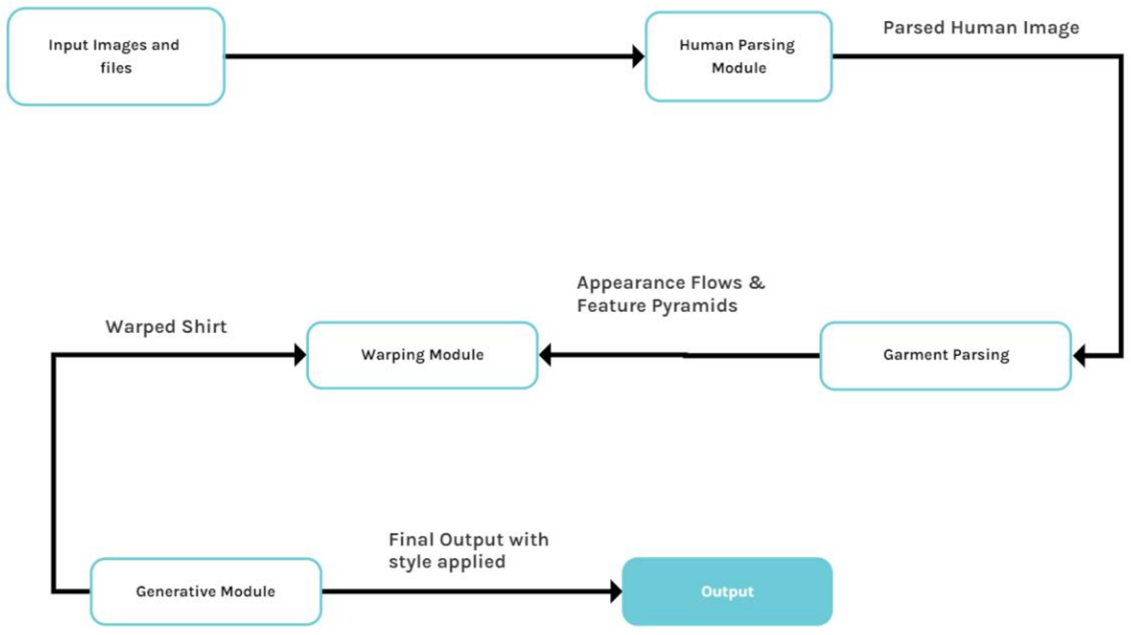


Figure 12: Generative Model Architecture

This module of the project is responsible for the seamless transfer of style from target shirt to the target person. It consists of a number of machine learning models to carry out different sub tasks. All of the models are convolutional neural networks, some of them rely on open source frameworks and libraries to get the job done.

First of all the data is passed on to the human parsing module, which is responsible for parsing the person present inside the image. This doesn't give us all the details since every

feature is extracted step by step in their own sections. The importance of human parsing is that it is required by the generative module and warping module to carry out their computations. Human parsing helps the model understand, which region contains shirt and which region contains arms and so forth. This helps the model to generate realistic images. After that the input image is passed to the garment parsing section which parses the garment/shirt present inside the image, constructing a feature pyramid network along with its appearance flows. This is an important step as it is required by the warping module to appropriate warp the target shirt on the target person. After that the feature pyramids and the appearance flows along with the inputs are passed to the warping module which warps the target according to the person's body. When this is done the warped results are passed to the generative module. The generative module then uses the deep generative model to construct a photo realistic image which has the style of the shirt applied.
Here's the high level system architecture diagram .

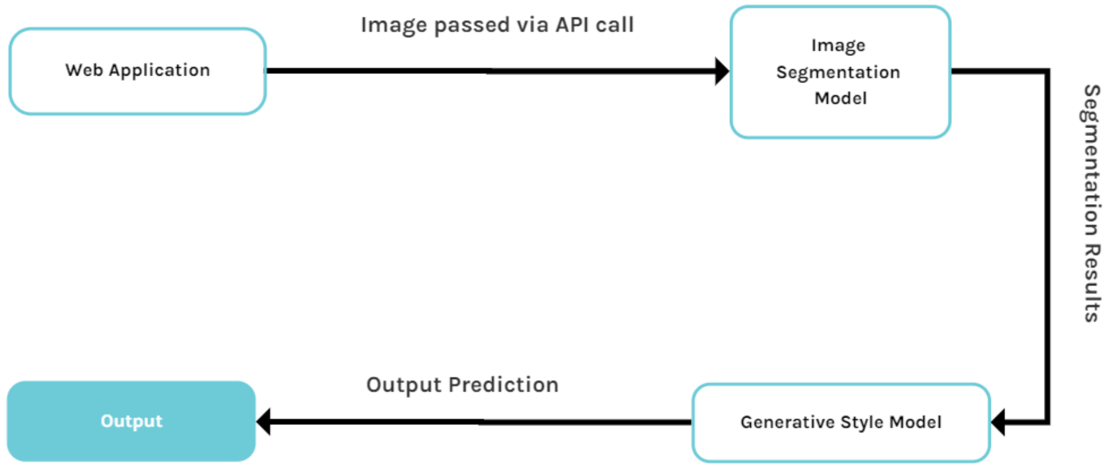


Figure 13:High level system architecture diagram

Dataset

Let's explore the dataset that we had at hand for the training of our deep convolutional model. The dataset was a publicly available Viton Dataset. It's the standard benchmark dataset used by everyone working on the shirt on projects. It was published in the computer vision and pattern recognition 2017 and since then it's publicly available for everyone working on such problems. The dataset consists of 15, 320 training images along with 4370 test images. Each image is 192 x 256 pixel wide with 3 standard channels (R, G and B). Dataset consists of image pairs. Every image pair has 3 images namely the target shirt image, target cloth image along with the shirts binary mask. Since the data is labelled every corresponding pair has an output image where the target shirt is applied on to the target person. Let’s look at sample pair in the dataset:

