Chapter 1

Introduction

The garment manufacturing industry have to assure the quality of the produced apparel to their customers. In particular, for jeans manufacturing, the inspection of various measurements such as fit, sewing lines, and pockets is an important part of the quality assurance process. These measurements are required to be within the tolerances specified by the customer.

1.1 Description of the Required Measurements

Although the exact description of the required measurements varies between companies and brands, there is a lot of overlap. The two big jeans brands: Levi's and Jack & Jones both have defined the following measurements:

Note: The following terms are interchangeable: Coin Pocket / Watch Pocket, Riser / Yoke, and Bottom / Hem.

1. Waist: Measure from side to side at the middle of the waist band then double
2. Inseam: Measured as the length of the natural curve of the inseam till the crotch point
3. Outseam: Measured as the full length of the jeans along the side seam
4. Seat: Measured 4 inches above the crotch point as the horizontal distance
5. Thigh: Measured 2 inches below the crotch point for one leg
6. Knee: Measured 2 inches above the midpoint of the inseam for one leg
7. Front rise: Measured upwards from the crotch point and including the waist band
8. Bottom: Measured across the hem of one leg
9. Fly opening: Measured as the length of the zip fly
10. J-stitch width: Measured as the horizontal distance between zip flap and J-stitch
11. Belt loop length: Measured as the length of one belt loop element
12. Front pocket width: Measured as the width of the front pocket along the waist band
13. Front pocket depth: Measured as the depth of the front pocket along the side seam
14. Coin pocket width at top: Measured as the width of the coin pocket at its top edge
15. Coin pocket width at bottom: Measured as the width of the coin pocket at its bottom edge
16. Coin pocket length at center: Measured as the vertical length of the coin pocket at its center between the bottom corner and midpoint of the top edge
17. Coin pocket length at side: Measured as the vertical length of the coin pocket at one of its side between the upper-right corner and bottom-right corner

18. Coin pocket position under waist band: Measured as the distance from the top edge of the coin pocket till the waist band
19. Coin pocket position from side seam: Measured as the distance from edge closer to the side seam of the coin pocket till the side seam
20. Riser height at center back: Measured as the distance between riser seam and bottom edge of the waist band at center back position
21. Riser height at side seam: Measured as the distance between riser seam and waist band at side seam
22. Back pocket width at top: Measured as the width of the back pocket at its top edge
23. Back pocket width at bottom: Measured as the back pocket's width at its bottom edge
24. Back pocket length at center: Measured as the length of the back pocket at its center
25. Back pocket length at side: Measured as the length of the back pocket at its side
26. Back pocket position from riser seam at center back: Vertical distance till riser seam at center back
27. Back pocket position from riser seam at side seam: Vertical distance till riser seam at side seam
28. Back pocket position from side seam: Measured as the horizontal distance till the side seam

1.2 Diagrams for the Required Measurements

The measurements 1-8 are described visually in Figure 1, measurements 9-19 are described visually in Figure 2, and measurements 20-28 are described visually in Figure 3.

These diagrams are not exhaustive and do not include all the listed measurements, but instead serve as a rough guide to how the measurements should be taken. The measurements required for the back pocket are very similar to the those required for the coin pocket, so Figure 3 depends on Figure 2 for completeness.

It should be noted that the following measurements are not shown:

1. Coin pocket position under waist band
2. Coin pocket position from side seam
3. Back pocket position from riser seam at center back
4. Back pocket position from riser seam at side seam
5. Back pocket position from side seam

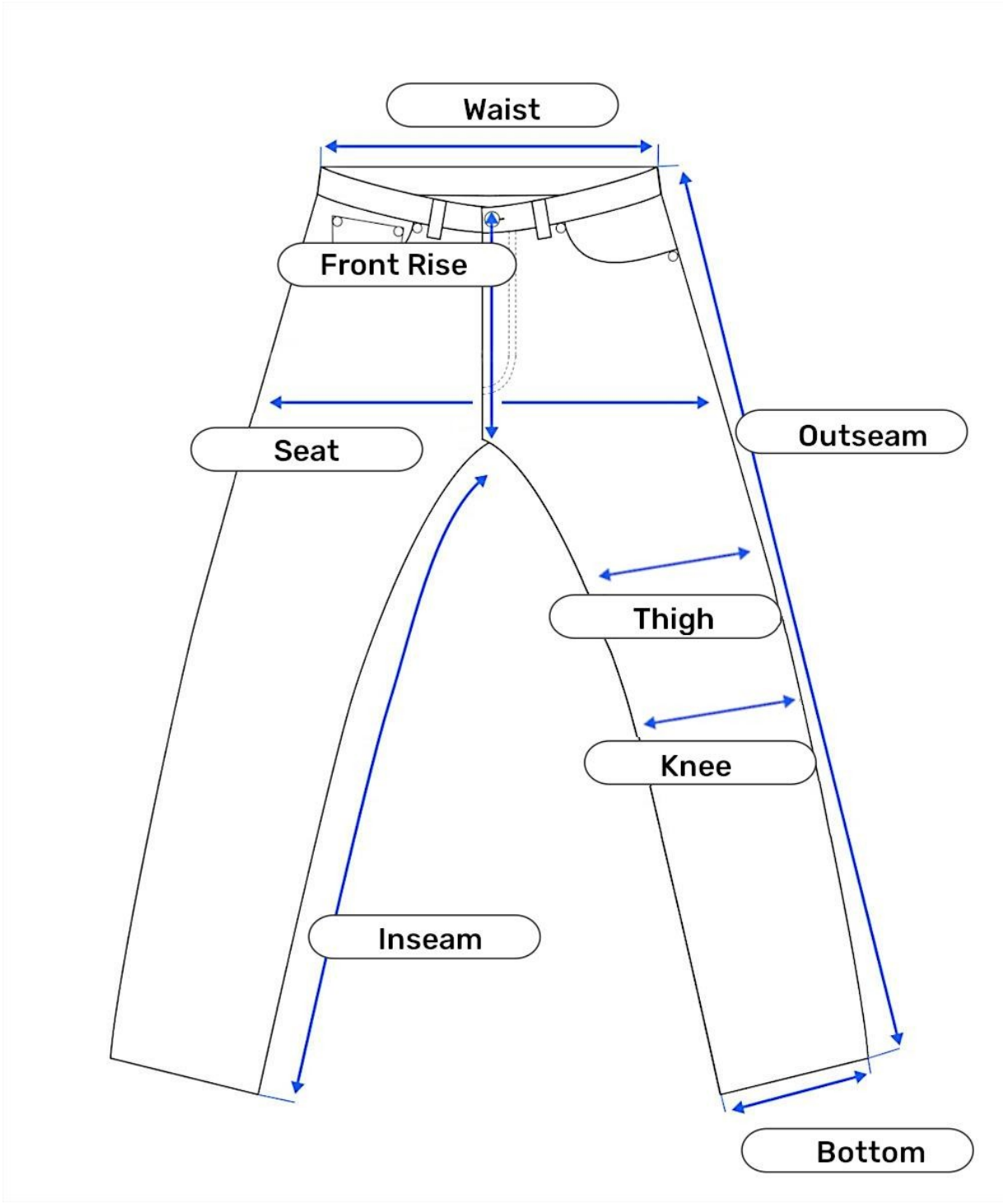


Figure 1 – Jeans Measurement Template [Front View]

