Virtual Try-On of Products

Project Report

Project X Mentorship Programme

at

Community of Coders, Veermata Jijabai Technological Institute

October 2024

Acknowledgement

We would like to extend a heartfelt gratitude to our project mentor, **Mrudul Pawar**, for being an extraordinary mentor and showing us the way throughout the course of this project. His timely guidance, insights and expertise helped us in our research and completing this project.

We would also like to thank ProjectX for giving us such a wonderful opportunity to learn and provide such quality mentorship.

Afreen Kazi

Aikazi_b23@ce.vjti.ac.in

Ishaan Shaikh

iishaikh_b23@ce.vjti.ac.in

Mahi Palimkar

mspalimkar_b23@ce.vjti.ac.in

Contents:

Overview

Virtual Try-On is a project that aims on clothes try-on for people. It has a huge scope in the fashion industry to provide a seamless try-on experience to users while shopping online.

Our project has 2 stepping stones. One is the 2-d virtual try-on, where we implemented try-on on 2d images. We tried various approaches for the same but an end-to-end approach using a deep U-net worked the best.

The next and the bigger stepping stone was 3-d try-on. We explored a lot of AR softwares like Unity, Blender and Maya. Finally, we researched and approach called DigiHuman that simulated the movement of an avatar according to the movement of the person in a certain video we upload.

We used that repo, cut out only the mesh section of the moving avatar and superimposed it on the video to give an effect that the person in the video is trying on the mesh.

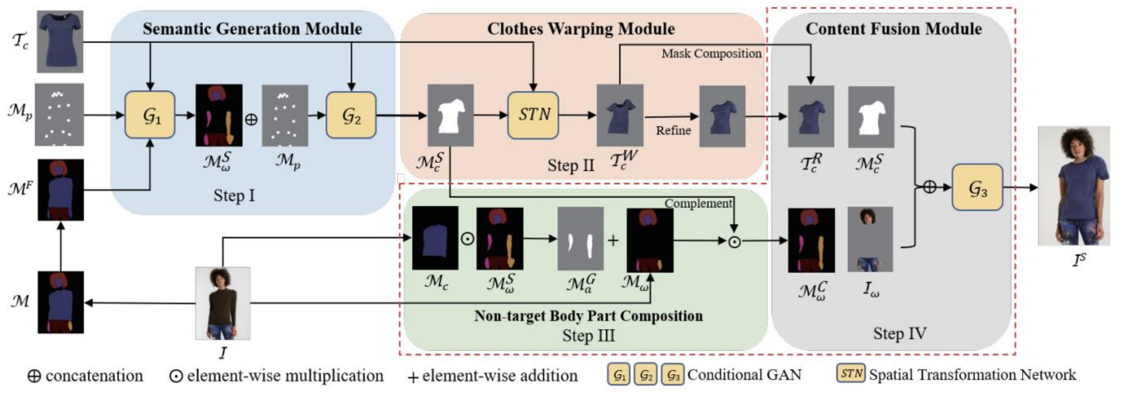
2D Tryon using ACGPN Approach

ACGPN (Adaptive Content Generating and Preserving Network) was our very first research to implement 2-D try-on.

ACGPN is based on a research paper Towards Virtual Try-on by Adaptively Generating and/or Preserving Image Content. ACGPN first predicts semantic layout of the reference image that will be changed after try-on and then determines whether its image content needs to be generated or preserved according to the predicted semantic layout, leading to photo-realistic try-on and rich clothing details.

ACGPN consists of 3 modules

- a Semantic Generation segmentation module
- a cloth warping module
- an image inpainting module.



1. Semantic Generation module

The **Semantic Generation Module** is responsible for generating a parsed semantic layout of the person and the target clothing. This module helps in understanding the structure of the human body and the location where the clothing will be placed. The steps involved are:

Inputs:

- **Target Clothing (T_c)**: The clothing image that the user wants to virtually try on.

- **Human Pose Map (M_p)**: A map that contains the human body parts (like arms, legs, torso, etc.), showing their relative positions.
- **Mask of the person wearing the original clothing (M_f):**

  M_f is a crucial input in the **Semantic Generation Module** because it provides information about the original clothing and body structure. This helps the model understand what parts of the body are currently covered by clothing and which parts are visible (such as the face, hands, etc.).

