CHAPTER 1
INTRODUCTION

1.1 OBJECTIVE

To improve the quality of the generated on-garment textures and logotypes, this project design a novel geometric matching module that conditions the pose matching procedure on body segmentations only, and, therefore, minimizes the dependence on multiple (potentially error prone) pre-processing steps. Additionally, this project formulate learning objectives for the module's training procedure that penalize the appearance of the aligned clothing solely within the body area (while ignoring other body parts) to ensure that challenging pose configuration and self occlusions (e.g. from hands) do not adversely affect performance.

1.2 MOTIVATION OF THE PROJECT

Image -based virtual try-on solutions that do not require specialized hard ware and dedicated imaging equipment, but are applicable with standard intensity images. This allows for the design of virtual fitting rooms that let consumers try-on clothes without visiting (brick and mortar) stores, and also benefits retailers by reducing product returns and shipping costs. Existing solutions to image-based virtual try-on are dominated by two-stage approaches (and their extensions) that typically include:

(i) geometric matching stage that aligns the target clothing to the pose of the person in the input image, and estimates an approximate position and shape of the target garment in the final try-on result.

(ii) an image synthesis stage that uses dedicated generative models (e.g., Generative Adversarial Networks (GANs), together with various refinement strategies to synthesize the final try-on image based on the aligned clothing and different auxiliary sources of information.

(iii) various enhancements have also been proposed, including:

(i) Refinements of the data fed to the geometric matching stage.

(ii) Integration of clothing segmentations into the synthesis procedure .

(iii) The use of knowledge distillation schemes to minimize the impact of parser-related errors .

1.3 SCOPE OF THE PROJECT

While the outlined advances greatly improved the quality of the generated try-on results, the loss of details on the transferred garments that is often a consequence of difficulties with the geometric matching stage still represents a major challenge to image-based virtual try-on solutions. Additionally, poor quality (human/clothing) parsing results typically still lead to unconvincing try-on images with garment textures being placed over incorrect body parts. Although recent (distilled) parser-free models, address this issue to some degree, they still inherit the main characteristics of the teacher models (including parsing issues) and often struggle with the generation of realistic body-parts, such as hands or arms this project propose a Context-Driven Virtual Tryon Network (C-VTON) that aims to address these issues.

1.4 PROPOSED SYSTEM

To overcome the existing problem this project proposed a new stylized virtual try on clothes, this project used the U2-net is convolutional network architecture for fast and precise segmentation of images, which can not only retain the authenticity of clothing texture and pattern, but also obtain the undifferentiated stylized try on.

this project propose a new image-based virtual Tryon network, i.e. ACGPN, which greatly improves the try-on quality in semantic alignment, character retention and layout adaptation.

this project for the first time take the semantic layout into consideration to generate the photorealistic Tryon results. A novel adaptive content generation and preservation scheme is proposed. A novel second-order difference constraint makes the training process of warping module more stable, and improves the ability of our method to handle complex textures on clothes. Experiments demonstrate that the proposed method can generate photorealistic images that outperform the state-of-the-art methods both qualitatively and quantitatively.

- Incorporate clothing-agnostic person portrayal in order to eliminate the reliance on the attire donned by the reference human figure in the first instance.
- The existing system produces inaccurate output with taking long duration to process the user input.
- The proposed method can capture the reference model in real time.

1.5 OVERVIEW OF THE PROJECT

1. Fashion is the way this project present ourselves to the world and has become one of the world's largest industries. Fashion, mainly conveyed by vision, has thus attracted much attention from computer vision researchers in recent years. Given the rapid development, this paper provides a comprehensive survey of more than 200 major fashion-related works covering four main aspects for enabling intelligent fashion.
2. Fashion detection includes landmark detection, fashion parsing, and item retrieval, Fashion analysis contains attribute recognition, style learning, and popularity prediction.

Fashion synthesis involves style transfer, pose transformation, and physical simulation, and Fashion recommendation comprises fashion compatibility, outfit matching, and hairstyle suggestion. For each task, the benchmark datasets and the evaluation protocols are summarized. Furthermore, this project highlight promising directions for future research.

3. Utilize image-based virtual try-on methodology called Attire Fit-In to generate synthetic image of the client trying the apparel.
4. Virtual Try on has even more benefits for customers through online shopping.
5. A customers can enjoy the convenience of shopping from home but can get a better idea of what they look like wearing on the item.

1.6 PROBLEM DEFINITION

The aim of this project is to try the frontal women top virtually using Deep learning Algorithm. Many issues have been founded in the literature for Virtual Top-try on. Both algorithm is time consuming. As it gradually takes more time to provide the outcome.

- In online shopping this project can only view the colour and brand, price the doesn’t tries on clothes in online shopping.
- The existing system produces inaccurate output with taking long duration to process the user input.
- This project aim to build a virtual try-on, ‘Top-Try-On’, which would help people visualizing particular piece of clothing would look on them.

CHAPTER 2

LITERATURE REVIEW

1. **Author Name:** Chia-Wei Hsieh, Chei-Yun Chen, Chein-Lung Chou

   Published Year: 2022

   Publication: IEEE a vibrational u-net for conditional appearance and shape.

   Introduction: The revenue of e-commerce market has increased recently, where the market's largest segment is Fashion with a market volume of $598,631 million in 2022 and it is expected to present a market volume of US $835,781 million by 2023.1 Although people get used to going online shopping for fashion items, the average conversion rate of online shopping platforms worldwide in 2018 is only 2.42%.2 One of the major reasons is that people cannot try fashion items to find whether they are suitable or not when shopping online.

   Result and Discussion: In this work, this project propose a new virtual try-on system (Fit Me) that can generate virtual try-on images with arbitrary customer poses. By first transforming the pose of the user to a target pose according to the joints of the target pose, this project warp the in shop clothing to fit the target pose and preserve characteristics of both customers and clothing.

   Advantages: To achieve the transformed task, conditional GANs can be utilized to generate a realistic image.

   Disadvantages: It is challenging to synthesize virtual try on images with arbitrary poses since it requires pose transformation and clothing warping at the same time.

   Future Scope: In the future, this project plan to take the human segmentation information into consideration for generating better results.

2. **Author Name :** Shanchen Pang, Xixi Tao, Neal N Xiong, Yukun Dong

   Published Year: 2021

   Publication: Proceedings of the IEEE/CVF International conference and computer vision.

   Introduction: Virtual try on is based on 3D matching, which is similar to beauty effects. The makeup effects are all virtual images of standard human faces, and the relative position information of facial features is not learned. And the virtual outfit change based on image makes use of generation adversarial network to make the generated pictures close to the real results of the fitting.

   Result and Discussion: Structural similarity (SSIM) was used to evaluate the similarity between the composite image and the real image.The higher the score on these two indicators, the higher the quality of the results. Table I lists the SSIM and IS scores given by Style-VTON. In the stylized fitting stage, this paper evaluates the model performance.

   Advantages : On the trying on effect, the editor with diversified design styles makes the fitting result closer to the aesthetic needs of users.

   Disadvantages: Virtual trying on is one of the most challenging tasks.

   Future Scope : This paper proposes a simple and effective virtual try on model based on images, using a three-stage design strategy, so that the texture in the fitting results can be well mapped in the context of complex figure posture and background. Finally, the resulting fitting results are stylized and minimized to produce attractive results. This paper introduces the innovation of pix22Dsurf method to improve the detail quality of image synthesis. Both the quantitative evaluation and the user perception opinion prove that the method in this paper is able to produce better results

3. **Author Name :** Shervin Minaee, Yuri Boykov, Fatih Porikli, Antonio Plaza

   Published Year: 2021

   Publication: IEEE Transaction on Image Processing,

   Introduction : Image segmentation has been a fundamental problem in computer vision since the early days of the field. Image segmentation can be formulated as the problem of classifying pixels with semantic labels or partitioning of individual objects. Semantic segmentation performs pixel-level labeling with a set of object categories for classification, which predicts a single label for the entire image.

   Result and Discussion : This project have surveyed image segmentation algorithms based on deep learning models, which have achieved impressive performance in various image segmentation tasks and benchmarks, grouped into architectural categories such as: CNN and FCN, RNN, R-CNN, dilated CNN, attention based models, generative and adversarial models, among others.

   Advantages : In recent years, however, deep learning (DL) models have yielded a new generation of image segmentation models with remarkable performance improvements, often achieving the highest accuracy rates on popular benchmarks compared to earliest methods.

   Disadvantages: There is now broad agreement that the performance of DL based segmentation algorithms is plateauing, especially in certain app domains such as medical image analysis.

   Future Scope : This project will next discuss some of the promising research directions that this project believe will help in further advancing image segmentation algorithms. generative and adversarial models, among others. This project have summarized the quantitative performance of these models on some popular benchmarks, such as the PASCAL VOC, MS COCO, Cityscapes, and ADE20k datasets.