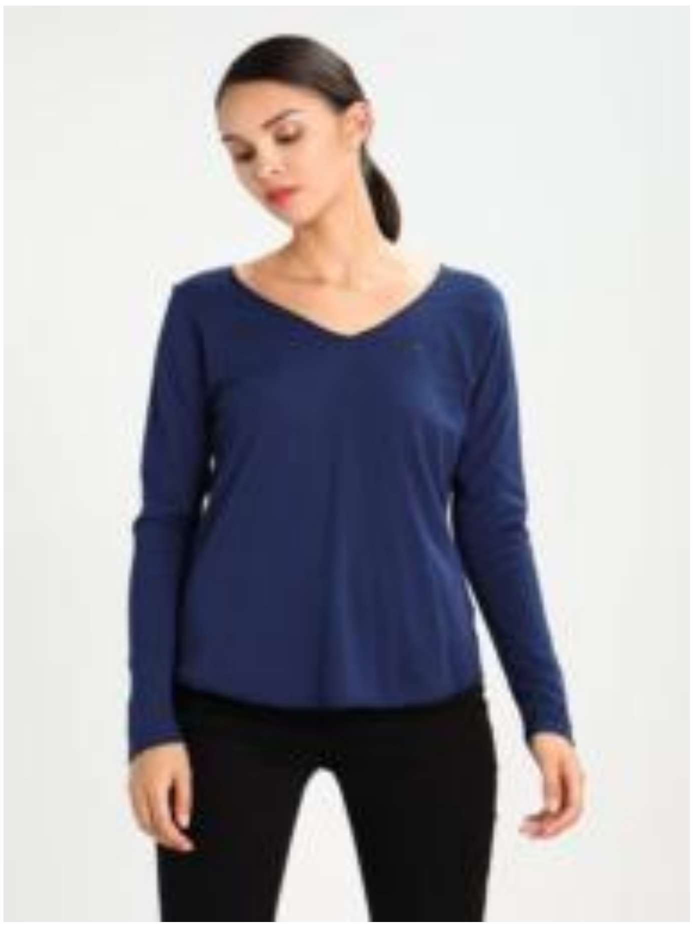


Figure 14: Target person

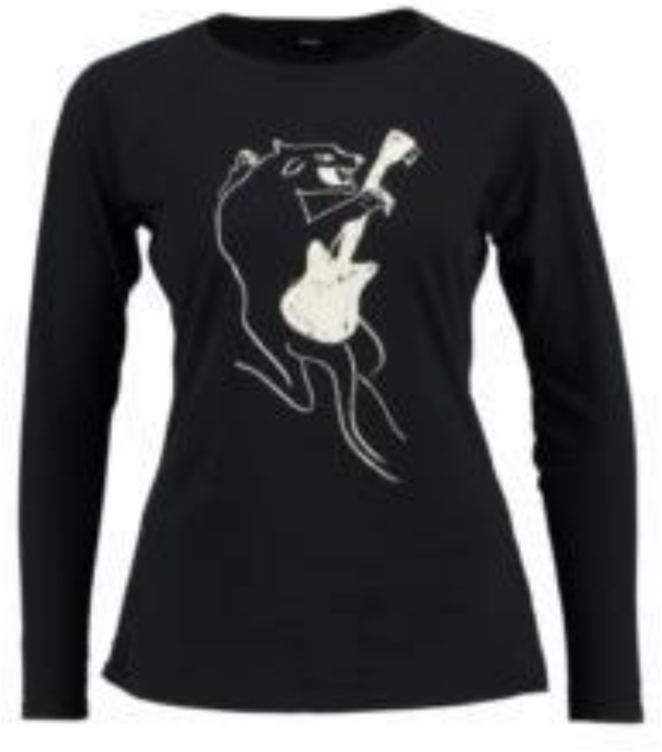


Figure 15: Target shirt

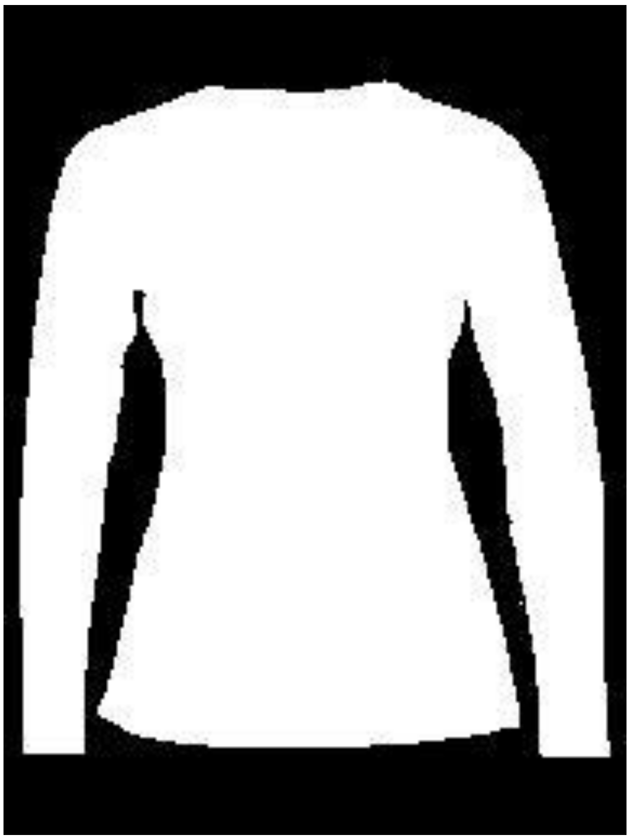


Figure 16: Binary mask

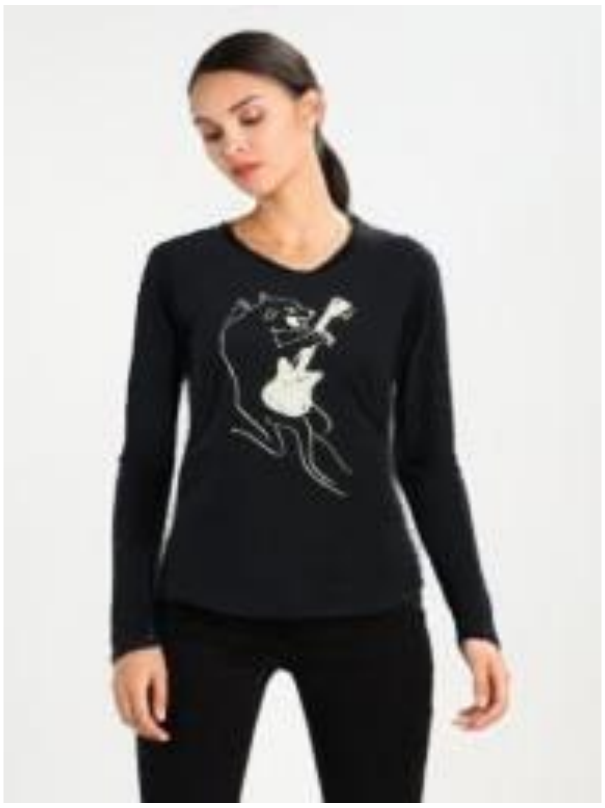


Figure 17: Output

All of these files have .jpg extension with standard R,G,B channels. There’s also a Viton HD dataset which has higher resolution images but training such a model on such HD Images requires much more computing power. We only used the Viton dataset, but given the right resources, our model has the ability to work with Viton HD without any hassle.

5.2 DETAILED SYSTEM DESIGNING

Our project makes use of a number of underlying models to generate the end results. The process starts by uploading target person and target garment photo to the website. A number of different convolution neural networks make step by step prediction and feature extraction followed by a generative model which combines everything and applies styling using all the features extracted and the inputs. Our System is available both as a website as well as a API which would be available to users on subscription basis.

5.2.1 Classification

The main components of our systems are the binary mask finder and the generative model developed using pytorch and opencv.

5.2.2 Definition

The components that have been classified above can be described as follows:

- Image Segmentation and Mask Finder:

  The job of the image segmentation and mask finder is to segment the image and ultimately find the masks of the shirts. It is very important to do that since this is required by the upcoming generative model to generate the styles. It is divided into two subsystems:

  0. Image Segmentation Model:

     This is a UNet based segmentation model which segments the shirt/garment present in the image. The job of this model is to separate the shirt off the original image so that the model can use that shirt to apply the styles on the target person. The input is an image and output is the image with shirt segmented.

  1. Mask Finder:

     This is the Image Processing based system which heavily uses OpenCV to take the image segmentation as input and generates the binary masks as output. Binary mask represents the shirt as white colored pixels while the rest of the background as black colored. The input is the segmented image and the output is the binary masked image.

- Generative Model:

  The job of the generative model is to apply styles to the target person using the target shirt. It's subdivided into 2 sub sections:

  1. Warping Model:

     This is the neural network convolutional model which would warp (change) the target shirt image according to the target person image. It would take the post and size of the person into consideration to do that.

2. Generative model:

   This is the neural network generative model which would take the warped image as an input and apply it to the input to apply the styles to the target person. The results is target person wearing the target shirt with the styles applied.

5.2.3 Responsibilities

1. The primary purpose of the website is to provide user with the ability to apply styles to the target person. It has a very simple to use interface where the user only has to upload two images, the image of the target person and the image of the target shirt. The website would forward it to the deployed deep learning model which would apply the styles to the person and download the result back to user's computer.
2. The primary purpose of the Image segmentation module is to segment the shirt present in the image and then use the results of segmentation to create a binary mask for the input shirt. The binary is created so that the generative model could make use of it to apply styles.
3. The primary purpose of the Generative Style module is to apply styles using the inputs and the features extracted. It would also extract some features of it's own like appearance flows to warp the input image. The warped image would then be used by the generative model to transfer styles from the target garment to the target person.
4. The primary purpose of the Dataset used in this project was to let the model train on the VITON dataset. It was publicly available for general public so it made the job easiser.

5.2.4 Constraints

There are a number of constraints related to the working of the model which effect our deep learning models used:

- Learning rate:
  This defines the speed of the downhill for our gradient descent learning algorithms. It was set as 0.00003 which formed a nice balance between gradient explosion and gradient vanishing.
- FPN_N:
  This indicates the N scales to use for the Feature Pyramids Network for feature extraction. This was set as N = 5.
- N_t:
  This denotes the t points around the neighbourhood of n in horizontal, vertical and diagonal position during the warping of shirts. It was set as 7.
- Channel_RGB:
  This denoted the channel to exploit for the Image segmentation module. Based on the experimentation, the R channel was heavily exploited for image segmentation results.

5.2.5 Composition

- Image Segmentation Model:
  The job of the image segmentation and mask finder is to segment the image and ultimately find the masks of the shirts. It is very important to do that since this is required by the upcoming generative model to generate the styles. It is divided into two subsystems:
  - Image Segmentation Model:
    This is a UNet based segmentation model which segments the shirt/garment present in the image. The job of this model is to separate the shirt off the original image so that the model can use that shirt to apply the styles on the target person. The input is an image and output is the image with the shirt segmented.
  - Mask Finder:
    This is the Image Processing based system which heavily uses OpenCV to take the image segmentation as input and generates the binary masks as output. Binary mask represents the shirt as white colored pixels

while the rest of the background as black colored. The input is the segmented image and the output is the binary masked image.

- Generative Model:

  The job of the generative model is to apply styles to the target person using the target shirt. It's subdivided into 2 sub sections:

  1. Warping Model:

     This is the neural network convolutional model which would warp (change) the target shirt image according to the target person image. It would take the post and size of the person into consideration to do that.
  2. Generative model:

     This is the neural network generative model which would take the warped image as an input and apply it to the input to apply the styles to the target person. The result is target person wearing the target shirt with the styles applied.

5.2.6 User Interaction

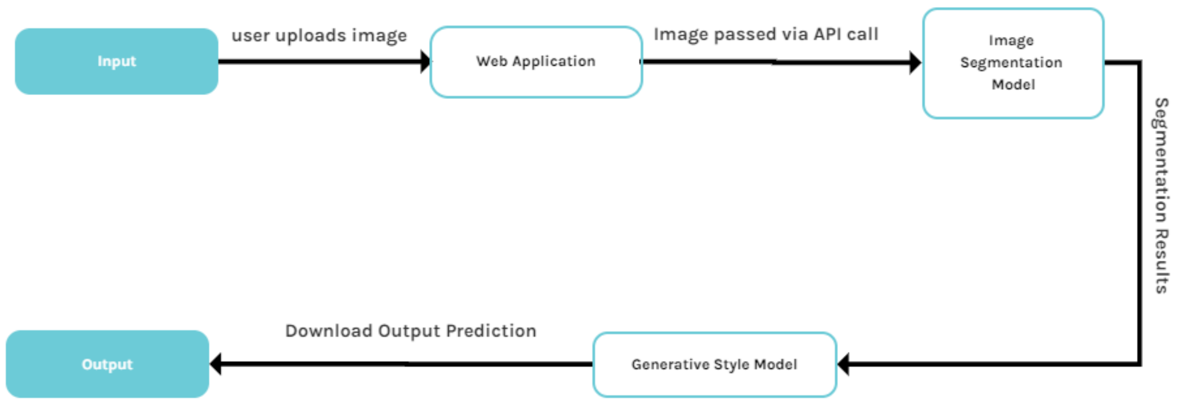


Figure 18: User interaction

The user would upload the image to the web application. It would pass it via api calls to the image segmentation module which would make shirt segmentation and generate a binary mask for the shirt. It would then forward it to the generative style model module which would calculate features for it, parse the garment, then warp the image and finally apply the styles using generative deep learning model. The end results would be the styles of the target shirt applied to the target person. Once the results have been generated it would be downloaded to the user's PC for further usages. The UI is very simple and offers minimal effort from the user's end.

5.2.7 Resources

- Prerequisites for training a model:
  The following list shows the supported platforms for running the model. The platform needs to have:
  1. Pytorch GPU (We used Nvidia RTX 2060 Gpu)
  2. CUDA
  3. CUDNN
  4. OpenCV 4
  5. Windows (but any platform with all the dependencies installed would work).

5.2.8 Interface/Exports

The following libraries were used in our project:

1. Website:

2. Image Segmentation Model:
   - Cv2
   - Os
   - Numpy
   - OpenCV
   - Matplotlib
   - Pytorch
   - Tensorflow
   - Pickle
   - Random

- Torch-vision
- Pickle
- Pillow-imaging
- Pip
- networks
- Anaconda

1. Generative Style Model:

- Cv2
- Os
- Numpy
- OpenCV
- Matplotlib
- Pytorch
- Torch.nn
- Time
- Tensorboard
- Seaborn
- correlation
- Tensorflow
- Pickle
- Random
- Torch-vision
- Pickle
- Pillow-imaging
- Pip
- networks
- Anaconda
- Cupy
- Math
- Re

5.2.9 Detailed Sub System Design

- **Image Segmentation Model**

  This is a convolutional neural network built in python and pytorch framework. The activation function used in the hidden layers is ReLU and softmax. This is the architecture for our mask RCNN based model