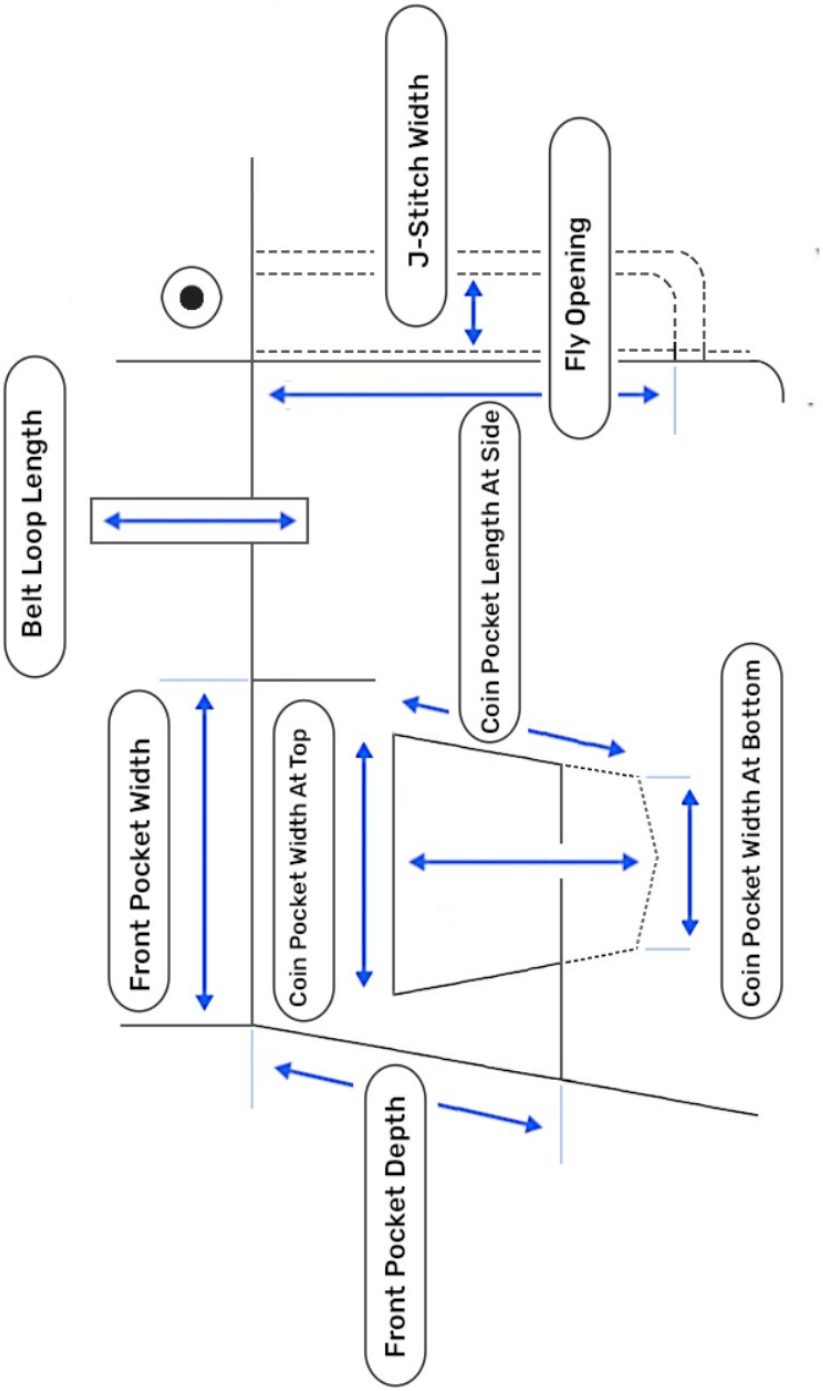


Figure 2 – Jeans Measurement Template [Front Pockets View]

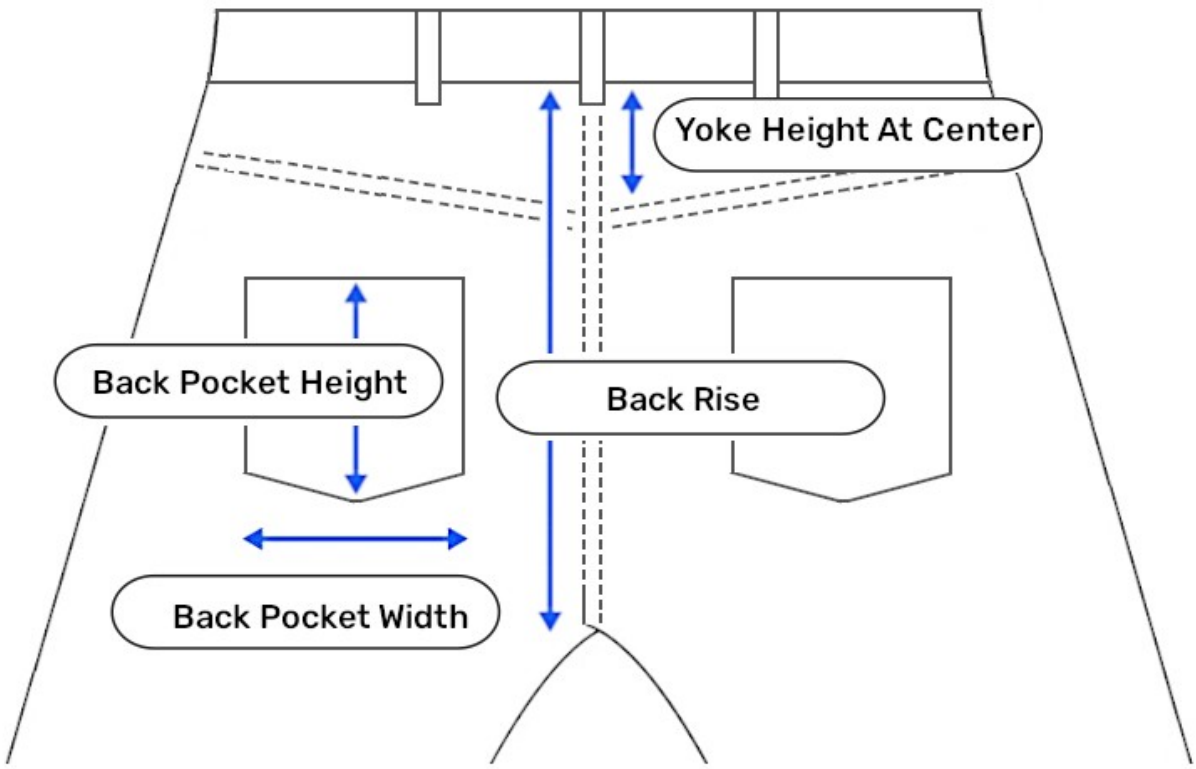


Figure 3 – Jeans Measurement Template [Back View]

1.3 Limitations of Manual Inspection

Usually, these measurements are taken a worker manually. The worker randomly selects a few (generally 5) jeans from each bundle and inspects them. For each pant, it takes around 3-5 minutes to take the necessary measurements.

This process is terribly inefficient since the worker spends unnecessarily long time to inspect a single pant. This also severely limits the number of jeans that can be inspected from each bundle, and limits the number of parameters that can be inspected for each jean – the worker usually skips the insignificant parameters such as the belt loop length, riser height, coin pocket width, and so on.

Besides, manual inspection introduces human error into the tolerances. The worker can accidentally take the wrong reading, or mistype the reading into the chart. But an even more convincing argument is that, this process is well-defined, redundant, and repeatable, so it can and should be automated.

1.4 Potential for Automating Jeans Measurement

Measurement of an object through computer vision and deep learning is not novel, and has been used extensively in the past for human pose estimation and biomedical image analysis. This problem is generally classified under keypoint detection or landmark localization task in the deep learning context.

In its simplest form, an image of the object is fed to a convolutional neural network that identifies the required keypoint position on the image. Then these keypoints are used to measure the required parameters.

For the jeans measurement task, these keypoints can refer to the vertices of the pockets, midpoints of the waist band edges, vertices of the riser seam, points to approximate the curve of the inseam, points on the stitch lines, and so on.

A hardware setup can be arranged consisting of a camera and lightening system that takes images of the jeans from both front and back. These images can be fed to a computer system that predicts the desired keypoints using a deep learning model, and then these keypoints are used to calculate the required measurements. Finally, these measurements can be presented to the worker through an intuitive graphical user interface or fed to an upstream server which records these measurements.

1.5 Organization of this Report

In this thesis, we first state the problem of automatic jeans measurement and conclude that it can be solved using techniques from computer vision and deep learning. We then review existing literature to identify techniques used for the keypoint detection task. We also review existing techniques for other problems encountered in this project such as camera calibration.

Then we describe the complete methodology we used in sufficient detail that it can be easily replicated and extended. The methodology chapter covers the following components:

1. Hardware setup
2. Image preprocessing
3. Neural network
4. Camera calibration
5. Graphical user interface

For each section, we have tried to enclose all the necessary details relevant and required within that section. For instance, someone implementing the neural network part can review all the required information solely from that section.

Then we present the results achieved from our work, and conclude with reasons for why the accuracy was lower than required. We then propose an improved method for training a much more reliable and accurate model.

Then, we go over the societal impact of our work, and list the environmental and sustainable development goals. We also present a business plan incorporating this project. Lastly, we describe our learning throughout this project and give future recommendations for enhancing our work.

Chapter 2

Literature Review and Problem Statement

The process of automatic garment measurement is mostly dependent on the successful detection of keypoints, which are points of interest on the jeans pant, such as the corners of pockets and sewing lines. Once these keypoints are detected, the required measurements are simply the distance between the two relevant keypoints, which can be trivially calculated. Therefore, we frame the automatic jeans measurement problem as a keypoint detection (or landmark localization) problem.