CHAPTER 3
SYSTEM REQUIREMENTS

3.1 SOFTWARE REQUIREMENT SPECIFICATION

Operating System - Windows 7/8/10

Platform - Google Colab (It will allow anybody to write and execute arbitrary python code through the browser and it specially well suited to machine learning and deep learning and it supports both GPU and TPU instance, which makes it perfect tools for deep learning and data analytics, It has inbuilt libraries like tensor flow, pillow, pytorch, open cv)

Server side Script - Python

3.2 HARDWARE REQUIREMENTS SPECIFICATION

- Processor – I3 (min)
- Speed - 2.5 GHz
- Hard Disk - 10 GB

3.3 FUNCTIONAL REQUIREMENTS

Online Learning

As the data keeps on appending i.e., this project get more and more data every day, the model will keep on learning more by getting more accurate data and thus increasing the accuracy.

User Interface

The proposed system should possess simple, fluid and a dynamic user interface that gives better results of the gestures in an intuitive fashion. Accurate Analysis The proposed system should accurately analyse the given dataset and predict outliers in the dataset without any fatal errors.

Data Analysis

It is a process of analysing the raw data to use the information in a system. This process involves various modules like data acquisition, data pre-processing and data visualization.

3.4 NON – FUNCTIONAL REQUIREMNETS

- **Usability:**

  System should be user-friendly as per the user requirements to get better accuracy and it should be interactive.

- **Reliability:**

  The system is more reliable because the qualities are inherited from Deep learning.

- **Availability:**

  The APIs should be always available for seamless behaviour.

- **Security:**

  User needs to login when they start the program option is also provided to create the additional user level security. Presently user level security is not set but can be implemented.

- **Performance:**

  The predictions and classifications should be pretty accurate to detect outliers in the data.

CHAPTER 4
SYSTEM ANALYSIS

4.1 SYSTEM DESIGN

4.1.1 System Architecture

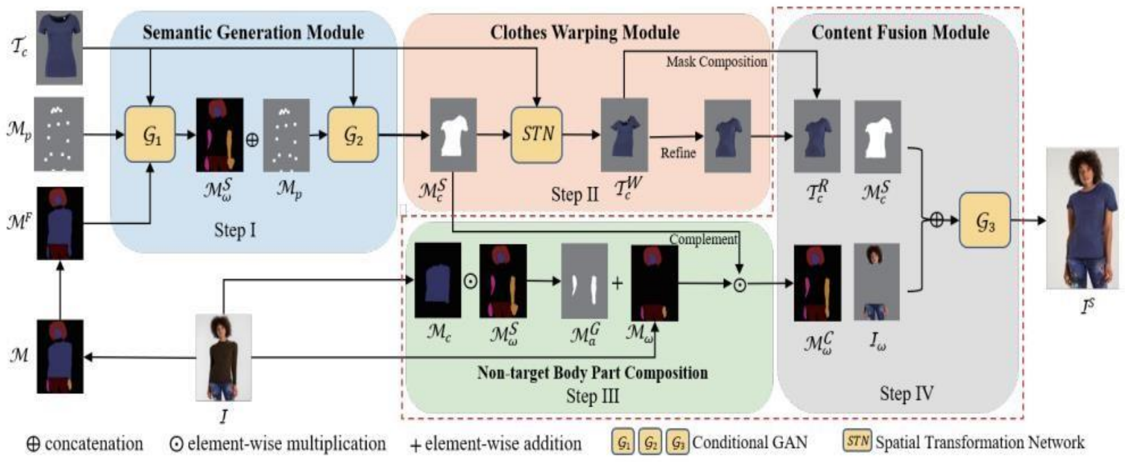


Figure 4.1 System Architecture

T_c - target cloth , M- Masking , M_c – Cloth Mask, M_p- pose map, MF fused body part mask M_V^s - synthesized body part mask , M_c^s - synthesized clothing mask , M_w^s - Synthesized clothing mask, T_c^w - Wrapped Cloth Image, T_c^r -Refined Cloth. The proposed ACGPN is composed of three modules, as shown in figure First, the Semantic Generation Module (SGM) progressively generates the mask of the body parts and the mask of the warped clothing regions via semantic segmentation, yielding semantic alignment of the spatial layout. Second, the Clothes Warping Module (CWM) is designed to warp target clothing image according to the warped clothing mask, where this project introduce a second-order difference constraint on Thin-Plate Spline (TPS) to produce geometric matching yet character retentive clothing images.in the Content Fusion Module (CFM), which integrates the information from previous modules to adaptively determine the generation or preservation of the distinct human parts in output synthesized image.

Non-target body part composition is able to handle different scenarios flexibly in try-on task while mask in painting fully exploits the layout adaptation ability of the ACGPN when dealing with the images from easy, medium, and hard levels of difficulty.

4.2 LOW LEVEL DESIGN

The model consists of four main modules:

1. Semantic Generation
2. Clothes Warping
3. Non target body part Composition
4. Fusion Module

1. Semantic Generation Module

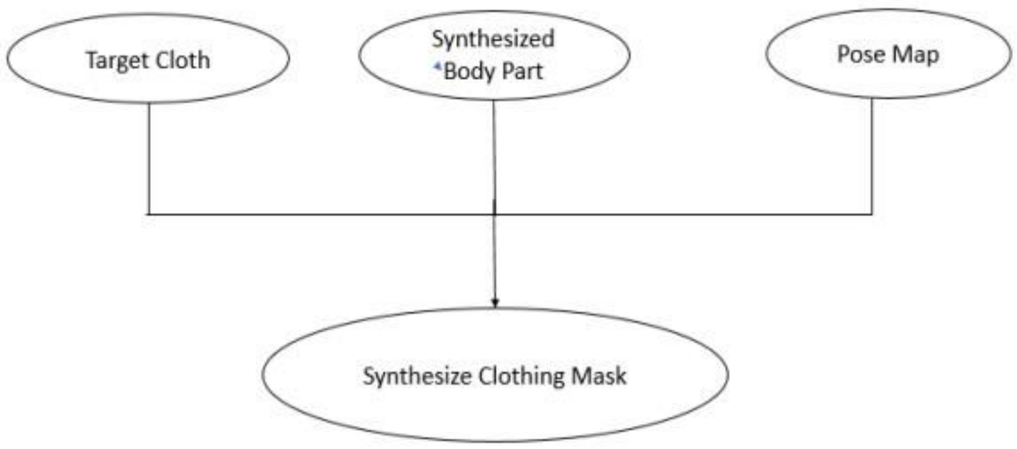


Figure 4.2 Semantic Generation Module

Semantic Segmentation is a computer vision task in which the goal is to categorize each pixel in an image into a class or object. The goal is to produce a dense pixel-wise segmentation map of an image, where each pixel is assigned to a specific class or object. The semantic generation module (SGM) is proposed to separate the target clothing region as well as to preserve the body parts (i.e. arms) of the person, without changing the pose and the rest human body details. Many previous works focus on the target clothes but overlook human body generation by only feeding the coarse body shape directly into the network, leading to the loss of the body part.

To address this issue, the mask generation mechanism is adopted in this module to generate semantic segmentation of body parts and target clothing region precisely. The Semantic Generation module receives the image of a target clothing and its mask, data on the person's pose, a segmentation map with all the body parts (hands are especially important), and clothing items identified.

2. Clothes Warping Module

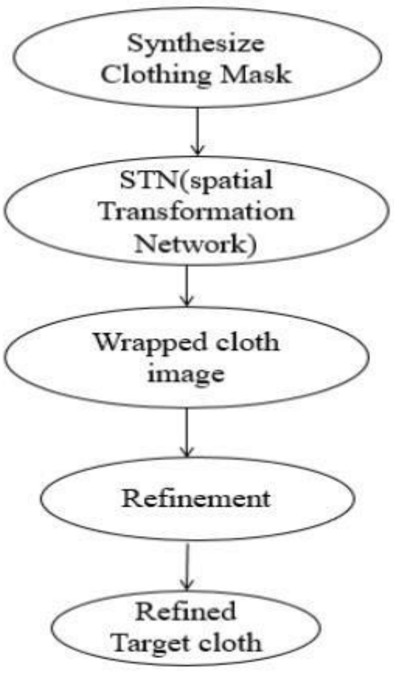


Figure 4.3 Clothes Warping Module

Clothes warping aims to fit the clothes into the shape of target clothing region with visually natural deformation according to human pose as well as to retain the character of the clothes. However, simply training a Spatial Transformation Network (STN) cannot ensure the precise transformation especially when dealing with hard cases (i.e. the clothes with complex texture and rich colors), leading to misalignment and blurry results.

To address these problems, this project introduce a second-order difference on the clothes warping network to realize geometric matching and character retention.

• Spatial Transformation Network (STN): STN is used in computer vision tasks to improve the accuracy of neural network models by allowing them to handle spatial transformations such as rotations, translations, scaling, and cropping.