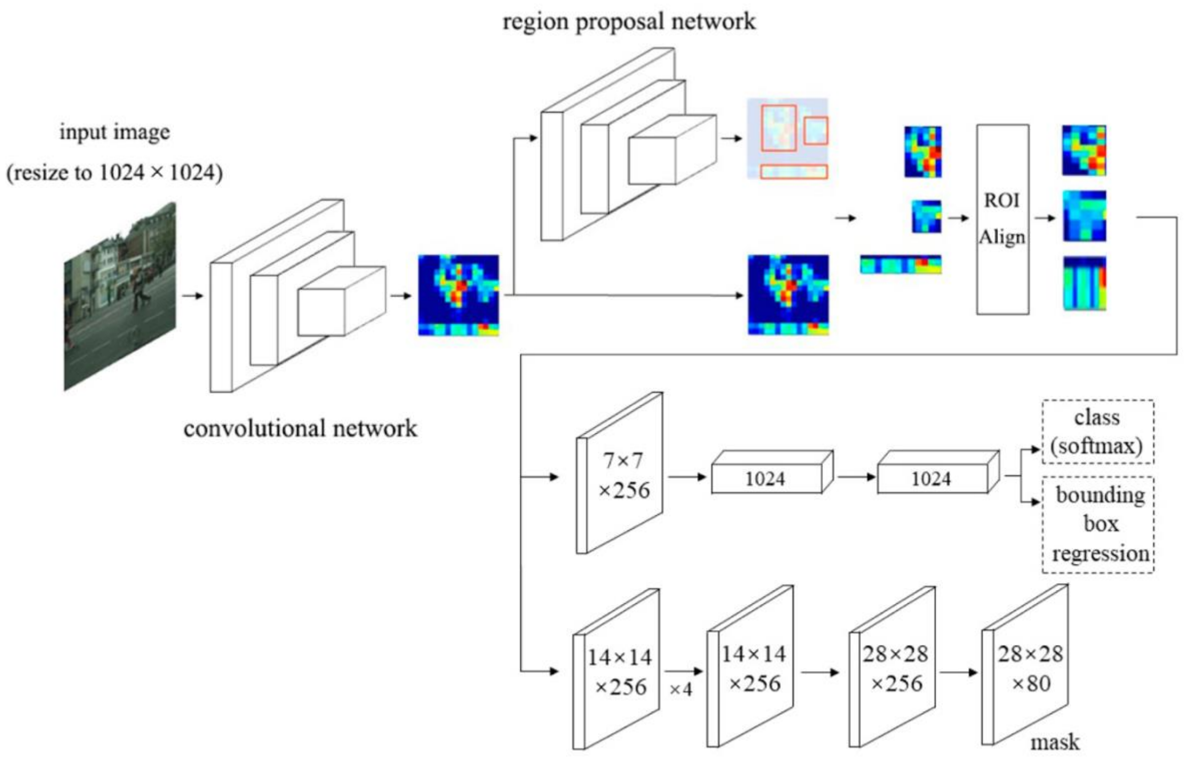


Figure 19: Mask RCNN model

Initially it uses the filter of 7 x 7 for detecting bigger and prominent features. Then it breaks the channels to 256 and 124 for further feature detection.
In the second sub layer it keeps on increasing the size of the window while keeping decreasing the number of channels from 256 to 80. For this Model we have used the implementation of a python framework [27] and simply call the library in our application for image segmentation.
Loss function:

$$L = L_{cls} + L_{box} + L_{mask}$$

Figure 20: Loss function

- **Generative Style Model**

  This model is responsible for applying styles to the target person based on the target shirt. It takes as input the appearance flows from previous layers along with human parsing and inputs. It uses a generative deep learning model to apply styles to the images. It's a GAN based architecture:

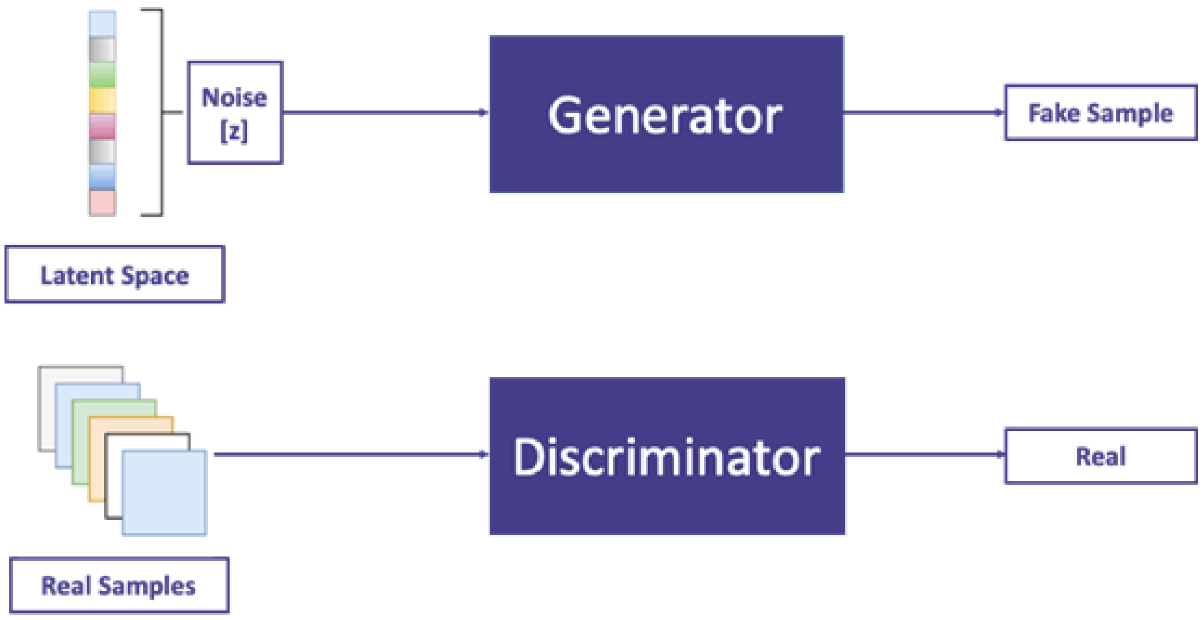


Figure 21: GAN generator and discriminator

There is one discriminator network and one generative network. Our architecture for the generative style model is inspired by this paper [16]. The generator generates the sample and the discriminator discriminates whether it is a fake or real sample. In this battle they learn together and improve style generation.

Loss

We use the Fréchet Inception Distance (FID) [18] as the assessment metric because we don't have ground-truth photographs (i.e., a reference person wearing the target apparel), which captures the similarity of generated images to real images (i.e., reference person images). A lower FID score means that the results are of higher quality.

First, we will define the formula of FID score:

$$d^2((\boldsymbol{m}, \boldsymbol{C}), (\boldsymbol{m}_w, \boldsymbol{C}_w)) = \|\boldsymbol{m} - \boldsymbol{m}_w\|_2^2 + \mathrm{Tr}\big(\boldsymbol{C} + \boldsymbol{C}_w - 2\big(\boldsymbol{C}\boldsymbol{C}_w\big)^{1/2}\big)$$

Figure 22: FID formula

We follow these steps to calculate the FID score:

- Extract the feature vectors of actual and false images using the Inception V2 pre-trained model
- Calculate the mean of the feature vectors obtained in step 1 by feature.
- Create the feature vectors' covariance matrices — C, C w
- Calculate the trace (the sum of the elements along the square matrix's major diagonal) of (C+C_w-2*(C*C_x)^1/2).
- Calculate the squared difference of the calculated mean vectors in step 2 — ||m-m_w||^2
- Finally, combine the results of steps 4 and 5.

Chapter 6

IMPLEMENTATION AND TESTING

The Figure below demonstrates the complete technology stack used in our virtual shirt try-on project.

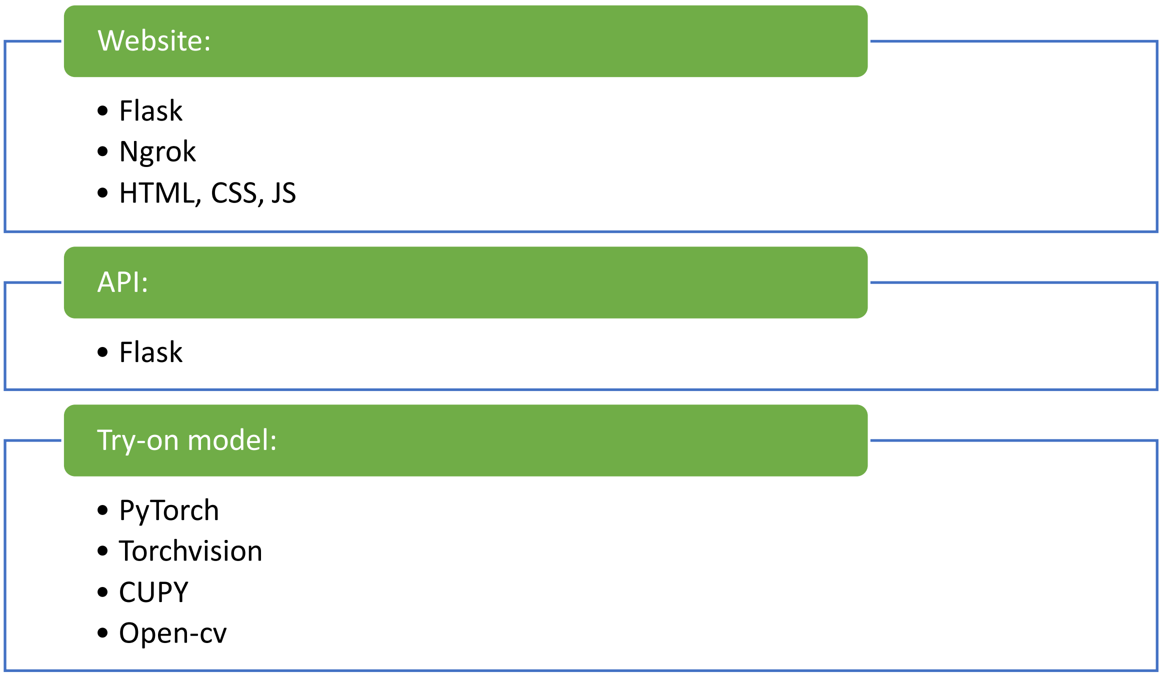


Figure 23: Technology stack

Website

Even though the main component of our project is the API, still we have developed a very simple website to demonstrate how our API can be called from a client's web app. Flask is being used to develop the website of our project because it is very lightweight and ideal for small project. We chose flask so that we can spend more time on our try-on model instead of web app. Moreover, REST API is developed so that it can be called from web, android or IOS app. We call the API using Flask http post form, which includes 2 images. One image of the shirt that user wants to try and second image of the user. API returns the processed image of user with new shirt. We will show this image to the user on the website.

The front-end of the website is built with Bootstrap and is entirely responsive. CSS, html, JSON, JavaScript, and static files are all saved separately. The website is hosted on a developer server called "Ngrok." Ngrok creates a dynamic link, which is forwarded to the application via a developer server. For the development web server, the great Ngrok utility is utilized to generate temporary public URLs. Ngrok is a fantastic utility that creates a tunnel between a URL on the ngrok.io domain and a computer model.

API

Flask is being used to develop the rest API of our project because it is very lightweight and ideal for our small project. Since we want to implement a paid subscription model for our try-on service, we created a rest API to keep track of how many calls are being used by our clients.

APIs are being used by almost all major websites today. If any of our clients wants to integrate our try-on service in their website, they can simply call our API and show the result on their website or app with their branding. Users will never have to leave our client’s website or app.

Try-on model

Two popular frameworks pytorch and tensor flow both were available to us, we decided to use pytorch since it aligned well with our end goals. To get our project we needed a few other libraries, and they were all written in pytorch, so it was easier for us to use pytorch and it integrate other libraries with our system. Pytorch also offers more customization of the layers and their internal loss function which was a requirement for our project so hence keeping all that in mind we decided to use pytorch and it turned out to be the right decision. Cupy was alternatively with Numpy. Cupy is a Numpy for GPU, we needed to make use of GPU available to reduce our processing time so that's why at places we are using Cupy to speed the processing speed. The usage of OpenCV doesn't require any explanations, it's the standard and most enriched image processing package in python which is why it was chosen.