Output:

The module has two primary outputs:

1. **M^s_w (Intermediate Semantic Map):**
   - This is an initial semantic segmentation map generated by the first generator G1. It contains predictions of body part locations, including occlusions (i.e., parts of the body covered by the clothing). It includes the regions where the target clothing will be applied.
   - **M^s_w** is a rough, intermediate map that helps identify the basic structure of the body and gives an early idea of how the clothing will overlay on the person.
2. **M^s_c (Final Semantic Map):**
   - After further refinement by the second generator G_2, the final semantic map M^s_c is produced. This map contains the segmented body parts (like torso, arms, neck, etc.) as well as the target clothing area where the new clothing is intended to be placed.
   - **M^s_c** is a more detailed and accurate version of the intermediate map, with precise boundaries between different body parts and the clothing. This map guides the warping process in the next module by indicating exactly where and how the target clothing should fit.

Process:

1. **Generator G1**:
   - Takes the human image, clothing, and pose map as inputs.

   - Outputs an initial estimate of the human body with some parts like the face and hands occluded by the clothing.

2. **Refinement with Generator G2**:
   - Generator G2 refines the initial estimate by adding more precise details about the body's segmentation and filling gaps that might occur when clothing occludes body parts.

3. **Element-Wise Operations**:
   - The outputs are combined through element-wise multiplication and concatenation operations to form the **final segmentation map (M^s_c)**, which accurately maps the person's body parts and how the clothing fits on them.

In the **ACGPN (Adaptive Content Generating and Preserving Network)** architecture, the **generator** is a neural network model that generates new content or transforms images based on certain inputs. Specifically, the generators in ACGPN are **deep convolutional networks** used to produce intermediate representations of the human body, clothing, and final try-on images. These generators play a critical role in various stages of the virtual try-on process.

Generators in the ACGPN Architecture:

1. **G_1 and G_2 in the Semantic Generation Module (Step I)**:
   - These generators are responsible for generating a **semantic map** of the human body wearing the target clothing.
   - **G1** takes as input the human pose map M_P, the original clothing mask M_F, and the clothing image T_c. It generates an initial semantic map M^s_W that represents the segmentation of various body parts (e.g., arms, torso, etc.).
   - **G2** takes the semantic map generated by **G1** and refines it by incorporating more detailed body part segmentation and ensuring the clothing layout is accurate, resulting in the final semantic map M^s_c. The output of **G2** guides the warping of the target clothing in the next step.

2. Cloth Warping Module

The **Clothes Warping Module** performs the task of adjusting the shape and size of the clothing to fit the pose and structure of the person in the input image. This module ensures that the clothing is correctly warped to match the body's contours, making the virtual try-on look natural.

Inputs:

- **Segmented Clothing Mask (M^s_c)**: A segmentation map generated from the previous module, which defines the boundaries of the clothing.
- **Target Clothing (T_c)**: The clothing image that is to be warped onto the person.
- **Target Key points:** These key points in the source image are the points to which the source points have to be warped.

Output:

- **Warped Clothing (T^w_c)**: The clothing image adjusted to fit the shape and pose of the person in the human image.

Process:

1. **Spatial Transformation Network (STN)**:
   - The STN takes the target clothing image and the mask that defines the clothing region. The clothing is warped using the body part information to ensure it fits the given human pose.
   - Thin Plate Spline (TPS) based STN is used for warping the source points in the target cloth image to the target points in the segmented cloth image.
2. **Mask Composition and Refinement**:
   - After warping, the clothing is combined with the body segmentation mask.
   - A second order difference constraint is introduced in the TPS warping to ensure that the geometrical features and patterns of the cloth do not give in to non-affine warping and get distorted.

- A refinement step ensures that the edges of the warped clothing align perfectly with the body, improving the realism of the result.

How TPS based STN works?

Thin Plate Spline (TPS) transformation integrates seamlessly with a Spatial Transformer Network (STN) to enable flexible and non-linear warping.