• Refine: Refinement in deep learning generally refers to the process of improving the accuracy and performance of a pre-trained deep learning model on a specific task or dataset.

3. Non-target body part Composition

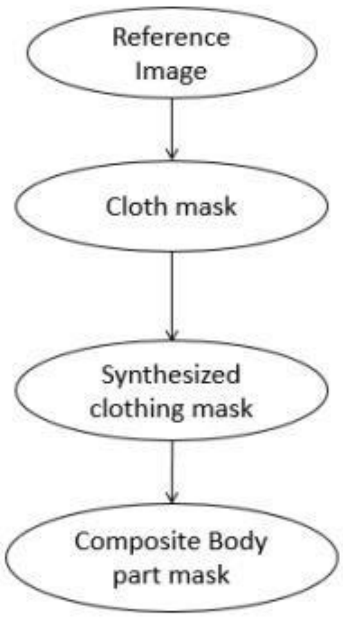


Figure 4.4 Non target body part Composition

Finally, the warped clothing image, the modified segmentation map from Semantic Generation Module, and a person's image are fed into the third generative module, and the final result is produced. It precisely preserves the non-target body part by which are used to fully recover the non-targeted details and generate coherent body parts with the guidance.

4. Content Fusion Module

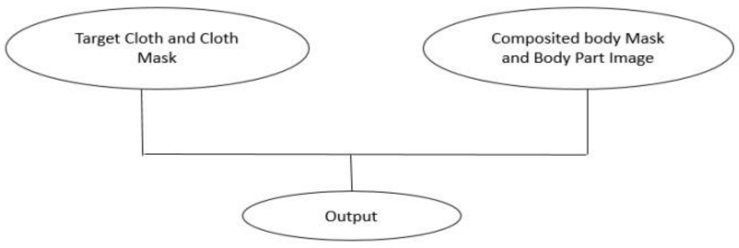

Target Cloth and Cloth Mask
Composited body Mask and Body Part Image
Output

Figure 4.5 Content Fusion Module

Going beyond semantic alignment and character retention, it remains a great challenge to realize layout adaptation on visual try-on task. To this end, both the target clothing region is required to clearly rendered, and fine-scale details of body parts (i.e. finger gaps) are needed to be adaptively preserved. Existing methods usually adopt the coarse body shape as a cue to generate the final try-on images, and fail to reconstruct fine details. In contrast, the proposed content fusion module (CFM) is composed designed to fully maintain the untargeted body parts as well as adaptively preserve the changeable body part (i.e. arms). It fills in the changeable body part by utilizing the masks and images generated from previous steps accordingly by an in painting based fusion GAN.

4.3 HIGH LEVEL DESIGN

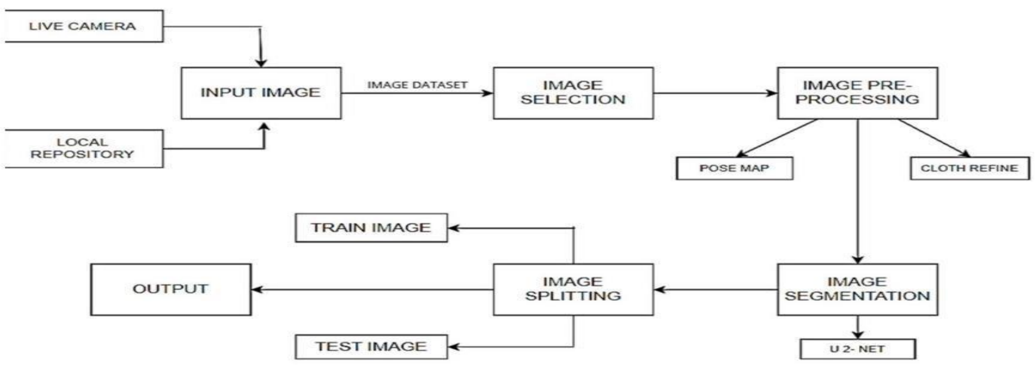


Figure 4.6 High Level Design

This project can select the image from local storage or through live camera and Image dataset (Frontal women and top images) is implemented as input. The input dataset is taken from dataset repository. The dataset is in the format .jpg or png. Image selection: In this step, this project can choose the input image as top and frontal women image. Pre-processing the collected input images are subjected to preprocessing. Here, pose map and cloth refinement is done in order get the desired cloth image. In Image Segmentation this project can segment the preprocessed image using u2-net architecture (The u-net is convolutional network architecture for fast and precise segmentation of images. Up to now it has outperformed the prior best method. Image segmentation is a computer vision task that segments an image into multiple areas by assigning a label to every pixel of the image. It provides much more information about an image than object detection, which draws a bounding box around the detected object, or image classification, which assigns a label to the object.). After this step ,the preprocessed data's are split into train set and test set for decision Train image is used for evaluate the model. Test image is used for predict the model. In this step, this project used to segmentation with U-Net. Final step is output, this project can fit or suggest the appropriate top to corresponding input frontal women image.

4.4 DATA FLOW DIAGRAM

DFD graphically representing the functions, or processes, which capture, manipulate, store, and distribute data between a system and its environment and between components of a system. The visual representation makes it a good communication tool between User and System designer. Structure of DFD allows starting from a broad overview and expand it to a hierarchy of detailed diagrams. DFD has often been used due to the following reasons:

• Logical information flow of the system • Determination of physical system construction requirements • Simplicity of notation • Establishment of manual and automated systems requirement.

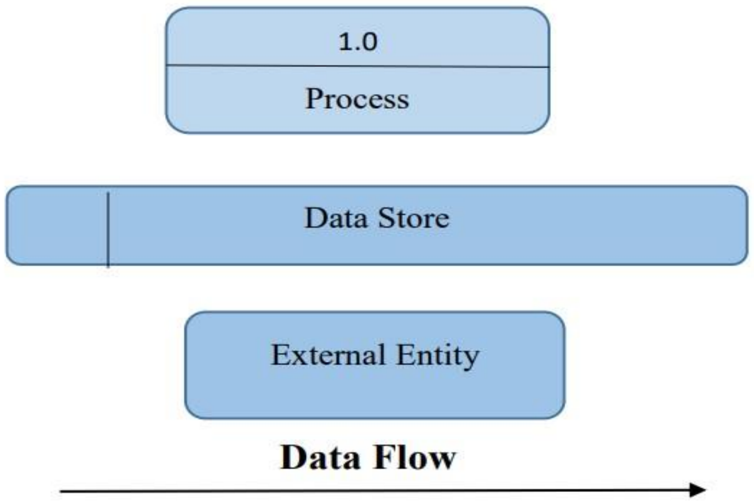


Figure 4.7 Data Flow Basic Notation

Process: any process that changes the data, producing an output. It might perform computations sort data based on logic, or direct the data flow based on business rules. A short label is used to describe process, such as the “Submit payment.

External entity: an outside system that sends or receives data, communicating with the system being diagrammed. They are the sources and destinations of information entering or leaving the system.
They might be an outside organization or person, a computer system or a business system. They are also known as terminators, sources and sinks or actors.

Data flow: the route that data takes between the external entities, processes and data stores. It portrays the interface between the other components and is shown with arrows, typically labelled with a short data name, like “Billing details.

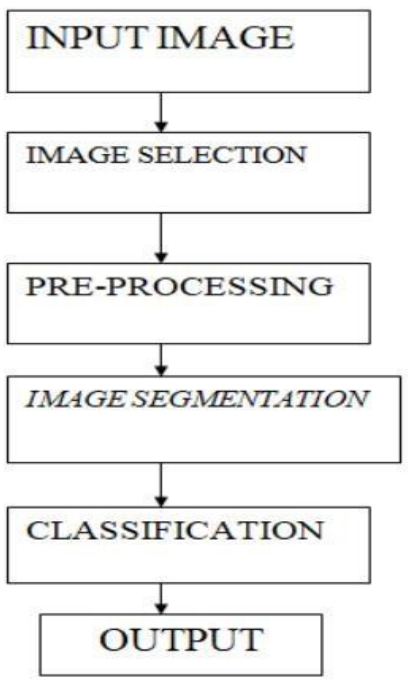


Figure 4.8 Data Flow Diagram

4.5 USE-CASE DIAGRAM

Use case diagram is a graph of actors, a set of use cases enclosed by a system boundary, communication associations between the actor and the use case. The use case diagram describes how a system interacts with outside actors; each use case represents a piece of functionality that a system provides to its users.

A use case is known as an ellipse containing the name of the use case and an actor is shown as a stick figure with the name of the actor below the figure. The use cases are used during the analysis phase of a project to identify and partition system functionality. They separate the system into actors and use case. Actors represent roles that are played by user of the system. Those users can be humans, other computers, pieces of hardware, or even other software systems.