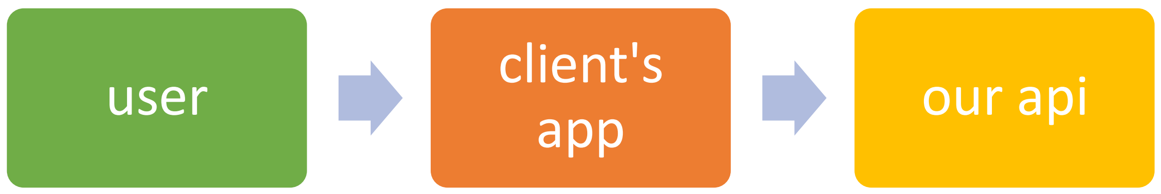


Figure 24: User interaction flow

Implementation

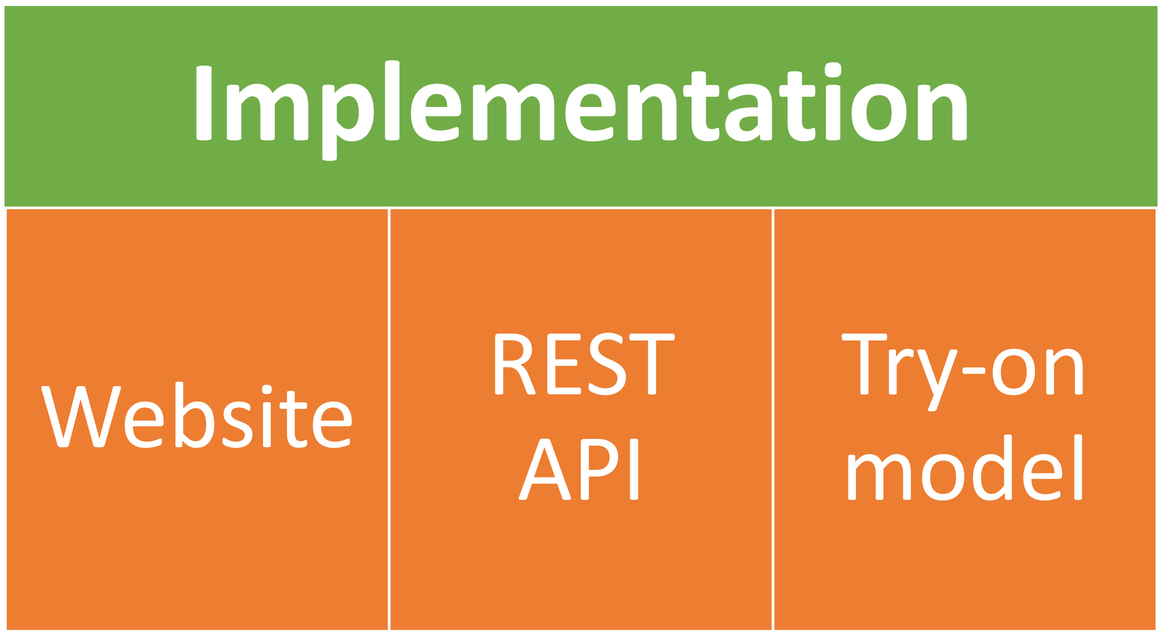


Figure 25: Implementation

Website Code

Table 1: Website code

```
from flask import Flask, request, render_template, redirect
import requests
app = Flask(__name__)
url = 'http://23ee-35-197-60-80.ngrok.io/'

@app.route('/', methods=['GET', 'POST'])
def homepage():
    if request.method.lower() == 'get':
        return render_template('main.html')
    else:
        person = request.files['person']
        shirt = request.files['shirt']
        files = {'person': person, 'shirt': shirt}
        r = requests.post(f'{url}tryon', files=files)
        print(type(r))
        print(r.text)
        if r.text == 'done':
            return redirect(f'{url}result')
        else:
            return 'error'



if __name__ == '__main__':
    app.run()
```

This is the code for a very basic website that is integrating our API. We are keeping the code short to emphasize that our API can be implemented in any website in just 10 lines.

API Code

A very basic code for flask API demo is provided below.

Table 2: API code

```
import sys
sys.path
from flask import Flask, render_template , request , send_file
from flask_ngrok import run_with_ngrok
import os
import cv2
#from google.colab import drive
app = Flask(__name__)
run_with_ngrok(app)
@app.route('/tryon', methods=['POST'])
def home():
  shirt = request.files['shirt']
  person = request.files['person']
  shirt.save(‘model_test/dataset/test_shirt_img/1.png')
  person.save(‘model_test/dataset/test_img/1.png')
  img = cv2.imread('model_test/dataset/test_shirt_img/1.png')
  resized = cv2.resize(img, (192,256), interpolation = cv2.INTER_AREA)
  cv2.imwrite('model_test/dataset/test_shirt_img/1.png', resized)
  img = cv2.imread('model_test/dataset/test_img/1.png')
  resized = cv2.resize(img, (192,256), interpolation = cv2.INTER_AREA)
  cv2.imwrite('model_test/dataset/test_img/1.png', resized)
  !python image_segmentor.py; source activate test36;  ./api.sh;
  #os.system('sh api.sh')
  return 'done'

@app.route('/result')
def d():
    return send_file(
    'model_test/results/demo/PFAFN/new_output.jpg',
    as_attachment=True,
    attachment_filename='result.jpg',
    mimetype='image/jpg'
   )

if __name__ == '__main__':
  app.run()
```

We get the person image and shirt image from client. Client can call this API from a website, android app or IOS app. Same API will be used, no matter what platform the client is using. That is the main advantage of creating REST API.

Try-on Code

Table 3: Try-on code

```
# IMAGE SEGMENTATION CODE

import cv2
import os
import numpy as np
import matplotlib.pyplot as plt

def generate_shirt_mask(path, name, orig_path, mask_dir):
  orig_img = cv2.imread(orig_path)
  orig_img = cv2.cvtColor(orig_img, cv2.COLOR_BGR2RGB) # BGR TO RGB

  print(os.path.join(path, name))
  img = cv2.imread(os.path.join(path, name))

  img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) # BGR TO RGB
  red = img[:, :, 0]
  ##plt.imshow(img)
  ##plt.figure()
  kernel = np.ones((5,5), np.uint8)
  red = cv2.erode(red, kernel, iterations=1)
  red[red>0] = 255

  orig_img[red == 0] = [255,255,255]
  ##plt.imshow(orig_img, cmap = 'gray')
  ##plt.figure()
  ##plt.imshow(red)
  #cv2.imwrite(os.path.join(mask_dir, name), red)
  print(os.path.join(mask_dir, name))
  orig_img = cv2.cvtColor(orig_img, cv2.COLOR_RGB2BGR)
  return red, orig_img

import os
# from tqdm import tqdm
#from tqdm.notebook import tqdm
from PIL import Image
import numpy as np

import torch
import torch.nn.functional as F
import torchvision.transforms as transforms

from ImageSegmentation.data.base_dataset import Normalize_image
from ImageSegmentation.utils.saving_utils import load_checkpoint_mgpu

from ImageSegmentation.networks import U2NET
#device = 'cuda'

#image_dir = 'model_test/dataset/test_shirt_img'
#result_dir = 'model_test/dataset/test_clothes'
#mask_dir = 'model_test/dataset/test_edge'
#checkpoint_path =
'ImageSegmentation/trained_checkpoint/cloth_segm_u2net_latest.pth'

def get_palette(num_cls):
    """ Returns the color map for visualizing the segmentation mask.
    Args:
        num_cls: Number of classes
    Returns:
        The color map
    """
    n = num_cls
    palette = [0] * (n * 3)
    for j in range(0, n):
        lab = j
        palette[j * 3 + 0] = 0
        palette[j * 3 + 1] = 0
        palette[j * 3 + 2] = 0
        i = 0
        while lab:
            palette[j * 3 + 0] |= (((lab >> 0) & 1) << (7 - i))
            palette[j * 3 + 1] |= (((lab >> 1) & 1) << (7 - i))
            palette[j * 3 + 2] |= (((lab >> 2) & 1) << (7 - i))
            i += 1
            lab >>= 3
    return palette

def run_model():
    device = 'cuda'
    image_dir = 'model_test/dataset/test_shirt_img'
    result_dir = 'model_test/dataset/test_clothes'
    mask_dir = 'model_test/dataset/test_edge'
    checkpoint_path = 
'ImageSegmentation/trained_checkpoint/cloth_segm_u2net_latest.pth'

    transforms_list = []
    transforms_list += [transforms.ToTensor()]
    transforms_list += [Normalize_image(0.5, 0.5)]
    transform_rgb = transforms.Compose(transforms_list)

    net = U2NET(in_ch=3, out_ch=4)
    net = load_checkpoint_mgpu(net, checkpoint_path)
    net = net.to(device)
    net = net.eval()

    palette = get_palette(4)

    images_list = sorted(os.listdir(image_dir))
    print(images_list)
    #pbar = tqdm(total=len(images_list))
    for image_name in images_list:
        img = Image.open(os.path.join(image_dir, 
image_name)).convert('RGB')
        img_size = img.size
        img = img.resize((768, 768), Image.BICUBIC)
        image_tensor = transform_rgb(img)
        image_tensor = torch.unsqueeze(image_tensor, 0)

        output_tensor = net(image_tensor.to(device))
        output_tensor = F.log_softmax(output_tensor[0], dim=1)
        output_tensor = torch.max(output_tensor, dim=1, 
keepdim=True)[1]
        output_tensor = torch.squeeze(output_tensor, dim=0)
        output_tensor = torch.squeeze(output_tensor, dim=0)
        output_arr = output_tensor.cpu().numpy()

        output_img = Image.fromarray(output_arr.astype('uint8'), 
mode='L')
        output_img = output_img.resize(img_size, Image.BICUBIC)

        output_img.putpalette(palette)
        output_img.convert('RGB').save(os.path.join(result_dir,
image_name[:-4]+'.png'))

        mask, shirt = generate_shirt_mask(result_dir, image_name[:-
4]+'.png', os.path.join(image_dir, image_name), mask_dir)



        cv2.imwrite(os.path.join(mask_dir, image_name[:-4]+'.png'),
mask)
        cv2.imwrite(os.path.join(result_dir, image_name[:-4]+'.png'),
shirt)

        mask = cv2.cvtColor(mask,cv2.COLOR_GRAY2RGB)
        #vis = np.concatenate((mask, shirt), axis=1)

        img = cv2.imread(os.path.join(image_dir, image_name))
        #res = cv2.vconcat([img, shirt])

        #res = cv2.vconcat([res, mask])

        #cv2.imwrite(os.path.join(result_dir, 'combined.jpg'), res)



        print(os.path.join(result_dir, image_name[:-4]+'.png'),
'SAVED')

        #pbar.update(1)

    #pbar.close()

    demo_file = 'model_test/demo.txt'
    img_name = images_list[0]

    with open(demo_file, 'w') as file:
      file.write(img_name+' '+img_name)
      file.close()
    print('All DONE')

    #os.chdir('model_test/')
    #os.system('python test.py')
    #os.chdir('..')
    return True

run_model()
```

