Help Document
Virtual Try-On System for Shalwar Qameez

Your Name
Your Institution

January 6, 2025

Contents

1 Introduction

Welcome to the **Virtual Try-On System for Shalwar Qameez**! This guide will walk you through the process of running the prototype, from setting up your environment to processing your images and accessing the results. Follow the steps below to utilize the system effectively.

2 Prerequisites

Before you begin, ensure you have the following:

- **Google Account**: To access Google Colab and Google Drive.
- **Images**: High-quality images of models wearing shalwar qameez in supported formats (`.png`, `.jpg`, `.jpeg`, `.webp`, `.bmp`, `.gif`).
- **Stable Internet Connection**: For uploading files and running the Colab notebook.

3 Getting Started

3.1 Accessing Google Colab

a. **Navigate to Google Colab**:

   - Open your web browser and go to Google Colab.

b. **Create a New Notebook**:

   - Click on `File ¿ New Notebook`.
   - Alternatively, you can open an existing notebook if you have one prepared.

c. **Set Up GPU Acceleration**:

   - Click on `Runtime` in the top menu.
   - Select `Change runtime type`.
   - Under `Hardware accelerator`, choose `GPU`.
   - Click `Save`.

d. **Install Dependencies**:

   - In a new Colab cell, enter and execute the following code to install necessary libraries:

```
!pip install -q tensorflow==2.9.0
!pip install -q tensorflow-hub
!pip install -q opencv-python
!pip install -q matplotlib
!pip install -q tqdm
```

3.2 Mounting Google Drive

Mounting your Google Drive allows Colab to access your files directly, enabling easy management of input images and output results.

a. **Run the Mount Command**:

- In a new Colab cell, enter and execute the following code:

```
from google.colab import drive
drive.mount('/content/drive')
```

b. **Authorize Access**:

- After executing, a prompt will appear.
- Click on the authorization link provided.
- Select your Google account and grant access.
- Copy the authorization code provided.
- Paste the code back into the Colab prompt and press `Enter`.

c. **Verify Mounting**:

- Your Google Drive is now mounted and accessible at `/content/drive/MyDrive/`.
- You can verify by listing the contents:

```
!ls /content/drive/MyDrive/
```

4 Uploading Your Dataset

You can provide your dataset to Colab in two ways:

4.1 Option 1: Direct Upload to Colab

Suitable for smaller datasets (a few dozen images).

a. **Create an Input Directory**:

- In a new Colab cell, enter and execute:

```
import os

# Create a directory to store uploaded images
os.makedirs('input_images', exist_ok=True)
```

b. **Upload Images**:

- In a new Colab cell, enter and execute:

  ```
  from google.colab import files

  # Upload files
  uploaded = files.upload()

  # Move uploaded files to 'input_images' directory
  for filename in uploaded.keys():
      os.rename(filename, os.path.join('input_images', filename))

  print("Uploaded images are saved in 'input_images/' directory.")
  ```

- **Instructions**:
  i. Run the above cell.
  ii. Click on the `"Choose Files"` button that appears.
  iii. Select and upload your images.
  iv. The images will be stored in the `input_images/` directory within the Colab environment.

4.2 Option 2: Using Google Drive

Ideal for larger datasets and seamless integration.

a. **Organize Your Images in Google Drive**:

- Create a folder (e.g., `shalwar_qameez_images`) in your Drive.
- Upload all your images into this folder.

b. **Define Directory Paths in Colab**:

- In a new Colab cell, enter and execute:

  ```
  import os

  # Define paths based on Google Drive structure
  input_dir = '/content/drive/MyDrive/shalwar_qameez_images/'       # Folder
  output_json_dir = '/content/drive/MyDrive/keypoints_json/'       # Directo
  output_rendered_dir = '/content/drive/MyDrive/rendered_images/' # Director
  ```

```
    # Create output directories if they don't exist
    os.makedirs(output_json_dir, exist_ok=True)
    os.makedirs(output_rendered_dir, exist_ok=True)

    print("Input and output directories are set up.")
```

c. **Verify Directory Setup**:

   - List the contents to ensure paths are correct:

     ```
     print("Input Directory Contents:")
     print(os.listdir(input_dir))
     ```

5 Running the Keypoint Extraction Script

The keypoint extraction process involves processing each image to detect keypoints and generating both JSON data and annotated images.