Because the position of the required keypoints cannot be found deterministically — and to complicate things further, the jeans image can contain occlusions, pose variations, or other deformations — keypoint estimation is usually accomplished using deep learning. Wherein, a neural network, usually a cascaded convolution neural network, such as HRNet (High Resolution Network), is trained on a dataset of labelled images. Each dataset image is labelled manually with the 24 keypoints necessary for the computation of the required measurements.

Keypoint detection aims to estimate the precise location of an object's keypoints, which are sometimes referred to as landmarks. Landmark localization has been extensively studied in the domains of face recognition [1], human pose estimation [2], and biomedical image analysis [3]. However, there is little work done on the landmark localization problem for garments. This is due to the unavailability of an appropriately labelled dataset for garments. The only large-scale dataset for the task of garment landmark detection is the DeepFashion2 dataset [4]. Although DeepFashion2 is a huge dataset of 491K images, it does not contain many images of jeans specifically and only labels 14 keypoints on each jeans image, which can only be used to determine some very basic measurements, and is therefore, insufficient for our task, which requires at least 40 accurately labelled keypoints.

In recent literature, fully convolutional neural networks (CNNs) have established themselves as the de facto method in landmark detection overtaking previous approaches such as random forests [5]. This began with Thomas et al. who used a CNN to regress target heatmaps achieving state-of the-art performance on the human pose estimation task. Shortly afterwards, fully convolutional neural networks such as U-Net became very popular for segmentation tasks and its encoder-decoder architecture began to be applied to landmark detection as well, such as in Payer et al. [6].

Therefore, the techniques for landmark detection are readily available, however, the availability of a suitable dataset is a challenge that needs to be solved for the development of our system. Creation of a dataset with millions of images is infeasible. To overcome this problem, we used a type of transfer learning called model fine-tuning [7].

Model fine-tuning is a method designed to overcome the problem of a small dataset. Instead of training entirely on just a single dataset, the model is trained progressively on multiple datasets. These datasets can be arbitrarily any dataset — even images of cats can be used. The model is able to learn important details about objects from these foundational datasets, instead of learning them from the main dataset. This allows the main dataset to be small, since the model learns general details about objects from these other datasets, and only uses the main dataset to learn the specific problem at hand.

For our task of garment landmark detection, we train the model on ImageNet [8] and DeepFashion2 prior to training it on our own dataset. Our own dataset contains extensively labelled images of jeans that we collected by visiting garment stores and manually annotating them using annotation tools such as Roboflow Annotate.

The state-of-the-art methods used to solve landmark detection problems on the DeepFashion2 dataset rely on heavy architectures such as DeepMark++ [9] (with a mean average precision, mAP, of 0.58). These heavy models require large amounts of GPU resources, extended training time, and cannot be considered to run on mobile devices. However, these state-of-the-art models are complicated because they have to run in the wild, which means, they are designed to handle different types of clothing items with large amounts of variations and deformations. This is not the case with our task.

James McCouat et al. [10] proposed an effective and easy-to-implement method for simultaneously performing landmark detection in images and obtaining an ingenious uncertainty measurement for each landmark. His methods have an impressive accuracy (mAP of 0.89) and have an open-source implementation. However, his technique is concerned with the task of cephalometric landmark detection from x-rays of the head. Therefore, we follow the methods described by James McCouat et al. in his paper: Contour Hugging Heatmaps for Landmark Detection, but implement them for the task of sizing landmark detection from images of jeans.

We have limited our problem to the accurate landmark estimation on only jeans images and under a controlled environment, where the jeans are folded in a well-defined pattern. This allows us to use simpler models such as HRNet [11] or U-Net [12], while still achieving better accuracy (mAP ≈ 0.83) than state-of-the-art models.

2.1 Problem Statement

We state the problem as the detection of the keypoints specified in Figure 4 through a convolutional neural network. The processes prior to the keypoint detection (image capture and preprocessing) and processes after the keypoint detection are trivial, so we focus mostly on the keypoint detection part.

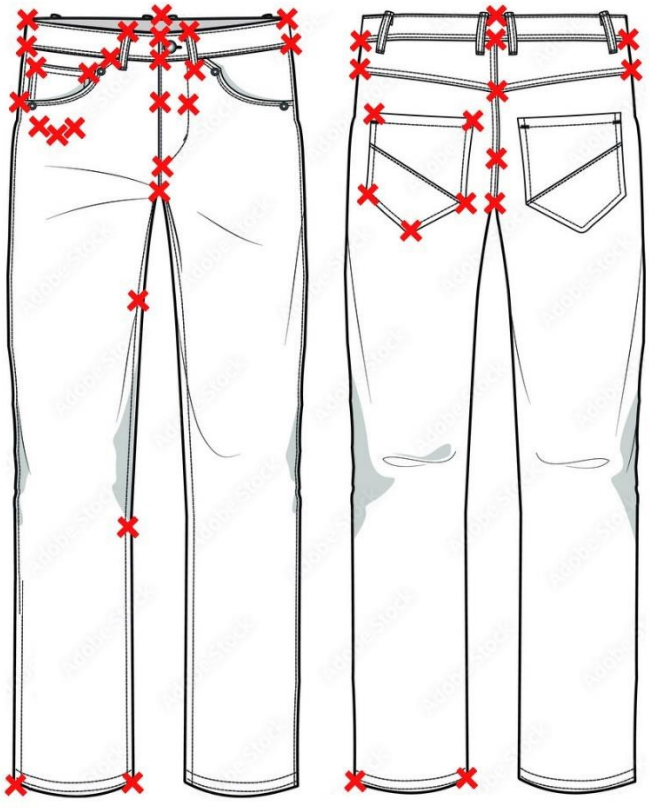


Figure 4 – Desired Keypoints

2.2 Techniques used for Automatic Garment Measurement

In the past, the most widely used technique for automatic garment measurement was to use a garment template to recognize the garment type and feature points, which are then used to calculate garment sizes, such as by Chunxiao Li et al. in [13].

Another technique was to use a fuzzy edge detection algorithm and then use a corner detection algorithm to locate the corner points. The corner points are then used to compute relevant measurements, such as by Brian Coffey et al. in [14].

However, both of these techniques are limited in the sense that although the authors have showed high accuracy (up to 1.0 cm error), the garments must be laid down in a very controlled setting. Slight changes in the position of the garment or color change in background can severely impact the system output.

These techniques are also limited in the sense that they only allow a small number of parameters to be measured. Only those measurements can be meaningfully computed that are defined at the outer boundary of the garment outline, such as waist and leg opening. Those are defined inside the boundary of the garment outline, such as pocket width, pocket position, zip fly opening, and belt loop length, cannot be measured.

In recent literature, the most widely used technique is to detect keypoints on the garment and then compute the desired measurements, such as in [15], [16], and [17]. If the garment is laid flat, the measurements can be computed as simple distances in 2D space. However, if the garment is not laid flat on a table, such as on a mannequin, Seounggeun Kim et al. presented a variation of keypoint detection that uses point cloud data from a LiDaR sensor to compute the 3D structure of the garment and then detect the keypoints on this 3D surface. However, since we laying the garments flat on a table, we do not need LiDaR sensors, and can restrict ourselves to the 2D plane.

2.3 Keypoint Detection

Keypoint detection, also known as keypoint localization or landmark detection, is a computer vision task that involves identifying and localizing specific points of interest in an image.

In computer vision tasks, keypoints represent human body joints, facial landmarks, or salient points on objects. Keypoint detection provides essential information about the location, pose, and structure of objects or entities within an image, playing a critical role in computer vision applications such as the following:

1. Pose estimation
2. Object detection and tracking
3. Facial analysis
4. Augmented reality

Keypoints are typically defined by certain characteristics that set them apart from the surrounding pixels. These characteristics include:

1. **Uniqueness**: Keypoints should be unique and easily distinguishable from other points in the image. They stand out due to specific visual attributes, such as color, intensity, or texture.
2. **Invariance**: Keypoints should exhibit a degree of invariance to common image transformations, such as rotation, scaling, and changes in lighting conditions. In other words, the same keypoint should be detectable in different versions of the same object or scene.
3. **Repeatability**: Keypoints should be reliably detectable across different instances of the same object or scene. This repeatability is essential for various applications, including object recognition and tracking.

There are several architectures designed for keypoint detection, such as HRNet and U-Net. The HRNet was designed for human pose estimation in high resolution images, while U-Net was designed for biomedical image segmentation. Due to its high accuracy and simplicity, we chose the U-Net architecture.

However, the plain U-Net architecture has limitations in terms of the "token-flatten" problem and the "scale-sensitivity" problem for medical image segmentation. The U-Net architecture has limitations in understanding long distance spatial relations in medical images, which hampers its

performance in segmentation tasks. To resolve these issues, the encoder of U-Net is often replaced with a Resnet encoder, such as Resnet34.