1. Spatial Transformation Network (STN):

STN consists of 3 main parts:

- Localization Network

- Grid generator

- Sampler

a. Localization Network: The Localization Network is a small CNN that learns to predict parameters that are required for the transformation. These parameters are essentially the Control points in the target cloth image that are to be warped to the target points in the segmented cloth image.

b. Grid Generator: The grid generator creates a regular grid over the target space (usually the target clothing image or the person's pose keypoints). This grid represents the points where the source image should be warped. This is followed by the TPS transformation.

c. Sampler: The sampler is the final step which handles where the pixels of the source image are moved to their new locations (calculated by the grid generator and TPS warping) .

2. Thin Plate Spline:

Thin Plate Spline (TPS) is a smooth, non-rigid transformation technique used to warp one set of points to another, while minimizing distortions. It's often used in computer graphics and machine learning for tasks like image warping and alignment. The key idea behind TPS is to find a smooth mapping between two sets of control points, while minimizing bending energy.

The process flow of TPS is as follows:

- TPS requires two sets of corresponding control points:
  - Source Points: Points from the original image (target cloth image) that you want to warp.
  - Target Points: Points that define the desired positions of the source points after the transformation. (points in the segmented cloth)
- The TPS transformation is defined by a smooth function that maps every source point to its corresponding target point. It consists of two parts:
  - Affine Transformation: A global linear transformation (rotation, scaling, translation).
  - Non-affine Warping: A non-linear part that introduces smooth deformations based on the positions of the control points.

- The function f(x) for each output coordinate is defined as:

  $$f(x)=A\cdot x+W\cdot U(\|x-xi\|)$$

  - $A\cdot xA$ is the affine part.
  - $W\cdot U(\|x-xi\|)$ is the non-affine warping, where $U(\|x-xi\|)$ is a radial basis function based on the distance between the control points.

- TPS uses a radial basis function $U(r)=r^2\log(r)$ to describe how much influence each control point has on the surrounding points. This ensures that the transformation is smooth, and the influence of a control point decreases with distance.

- Once the transformation function is computed using the control points, TPS can be applied to warp the entire image grid. Each pixel in the source image is moved according to the TPS function, smoothly warping the image to align with the target.

3. **Second Order Difference Constraint:**
   The second-order difference constraint ensures that the transformation between neighbouring points remains smooth. It penalizes large differences in the rate of change between adjacent

control points, effectively ensuring that the transformed surface (or image) doesn't bend or distort too sharply. This smoothness is particularly important when transforming objects with complex textures, as it helps maintain the geometric consistency of the original object.
It ensures that the warping happens in a visually natural manner, especially when dealing with complex textures or fabrics with intricate patterns and by maintaining geometric consistency, the original shape and character of the object being warped are preserved.

3. Content Fusion Module

The **Content Fusion Module** is responsible for merging the warped clothing with the body in a realistic manner. It addresses issues such as occlusions, where body parts might block some parts of the clothing, and ensures a smooth blending of the clothing with the body.

Inputs:

- **Warped Clothing (T^w_c)**: The warped clothing from the Clothes Warping Module.
- **Masked Body Parts (M^c_w)**: A body part mask indicating which parts of the body are occluded by clothing and which are visible.
- **Original Human Image (I_w)**: The original image of the person, which includes the face, arms, and other body parts.

Outputs:

- **Final Composite Image (I^s)**: The final result of the virtual try-on, where the new clothing has been realistically applied to the person's body.

Process:

1. **Non-target Body Part Composition**:
    - The non-target body parts (those not covered by the clothing, such as hands, face, and neck) are processed separately.
    - The module makes sure these parts are accurately visible and complements the warped clothing.
2. **Complementing the Occluded Parts**:

    - Areas of the body that were previously occluded by the original clothing (like the torso or arms) are completed by this module to avoid artifacts.
    - Missing body parts are reconstructed using information from the original human image.
3. **Final Fusion**:
    - Generator G3 performs the final fusion of the warped clothing, non-target body parts, and occlusion handling results.
    - The blending process ensures that the clothing fits seamlessly with the body, handling any color or texture mismatches between the body and the clothing.

Challenges faced in Building ACGPN from Scratch

1. While building the Cloth Warping Module, the localization network in STN could not be trained to properly predict the control points as there was no dataset available to train it. Since, the task of the Localization Network was to predict the control points on the target cloth so as to warp it to the source points in the segmented cloth map, pose estimation could not be used as we had to work on a cloth image. As a result, we had to train an unsupervised model, which would require an excessively large number of epochs to learn to predict the correct transformation points properly. Lack of resources which included dataset and GPUs resulted in a failed implementation.
2. There was no data related to warped clothing images, so Content Fusion Module couldn’t go ahead independent of the warping module. If the data was available both the issues could have been solved.