5.1 a. Setting Up Directories

Ensure that your input and output directories are correctly defined and exist.

```
import os

# Define paths
input_dir = '/content/drive/MyDrive/shalwar_qameez_images/'       # Update as per y
output_json_dir = '/content/drive/MyDrive/keypoints_json/'       # Update as per yo
output_rendered_dir = '/content/drive/MyDrive/rendered_images/' # Update as per you

# Create output directories if they don't exist
os.makedirs(output_json_dir, exist_ok=True)
os.makedirs(output_rendered_dir, exist_ok=True)

print("Directories are set and ready.")
```

5.2 b. Executing the Script

1. Import Necessary Libraries

```
import tensorflow as tf
import tensorflow_hub as hub
import numpy as np
import cv2
import json
import os
from tqdm import tqdm
import matplotlib.pyplot as plt
```

2. Define the Remapping Function

```
def movenet_to_coco18(movenet_keypoints):
    """
    Remaps MoveNet’s 17 keypoints to COCO’s 18 keypoints.
    """
    NOSE = 0
    LEFT_EYE = 1
    RIGHT_EYE = 2
    LEFT_EAR = 3
    RIGHT_EAR = 4
    LEFT_SHOULDER = 5
    RIGHT_SHOULDER = 6
    LEFT_ELBOW = 7
    RIGHT_ELBOW = 8
    LEFT_WRIST = 9
    RIGHT_WRIST = 10
    LEFT_HIP = 11
    RIGHT_HIP = 12
    LEFT_KNEE = 13
    RIGHT_KNEE = 14
    LEFT_ANKLE = 15
    RIGHT_ANKLE = 16

    coco_keypoints = np.zeros((18, 3), dtype=np.float32)

    # 0: Nose
    coco_keypoints[0] = movenet_keypoints[NOSE]

    # 1: Neck (average of left and right shoulders)
    left_shoulder = movenet_keypoints[LEFT_SHOULDER]
    right_shoulder = movenet_keypoints[RIGHT_SHOULDER]
    neck_y = (left_shoulder[0] + right_shoulder[0]) / 2.0
    neck_x = (left_shoulder[1] + right_shoulder[1]) / 2.0
    neck_conf = (left_shoulder[2] + right_shoulder[2]) / 2.0
    coco_keypoints[1] = np.array([neck_y, neck_x, neck_conf], dtype=np.float32)

    # 2: RShoulder
    coco_keypoints[2] = movenet_keypoints[RIGHT_SHOULDER]
    # 3: RElbow
    coco_keypoints[3] = movenet_keypoints[RIGHT_ELBOW]
    # 4: RWrist
    coco_keypoints[4] = movenet_keypoints[RIGHT_WRIST]

    # 5: LShoulder
    coco_keypoints[5] = movenet_keypoints[LEFT_SHOULDER]
    # 6: LElbow
    coco_keypoints[6] = movenet_keypoints[LEFT_ELBOW]
    # 7: LWrist
    coco_keypoints[7] = movenet_keypoints[LEFT_WRIST]

    # 8: RHip
    coco_keypoints[8] = movenet_keypoints[RIGHT_HIP]
    # 9: RKnee
    coco_keypoints[9] = movenet_keypoints[RIGHT_KNEE]
    # 10: RAnkle
    coco_keypoints[10] = movenet_keypoints[RIGHT_ANKLE]

    # 11: LHip
    coco_keypoints[11] = movenet_keypoints[LEFT_HIP]
    # 12: LKnee
    coco_keypoints[12] = movenet_keypoints[LEFT_KNEE]
    # 13: LAnkle
    coco_keypoints[13] = movenet_keypoints[LEFT_ANKLE]

    # 14: REye
    coco_keypoints[14] = movenet_keypoints[RIGHT_EYE]
    # 15: LEye
    coco_keypoints[15] = movenet_keypoints[LEFT_EYE]
    # 16: REar
    coco_keypoints[16] = movenet_keypoints[RIGHT_EAR]
    # 17: LEar
    coco_keypoints[17] = movenet_keypoints[LEFT_EAR]

    return coco_keypoints
```