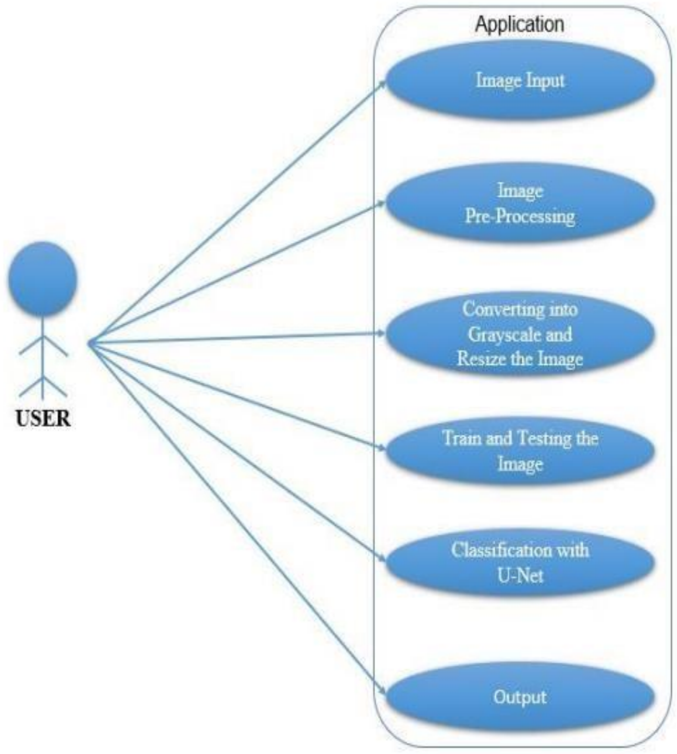

Application
Image Input
Image Pre-Processing
Converting into Grayscale and Resize the Image
Train and Testing the Image
Classification with U-Net
Output
USER

Figure 4.9 Use Case Diagram

4.6 SEQUENCE DIAGRAM

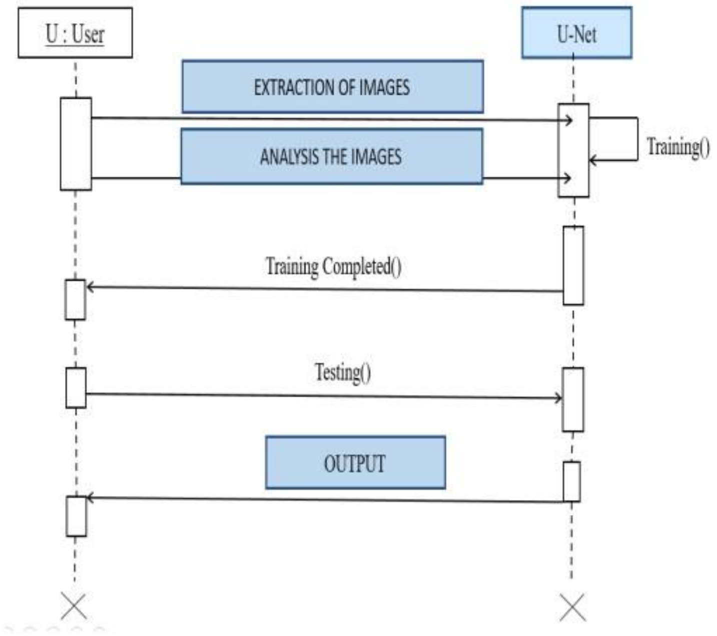


Figure 4.10 Sequence Diagram

CHAPTER 5
REQUIREMENT ANALYSIS

5.1 LIBRARIES:

1. **Tensor Flow**: TensorFlow is an open-source machine learning library that is widely used for deep learning applications. It provides a wide range of tools for building and training neural networks, and is commonly used for tasks such as image recognition and natural language processing.
2. **PyTorch** : PyTorch is another open-source deep learning library that is widely used in virtual try-on systems. It is known for its dynamic computational graph, for scaling rotating which allows for more flexible model design and training.
3. **IPYTHON-ENABLED PYTHON DEBUGGER**: IPDB is a Python debugger that allows you to set breakpoints, inspect variables, and step through code execution. While debugging is an important part of the development process in any field, including deep learning, there are other more commonly used libraries and tools that are specifically designed for deep learning tasks.
4. **OpenCV**: OpenCV (Open Source Computer Vision Library) is a popular open-source computer vision library that is commonly used in deep learning for tasks such as image and video processing, object detection, and facial recognition. OpenCV provides a wide range of tools and algorithms for image and video processing, including tools for filtering, thresholding, edge detection, and feature detection. These tools can be used to preprocess images and extract useful features for use in deep learning models.
5. **Pillow** : The Pillow library contains all the basic image processing functionality. You can do image resizing, rotation and transformation.

6. **TensorBoardX**: TensorBoardX is a library that provides visualization tools for deep learning tasks in PyTorch. It is based on the original TensorBoard library from TensorFlow and allows you to monitor the training and evaluation process of your PyTorch models. With TensorBoardX, you can visualize a variety of metrics such as loss, accuracy, and learning rate during training, as well as visualize the model's structure and the distribution of weights and biases. You can also visualize the gradients and activations of different layers of the model, which can help with debugging and fine-tuning.
7. **NINJA**, which stands for "Network In Network with Squeeze and Excitation blocks and Self Attention" is a deep learning architecture that was proposed by authors of the paper "NINJA: Nonlinear Independent feature Analysis for Joint Diagonalization" in 2021.
8. **NumPy** is a fundamental library in Python that is commonly used in deep learning for numerical operations and data manipulation. NumPy provides an array data structure that is similar to the native Python list, but with several key advantages, including faster execution speed and built-in functions for mathematical operations.
9. **PIL**: The Python Imaging Library, or PIL for short, is an open source library for loading and manipulating images. It was developed and made available more than 25 years ago and has become a de facto standard API for working with images in Python.

5.2 ACGPN MODEL

(Adaptive Content Generating and Preserving Network) is a deep learning model that is specifically designed for image-to-image translation tasks, with a focus on generating realistic images of human faces with controllable attributes.

The ACGPN model is based on the which involves a generator network that generates images and a discriminator network that evaluates the realism of the generated images. In ACGPN, the generator network is an attribute controlled generator that can generate images with specific attributes such as gender, age, and expression. The discriminator network is a geometry-preserving discriminator that ensures that the generated images maintain the geometry and structure of the original images, such as the shape of the face and the placement of facial features. The ACGPN model employs a multi-stage training process that involves training the attribute controlled generator network to generate images with specific attributes, and then training the geometry-preserving discriminator network to ensure that the generated images maintain the geometry and structure of the original images. The training process is repeated iteratively until the generated images are indistinguishable from real images. The ACGPN model has achieved impressive results on various benchmark datasets, including the Celeb A dataset, where it outperformed other state of-the-art models in terms of both quantitative and qualitative measures. ACGPN has also been applied to other image-to-image translation tasks, such as style transfer and image colorization. Overall, the ACGPN model is an important contribution to the field of deep learning for image to-image translation tasks such as virtual try on and face editing. This project are using the ACGPN MODEL for the Virtual Try-On System. In this algo, this project are using a mix of U2 -Net and Human Parser to make a generative model that can swap dresses of people in any pose with the desired cloth.

4.4.1 U2-NET Architecture

U2-Net is a type of artificial intelligence architecture that can identify and highlight important objects in an image. It works by analyzing an image at different levels of detail, and then generating a map that shows which parts of the image are the most significant.

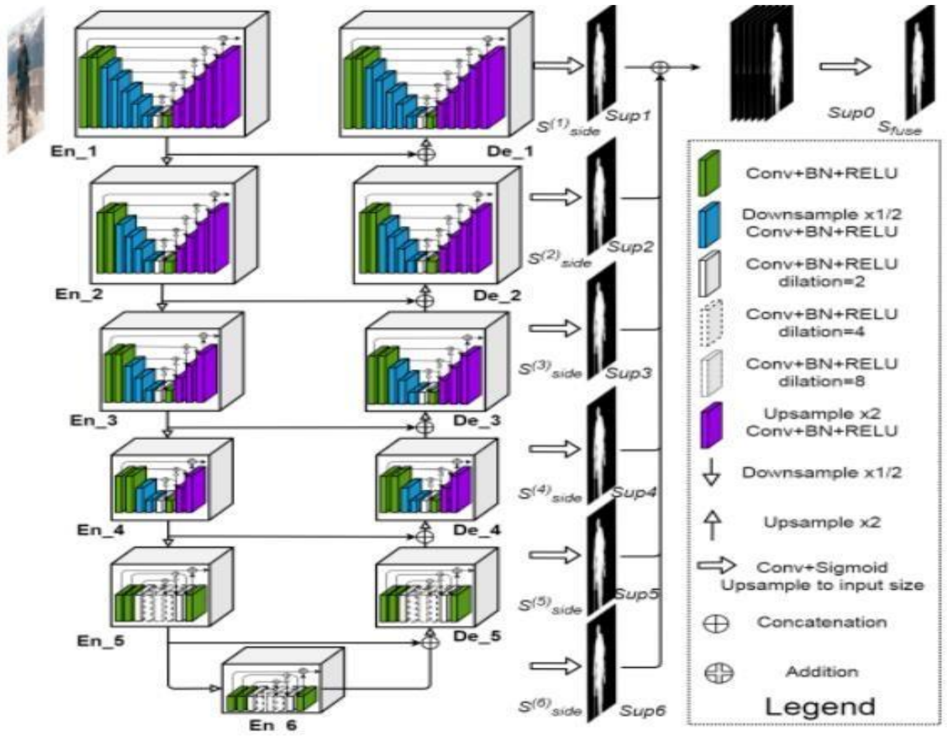


Figure 5.1 U2 – NET Architecture

U2-Net extends the U2-Net architecture by adding a residual connection to each stage of the encoder and decoder. This improves the flow of information through the network and helps to preserve fine details in the segmentation masks. Additionally, U2-Net uses dilated convolutions in the encoder to increase the receptive field of the network without increasing the number of parameters.