# Try-on model code

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import numpy as np
from options.train_options import TrainOptions
from .correlation import correlation # the custom cost volume layer
opt = TrainOptions().parse()

def apply_offset(offset):
    sizes = list(offset.size()[2:])
    grid_list = torch.meshgrid([torch.arange(size, device=offset.device)
for size in sizes])
    grid_list = reversed(grid_list)
    # apply offset
    grid_list = [grid.float().unsqueeze(0) + offset[:, dim, ...]
        for dim, grid in enumerate(grid_list)]
    # normalize
    grid_list = [grid / ((size - 1.0) / 2.0) - 1.0
        for grid, size in zip(grid_list, reversed(sizes))]

    return torch.stack(grid_list, dim=-1)
```

```
def TVLoss(x):
    tv_h = x[:, :, 1:, :] - x[:, :, :-1, :]
    tv_w = x[:, :, :, 1:] - x[:, :, :, :-1]

    return torch.mean(torch.abs(tv_h)) + torch.mean(torch.abs(tv_w))



# backbone
class ResBlock(nn.Module):
    def __init__(self, in_channels):
        super(ResBlock, self).__init__()
        self.block = nn.Sequential(
            nn.BatchNorm2d(in_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(in_channels, in_channels, kernel_size=3, padding=1,
bias=False),
            nn.BatchNorm2d(in_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(in_channels, in_channels, kernel_size=3, padding=1,
bias=False)
            )

    def forward(self, x):
        return self.block(x) + x



class DownSample(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(DownSample, self).__init__()
        self.block=  nn.Sequential(
            nn.BatchNorm2d(in_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=2,
padding=1, bias=False)
            )

    def forward(self, x):
        return self.block(x)
```

```
class FeatureEncoder(nn.Module):
    def __init__(self, in_channels, chns=[64,128,256,256,256]):
        # in_channels = 3 for images, and is larger (e.g., 17+1+1) for
agnositc representation
        super(FeatureEncoder, self).__init__()
        self.encoders = []
        for i, out_chns in enumerate(chns):
            if i == 0:
                encoder = nn.Sequential(DownSample(in_channels,
out_chns),
                                        ResBlock(out_chns),
                                        ResBlock(out_chns))
            else:
                encoder = nn.Sequential(DownSample(chns[i-1], out_chns),
                                         ResBlock(out_chns),
                                         ResBlock(out_chns))

            self.encoders.append(encoder)

        self.encoders = nn.ModuleList(self.encoders)



    def forward(self, x):
        encoder_features = []
        for encoder in self.encoders:
            x = encoder(x)
            encoder_features.append(x)
        return encoder_features

class RefinePyramid(nn.Module):
    def __init__(self, chns=[64,128,256,256,256], fpn_dim=256):
        super(RefinePyramid, self).__init__()
        self.chns = chns

        # adaptive
        self.adaptive = []
        for in_chns in list(reversed(chns)):
            adaptive_layer = nn.Conv2d(in_chns, fpn_dim, kernel_size=1)
            self.adaptive.append(adaptive_layer)
        self.adaptive = nn.ModuleList(self.adaptive)
        # output conv
        self.smooth = []
        for i in range(len(chns)):
            smooth_layer = nn.Conv2d(fpn_dim, fpn_dim, kernel_size=3,
padding=1)
            self.smooth.append(smooth_layer)
        self.smooth = nn.ModuleList(self.smooth)

    def forward(self, x):
        conv_ftr_list = x

        feature_list = []
        last_feature = None
        for i, conv_ftr in enumerate(list(reversed(conv_ftr_list))):
            # adaptive
            feature = self.adaptive[i](conv_ftr)
            # fuse
            if last_feature is not None:
                feature = feature + F.interpolate(last_feature,
scale_factor=2, mode='nearest')
            # smooth
            feature = self.smooth[i](feature)
            last_feature = feature
            feature_list.append(feature)

        return tuple(reversed(feature_list))


class AFlowNet(nn.Module):
    def __init__(self, num_pyramid, fpn_dim=256):
        super(AFlowNet, self).__init__()
        self.netMain = []
        self.netRefine = []
        for i in range(num_pyramid):
            netMain_layer = torch.nn.Sequential(
                torch.nn.Conv2d(in_channels=49, out_channels=128,
kernel_size=3, stride=1, padding=1),
                torch.nn.LeakyReLU(inplace=False, negative_slope=0.1),
                torch.nn.Conv2d(in_channels=128, out_channels=64,
kernel_size=3, stride=1, padding=1),
                torch.nn.LeakyReLU(inplace=False, negative_slope=0.1),
                torch.nn.Conv2d(in_channels=64, out_channels=32,
kernel_size=3, stride=1, padding=1),
                torch.nn.LeakyReLU(inplace=False, negative_slope=0.1),
                torch.nn.Conv2d(in_channels=32, out_channels=2,
kernel_size=3, stride=1, padding=1)
            )

            netRefine_layer = torch.nn.Sequential(
                torch.nn.Conv2d(2 * fpn_dim, out_channels=128,
kernel_size=3, stride=1, padding=1),
                torch.nn.LeakyReLU(inplace=False, negative_slope=0.1),
                torch.nn.Conv2d(in_channels=128, out_channels=64,
kernel_size=3, stride=1, padding=1),
                torch.nn.LeakyReLU(inplace=False, negative_slope=0.1),
                torch.nn.Conv2d(in_channels=64, out_channels=32,
kernel_size=3, stride=1, padding=1),
                torch.nn.LeakyReLU(inplace=False, negative_slope=0.1),
                torch.nn.Conv2d(in_channels=32, out_channels=2,
kernel_size=3, stride=1, padding=1)
            )
            self.netMain.append(netMain_layer)
            self.netRefine.append(netRefine_layer)

        self.netMain = nn.ModuleList(self.netMain)
        self.netRefine = nn.ModuleList(self.netRefine)


    def forward(self, x, x_edge, x_warps, x_conds, warp_feature=True):
        last_flow = None
        last_flow_all = []
        delta_list = []
        x_all = []
        x_edge_all = []
        cond_fea_all = []
        delta_x_all = []
        delta_y_all = []
        filter_x = [[0, 0, 0],
                    [1, -2, 1],
                    [0, 0, 0]]
        filter_y = [[0, 1, 0],
                     [0, -2, 0],
                     [0, 1, 0]]
        filter_diag1 = [[1, 0, 0],
                         [0, -2, 0],
                         [0, 0, 1]]
        filter_diag2 = [[0, 0, 1],
                         [0, -2, 0],
                         [1, 0, 0]]
        weight_array = np.ones([3, 3, 1, 4])
        weight_array[:, :, 0, 0] = filter_x
        weight_array[:, :, 0, 1] = filter_y
        weight_array[:, :, 0, 2] = filter_diag1
        weight_array[:, :, 0, 3] = filter_diag2

        weight_array =
torch.cuda.FloatTensor(weight_array).permute(3,2,0,1)
        self.weight = nn.Parameter(data=weight_array,
requires_grad=False)

        for i in range(len(x_warps)):
              x_warp = x_warps[len(x_warps) - 1 - i]
              x_cond = x_conds[len(x_warps) - 1 - i]
              cond_fea_all.append(x_cond)

              if last_flow is not None and warp_feature:
                  x_warp_after = F.grid_sample(x_warp,
last_flow.detach().permute(0, 2, 3, 1),
                       mode='bilinear', padding_mode='border')
              else:
                  x_warp_after = x_warp

              tenCorrelation =
F.leaky_relu(input=correlation.FunctionCorrelation(tenFirst=x_warp_after,
tenSecond=x_cond, intStride=1), negative_slope=0.1, inplace=False)
              flow = self.netMain[i](tenCorrelation)
              delta_list.append(flow)
              flow = apply_offset(flow)
              if last_flow is not None:
                  flow = F.grid_sample(last_flow, flow, mode='bilinear',
padding_mode='border')
              else:
                  flow = flow.permute(0, 3, 1, 2)

              last_flow = flow
              x_warp = F.grid_sample(x_warp, flow.permute(0, 2, 3, 
1),mode='bilinear', padding_mode='border')
              concat = torch.cat([x_warp,x_cond],1)
              flow = self.netRefine[i](concat)
              delta_list.append(flow)
              flow = apply_offset(flow)
              flow = F.grid_sample(last_flow, flow, mode='bilinear', 
padding_mode='border')

              last_flow = F.interpolate(flow, scale_factor=2, 
mode='bilinear')
              last_flow_all.append(last_flow)
              cur_x = F.interpolate(x, scale_factor=0.5**(len(x_warps)-1-
i), mode='bilinear')
              cur_x_warp = F.grid_sample(cur_x, last_flow.permute(0, 2, 
3, 1),mode='bilinear', padding_mode='border')
              x_all.append(cur_x_warp)
              cur_x_edge = F.interpolate(x_edge, 
scale_factor=0.5**(len(x_warps)-1-i), mode='bilinear')
              cur_x_warp_edge = F.grid_sample(cur_x_edge, 
last_flow.permute(0, 2, 3, 1),mode='bilinear', padding_mode='zeros')
              x_edge_all.append(cur_x_warp_edge)
              flow_x,flow_y = torch.split(last_flow,1,dim=1)
              delta_x = F.conv2d(flow_x, self.weight)
              delta_y = F.conv2d(flow_y,self.weight)
              delta_x_all.append(delta_x)
              delta_y_all.append(delta_y)

        x_warp = F.grid_sample(x, last_flow.permute(0, 2, 3, 1),
                     mode='bilinear', padding_mode='border')
        return x_warp, last_flow, cond_fea_all, last_flow_all, 
delta_list, x_all, x_edge_all, delta_x_all, delta_y_all



class AFWM(nn.Module):

    def __init__(self, opt, input_nc):
        super(AFWM, self).__init__()
        num_filters = [64,128,256,256,256]
        self.image_features = FeatureEncoder(3, num_filters)
        self.cond_features = FeatureEncoder(input_nc, num_filters)
        self.image_FPN = RefinePyramid(num_filters)
        self.cond_FPN = RefinePyramid(num_filters)
        self.aflow_net = AFlowNet(len(num_filters))
        self.old_lr = opt.lr
        self.old_lr_warp = opt.lr*0.2

    def forward(self, cond_input, image_input, image_edge):
        cond_pyramids = self.cond_FPN(self.cond_features(cond_input)) #
maybe use nn.Sequential
        image_pyramids = self.image_FPN(self.image_features(image_input))

        x_warp, last_flow, last_flow_all, flow_all, delta_list, x_all,
x_edge_all, delta_x_all, delta_y_all = self.aflow_net(image_input,
image_edge, image_pyramids, cond_pyramids)

        return x_warp, last_flow, last_flow_all, flow_all, delta_list,
x_all, x_edge_all, delta_x_all, delta_y_all



    def update_learning_rate(self,optimizer):
        lrd = opt.lr / opt.niter_decay
        lr = self.old_lr - lrd
        for param_group in optimizer.param_groups:
            param_group['lr'] = lr
        if opt.verbose:
            print('update learning rate: %f -> %f' % (self.old_lr, lr))
        self.old_lr = lr