2D Tryon by End-to-End Learning

An end-to-end learning model was developed using a U-Net CNN architecture to facilitate the virtual try-on of clothes.

Input: The model takes as input the segmentation masks of the person, clothing agnostic image of the person (Image with only facial and body details and no cloth details), the target cloth, and its corresponding mask, effectively isolating the relevant features necessary for a realistic overlay.

Output: The performance of the model was evaluated using the Peak Signal-to-Noise Ratio (PSNR), a common metric for assessing the quality of reconstructed images. The achieved PSNR value of approximately 20 indicates a reasonable level of fidelity in the generated output, suggesting that the model successfully retained significant visual details of both the person and the clothing.

Process:

A U-Net Convolutional Neural Network (CNN) architecture was used to implement an end-to-end learning model capable of "trying on" clothes directly onto a person by referring to the **input segmentation mask** of the person, the clothing agnostic image of the person, **target clothing image**, and **cloth mask**. The U-Net structure allows the model to focus on the fine details of both the person and the clothing while learning how to realistically superimpose the clothing over the person’s body.

U-Net Structure: The U-Net architecture consists of two main components:

- **Encoder (Contracting Path)**: This part of the network downsamples the input through multiple convolution and pooling layers, allowing it to capture the most important features and spatial relationships of the person and the clothing.
  - Each level in the encoder progressively reduces the spatial dimensions while increasing the feature depth, capturing the hierarchical structure of the person and the target clothing.
- **Decoder (Expanding Path)**: The decoder upsamples the feature maps back to the original resolution through transposed convolutions.

Importantly, **skip connections** between corresponding layers in the encoder and decoder ensure that the network retains fine-grained details necessary for the task. This enables the U-Net to reconstruct the output image of the person with the tried-on clothing in high resolution.

PSNR as an evaluation Metric:

- PSNR stands for Peak Signal To Noise Ratio which is a metric used to measure the quality of a reconstructed or compressed image compared to the original image. Its goal is to minimize distortion and preserve image quality.
- PSNR is defined as the ratio between the maximum possible power of a signal (in this case, pixel values) and the power of noise that affects the quality of the image. The "noise" in this context is the difference between the original image and the generated image.
- The higher the PSNR value, the closer the generated image is to the original, meaning better visual quality.
- PSNR is typically expressed in decibels (dB). It is calculated based on the Mean Squared Error (MSE) between the original image and the generated image. The formula for PSNR is:

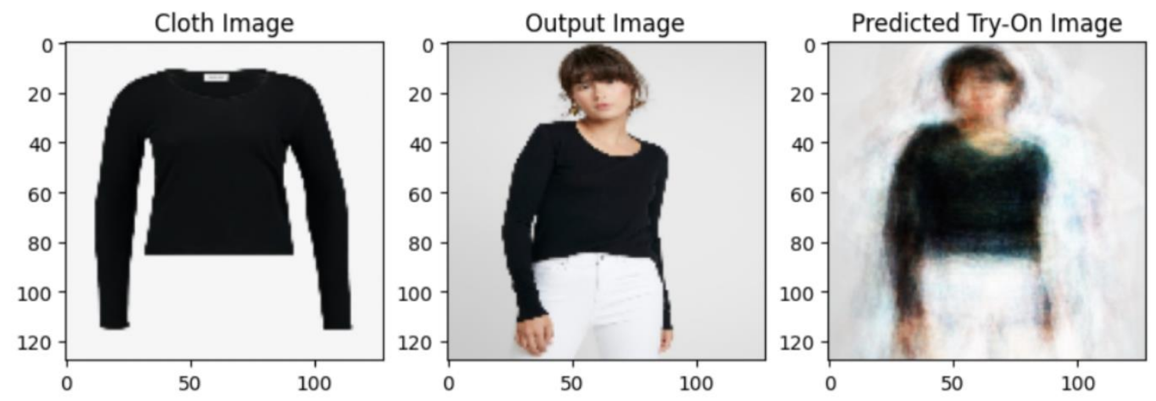
$$PSNR = 10 \cdot \log_{10}\left(\frac{MAX^2}{MSE}\right)$$