1. Architecture Details: The U-2 Net architecture consists of an encoder-decoder structure, where the encoder is composed of a series of convolutional blocks, and the decoder is composed of a series of up-sampling blocks. In each convolutional block of the encoder, there are two convolutional layers followed by a residual connection.

   In the decoder, each up-sampling block consists of an up-sampling layer, a convolutional layer, and residual connection. The final layer of the network is a sigmoid layer, which generates the segmentation mask.

2. Dilated Convolutions: U-2 Net employs dilated convolutions in the encoder to increase the receptive field of the network without increasing the number of parameters. Dilated convolutions have been shown to be effective in capturing contextual information in images, which is important for accurate segmentation.

3. Hybrid Loss Function: U-2 Net uses a hybrid loss function that combines both binary cross-entropy loss and dice loss. The binary cross-entropy loss is used to penalize false positives and false negatives, while the dice loss is used to ensure that the segmentation masks are accurate. The dice loss is calculated based on the overlap between the predicted segmentation mask and the ground truth mask.

4. Training Details: U-2 Net is trained using a combination of the Adam optimizer and the learning rate scheduler. The model is trained on image patches of size 320x320, and data augmentation techniques such as random flipping and rotation are applied to increase the diversity of the training data.

5. Applications: U-2 Net has been used for various image segmentation applications, such as salient object detection, medical image segmentation, and video object segmentation.

DUTS dataset, the PASCAL VOC dataset, and the Cityscapes dataset.

Figure 5.2 Cloth and Mask Image

Overall, the U2-Net architecture is an important contribution to the field of deep learning for image segmentation tasks, and hybrid loss function.

5.2.2 Human Parsing

Human parsing is a fundamental visual understanding. Task, requiring segmenting human instances into explicit body parts as well as some clothing classes at the pixel level Self-human parsing is a task in computer vision that involves segmenting an image of a human into different semantic parts such as head, torso, arms, and legs. This task is important for various applications such as human-computer interaction, virtual try-on, and augmented reality.

Deep learning methods have shown great success in solving self-human parsing tasks. There are various deep learning models and architectures that have been proposed for self human parsing. Once the model is trained, it can be used to segment new images of humans into different semantic parts. This can be done in real-time using a camera or in batch mode using a set of images. Overall, self-human parsing is an important task in computer vision and deep learning methods.

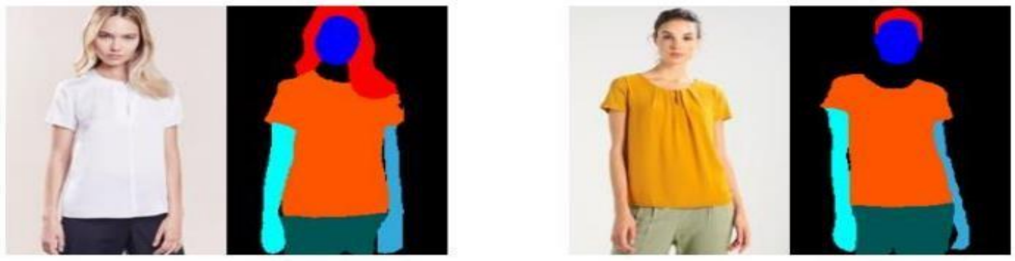


Figure 5.3 Reference Image And Synthesized Human Image

CHAPTER 6

TESTING AND CODING

6.1 TEST CASES

CAMERA	TARGET CLOTH	SOME LOSS DUE TO RESOLUTION OF IMAGE
Frontal image	Target cloth	Good result
Side image	Target cloth	Some loss
Full sleeve image	Half sleeve target cloth	Good result
Half sleeve image	Full sleeve target cloth	Some loss

6.2 TRAINED MODULE

```
import time
from collections import OrderedDict
from options.test_options import TestOptions
from data.data_loader import CreateDataLoader
from models.models import create_model
import util.util as util
import os
import numpy as np
import torch
from torch.autograd import Variable
#from tensorboardX import SummaryWriter
import cv2
#writer = SummaryWriter('runs/G1G2')
SIZE = 320
NC = 14
def generate_label_plain(inputs):
    size = inputs.size()
    pred_batch = []
    for input in inputs:
        input = input.view(1, NC, 256, 192)
        pred = np.squeeze(input.data.max(1)[1].cpu().numpy(), axis=0)
        pred_batch.append(pred)
        pred_batch = np.array(pred_batch)
        pred_batch = torch.from_numpy(pred_batch)
       label_batch = pred_batch.view(size[0], 1, 256, 192)
```

```
 return label_batch
def generate_label_color(inputs):
   label_batch = []
   for i in range(len(inputs)):
      label_batch.append(util.tensor2label(inputs[i], NC))
   label_batch = np.array(label_batch)
   label_batch = label_batch * 2 - 1
   input_label = torch.from_numpy(label_batch)
   return input_label
def complete_compose(img, mask, label):
   label = label.cpu().numpy()
   M_f = label > 0
   M_f = M_f.astype(np.int)
   M_f = torch.FloatTensor(M_f).cuda()
   masked_img = img*(1-mask)
   M_c = (1-mask.cuda())*M_f
   M_c = M_c+torch.zeros(img.shape).cuda()  # broadcasting
   return masked_img, M_c, M_f
def compose(label, mask, color_mask, edge, color, noise):
   masked_label = label*(1-mask)
   masked_edge = mask*edge
   masked_color_strokes = mask*(1-color_mask)*color
   masked_noise = mask*noise
   return masked_label, masked_edge, masked_color_strokes, masked_noise
def changearm(old_label):
   label = old_label
   arm1 = torch.FloatTensor((old_label.cpu().numpy() == 11).astype(np.int))
   arm2 = torch.FloatTensor((old_label.cpu().numpy() == 13).astype(np.int))
   noise = torch.FloatTensor((old_label.cpu().numpy() == 7).astype(np.int))
   label = label*(1-arm1)+arm1*4
   label = label*(1-arm2)+arm2*4
   label = label*(1-noise)+noise*4
   return label
def main():
   os.makedirs('sample', exist_ok=True)
   opt = TestOptions().parse()
   data_loader = CreateDataLoader(opt)
   dataset_size = len(data_loader)
   print('# Inference images = %d' % dataset_size)
   model = create_model(opt)
   for i, data in enumerate(dataset):
      # add gaussian noise channel
      # wash the label
      t_mask = torch.FloatTensor(
         (data['label'].cpu().numpy() == 7).astype(np.float))
```

```
        #
        # data['label'] = data['label'] * (1 - t_mask) + t_mask * 4
        mask_clothes = torch.FloatTensor(
          (data['label'].cpu().numpy() == 4).astype(np.int))
        mask_fore = torch.FloatTensor(
          (data['label'].cpu().numpy() > 0).astype(np.int))
        img_fore = data['image'] * mask_fore
        img_fore_wc = img_fore * mask_fore
        all_clothes_label = changearm(data['label'])

        ############## Forward Pass ######################
        fake_image, warped_cloth, refined_cloth = model(Variable(data['label'].cuda()),
Variable(data['edge'].cuda()), Variable(img_fore.cuda()), Variable(
          mask_clothes.cuda()),                              Variable(data['color'].cuda()),
Variable(all_clothes_label.cuda()),                        Variable(data['image'].cuda()),
Variable(data['pose'].cuda()), Variable(data['image'].cuda()), Variable(mask_fore.cuda()))

        # make output folders
        output_dir = os.path.join(opt.results_dir, opt.phase)
        fake_image_dir = os.path.join(output_dir, 'try-on')
        os.makedirs(fake_image_dir, exist_ok=True)
        warped_cloth_dir = os.path.join(output_dir, 'warped_cloth')
        os.makedirs(warped_cloth_dir, exist_ok=True)
        refined_cloth_dir = os.path.join(output_dir, 'refined_cloth')
        os.makedirs(refined_cloth_dir, exist_ok=True)
        # save output
        for j in range(opt.batchSize):
          print("Saving", data['name'][j])
          util.save_tensor_as_image(fake_image[j],
                              os.path.join(fake_image_dir, data['name'][j]))
          util.save_tensor_as_image(warped_cloth[j],
                              os.path.join(warped_cloth_dir, data['name'][j]))
          util.save_tensor_as_image(refined_cloth[j],
                              os.path.join(refined_cloth_dir, data['name'][j]))
if _name_ == '_main_':
  main(
```