    def update_learning_rate_warp(self,optimizer):
        lrd = 0.2 * opt.lr / opt.niter_decay
        lr = self.old_lr_warp - lrd
        for param_group in optimizer.param_groups:
            param_group['lr'] = lr
        if opt.verbose:
            print('update learning rate: %f -> %f' % (self.old_lr_warp,
lr))
        self.old_lr_warp = lr
```

System Testing

Before we present the system testing results, we must define the functional and non-functional requirements. Although they are defined in the SRS document, we will provide a brief overview of these requirements in this report.

Functional Requirements

- The implemented models shall be able to run on any picture format i.e jpg, jpeg, png.
- The mask should be generated accurately, and system must be able to see where actually the shirt is.
- The system shall perform this function in a way that it accurately applies the targeted shirt with the user's image with the minimum time.
- The user should be able to download the image.
- The image of the person should not be stored on the server to maintain privacy.
- The image should only be visible to the ecommerce company who are using the service.

Non-functional Requirements

Performance Requirements

- The output image must be available to user within 5-15 seconds.
- GPU services should be used to minimized to the response time of API
- The overall time from submitting the image and getting it back should not exceed 30 seconds.

Software Quality Attributes

Usability

- The API should not be hard to implement within the existing solutions.
- There should a message returned to guide the user in any case it goes wrong.
- The API shall be able to handle 100 Try ON requests per minute.

Correctness

- There is no way to get the accuracy in the user perspective case however we can determine that how accurately the targeted shirt is being applied. We should make sure that the end user is satisfied.

Availability

- The API shall be up 24/7. in case of maintenance, it must not be more than 30 minutes.

Robustness

- The system shall be able to handle invalid format and respond to user. It should also be able to work with low quality images.

System Testing results

Accuracy:

Please see the FID table.

Performance:

Please see the API performance section.

Usability:

Please see the website code section to see how easy it is to implement our API in any website or app.

Correctness:

Please see the user testing section.

Security:

We are not storing images of users in any database for privacy concerns.

Chapter 7

RESULTS AND DISCUSSION

Model Results

To generate the try-on results during the test, a target garment and a reference person image are provided. We use the Fréchet Inception Distance (FID) [18] as the assessment metric because we don't have ground-truth photographs (i.e., a reference person wearing the target apparel), which captures the similarity of generated images to real images (i.e., reference person images). A lower FID score means that the results are of higher quality. There is another type of score called Inception Score [19] that is used to automatically evaluate the quality of image generative models. But [20] proved that if Inception Score is used on models that are not trained on ImageNet dataset, it can lead to wrong results. Since we are training our model on VITON dataset [4], FID is used instead of Inception Score.

First, we will define the formula of FID score:

$$d^2((\boldsymbol{m}, \boldsymbol{C}), (\boldsymbol{m}_w, \boldsymbol{C}_w)) = \|\boldsymbol{m} - \boldsymbol{m}_w\|_2^2 + \mathrm{Tr}\big(\boldsymbol{C} + \boldsymbol{C}_w - 2\big(\boldsymbol{C}\boldsymbol{C}_w\big)^{1/2}\big)$$

Figure 26: FID formula

We follow these steps to calculate the FID score:

- Extract the feature vectors of actual and false images using the Inception V2 pre-trained model
- Calculate the mean of the feature vectors obtained in step 1 by feature.
- Create the feature vectors' covariance matrices — C, C w
- Calculate the trace (the sum of the elements along the square matrix's major diagonal) of (C+C_w-2*(C*C$_x$)^1/2).
- Calculate the squared difference of the calculated mean vectors in step 2 — ||m-m_w||^2

- Finally, combine the results of steps 4 and 5.

Now that we have defined the evaluation metric of our model, let us see how our model compares with other try-on models.

Table 4: FID comparison

Model	Dataset	FID
CP-VTON [21]	VITON	24.43
ClothFlow [22]	VITON	14.43
CP-VTON+ [23]	VITON	21.08
ACGPN [24]	VITON	15.67
Our model	VITON	16.21

It is not possible to visualize the quality of a model from FID score. So, the results from various models listed in the table are given below:

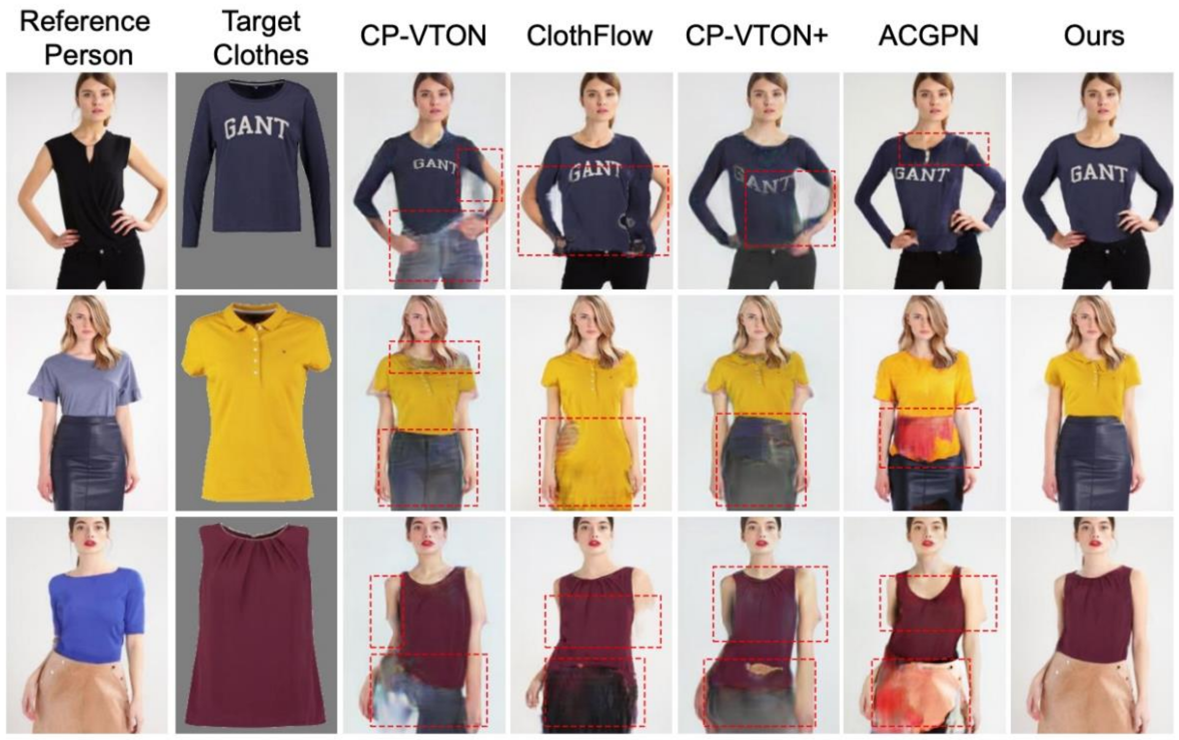


Figure 27: Model comparison

We can observe the shortcomings of each mode. For example, CP-VTON and ACGPIN fail completely to preserve the original characteristics of the clothes. Similarly, there are distortions present in the other 2 models as well.

Now, we will discuss the results of our model one by one.

In the first row, not only, our model preserves the wrap and color of cloth, it also preserves the logo perfectly. All other models failed to preserve the logo (as seen in the comparison).

In the second row, our model adjusts the length of the yellow t-shirt according to the dress of the person. Only CP-VTON and CP-VTON+ succeeded in this regard but failed in preserving the color of the cloth. our model on the other hand adjusts the length of the cloth and retains the original color as well. One major thing to notice is that the length of the sleeves of the yellow t-shirt is smaller than the original t-shirt. So, our model uses GANs [26] to generate a portion of arms that was not visible in the original image. The arm's skin tone is perfectly maintained as well.