2.4 Dataset

In order to train a deep learning model for keypoint detection, we need a huge dataset of jeans images annotated with the desired keypoints. However, there are no datasets available for jeans that labels all the desired keypoints.

DeepFashion2 (and its prequel DeepFashion) are the only large fashion datasets available for keypoint detection task. However, they only label a small subset of the required keypoints, as shown in Figure 5 below. Nonetheless, this dataset serves as a good starting point for testing our methodology, before we can collect our own dataset.

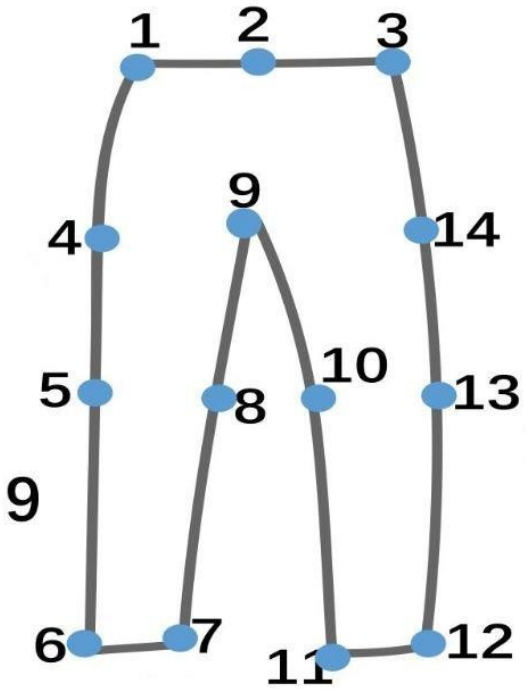


Figure 5 – Keypooints Labelled by DeepFashion2

Chapter 3

Methods and Materials

Our automatic measurement system consists of a hardware setup, image preprocessing, deep learning model, and graphical user interface. To capture a garment image with high quality, we design a hardware system, which includes a camera device, a lightening device, a shooting stand, and an operating table with a white color sheet laid throughout over it. A rough overview of this system is shown in Figure 6.

3.1 Hardware Setup

The hardware setup consists of a table, a camera, and a lightening device, such as an LED ring light. The camera and the lightening device are attached around 4 foot above the table surface through a table stand to ensure that the camera is able to capture the complete jeans.

The camera device used is a Google Pixel 6 smartphone. Although we initially planned on using webcams due to their versatility and low cost, we finalized on using a smartphone since we captured our dataset using smartphone camera as well. To ensure compatibility with the training set, we used a smartphone with a high-resolution camera, such as Google Pixel 6.

The lightening device used is a low-cost LED bar light. A better alternative would have been to use a ring light to ensure that the light is diffused evenly onto the garment, and not focused on a small region such as by the use of an LED bar light. However, due to its low cost, we continued using the LED bar light.

The shooting stand used is an adjustable mobile stand. Since we ended up using a smartphone as our camera device, we needed a mobile stand to hold the smartphone. The LED bar light was attached right next to the phone holder. However, this resulted in the phone getting heated quickly due to the high temperature of the LED bar light. To counteract this, we ended up placing a small wood piece between the bar light and the smartphone to add some heat insulation and taping everything together.

Finally, the table was large enough to ensure that the jeans can be laid flat, and a white sheet was used to cover the table surface. This white sheet would act as the background, and would be removed during image preprocessing.

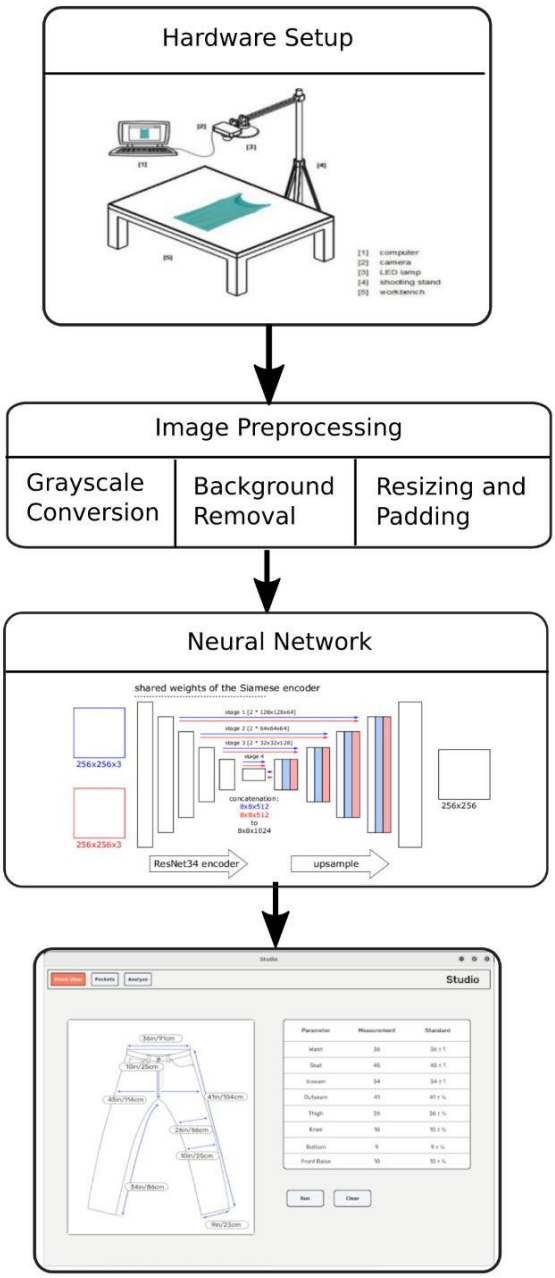


Figure 6 – Overview of the Proposed System

3.2 Image Preprocessing

The image preprocessing step consists of the following steps:

1. Data Augmentation
2. Grayscale Conversion
3. Background Removal
4. Padding to Aspect Ratio and Resizing

3.2.1 Data Augmentation

The performance of any machine learning model almost always increases with increased dataset size. Therefore, it is always a good idea to perform data augmentation, which is the process of adding slight variations to the existing data in order to artificially generate more data. Our dataset was considerably small, so we performed a variety of data augmentation in order to increase the dataset size by many folds. We implemented following types of data augmentation:

1. Affine Transformation: Translation, Scaling, and Rotation
2. Intensity Transformation: Brightness and Gamma Contrast
3. Elastic Transformation

We used a Python library called `imgaug` to perform data augmentation. Following is the snippet of code used to implement data augmentation using `imgaug:`

```
iaa.Sequential([
    iaa.Affine(
        translate_px = {"x": (-50, 50), "y": (-50, 50)},
        scale        = [0.8, 1],
        rotate       = [-5, 5],
    ),
    iaa.Multiply(mul = (0.5, 1.5)),
    iaa.GammaContrast(),
    iaa.ElasticTransformation(
        alpha = (0, 500),
        sigma = 30,
        order = 3,
    )
])
```

3.2.2 Grayscale Transformation

Since the detection of keypoints on jeans images does not depend on color information, we transformed our images to grayscale using a Python library called `skimage` with the following code snippet:

```
image = skimage.io.imread(image_path, as_gray=True)
```

3.2.3 Background Removal

The background removal process was not implemented in our project, however, there are several techniques that can be used for this task. The simplest and the most inflexible technique is to record an image of the static background and subtract it from each captured image. However, this technique can introduce large errors if the garment is not placed exactly in its place.

A second technique is to select a point on the background and use flood fill to connect all similarly colored points together, then deleting them. This is the preferred method if the background is of a single color with very small variations, such as a single white colored sheet.

3.2.4 Padding to Aspect Ratio and Resizing

The U-Net architecture requires the image dimensions (width and height) to be multiples of 32. Moreover, any convolutional neural network will benefit from having smaller sized images in order to improve training and inference time. Therefore, we resized the images to 320 pixels by 480 pixels. However, in order to prevent distortion, we first padded the image to the desired aspect ratio, and then resized. To implement this part, we again used the `imgaug` library.

```
iaa.Sequential( [
    iaa.PadToAspectRatio(320 / 480, position="right-bottom"),
    iaa.Resize({"width": 320, "height": 480}),
])
```

3.3 Neural Network

The core component of our system is a convolutional neural network based on the U-Net architecture with Resnet34 encoder. We explored several options for the choice of architecture, and we finalized with the following two options:

- HRNet
- U-Net with Resnet34 encoder

The High Resolution Network (HRNet) was designed for the task of human pose estimation in very high resolution images (4K), and is therefore overkill for our task. The U-Net architecture (named after its U shape) was designed for biomedical image segmentation, and is therefore very accurate. There is no architecture that beats a well-tuned U-Net in terms of localization accuracy.