6.3 HUMAN PARSING

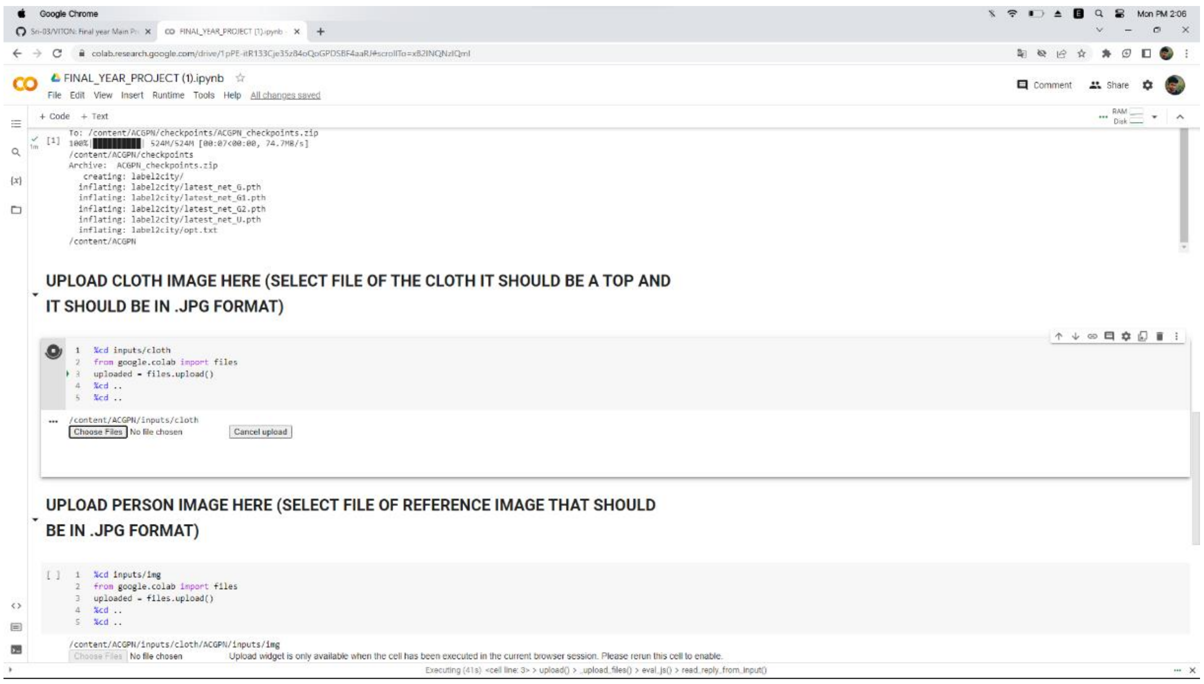
```
    import os
    import torch
    import argparse
    import numpy as np
    from PIL import Image
    from tqdm import tqdm

    from torch.utils.data import DataLoader
    import torchvision.transforms as transforms

    import networks
    from utils.transforms import transform_logits
    from datasets.simple_extractor_dataset import SimpleFolderDataset

    dataset_settings = {
      'lip': {
        'input_size': [473, 473],
        'num_classes': 20,
        'label': ['Background', 'Hat', 'Hair', 'Glove', 'Sunglasses', 'Upper-clothes', 'Dress', 'Coat',
             'Socks', 'Pants', 'Jumpsuits', 'Scarf', 'Skirt', 'Face', 'Left-arm', 'Right-arm',
             'Left-leg', 'Right-leg', 'Left-shoe', 'Right-shoe']
      },
      'atr': {
        'input_size': [512, 512],
        'num_classes': 18,
        'label': ['Background', 'Hat', 'Hair', 'Sunglasses', 'Upper-clothes', 'Skirt', 'Pants', 'Dress',
    'Belt',
             'Left-shoe', 'Right-shoe', 'Face', 'Left-leg', 'Right-leg', 'Left-arm', 'Right-arm', 'Bag',
    'Scarf']
      },
      'pascal': {
      'input_size': [512, 512],
      'num_classes': 7,
      'label': ['Background', 'Head', 'Torso', 'Upper Arms', 'Lower Arms', 'Upper Legs', 'Lower
  Legs'],
    }
  }
  def get_arguments():
    """Parse all the arguments provided from the CLI.
    Returns:
     A list of parsed arguments.
    """
    parser = argparse.ArgumentParser(description="Self Correction for Human Parsing")
    parser.add_argument("--dataset", type=str, default='lip', choices=['lip', 'atr', 'pascal'])
    parser.add_argument("--model-restore", type=str, default='', help="restore pretrained model
parameters.")
    parser.add_argument("--gpu", type=str, default='0', help="choose gpu device.")
    parser.add_argument("--input-dir", type=str, default='', help="path of input image folder.")
    parser.add_argument("--output-dir", type=str, default='', help="path of output image
folder.")
    parser.add_argument("--logits", action='store_true', default=False, help="whether to save
the logits.")
    return parser.parse_args()


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 main():
   args = get_arguments()
   gpus = [int(i) for i in args.gpu.split(',')]
   assert len(gpus) == 1
   if not args.gpu == 'None':
      os.environ["CUDA_VISIBLE_DEVICES"] = args.gpu
   num_classes = dataset_settings[args.dataset]['num_classes']
   #num_classes = 14 for ACGPN
   input_size = dataset_settings[args.dataset]['input_size']
   label = dataset_settings[args.dataset]['label']
   #print("Evaluating total class number {} with {}".format(num_classes, label))

   model = networks.init_model('resnet101', num_classes=num_classes, pretrained=None)
```

```
    state_dict = torch.load(args.model_restore)['state_dict']
    from collections import OrderedDict
    new_state_dict = OrderedDict()
    for k, v in state_dict.items():
        name = k[7:]  # remove `module.`
        new_state_dict[name] = v
    model.load_state_dict(new_state_dict)
    model.cuda()
    model.eval()

    transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.406, 0.456, 0.485], std=[0.225, 0.224, 0.229])
    ])
    trans_dict = {
        0:0,
        1:1, 2:1,
        5:4, 6:4, 7:4,
        18:5,
        19:6,
        9:8, 12:8,
        16:9,
        17:10,
        14:11,
        4:12, 13:12,
        15:13
    }

dataset=SimpleFolderDataset(root=args.input_dir,input_size=input_size,transform=transform
)
    dataloader = DataLoader(dataset)

    if not os.path.exists(args.output_dir):
        os.makedirs(args.output_dir)

    #palette = get_palette(14)
    with torch.no_grad():
        for idx, batch in enumerate(tqdm(dataloader)):
            image, meta = batch
            img_name = meta['name'][0]
            c = meta['center'].numpy()[0]
            s = meta['scale'].numpy()[0]
            w = meta['width'].numpy()[0]
            h = meta['height'].numpy()[0]
        output = model(image.cuda())
        upsample = torch.nn.Upsample(size=input_size, mode='bicubic', align_corners=True)
        upsample_output = upsample(output[0][-1][0].unsqueeze(0))
        upsample_output = upsample_output.squeeze()
        upsample_output = upsample_output.permute(1, 2, 0)  # CHW -> HWC

        logits_result = transform_logits(upsample_output.data.cpu().numpy(), c, s, w, h,
input_size=input_size)
        parsing_result = np.argmax(logits_result, axis=2)
        parsing_result_path = os.path.join(args.output_dir, img_name[:-4] + '.png')
        output_arr = np.asarray(parsing_result, dtype=np.uint8)

        new_arr = np.full(output_arr.shape, 7)
        for old, new in trans_dict.items():
            new_arr = np.where(output_arr == old, new, new_arr)
        output_img = Image.fromarray(np.asarray(new_arr, dtype=np.uint8))
        #output_img.putpalette(palette)
        output_img.save(parsing_result_path)
        if args.logits:
            logits_result_path = os.path.join(args.output_dir, img_name[:-4] + '.npy')
            np.save(logits_result_path, logits_result)
    return
if _name_ == '_main_':
    main()
```