Similarly, in the third row, our model performs better than other models and automatically generates the hidden portion of arms (note that no other model could do so).

Although, FID score is a good indicator of quality of generated image, we still need human judgment for quality assurance. So, we asked 10 of our friends to volunteer and rate the generated images from all models on a scale of 1-5. All images were shuffled and without any label. This table illustrates the results from this study:

Table 5: User rating comparison

Model	Average Rating
CP-VTON	2.4/5
ClothFlow	3.4/5
CP-VTON+	2.8/5
ACGPN	3.1/5
Our model	3.8/5

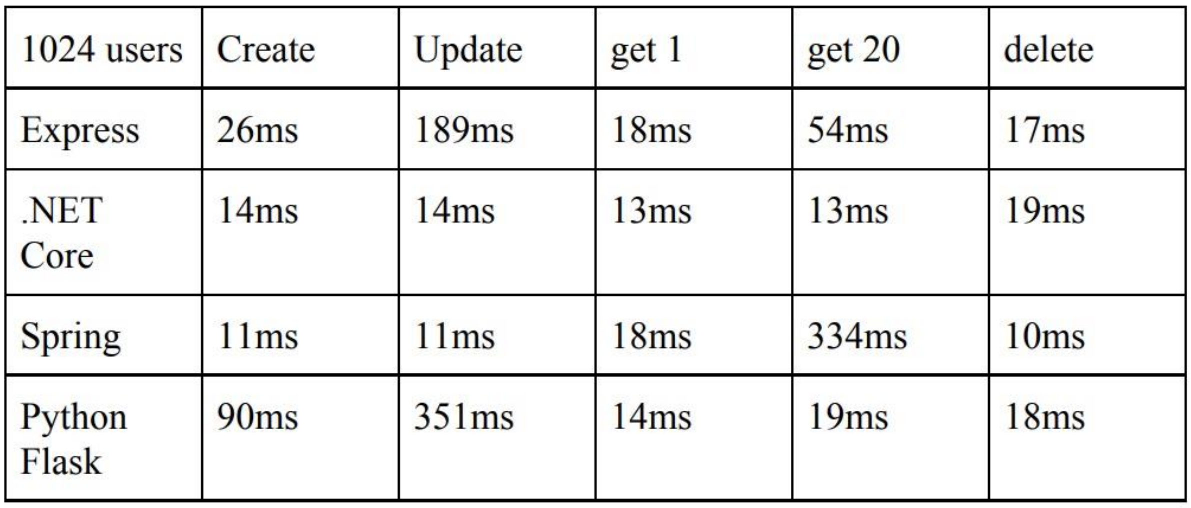
Both FID score and human judgement suggest that our model is very competitive to other models.

API Performance

Our model is currently taking (on average) 10.45 seconds to produce the result on Google Colab platform. This time can be significantly improved if a better GPU is used. [25] demonstrates the performance of various web frameworks including Flask REST API. The performance metric being used is the response time from the server.

Result of 1024 Users

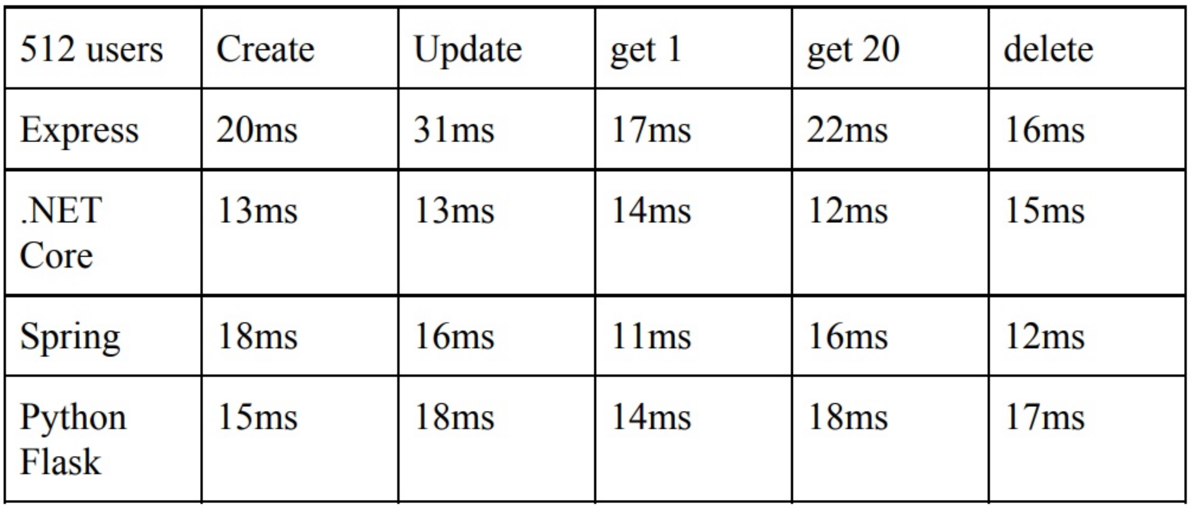
Table 6: 1024 users API performance



1024 users	Create	Update	get 1	get 20	delete
Express	26ms	189ms	18ms	54ms	17ms
.NET Core	14ms	14ms	13ms	13ms	19ms
Spring	11ms	11ms	18ms	334ms	10ms
Python Flask	90ms	351ms	14ms	19ms	18ms

Result of 512 Users

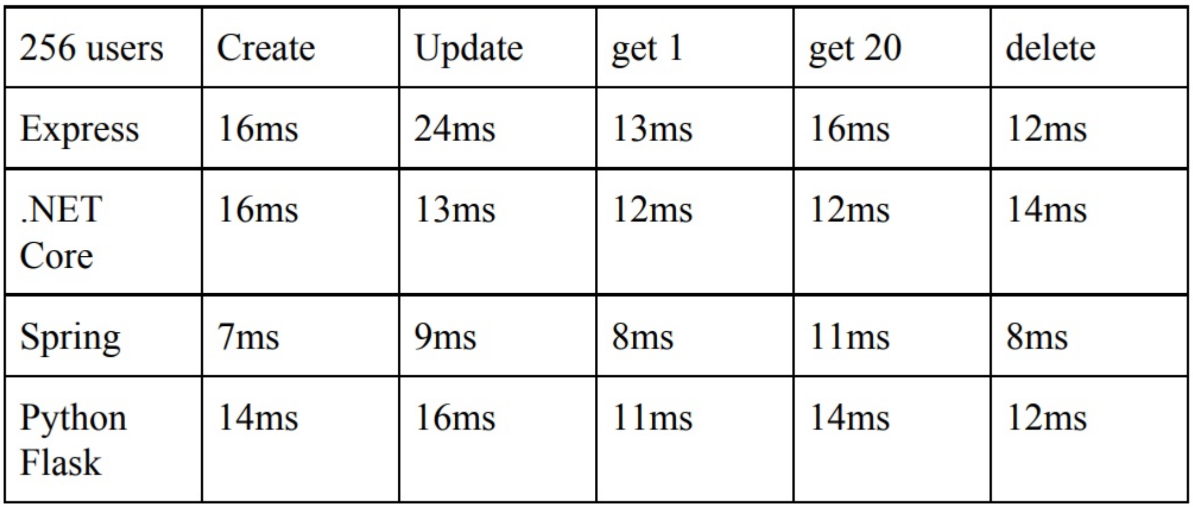
Table 7: 512 users API performance



512 users	Create	Update	get 1	get 20	delete
Express	20ms	31ms	17ms	22ms	16ms
.NET Core	13ms	13ms	14ms	12ms	15ms
Spring	18ms	16ms	11ms	16ms	12ms
Python Flask	15ms	18ms	14ms	18ms	17ms

Results of 256 Users

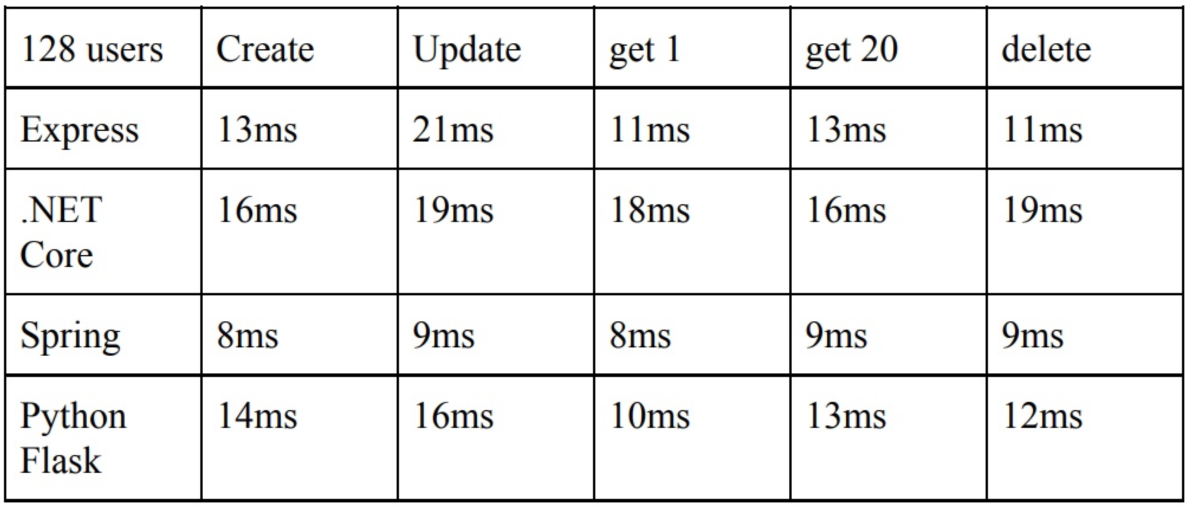
Table 8: 256 users API performance



256 users	Create	Update	get 1	get 20	delete
Express	16ms	24ms	13ms	16ms	12ms
.NET Core	16ms	13ms	12ms	12ms	14ms
Spring	7ms	9ms	8ms	11ms	8ms
Python Flask	14ms	16ms	11ms	14ms	12ms

Results of 128 Users

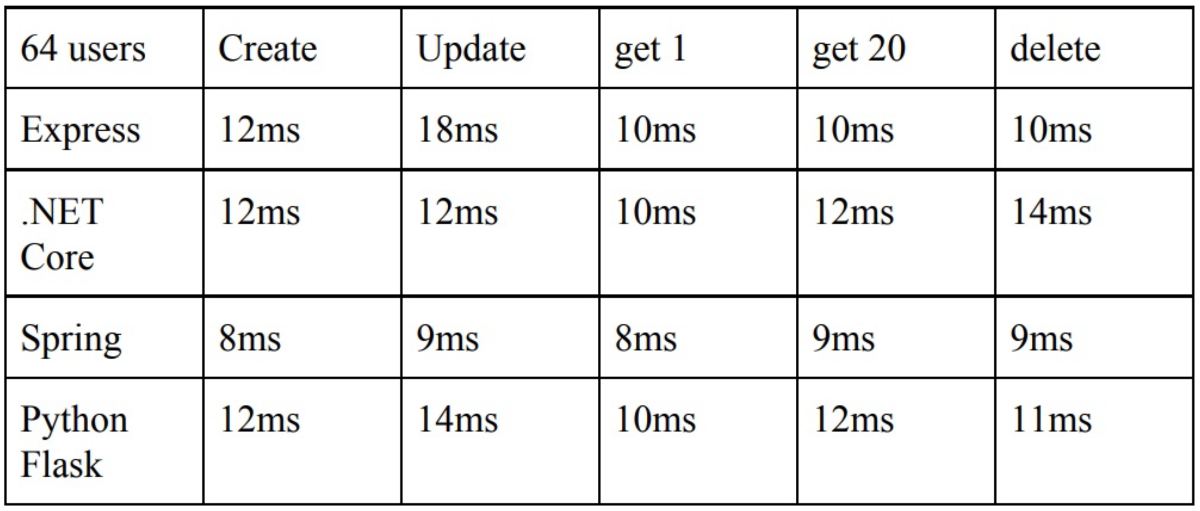
Table 9: 128 users API performance



128 users	Create	Update	get 1	get 20	delete
Express	13ms	21ms	11ms	13ms	11ms
.NET Core	16ms	19ms	18ms	16ms	19ms
Spring	8ms	9ms	8ms	9ms	9ms
Python Flask	14ms	16ms	10ms	13ms	12ms

Result of 64 Users

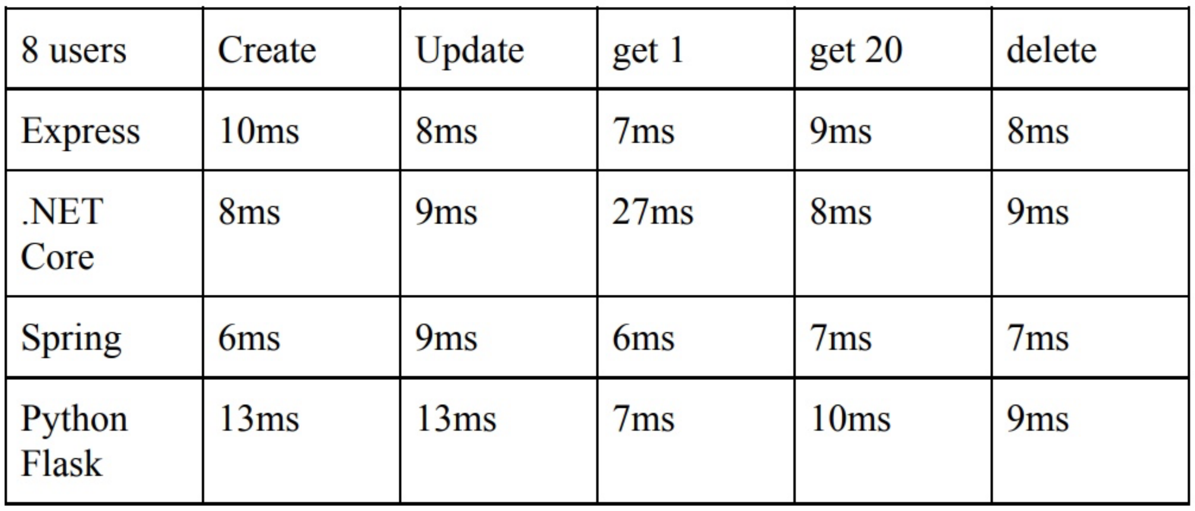
Table 10: 64 users API performance



64 users	Create	Update	get 1	get 20	delete
Express	12ms	18ms	10ms	10ms	10ms
.NET Core	12ms	12ms	10ms	12ms	14ms
Spring	8ms	9ms	8ms	9ms	9ms
Python Flask	12ms	14ms	10ms	12ms	11ms

Results of 8 Users

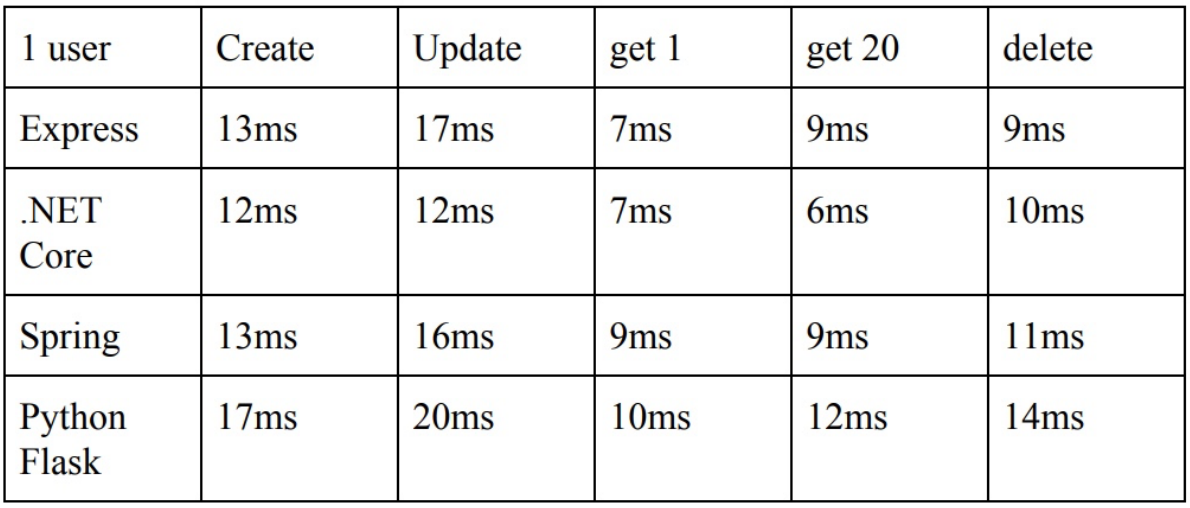
Table 11: 8 users API performance



8 users	Create	Update	get 1	get 20	delete
Express	10ms	8ms	7ms	9ms	8ms
.NET Core	8ms	9ms	27ms	8ms	9ms
Spring	6ms	9ms	6ms	7ms	7ms
Python Flask	13ms	13ms	7ms	10ms	9ms

Results of 1 User

Table 12: 1 user API performance



1 user	Create	Update	get 1	get 20	delete
Express	13ms	17ms	7ms	9ms	9ms
.NET Core	12ms	12ms	7ms	6ms	10ms
Spring	13ms	16ms	9ms	9ms	11ms
Python Flask	17ms	20ms	10ms	12ms	14ms

Flask is clearly not the best performing framework but its ease of use and integration with ML models makes it an ideal choice for our work. Note that to get the average time taken by our API to return the result, we just need to add the time taken by our model (i.e 10.45 seconds) to generate the output to the response time from the table.

Chapter 8

CONCLUSION AND FUTURE WORK

The overall project was a big success based on the mile stones we set, we were able to create a project that given an image of a target person followed by the image of the target shirt would seamlessly transfer the styles of the target shirt on to the target person. We were able to follow the professional software design practices and get the job done in the required timeframe while leaving enough time for testing and discussion. The system is available both as a web application and a REST API which would be available to companies on subscription basis. The project API is kept minimalistic so that companies can very easily integrate it with their system.

Our model's performance was tested both by the computer metrics as well as the humans, we were able to achieve an FID score less than 15 which is quite state of the art in terms of the try on technology community. On top of that we consulted people as a part of a survey asking them to rate our model's results. We found that they were quite satisfied with our project and showed interest in the usage of our model. Their positive feedback kept us going. The web application is flask based and the api is very simple, integration with any framework is really easy for the developer.

Try on Technology is on the hype these days, Nike and L'oreal Paris have already released their own try on technology for shoes and make products respectively. They have discovered that usage of their try on technology has increased their over all revenue and decreased the product return rate. Every person has a different body structure and different appearance, so this doesn't mean that what looked on the portrayed models would look good on you. By letting the user to try out products virtually before they actually buy is a good leap towards better online shopping and ecommerce and with products like this we can leap into the future of shopping by completely revolutionising it.