To improve the feature extraction ability of the U-Net, the encoder portion is often replaced with a residual network such as Resnet34. The residual network enhances the feature extraction process by providing skip connections. This improved architecture is shown in Figure 7 below.

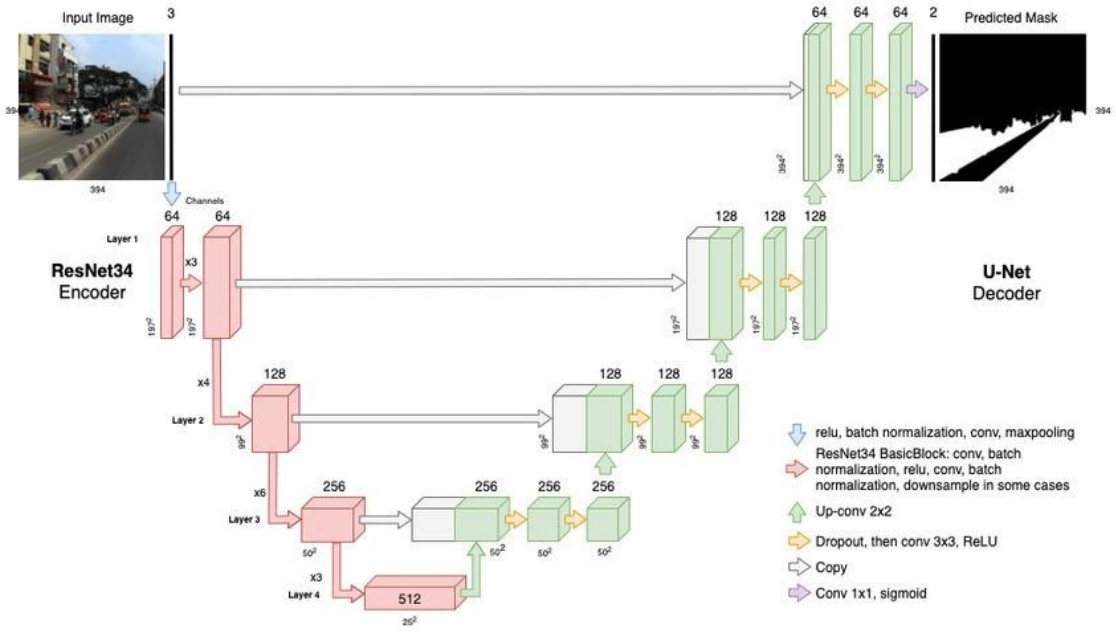


Figure 7 – U-Net with Resnet34 Encoder Architecture

To implement the model, we used a Python library called `segmentation_models_pytorch`.

```
class Unet(nn.Module):
    def __init__(self):
        super().__init__()
        self.unet = smp.Unet(
            encoder_name     = "resnet34",
            encoder_weights  = "imagenet",
            decoder_channels = [256, 128, 64, 32, 32],
            in_channels      = 1,
            classes          = 14
        )

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

A data loader was created that iterates over the dataset. In each iteration, it returns an image and its keypoints, along with some metadata. The keypoints are isolated into separate channels. So if the image is 320 by 480 pixels and has 14 keypoints, the target will be a 14 by 320 by 480 array called `channels` in which, for axis = 0, only the keypoint numbered `i` will be set to 1 while all other elements are 0. This acts like a very sharp heatmap for training the model, and the model learns to output the heatmap for each keypoint separately

```
channels = np.zeros([14, image.shape[0], image.shape[1]])
for i, (x, y) in enumerate(keypoints):
    # to handle edge case where x or y are outside the image boundary
    x = int(min(x, image.shape[1] - 1))
    y = int(min(y, image.shape[0] - 1))

    channels[i, y, x] = 1.0
```

The core logic for training the model involves wrapping a data loader around the dataset class, initializing an optimizer and schedular, and instantiating the model itself. Since we have access to a GPU (Nvidia GTX 1050ti), we moved the model tensors to the CUDA device for training. We set the learning rate to 0.001 (which is very commonly used), and used a multistep learning rate schedular that decays the learning rate by gamma once each milestone is reached.

The core loop for training the model runs for 10 epochs. In each iteration, the model is trained by loading the images and its annotations (as channels) from the data loader, and passing the image to the model to generate the output. The output is then passed to a 2D softmax function and cross entropy loss is calculated as the loss value. This loss value is then propagated backwards to train the model. The optimizer steps onto the next iteration, and once every milestone is reached, the schedular also steps. Finally, when the loop has completed 10 epochs, the model is saved as a serialized state dictionary in the desired location.

```
dataset   = LandmarkDataset(images_dir, labels_dir, do_augment=True)
loader    = DataLoader(dataset, batch_size=4, shuffle=True)
model     = Unet().cuda()
optimizer = Adam(model.parameters(), lr=0.001)
scheduler = MultiStepLR(optimizer, milestones=[4, 6, 8], gamma=0.1)

for epoch in range(10):
    model.train()

    for batch, (image, channels, meta) in enumerate(loader):
        image = image.cuda()
        channels = channels.cuda()

        output = model(image.float())
        output = softmax_2D(output)

        optimizer.zero_grad()
        loss = cross_entropy(output, channels)
        loss.backward()
        optimizer.step()

    scheduler.step()

torch.save(model.state_dict(), model_save_path)
```

We used the 2D softmax function as the final activation function, because softmax function has the property of converting a vector of real numbers into a probability distribution. This probability distribution is used as the output heatmap.

The 2D softmax and cross entropy loss functions are implemented according to the following code snippet:

```
def softmax_2D(x):
    exp_y = torch.exp(x)
    return exp_y / torch.sum(exp_y, dim=(2, 3), keepdim=True)


def cross_entropy(output, target):
    nll = -target * torch.log(output.double())
    return torch.mean(torch.sum(nll, dim=(2, 3)))
```

The statement `optimizer.zero_grad()` is required because in PyTorch, for every mini-batch during the training phase, we typically want to explicitly set the gradients to zero before starting to do backpropagation since PyTorch accumulates the gradients on subsequent backward passes. This accumulating behavior is convenient while training RNNs or when we want to compute the gradient of the loss summed over multiple mini-batches. So, the default action has been set to accumulate the gradients on every `loss.backward()` call.

Because of this, when we start the training loop, ideally, we should zero out the gradients so that the parameter update is done correctly. Otherwise, the gradient would be a combination of the old gradient, which we have already used to update your model parameters and the newly-computed gradient. It would therefore point in some other direction than the intended direction towards the minimum (or maximum, in case of maximization objectives).

For testing our model, we load the trained model state dictionary into the model, and start the inference loop with `torch.no_grad()` call to inform PyTorch that we do not want our model to be trained anymore. Within the inference loop, we again transfer the image and channels tensors to the CUDA device, and receive the output from the forward pass of the image as input to the model. The output is then passed to the 2D softmax function, its loss calculated using cross entropy loss. Finally, the radial errors, expected radial errors, and max probabilities are calculated as performance metrics.

```
dataset = LandmarkDataset(images_dir, labels_dir, do_augment=False)
loader = DataLoader(dataset, batch_size=1, shuffle=False)
model = Unet().cuda()
state_dict = torch.load(model_save_path)
model.load_state_dict(state_dict, strict=True)
model.eval()
with torch.no_grad():
    for idx, (image, channels, meta) in enumerate(loader):
        image = image.cuda()
        channels = channels.cuda()

        output = model(image.float())
        output = softmax_2D(output)

        loss = cross_entropy(output, channels)
        radial_errors,
        expected_radial_errors,
        max_probabilities = evaluate(
            output.cpu().detach().numpy(),
            meta["landmarks"].detach().numpy(),
            meta["pixel_size"].detach().numpy()
        )
```

The `evaluate()` calculates the radial errors as the radial distance between the target point and the predicted point. The expected radial error is calculated as the radial distance between the target point and the predicted point multiplied by the predicted probability as available in the output heatmap. The max probabilities are calculated as the hottest point in the heatmap.