  - MAX is the maximum possible pixel value of the image (255 for an 8-bit image).
  - MSE (Mean Squared Error) is the average squared difference between corresponding pixels in the two images

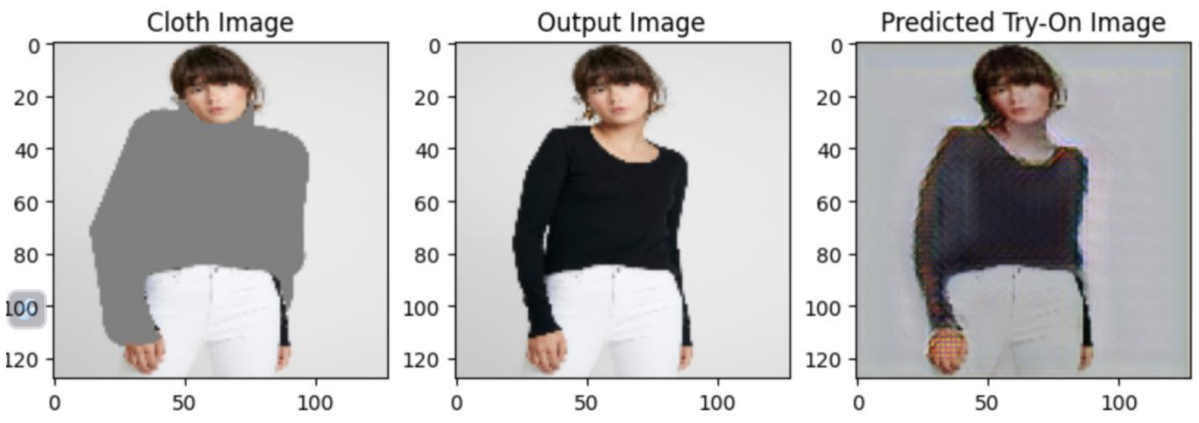
End-to-end approach output

1. Using CNNs: We first tried using basic CNN approach, using the same inputs and trained the model to learn the features directly from the input image and also supplying the output image. This gave a low PSNR of about 14

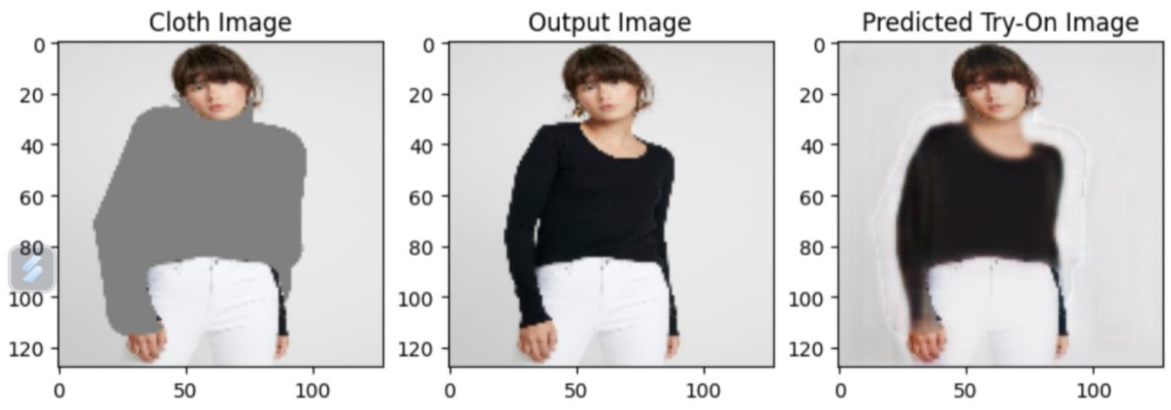
```
64/64 ━━━━━━━━━━━━━━━━━━━━ 26s 410ms/step
```



2. So we improvised it by changing the CNN architecture to UNET. This significantly improved the quality of the output as the UNET structure implements skip connections to ensure preservation of high-end features like facial details and the cloth details. This gave a PSNR of about 20



3. To further improvise the quality, we implemented a deeper UNet architecture that involved increasing the number of layers in the encoder and the decoder. It also needed more skip connections and at the end we had the result having a PSNR of about 22 which is pretty decent as seen in the image below.