CHAPTER 7

RESULT

7.1 CHOOSING IMAGE FROM LOCAL REPOSITIRY



```
To: /content/ACGPN/checkpoints/ACGPN_checkpoints.zip
100%|██████████| 524M/524M [00:07<00:00, 74.7MB/s]
/content/ACGPN/checkpoints
Archive:  ACGPN_checkpoints.zip
   creating: label2city/
  inflating: label2city/latest_net_G.pth
  inflating: label2city/latest_net_G1.pth
  inflating: label2city/latest_net_G2.pth
  inflating: label2city/latest_net_U.pth
  inflating: label2city/opt.txt
/content/ACGPN
```

UPLOAD CLOTH IMAGE HERE (SELECT FILE OF THE CLOTH IT SHOULD BE A TOP AND IT SHOULD BE IN .JPG FORMAT)

```
%cd inputs/cloth
from google.colab import files
uploaded = files.upload()
%cd ..
%cd ..
```

```
/content/ACGPN/inputs/cloth
Choose Files No file chosen        Cancel upload
```

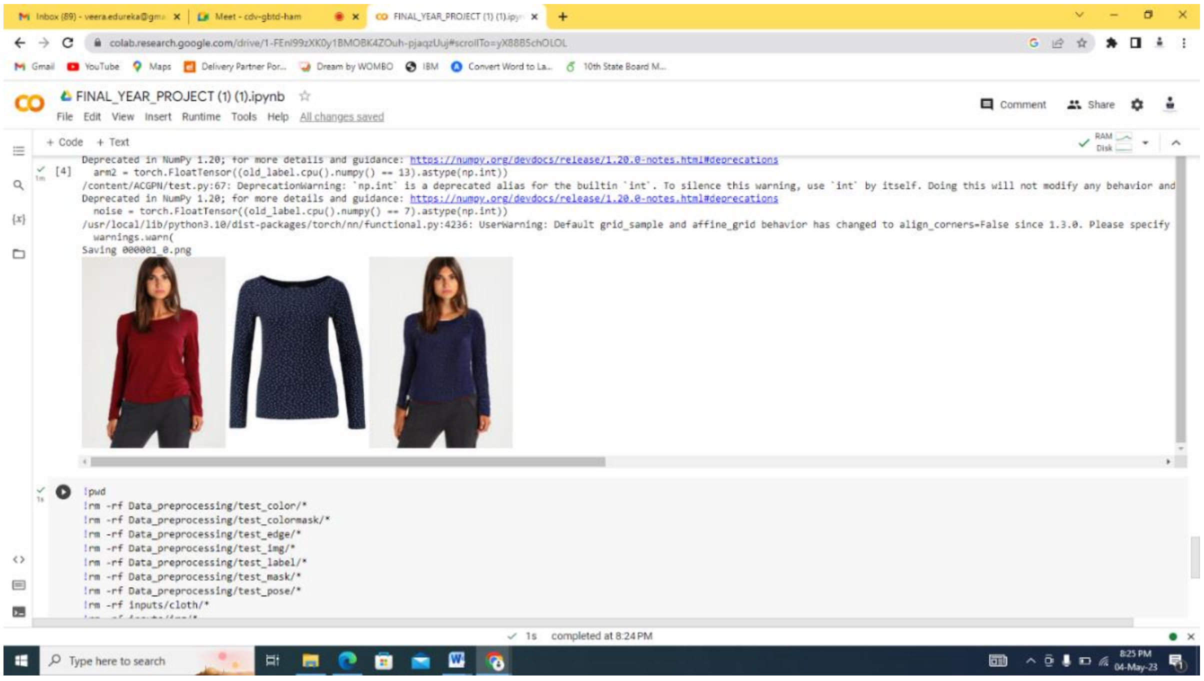
UPLOAD PERSON IMAGE HERE (SELECT FILE OF REFERENCE IMAGE THAT SHOULD BE IN .JPG FORMAT)

```
%cd inputs/img
from google.colab import files
uploaded = files.upload()
%cd ..
%cd ..
```

```
/content/ACGPN/inputs/cloth/ACGPN/inputs/img
Choose Files No file chosen        Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
```

Google Chrome
FINAL_YEAR_PROJECT (1).ipynb
File Edit View Insert Runtime Tools Help All changes saved
Executing (41s) <cell line: 3> > upload() > _upload_files() > eval_js() > read_reply_from_input()

Figure 7.1 Upload Clothing Images



```
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  arm2 = torch.FloatTensor((old_label.cpu().numpy() == 13).astype(np.int))
/content/ACGPN/test.py:67: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not modify any behavior and
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  noise = torch.FloatTensor((old_label.cpu().numpy() == 7).astype(np.int))
/usr/local/lib/python3.10/dist-packages/torch/nn/functional.py:4236: UserWarning: Default grid_sample and affine_grid behavior has changed to align_corners=False since 1.3.0. Please specify
  warnings.warn(
Saving 000001_0.png
```

```
!pwd
!rm -rf Data_preprocessing/test_color/*
!rm -rf Data_preprocessing/test_colormask/*
!rm -rf Data_preprocessing/test_edge/*
!rm -rf Data_preprocessing/test_img/*
!rm -rf Data_preprocessing/test_label/*
!rm -rf Data_preprocessing/test_mask/*
!rm -rf Data_preprocessing/test_pose/*
!rm -rf inputs/cloth/*
```

FINAL_YEAR_PROJECT (1) (1).ipynb
File Edit View Insert Runtime Tools Help All changes saved
1s completed at 8:24 PM
Type here to search

Figure 7.2 Generating Images

CHAPTER 8

CONCLUSION AND FUTUREWORK

Virtual try-on using deep learning has great potential to revolutionize the way people shop for clothes online. With advancements in computer vision and deep learning techniques, virtual try-on systems can provide a realistic and personalized experience for customers, increasing their confidence in purchasing clothes online. In conclusion, virtual try-on systems using deep learning have shown promising results in accurately and efficiently mapping virtual clothes onto a person's body. Future work in this area could include improving the realism and accuracy of the virtual try-on system by incorporating more advanced techniques, such as generative adversarial networks (GANs) and 3D modeling. Additionally, there is room for improvement in the integration of virtual try-on systems into e-commerce platforms, making the experience more seamless and accessible to customers. One area of future work for virtual try-on using deep learning is improving the diversity of body types and skin tones represented in the virtual models. Many current systems are trained on a limited dataset of body types and skin tones, which can lead to inaccuracies and biases in the virtual try-on experience. By incorporating a more diverse range of models and data, virtual try-on systems can better serve a wider range of customers. Another area of research in virtual try-on using deep learning is incorporating additional contextual information to improve the overall shopping experience. Additionally, virtual try-on systems could leverage environmental data, such as weather or occasion, to suggest appropriate clothing options. There is also potential for virtual try-on using deep learning to extend beyond just clothing and into other areas, such as beauty and cosmetics.