```
def get_hottest_point(heatmap):
    w, h = heatmap.shape
    flattened_heatmap = np.ndarray.flatten(heatmap)
    hottest_idx = np.argmax(flattened_heatmap)
    return np.flip(np.array(np.unravel_index(hottest_idx, [w, h])))


def get_max_probability(heatmap):
    return np.max(heatmap)
```

3.4 Graphical User Interface

The graphical user interface is created using the Python library called PyQt5. We selected this library due to its flexibility, such as allowing the user to provide stylesheets for widgets. The stylesheets can then describe the styles to be applied to the widget in case of certain events, similar to CSS.

The Qt Designer, shown in Figure 8, was used to create the layout for the application. It allows to drag and drop the widgets and enclose the widgets in layouts. Each widget can then be styled according to the provided stylesheet.

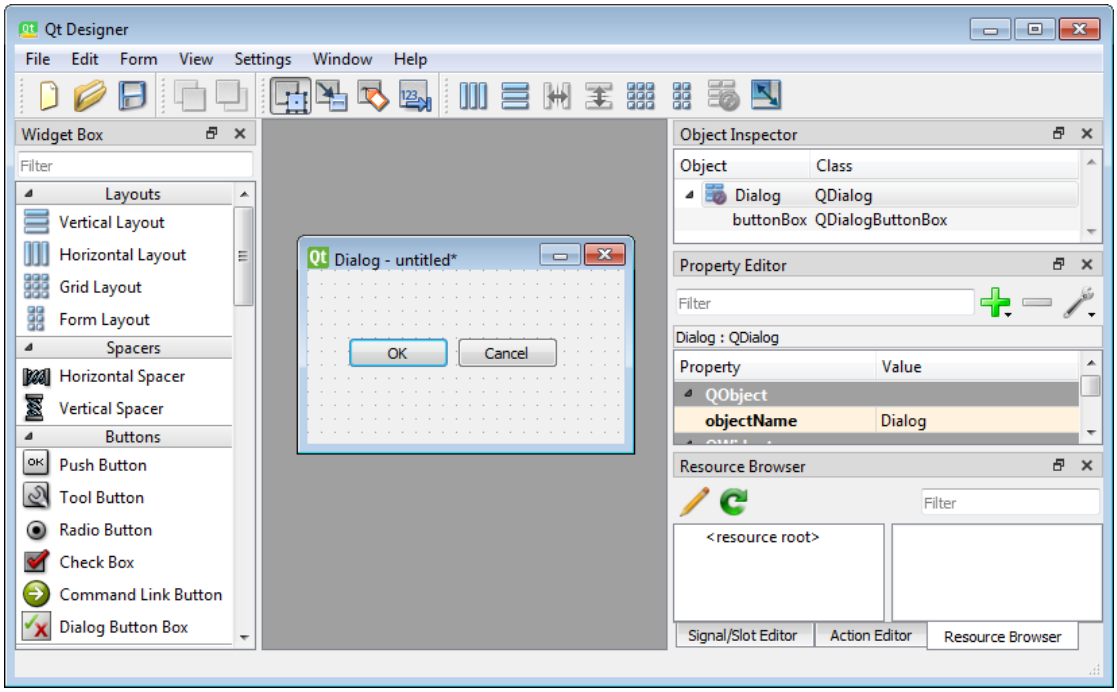


Figure 8 – QT Designer

We created an image template using Adobe Photoshop and then use the Python library Pillow to write the measurement readings over this template image, inside the rounded rectangles. We also created a table for quickly viewing the measurement readings and comparing them to the standard values. The complete graphical interface is shown in Figure 9 and Figure 10.

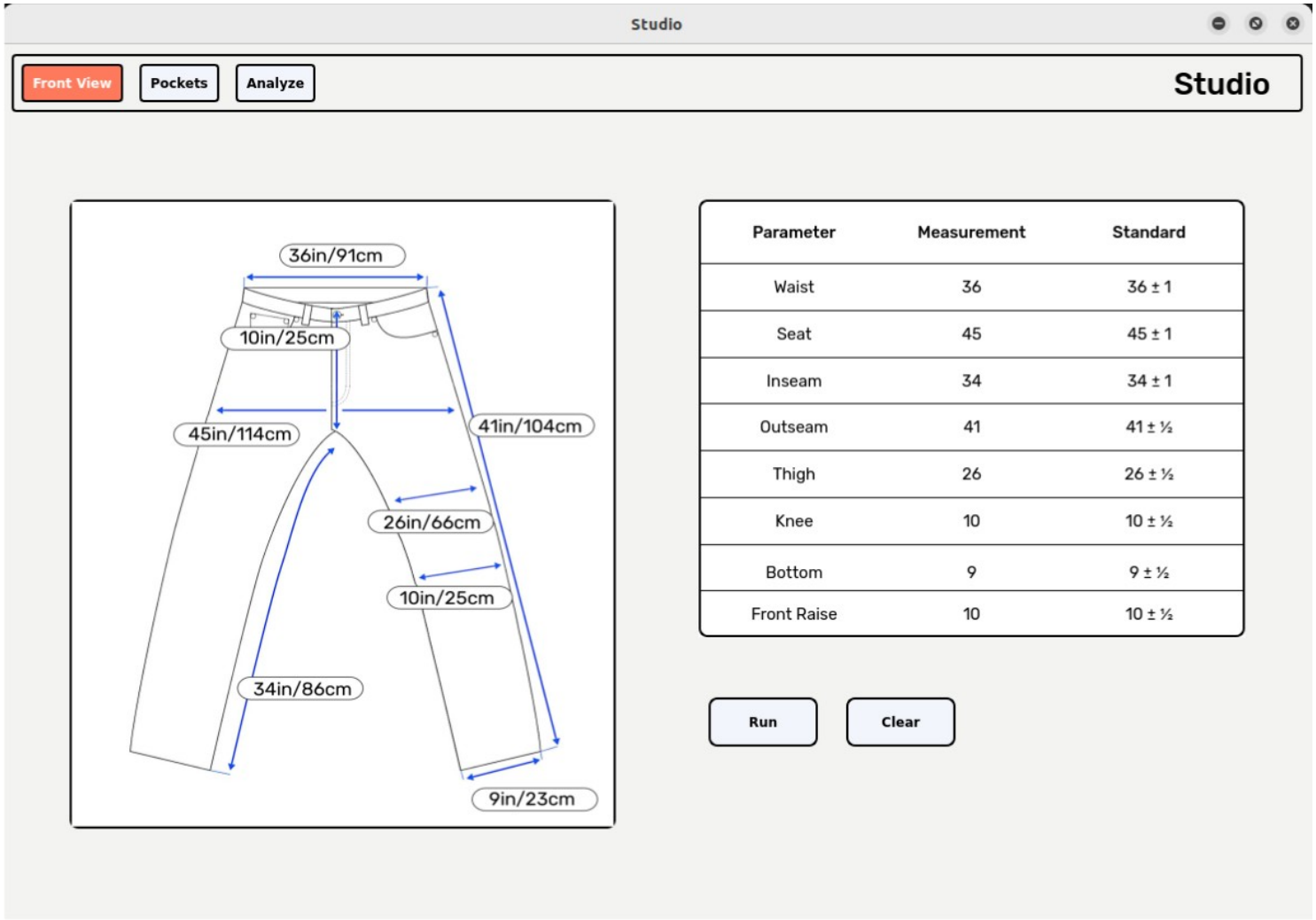
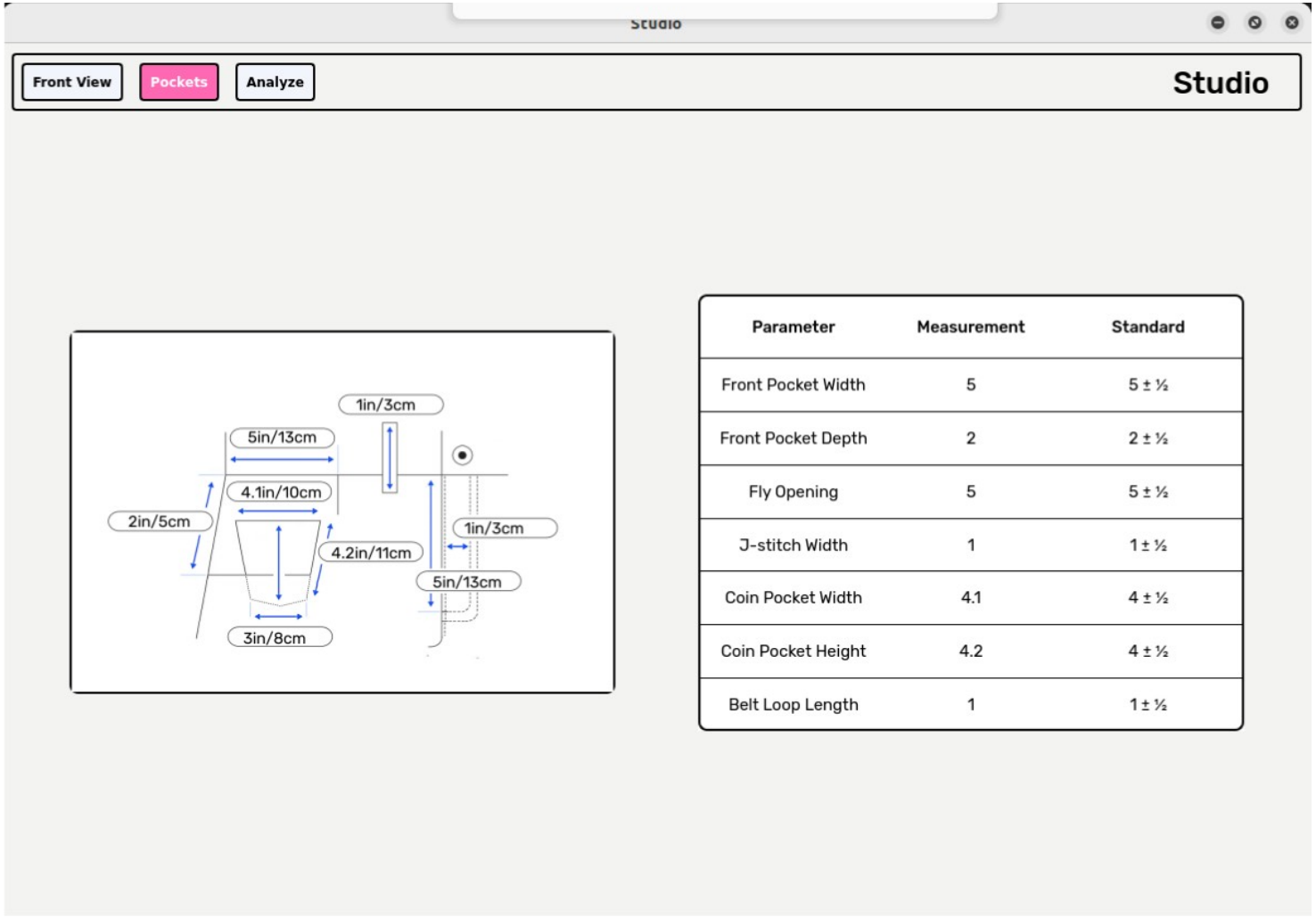


Figure 9 – GUI [First Page]



Parameter	Measurement	Standard
Front Pocket Width	5	5 ± ½
Front Pocket Depth	2	2 ± ½
Fly Opening	5	5 ± ½
J-stitch Width	1	1 ± ½
Coin Pocket Width	4.1	4 ± ½
Coin Pocket Height	4.2	4 ± ½
Belt Loop Length	1	1 ± ½

Figure 10 – GUI [Second Page]

Chapter 4

Results

The model was tested on a small number of jeans images and the resulting heatmaps and detected landmarks were manually analyzed. These results are shown for three jeans images. Figures 11-13 are for the first jean, Figures 14-16 are for the second jean, and Figures 17-19 are for the third jean.

These results apparently seem very satisfactory; however, it should be noted that these images have background removed and brightness adjusted. When these two conditions do not hold, or if there are occlusions or large folds in the jeans, the landmarks are detected are very wrong positions. This is because, the model mistakes shadows for seam lines and starts to confuse landmarks. Additionally, when the jeans have folds or if the brightness is not adjusted properly, there are sharp gradients in the image which the model also confuses for sewing lines and wrongfully detects landmarks on such regions.