3. Define the Rendering Function

```
def render_keypoints(image, keypoints, output_path):
    """
    Renders keypoints on the image and saves it.

    Parameters:
    - image: The original BGR image as a NumPy array.
    - keypoints: A NumPy array of shape (18, 3) representing COCO-18 keypoints.
    - output_path: Path to save the rendered image.
    """
    # Define colors for different keypoints
    color = (0, 255, 0)  # Green color in BGR
    thickness = 2
    radius = 4

    # Define connections based on COCO format
    connections = [
        (0, 1), (1, 2), (2, 3), (3, 4),        # Nose to Eyes and Ears
        (1, 5), (1, 8),                         # Neck to Shoulders and Hips
        (5, 6), (6, 7), (8, 9), (9, 10),        # Shoulders to Elbows to Wrists and
        (5, 11), (11, 12), (12, 13),            # Left side
        (8, 14), (14, 15), (15, 16)             # Right side
    ]

    # Draw keypoints
    for point in keypoints:
        y, x, conf = point
        if conf > 0.1:  # Threshold to filter out low-confidence keypoints
            cv2.circle(image, (int(x), int(y)), radius, color, thickness)

    # Draw connections
    for connection in connections:
        start, end = connection
        y1, x1, conf1 = keypoints[start]
        y2, x2, conf2 = keypoints[end]
        if conf1 > 0.1 and conf2 > 0.1:
            cv2.line(image, (int(x1), int(y1)), (int(x2), int(y2)), color, thicknes

    # Save the rendered image
    cv2.imwrite(output_path, image)
```

4. Define the Main Processing Function

```
def process_images(input_dir, output_json_dir, output_rendered_dir, model_url='http
    """
    Processes images to extract keypoints using MoveNet, saves keypoints as JSON,
    and saves rendered images with keypoints overlayed.

    Parameters:
    - input_dir: Directory containing input images.
    - output_json_dir: Directory to save JSON keypoints files.
    - output_rendered_dir: Directory to save rendered images.
    - model_url: URL of the MoveNet model on TensorFlow Hub.
    """
    # Load MoveNet model
    print("Loading MoveNet from TF-Hub...")
    model = hub.load(model_url)
    print("MoveNet model loaded successfully.")

    # Iterate over images in input_directory
    image_files = [f for f in os.listdir(input_dir) if f.lower().endswith(('.png',

    for filename in tqdm(image_files, desc="Processing Images"):
        image_path = os.path.join(input_dir, filename)
```

```
# Read & preprocess
image_bgr = cv2.imread(image_path)
if image_bgr is None:
    print(f"Warning: Could not read {image_path}, skipping.")
    continue
image_rgb = cv2.cvtColor(image_bgr, cv2.COLOR_BGR2RGB)

# Resize to 192x192 with padding for MoveNet
input_image = tf.image.resize_with_pad(
    np.expand_dims(image_rgb, axis=0),
    192, 192
)
input_image = tf.cast(input_image, dtype=tf.int32)

# Inference
outputs = model.signatures['serving_default'](input_image)
keypoints_17 = outputs['output_0'].numpy().reshape((17, 3))

# Scale keypoints back to original image size
height, width, _ = image_bgr.shape
keypoints_17[:, 0] *= height  # Y
keypoints_17[:, 1] *= width   # X
# (Confidence remains at index 2)

# Remap 17 -> COCO-18
coco18_keypoints = movenet_to_coco18(keypoints_17)

# Build final pose_keypoints_2d array in [x, y, conf] format
pose_keypoints_2d = []
for (y, x, conf) in coco18_keypoints:
    pose_keypoints_2d.extend([float(x), float(y), float(conf)])

# Create JSON structure
keypoints_dict = {
    "version": 1.3,
    "people": [
        {
            "person_id": [-1],
            "pose_keypoints_2d": pose_keypoints_2d,
            "face_keypoints_2d": [],
            "hand_left_keypoints_2d": [],
            "hand_right_keypoints_2d": [],
            "pose_keypoints_3d": [],
            "face_keypoints_3d": [],
            "hand_left_keypoints_3d": [],
            "hand_right_keypoints_3d": []
        }
    ]
}

# Save JSON file
json_filename = f'{os.path.splitext(filename)[0]}_keypoints.json'
output_json_path = os.path.join(output_json_dir, json_filename)
with open(output_json_path, 'w') as json_file:
    json.dump(keypoints_dict, json_file, indent=2)

# Render keypoints on image
rendered_image = image_bgr.copy()
rendered_image_path = os.path.join(output_rendered_dir, f"{os.path.splitext
render_keypoints(rendered_image, coco18_keypoints, rendered_image_path)

print(f"Processed {filename} -> {json_filename} & {os.path.splitext(filenam
```