2D Tryon Approach by Generating Agnostic Image of cloth

We tried to generate an agnostic warped clothing mask using the output of the above models using GANS and superimpose it on the person's body for better quality. The GANs implement a Generator Discriminator Architecture which are basically Two CNNs competing against each other to produce better- and better-quality images.

As GANs, were computationally expensive and we didn't have enough hardware resources we could not complete it.

3D Virtual Try-On

After getting satisfactory results in 2D Virtual Try-On we moved ahead with 3D Virtual Try-On.

Our initial approach towards 3D try-on was:

- 3D pose estimation of a person in real time, using camera feed.
- Acquiring/Building a cloth mesh
- Skinning (skin the mesh with the humanoid rig) and rigging (movement of the mesh occurs in accordance to the movement of the rig).
- Simulation of the mesh corresponding to the body keypoints obtained from the 3d pose estimation.
- Superimposition of the simulated mesh on the person in real time.

1. 3D pose estimation

We used Mediapipe Pose Solution and OpenCV for 3d pose estimation.

1. Pose Estimation with Mediapipe

- **Mediapipe Pose** is a framework that provides a pre-trained pose estimation model for detecting key body landmarks. The model is capable of detecting up to 33 key points (landmarks) on the human body in both 2D (pixel coordinates) and 3D (real-world coordinates).
- The code utilizes Mediapipe's Pose module for detecting these key landmarks from a live video feed (e.g., a webcam).
- **3D landmarks** include the (x, y, z) coordinates of each body landmark, where x and y are normalized coordinates relative to the image dimensions, and z represents the depth information (in meters, estimated based on the camera position).

2. How 3D Pose Estimation Works

- **Input**: The input to the system is a video feed, which can come from a live webcam (cap = cv2.VideoCapture(0)) or a prerecorded video file (by changing the 0 to a file path).
- **Preprocessing**: The captured video frames are converted from BGR (OpenCV's default color format) to RGB (frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)) as Mediapipe Pose expects the input to be in RGB format.
- **Pose Detection**: The pose_detector.process(frame_rgb) function processes the RGB frame and outputs the detected pose landmarks.
    - **2D Landmarks**: These are the positions of the detected body keypoints (such as the shoulders, elbows, hips, knees, etc.) in pixel coordinates.
    - **3D Landmarks**: The model also estimates real-world coordinates (x, y, z) for these keypoints. The z coordinate indicates the depth, which helps to infer the body's 3D structure.

3. Extracting and Displaying 3D Landmarks

- The **read_landmark_positions_3d()** function extracts the 3D coordinates (x, y, z) of each detected landmark from results.pose_world_landmarks. The landmarks are stored in a NumPy array for easy manipulation and printing.
    - results.pose_world_landmarks.landmark[lm]: Accesses the 3D position of a particular landmark (like the nose, shoulder, etc.) in real-world coordinates.
    - The landmarks are stored as a list of tuples containing (x, y, z) values. The x and y values are normalized between [0, 1], and the z-value represents depth in meters relative to the camera.
- The code prints the 3D coordinates of these landmarks in the console (print(f'3D Landmarks: {landmark_positions_3d}')), allowing the user to see real-time changes as the person in front of the camera moves.