We tested around 100 jeans images (all of which have background removed and brightness adjusted), and found out that the model was predicting around 80% of landmarks in their correct positions. For others, the landmarks were being detected in completely irrational locations such as outside the jeans outline, or even outside the image boundary. A possible explanation could have been that for such images, the jeans pant had no free space inside the image. There was no padding around the jeans inside the image boundary. So, the locations where the jeans pant touched the image boundaries created sharp gradients, which again the model confused for landmarks.

For the jeans images, for which the model predicted correctly, the detected landmarks were used to calculate the desired measurements. Since, the camera was not calibrated, we used a heuristic approach to find out that in our case, the pixel-to-mm ratio was 0.1318. We calculated the required distances between relevant keypoints and multiplied the resultant in pixels with this ratio to compute the distance in real world metric units.

We noted that these computed measurements were sensible when these measurements were large, such as for waist and inseam. However, for smaller measurements, such as front rise and knee. This could have been due to perspective errors, since we also did not perform perspective correction, and simply adjusted the camera position till we believed that it was centered.

Figure 11 Captured Image [Jeans 1]

Figure 12 – Heatmap [Jeans 1]

Figure 13 – Landmarks [Jeans 1]

Figure 14 – Captured Image [Jeans 2]

Figure 15 – Heatmap [Jeans 2]

Figure 16 – Landmarks [Jeans 2]

Figure 17 – Captured Image [Jeans 3]

Figure 18 – Heatmap [Jeans 3]

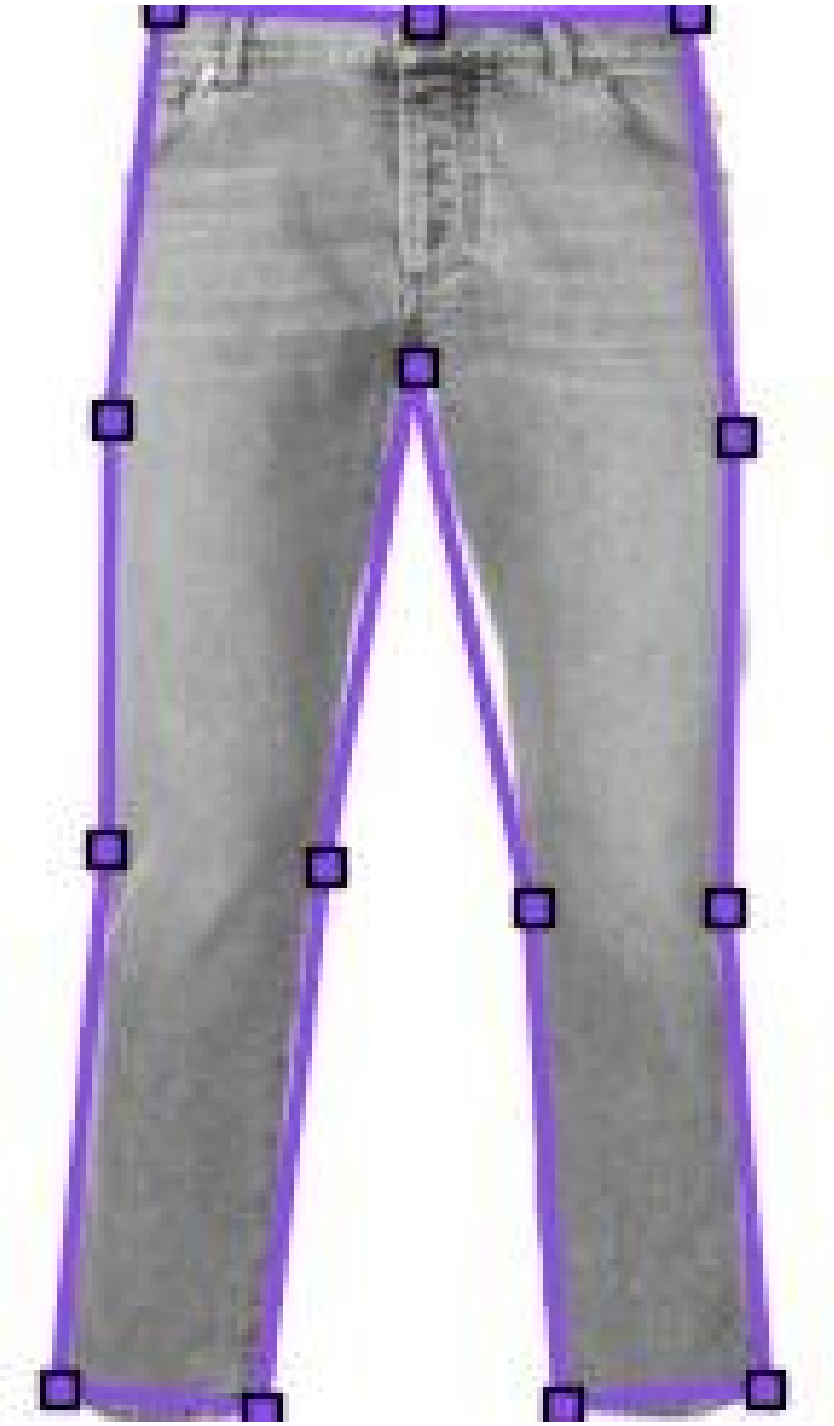


Figure 19 – Landmarks [Jeans 3]

The losses (as the cross-entropy loss compared to the ground truth), radial errors (for each of the 14 keypoints), and average error (for all keypoints) for the first 15 images are shown below.