No product is perfect and requires constant maintenance and improvements. Same is the case with our model. Although it works really well to apply styles on the target person based on the target shirt, there are areas that can be improved upon. Our model generates output of size 192 x 256 (RGB). Since we were limited by the GPU computing power, we couldn’t go for HD images. There is another version of VITON dataset available that has higher quality images. It goes by the name of VITON HD. It has 2-3 times better quality images compared to standard VITON. With the provision of more computing power and more time, we could train the model on the VITON HD dataset, which would be a big improvement in our current model architecture. We would be able to generate even higher quality images with this model. Moving forward, another aspect to focus upon would be to expand our current garment categories and make it work for other garment types as well. Right now our model works only on shirts (t shirt, long sleeve shirts, sleeveless shirts). It would be a massive improvement to make it for sweaters, jackets, coats and pants. But then comes in steps, we have a baseline model, with the provision of more dataset for such categories we would be able to build a similar model with minimal effort since we have the main model already developed.

Furthermore, we would like to integrate our current system with an online payment and subscription based model to make it available for the general public with a few lines of code. We would purchase more storage and make it available for everyone with quite ease just like google cloud models. Right now our API is available on-demand but we intend to make it available for everyone so that there is a system in place which would handle the payments based on the requests made.

Chapter 9

REFERENCES

[1] Seunghwan Choi, Sunghyun Park, Minsoo Lee, and Jaegul Choo. VITON-HD: High-resolution virtual try-on via misalignment-aware normalization. In IEEE Conf. Comput. Vis. Pattern Recog., pages 14131–14140, 2021.

[2] Haoye Dong, Xiaodan Liang, Xiaohui Shen, Bochao Wang, Hanjiang Lai, Jia Zhu, Zhiting Hu, and Jian Yin. Towards multi-pose guided virtual try-on network. In Int. Conf. Comput. Vis., pages 9026–9035, 2019.

[3] Xintong Han, Xiaojun Hu, Weilin Huang, and Matthew R Scott. Clothflow: A flow-based model for clothed person generation. In Int. Conf. Comput. Vis., pages 10471–10480, 2019.

[4] Xintong Han, Zuxuan Wu, Zhe Wu, Ruichi Yu, and Larry S Davis. VITON: An image-based virtual try-on network. In IEEE Conf. Comput. Vis. Pattern Recog., pages 7543–7552, 2018.

[5] Hyug Jae Lee, Rokkyu Lee, Minseok Kang, Myounghoon Cho, and Gunhan Park. LA-VITON: a network for lookingattractive virtual try-on. In Int. Conf. Comput. Vis., pages 0–0, 2019.

[6] Remi Brouet, Alla Sheffer, Laurence Boissieux, and MariePaule Cani. Design preserving garment transfer. ACM Trans. Graph., 31(4): Article–No, 2012.

[7] Peng Guan, Loretta Reiss, David A Hirshberg, Alexander Weiss, and Michael J Black. Drape: Dressing any person. ACM Trans. Graph., 31(4):1–10, 2012.

[8] Chaitanya Patel, Zhouyingcheng Liao, and Gerard Pons-Moll. Tailornet: Predicting clothing in 3D as a function of human pose, shape, and garment style. In IEEE Conf. Comput. Vis. Pattern Recog., pages 7365–7375, 2020.

[9] Gerard Pons-Moll, Sergi Pujades, Sonny Hu, and Michael J Black. Clothcap: Seamless 4D clothing capture and retargeting. ACM Trans. Graph., 36(4):1–15, 2017.

[10] Damien Rohmer, Tiberiu Popa, Marie-Paule Cani, Stefanie Hahmann, and Alla Sheffer. Animation wrinkling: augmenting coarse cloth simulations with realistic-looking wrinkles. ACM Trans. Graph., 29(6):1–8, 2010.

[11] Han Yang, Ruimao Zhang, Xiaobao Guo, Wei Liu, Wangmeng Zuo, and Ping Luo. Towards photo-realistic virtual try-on by adaptively generating-preserving image content. In CVPR, 2020.

[12] Ke Gong, Xiaodan Liang, Dongyu Zhang, Xiaohui Shen, and Liang Lin. Look into person: Self-supervised structuresensitive learning and a new benchmark for human parsing. In CVPR, 2017.

[13] Chongjian Ge, Yibing Song, Yuying Ge, Han Yang, Wei Liu, and Ping Luo. Disentangled cycle consistency for highlyrealistic virtual try-on. In CVPR, 2021.

[14] Bochao Wang, Huabin Zheng, Xiaodan Liang, Yimin Chen, Liang Lin, and Meng Yang. Toward characteristicpreserving image-based virtual try-on network. In ECCV, 2018.

[15] Ruiyun Yu, Xiaoqi Wang, and Xiaohui Xie. Vtnfp: An image-based virtual try-on network with body and clothing feature preservation. In ICCV, 2019.

[16] Yuying Ge, Yibing Song, Ruimao Zhang, Chongjian Ge, Wei Liu, and Ping Luo. Parser-free virtual try-on via distilling appearance flows. In CVPR, 2021.

[17] Thibaut Issenhuth, Jer´ emie Mary, and Cl ´ ement Calauzenes. ´ Do not mask what you do not need to mask: a parser-free virtual try-on. In ECCV, 2020

[18] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rul

[19] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In NeurIPS, 2016.

[20] Mihaela Rosca, Balaji Lakshminarayanan, David WardeFarley, and Shakir Mohamed. Variational approaches for auto-encoding generative adversarial networks. arXiv preprint arXiv:1706.04987, 2017.

[21] Bochao Wang, Huabin Zheng, Xiaodan Liang, Yimin Chen, Liang Lin, and Meng Yang. Toward characteristicpreserving image-based virtual try-on network. In ECCV, 2018.

[22] Xintong Han, Xiaojun Hu, Weilin Huang, and Matthew R Scott. Clothflow: A flow-based model for clothed person generation. In ICCV, 2019.

[23] Matiur Rahman Minar, Thai Thanh Tuan, Heejune Ahn, Paul Rosin, and Yu-Kun Lai. Cp-vton+: Clothing shape and texture preserving image-based virtual try-on. In CVPR Workshops, June 2020.

[24] Han Yang, Ruimao Zhang, Xiaobao Guo, Wei Liu, Wangmeng Zuo, and Ping Luo. Towards photo-realistic virtual try-on by adaptively generating-preserving image content. In CVPR, 2020.

[25] S. Söderlund, ‘Performance of REST applications: Performance of REST applications in four different frameworks’, Dissertation, 2017.

[26] I. J. Goodfellow *κ.ά.*, ‘Generative Adversarial Networks’, *arXiv [stat.ML]*. 2014.

[27] K. He, G. Gkioxari, P. Dollár, και R. Girshick, ‘Mask R-CNN’. arXiv, 2017.

[28] T. Zhou, S. Tulsiani, W. Sun, J. Malik, και A. A. Efros, ‘View Synthesis by Appearance Flow’, στο *European Conference on Computer Vision*, 2016.

[29] J. Zhao *κ.ά.*, ‘Understanding Humans in Crowded Scenes: Deep Nested Adversarial Learning and A New Benchmark for Multi-Human Parsing’, 2018.