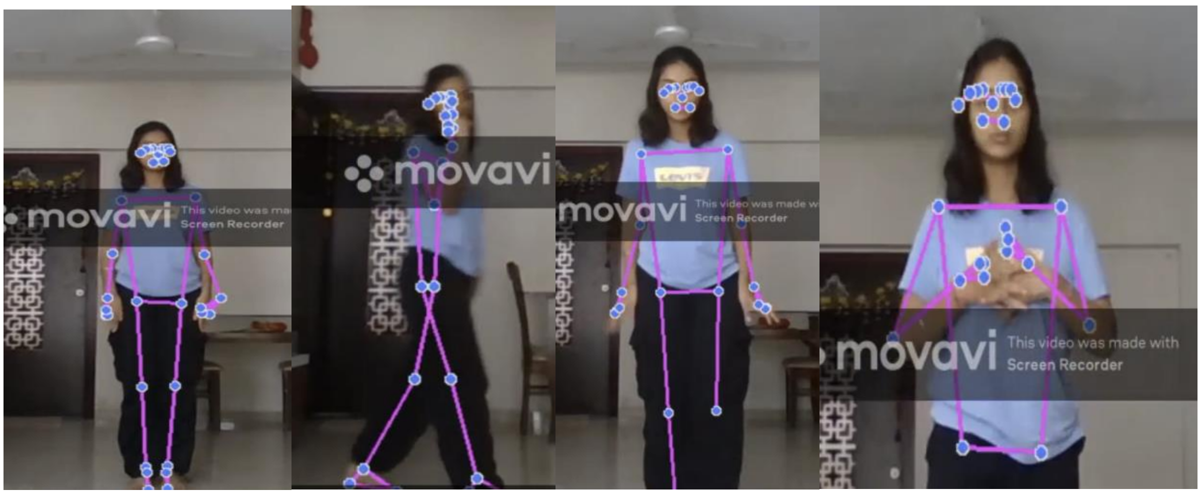
4. Drawing 2D Landmarks on the Image

- The **draw_landmarks_on_image()** function visualizes the detected landmarks on the image. It uses the Mediapipe drawing utilities (mp.solutions.drawing_utils) to draw connections between the detected landmarks and to overlay the pose on the video frame.
- A color scheme is applied to differentiate between the landmarks and their connections:
  - The landmarks are drawn with a circle of a specified radius.
  - The connections between landmarks (like between shoulder and elbow) are drawn as lines, creating a visual skeleton overlay.

5. Real-Time Video Display

- The processed video frames with pose annotations are displayed in real-time using OpenCV's imshow() function (cv2.imshow('Real-Time 3D Pose Estimation', frame)).
- The system captures video frames in a loop until the user presses the 'q' key, at which point the program terminates.

OUTPUT of 3d pose estimation



2. Mesh Rigging

To obtain a cloth mesh, you can:

1. **Download from 3D Marketplaces**: Use platforms like **CGTrader**, **TurboSquid**, or **Sketchfab** to get pre-made 3D cloth models in formats like .obj or .fbx.
2. **Create in 3D Software**:
   - Use **Blender**: Model and simulate cloth behavior using cloth physics tools.
   - Use **Marvelous Designer/CLO 3D**: Design garments with realistic fabric simulations.

We tried both approaches. We created a mesh in blender, but realized that designing a mesh from scratch would not result in a perfect mesh.

Then we opted for readymade meshes from CG trader.

1. Rigging

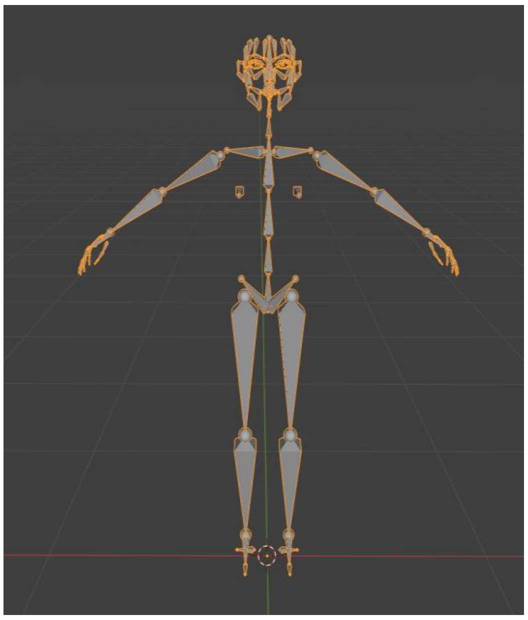
- **Rigging** is the process of creating a **skeleton** for a 3D model. This skeleton is made up of **bones** or **joints** that control the movement of different parts of the model (like arms, legs, or facial features).
- Once rigged, the character or object can be manipulated and posed by moving the bones, making it easier to animate.
- The rig acts like an invisible framework that defines how different parts of the model should move.

2. Skinning

- **Skinning** is the process of attaching the 3D model (or **skin**) to the **rig** or skeleton. It defines how the surface of the 3D model deforms when the bones move.
- Skinning ensures that when a bone or joint is moved, the corresponding part of the mesh moves along smoothly. This is done by assigning **weights** to the vertices of the mesh, determining how much influence each bone has on each vertex.