```
Image: ['00001'] loss: 6.533 1.237mm 1.727mm 38.998mm 2.337mm 1.479mm 0.437mm
0.371mm 1.812mm 0.073mm 1.124mm 0.752mm 0.559mm 0.553mm 2.795mm average: 3.875mm
Image: ['00002'] loss: 7.038 6.724mm 6.666mm 9.907mm 1.182mm 0.510mm 0.329mm
0.458mm 0.489mm 0.346mm 2.361mm 0.431mm 16.037mm 1.161mm 2.596mm average: 3.514mm
Image: ['00003'] loss: 6.886 1.173mm 1.730mm 2.904mm 3.956mm 1.325mm 0.123mm
0.243mm 0.855mm 3.518mm 0.804mm 0.450mm 0.279mm 1.348mm 1.506mm average: 1.444mm
Image: ['00004'] loss: 6.518 2.946mm 1.311mm 1.309mm 4.103mm 0.347mm 0.363mm
1.583mm 3.034mm 0.265mm 4.770mm 0.259mm 0.208mm 5.313mm 2.088mm average: 1.993mm
Image: ['00005'] loss: 6.353 3.511mm 3.363mm 5.307mm 8.013mm 2.810mm 0.402mm
0.276mm 3.337mm 3.916mm 0.938mm 0.492mm 0.496mm 2.224mm 1.572mm average: 2.618mm
Image: ['00006'] loss: 6.512 13.116mm 2.592mm 1.279mm 0.665mm 0.225mm 0.328mm
0.433mm 0.851mm 0.177mm 0.647mm 0.992mm 0.923mm 0.103mm 1.637mm average: 1.712mm
Image: ['00007'] loss: 7.286 12.835mm 1.201mm 1.906mm 1.497mm 4.409mm 1.634mm
0.166mm 1.018mm 2.549mm 0.382mm 1.686mm 1.614mm 2.372mm 0.772mm average: 2.432mm
Image: ['00008'] loss: 7.372 0.693mm 2.452mm 14.737mm 2.453mm 0.918mm 18.979mm
10.261mm 8.151mm 0.766mm 1.195mm 8.228mm 18.368mm 22.513mm 2.971mm average:
8.049mm
Image: ['00009'] loss: 5.343 0.337mm 1.405mm 0.609mm 2.063mm 0.895mm 2.436mm
0.286mm 0.134mm 0.525mm 2.864mm 0.243mm 0.151mm 2.008mm 7.643mm average: 1.543mm
Image: ['00010'] loss: 4.995 5.185mm 3.740mm 0.249mm 4.229mm 1.062mm 0.147mm
0.798mm 0.460mm 0.470mm 1.636mm 0.393mm 0.178mm 0.050mm 0.210mm average: 1.343mm
Image: ['00011'] loss: 5.100 5.462mm 4.365mm 1.446mm 1.497mm 0.728mm 0.025mm
0.588mm 1.103mm 0.140mm 1.000mm 0.069mm 0.355mm 1.142mm 0.812mm average: 1.338mm
Image: ['00012'] loss: 7.419 1.797mm 0.189mm 1.728mm 2.272mm 3.504mm 1.368mm
0.229mm 4.559mm 5.342mm 6.129mm 0.669mm 0.205mm 9.590mm 0.328mm average: 2.708mm
Image: ['00013'] loss: 6.444 2.538mm 0.068mm 0.286mm 0.860mm 4.735mm 0.530mm
1.284mm 1.767mm 0.363mm 0.629mm 0.265mm 0.419mm 1.380mm 0.460mm average: 1.113mm
Image: ['00014'] loss: 6.708 0.883mm 1.714mm 1.401mm 0.853mm 3.879mm 0.855mm
0.625mm 2.382mm 0.343mm 1.437mm 0.485mm 1.704mm 1.006mm 0.821mm average: 1.313mm
Image: ['00015'] loss: 7.195 1.701mm 1.788mm 31.120mm 0.689mm 2.829mm 7.208mm
0.263mm 0.824mm 0.937mm 0.965mm 0.399mm 0.798mm 0.473mm 1.019mm average: 3.644mm
```

These errors and losses were used to describe the performance of our model. We found out that the average loss was 6.921 and the average radial error (MRE) was 3.426mm across the 100 images tested.

The average radial error per landmark was:

- 3.454mm
- 2.339mm
- 4.739mm
- 4.222mm
- 3.634mm
- 4.399mm
- 2.702mm
- 2.874mm
- 2.302mm
- 2.126mm
- 3.563mm
- 4.584mm
- 3.394mm
- 3.636mm

The successful detection rate (SDR) for 2mm, 2.5mm, 3mm, 4mm category are summarized in the table below:

2mm	**2.5mm**	**3mm**	**4mm**
66.571%	72.429%	76.929%	81.929%

References

1. Köstinger, M., Wohlhart, P., Roth, P. M., & Bischof, H. (2011). Annotated Facial Landmarks in the Wild: A large-scale, real-world database for facial landmark localization. In 2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops) (pp. 2144-2151). Barcelona, Spain.
2. Sun, K., Xiao, B., Liu, D., & Wang, J. (2019). Deep High-Resolution Representation Learning for Human Pose Estimation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 5693-5703).
3. Schwendicke, F., Chaurasia, A., Arsiwala, L., et al. (2021). Deep learning for cephalometric landmark detection: systematic review and meta analysis. Clin Oral Invest, 25, 4299–4309.
4. Ge, Y., Zhang, R., Wang, X., Tang, X., & Luo, P. (2019). DeepFashion2: A Versatile Benchmark for Detection, Pose Estimation, Segmentation and Re-Identification of Clothing Images. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 5337-5345).
5. Ibragimov, B., Likar, B., Pernus, F., & Vrtovec, T. (2014). "Automatic cephalometric x-ray landmark detection by applying game theory and random forests." In Proc. ISBI Int. Symp. on Biomedical Imaging (pp. 1 8).
6. Payer, C., Stern, D., Bischof, H., & Urschler, M. (2019). "Integrating spatial configuration into heatmap regression based CNNs for landmark localization." Medical Image Analysis, 54, 207–219.
7. Guo, Y., Shi, H., Kumar, A., Grauman, K., Rosing, T., & Feris, R. (2019). SpotTune: Transfer Learning Through Adaptive Fine-Tuning. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 4805-4814).
8. Deng, J., Dong, W., Socher, R., Li, L. -J., Li, K., & Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. In 2009 IEEE Conference on Computer Vision and Pattern Recognition (pp. 248-255).
9. Sidnev, A., Krapivin, A., Trushkov, A., Krasikova, E., Kazakov, M., & Viryasov, M. (2020). "DeepMark++: Real-time Clothing Detection at the Edge." arXiv preprint arXiv:2006.03296, v3. Submitted on 1 Jun 2020, last revised 10 Nov 2020.
10. McCouat, J., & Voiculescu, I. (2022). "Contour-hugging heatmaps for landmark detection." In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 20597-20605).
11. Sun, K., Xiao, B., Liu, D., & Wang, J. (2019). "Deep High-Resolution Representation Learning for Human Pose Estimation." In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 5693-5703).

12. Ronneberger, O., Fischer, P., & Brox, T. (2015). "U-Net: Convolutional Networks for Biomedical Image Segmentation." In N. Navab, J. Hornegger, W. Wells, & A. Frangi (Eds.), Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015 (Vol. 9351, pp. 234-241). Springer, Cham.
13. C. Li, Y. Xu, Y. Xiao, H. Liu, M. Feng, and D. Zhang, "Automatic Measurement of Garment Sizes Using Image Recognition," in *Proceedings of the International Conference on Graphics and Signal Processing (ICGSP '17)*, 2017. doi: 10.1145/3121360.3121382.
14. B. Coffey and T. J. Torres, "Photo Based Clothing Measurements," [Online]. Available: https://multithreaded.stitchfix.com/blog/2016/09/30/photo-based-clothing-measurement. [Accessed: Jun. 3, 2024].
15. K. H. K. Boon, "U-Net with ResNet Backbone for Garment Landmarking Purpose," arXiv:2204.12084v1 [cs.CV], Feb. 15, 2022. Available: arXiv:2204.12084.
16. A. Paulauskaite-Taraseviciene, E. Noreika, R. Purtokas, I. Lagzdinyte-Budnike, V. Daniulaitis, and R. Salickaite-Zukauskiene, "An Intelligent Solution for Automatic Garment Measurement Using Image Recognition Technologies," *Applied Sciences*, vol. 12, no. 9, p. 4470, 2022. doi: 10.3390/app12094470.
17. S. Kim, H. Moon, J. Oh, Y. Lee, H. Kwon, and S. Kim, "Automatic Measurements of Garment Sizes Using Computer Vision Deep Learning Models and Point Cloud Data," *Applied Sciences*, vol. 12, no. 10, p. 5286, 2022. doi: 10.3390/app12105286