APPENDIX

SOURCE CODE

```
# -*- coding: utf-8 -*-
"""FINAL_YEAR_PROJECT (1) (1).ipynb

Automatically generated by Colaboratory.

Original file is located at
https://colab.research.google.com/drive/1-FEnl99zXK0y1BMOBK4ZOuh-pjaqzUuj

# **IMPORT THE MODEL AND INSTALL ALL THE REQUIRED LIBRARIES**
"""

# Commented out IPython magic to ensure Python compatibility.
!pwd
# Here this project import the main GITHUB repository
!git clone https://github.com/Sri-03/VITON
# %cd ACGPN
!pip install ninja
import gdown
import numpy as np
from PIL import Image
import IPython
import os
import sys

from predict_pose import generate_pose_keypoints
!mkdir Data_preprocessing/test_color
!mkdir Data_preprocessing/test_colormask
!mkdir Data_preprocessing/test_edge
!mkdir Data_preprocessing/test_img
!mkdir Data_preprocessing/test_label
!mkdir Data_preprocessing/test_mask
!mkdir Data_preprocessing/test_pose
!mkdir inputs
!mkdir inputs/img
!mkdir inputs/cloth
# %cd pose
!gdown --id 1IMoLEcTGO3wBQkt3h7XSSZ99E7rfsJVp
# %cd ..
!git clone https://github.com/Sri-03/Self-Correction-Human-Parsing-for-ACGPN




!git clone https://github.com/Sri-03/U-2-Net
#for segmentation mask generation
url = 'https://drive.google.com/uc?id=1IZaTLTCV6PP63TGhrW6ZUm0Z-aCq5Rsi'
```

```
output = 'lip_final.pth'
gdown.download(url, output, quiet=False)
# %cd U-2-Net
!mkdir saved_models
!mkdir saved_models/u2net
!mkdir saved_models/u2netp
!gdown --id 1InAEi6jTvvZ_VMOcSpgVaTKRmUfQiOi3 -O saved_models/u2netp/u2netp.pth
!gdown --id 1Ifr74yn9en-GT6b0RhT3kDM5oOsbKY99 -O saved_models/u2net/u2net.pth
import u2net_load
import u2net_run
u2net = u2net_load.model(model_name = 'u2netp')
# %cd ..
!mkdir checkpoints
gdown.download('https://drive.google.com/uc?id=1J2IzVAmtI4PfSx56Ow81YMfqVZUrMesf',o
utput='checkpoints/ACGPN_checkpoints.zip', quiet=False)
# %cd checkpoints
!unzip ACGPN_checkpoints
# %cd ..

"""

# **UPLOAD CLOTH IMAGE HERE (SELECT FILE OF THE CLOTH IT SHOULD BE A
TOP AND IT SHOULD BE IN .JPG FORMAT)** """

# Commented out IPython magic to ensure Python compatibility.
# %cd inputs/cloth
from google.colab import files
uploaded = files.upload()
# %cd ..
# %cd ..

"""# **UPLOAD PERSON IMAGE HERE (SELECT FILE OF REFERENCE IMAGE THAT
SHOULD BE IN .JPG FORMAT)**"""

# Commented out IPython magic to ensure Python compatibility.
# %cd inputs/img
from google.colab import files
uploaded = files.upload()
# %cd ..
# %cd ..

"""# **LIVE CAMERA CAPTURING** (*CLICK ON CPATURE BUTTON TO CAPTURE
YOUR IMAGE*)"""

from IPython.display import display, Javascript
from google.colab.output import eval_js
from base64 import b64decode


def take_photo(filename='/content/ACGPN/inputs/img/photo.jpg', quality=0.8):
  js = Javascript('''
    async function takePhoto(quality) {
      const div = document.createElement('div');
      const capture = document.createElement('button');
      capture.textContent = 'Capture';
      div.appendChild(capture);

      const video = document.createElement('video');
      video.style.display = 'block';
      const stream = await navigator.mediaDevices.getUserMedia({video:    true});

      document.body.appendChild(div);
      div.appendChild(video);
      video.srcObject = stream;
      await video.play();

      // Resize the output to fit the video element.
      google.colab.output.setIframeHeight(document.documentElement.scrollHeight,true);

      // Wait for Capture to be clicked.
      await new Promise((resolve) => capture.onclick = resolve);

      const canvas = document.createElement('canvas');
      canvas.width = video.videoWidth;
      canvas.height = video.videoHeight;
      canvas.getContext('2d').drawImage(video, 0, 0);
      stream.getVideoTracks()[0].stop();
      div.remove();
      return canvas.toDataURL('image/jpeg', quality);
    }
    ''')
  display(js)
  data = eval_js('takePhoto({})'.format(quality))
  binary = b64decode(data.split(',')[1])
  with open(filename, 'wb') as f:
    f.write(binary)
  return filename

  from IPython.display import Image
try:
  filename = take_photo()
  print('Saved to{}'.format(filename ))


  display (Image(filename ))
except Exception as err:
   #Errors be thrown if the user does not have a webcam or if they do not
   #grant the page permission to access it.
  print(str(err))

"""# **RUN THE MODEL**"""

sorted(os.listdir('inputs/cloth'))
from PIL import Image
cloth_name = '000001_1.png'
cloth_path = os.path.join('inputs/cloth', sorted(os.listdir('inputs/cloth'))[0])
cloth = Image.open(cloth_path)
cloth = cloth.resize((192, 256), Image.BICUBIC).convert('RGB')
cloth.save(os.path.join('Data_preprocessing/test_color', cloth_name))

u2net_run.infer(u2net, 'Data_preprocessing/test_color', 'Data_preprocessing/test_edge')
os.listdir('inputs/img')

import time

start_time = time.time()
img_name = '000001_0.png'
img_path = os.path.join('inputs/img', sorted(os.listdir('inputs/img'))[0])
img = Image.open(img_path)
img = img.resize((192,256), Image.BICUBIC)

img_path = os.path.join('Data_preprocessing/test_img', img_name)
img.save(img_path)
resize_time = time.time()
print('Resized image in {}s'.format(resize_time-start_time))

!python3 Self-Correction-Human-Parsing-for-ACGPN/simple_extractor.py --dataset 'lip' --model-
restore 'lip_final.pth' --input-dir 'Data_preprocessing/test_img' --output-dir
'Data_preprocessing/test_label'
parse_time = time.time()
print('Parsing generated in {}s'.format(parse_time-resize_time))

pose_path = os.path.join('Data_preprocessing/test_pose', img_name.replace('.png',
'_keypoints.json'))
generate_pose_keypoints(img_path, pose_path)
pose_time = time.time()
print('Pose map generated in {}s'.format(pose_time-parse_time))
!rm -rf Data_preprocessing/test_pairs.txt



with open('Data_preprocessing/test_pairs.txt','w') as f:
f.write('000001_0.png 000001_1.png')
!python test.py
```

```
# Here is a Sample Output of the Model
output_grid =
np.concatenate([np.array(Image.open('Data_preprocessing/test_img/000001_0.png')),
               np.array(Image.open('Data_preprocessing/test_color/000001_1.png')),
               np.array(Image.open('results/test/try-on/000001_0.png'))], axis=1)
image_grid = Image.fromarray(output_grid)
image_grid

# Commented out IPython magic to ensure Python compatibility.
!pwd
!rm -rf Data_preprocessing/test_color/*
!rm -rf Data_preprocessing/test_colormask/*
!rm -rf Data_preprocessing/test_edge/*
!rm -rf Data_preprocessing/test_img/*
!rm -rf Data_preprocessing/test_label/*
!rm -rf Data_preprocessing/test_mask/*
!rm -rf Data_preprocessing/test_pose/*
!rm -rf inputs/cloth/*
!rm -rf inputs/img/*
!rm -rf results/*
# %cd /content/ACGPN
```

REFERENCES

[1] Shervin Minaee, Yuri Boykov,Antonio Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos, **“**IMAGE SEGMENTATION USING DEEP LEARNING**”,** IEEE Journal, 2022.

[2] Shanchen Pang,Xixi Tao, Neal N. Xiong, Dong, “AN EFFICIENT STYLE VIRTUAL TRY ON NETWORK FOR CLOTHING BUSINESS INDUSTRY”**,** IEEE journal, 2021.

[3] Tewodros Legesse Munea, Yalew Zelalem Jembre, Halefom Tekle Weldegebriel , Longbiao Chen , Chenxi Huang , And Chenhui Yang, “A SURVEY AND TAXONOMY OF MODELS APPLIED IN 2D HUMAN POSE ESTIMATION”, O’reilly media publisher, 2021.

[4] Xintong Han, Zuxuan Wu, Zhe Wu, Ruichi Yu, Larry S. “VITON: AN IMAGE-BASED VIRTUAL TRY-ON NETWORK”, IEEE journal, 2021.

[5] I Dabolina , L Silina1 and P Apse - Apsitis “EVALUATION OF CLOTHING FIT”, Galgotia Publications 2021.

[6] Han Yang, Ruimao Zhang Xiaobao Guo Wei Liu Wangmeng Zuo Ping, “TOWARDS PHOTO-REALISTIC VIRTUAL TRY-ON BY ADAPTIVELY GENERATING ↔ PRESERVING IMAGE CONTENT”, John wiley & Sons Publishers, 2020.

[7] Bin Ren , Hao Tang , Fanyang Meng , Runwei Ding , Ling Shao , Philip H.S. Torr , Nicu Sebe, “CLOTH INTERACTIVE TRANSFORMER FOR VIRTUAL TRY-ON”, O’reilly media publisher, 2020.

[8] Amir Hossein Raffiee, Michael Sollami, Salesforce Einstein, “GARMENTGAN: PHOTO-REALISTIC ADVERSARIAL FASHION TRANSFER”, IEEE journal, 2020.

[9] Xintong Han Xiaojun Hu Weilin Huang Matthew R. Scott Malong Technologies, Shenzhen, China Shenzhen Malong, “CLOTHFLOW: A FLOW-BASED MODEL FOR CLOTHED PERSON GENERATION”, IEEE journal, 2020.

[10] Chia-Wei Hsieh, Chieh-Yun Chen, Chien-Lung Chou, Hong-Han Shuai, Wen-Huang Cheng , “FIT-ME: IMAGE-BASED VIRTUAL TRY-ON WITH ARBITRARY POSES”, IEEE Journal, 2020**.**

[11] G. Balakrishnan, A. Zhao, A. V. Dalca, F. Durand, and J. Guttag. “SYNTHESIZING IMAGES OF HUMANS IN UNSEEN POSES”, IEEE Journal, 2019.

[12] T. Alashkar, S. Jiang, S., and Y. Fu. “EXAMPLES-RULES GUIDED DEEP NEURAL NETWORK FOR MAKEUP RECOMMENDATION.”, O’reilly media publisher, 2019.

[13] Z. Al-Halah, R. Stiefelhagen, and K. Grauman. “FASHION FORWARD: FORECASTING VISUAL STYLE IN FASHION.”, O’reilly media publisher, 2019.