We tried doing both of them in blender as well as Maya using multiple meshes. (Both Blender and Maya are 3d creation tools used for animations, modelling, rendering etc.)

Firstly this is what a humanoid rig looks like in blender.



After skinning and rigging these were the outputs that we obtained.

For rigging of the model we also had to do weight painting.

Weight painting in Maya is the process of assigning how much influence each joint or bone in a skeleton has over different parts of a 3D model's surface. This helps control how the mesh deforms when the skeleton moves.

Analogy:

Imagine you're holding a puppet with multiple strings (the **bones**). Each string pulls on a section of the puppet's body (**mesh**). Weight painting is like adjusting the tension on each string: some strings pull more tightly on certain parts, while others have less effect. Without weight painting, the puppet's movements would look unnatural or messy because the pulling forces would not be evenly distributed. Weight painting ensures smooth and realistic deformations.

3. Simulation of the mesh in correspondence to real time body key points

This is where Unity comes into picture. Unity is a very Popular AR Engine which we used for real time simulation of the mesh in correspondence to the key points obtained during 3d Pose estimation.

Unity provides a comprehensive suite of tools and features specifically designed for **3D development**, making it one of the most versatile engines for creating 3D applications, games, simulations, and AR/VR experiences.

These are some functionalities specific to Virtual Try on that Unity offers.

1. Initialize AR Session

- **Function**: Start and configure the AR session when the application launches.
- **Usage**: Set up AR tracking, enabling the camera, and initializing necessary AR components.

2. Load 3D Models

- **Function**: Load and display 3D models of clothing and accessories from assets.
- **Usage**: Instantiate clothing items when a user selects them from the UI.

3. Attach Clothing to Avatar

- **Function**: Attach selected clothing items to the user's avatar or a predefined character model.
- **Usage**: Adjust the position and scale of clothing to fit the avatar correctly.

4. Integrate Mediapipe with Unity

UDP sockets were used to send packets of pose data detected by Mediapipe directly to unity.

5. **Update mapped keypoints on Skeleton according to updated Mediapose Keypoints**
   As the keypoints update , the transform points mapped to the bones on Unity3dD also update, thereby updating the bones and finally the mesh rigged to the bones updates.

6. . Change Clothing Item

- **Function**: Allow users to switch between different clothing items seamlessly.
- **Usage**: Remove the currently displayed clothing and attach the newly selected item.

7. Rotate and Scale Clothing

- **Function**: Enable users to rotate and scale clothing items for a better fit.
- **Usage**: Implement touch or gesture controls to manipulate clothing items.

8. Real-time Rendering

- **Function**: Render the avatar and clothing in real time as the user moves.
- **Usage**: Ensure that the clothing dynamically interacts with the avatar's movements.

What is DigiHuman?

Digihuman is an approach that we found while we were searching for other methods to complete our Virtual Try on.

What Digihuman basically offers:

- There are certain pre-created avatars which are fully rigged.
- We can upload any video of a person doing any sort of movement, whether it is purely facial, or purely body movement, or both, and the avatar that we choose will move accordingly.

Our approach:

We tried uploading multiple videos to check that the movement of the avatar was best in which.

Once we found the best one, we extracted only the cloth mesh of the avatar.

Now since the cloth mesh was already simulated according to the movement of the person in the video, we superimposed that mesh on the person to give an effect of virtual try on.

OUTPUT:

Here's the link to the output Video:
Virtual-Try-On-Products-/assets/output.gif at main · Mr-MVP/Virtual-Try-On-Products-

Future Prospects:

3D virtual try on is a very novel research project. One of the major issue we faced was that until now, no one has done extensive research on it. This is the reason why we had to try out so many approaches and methods to find out what works best.

One of the most important future prospects would be integrating real time cloth simulation in our project. We could not integrate the real time aspect now due to time and resource constraints, but we will do that in future.

Another future aspect is including multiple meshes for users to try on. We could not do this because the repo that we used finally, DigiHuman, had a limited number of avatars whose cloth meshes we could extract. If we integrate multiple meshes it will give a very realistic vton experience.

Deploy it on Phone- We can also make this work on mobile phones to give a seamless try on experience as offered by giants like Lens Studio.

Resources:

1. ACGPN Repo

2. ACGPN Paper
3. Building a CNN
4. U-net approach
5. 3d pose estimation
6. DigiHuman