5. Execute the Processing Function

Run the following cell to start processing your images:

```
# Execute the processing function
process_images(input_dir, output_json_dir, output_rendered_dir)
```

What Happens:

- The script processes each image in the `input_dir`.
- Extracts keypoints using MoveNet.
- Saves the keypoints as `name_keypoints.json` in `output_json_dir`.
- Saves rendered images with overlaid keypoints as `name_rendered.png` in `output_rendered_dir`.

Note: Processing time depends on the number and size of images. Using `tqdm` provides a progress bar for monitoring.

6 Accessing the Outputs

After processing, your outputs will be organized into two separate directories:

1. **Keypoints JSON Directory**: Contains JSON files with keypoint data.
    - `/content/drive/MyDrive/keypoints_json/` (if using Google Drive)
    - `model1_keypoints.json`, `model2_keypoints.json`, etc.
2. **Rendered Images Directory**: Contains images with rendered keypoints.
    - `/content/drive/MyDrive/rendered_images/` (if using Google Drive)
    - `model1_rendered.png`, `model2_rendered.png`, etc.

6.1 Accessing via Google Drive

a. **Navigate to Google Drive**:

- Go to Google Drive and log in to your account.

b. **Locate the Output Folders**:

- Find the `keypoints_json/` and `rendered_images/` folders within your Drive.

c. **Download Files**:

- Right-click on any file and select `Download`.

6.2 Accessing via Colab's Local Storage

a. **Use Colab's Files Sidebar**:

- Click on the `Files` tab on the left sidebar.
- Navigate to the `rendered_images/` or `keypoints_json/` directories.

b. **Download Specific Files**:

- Right-click on a file and select `Download`.

Note: Files extracted to Colab's local filesystem will be lost once the Colab session ends. To retain extracted files, ensure they are saved to Google Drive.

7 Visualizing Rendered Images

To ensure that keypoints are accurately detected and rendered, you can visualize the rendered images within Colab.

7.1 Define Visualization Function

```
import matplotlib.pyplot as plt
import cv2

def visualize_rendered_image(rendered_image_path):
    """
    Displays the rendered image with keypoints.

    Parameters:
    - rendered_image_path (str): Path to the rendered PNG image.
    """
    image = cv2.imread(rendered_image_path)
    if image is None:
        print(f"Error: Unable to read {rendered_image_path}")
        return
    image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    plt.figure(figsize=(10, 10))
    plt.imshow(image_rgb)
    plt.axis('off')
    plt.show()
```

7.2 Visualize an Image

```
# Replace with your actual filename
sample_rendered_image = '/content/drive/MyDrive/rendered_images/model1_rendered.png

# Visualize
visualize_rendered_image(sample_rendered_image)
```

Instructions:

i. Replace `'model1_rendered.png'` with the actual filename of a rendered image.

ii. Run the cell to display the image with overlaid keypoints.

8 Common Issues and Solutions

8.1 a. TensorFlow Compatibility Issues

Issue:

```
ImportError: No module named 'tensorflow_hub'
```

Solution:

i. Ensure all dependencies are installed correctly.

ii. Re-run the installation commands and restart the Colab runtime:

```
# Reinstall TensorFlow and TensorFlow Hub
!pip install -q tensorflow==2.9.0
!pip install -q tensorflow-hub
```

iii. Restart the runtime:

- Click on `Runtime ¿ Restart runtime`.

8.2 b. Unable to Read Images

Issue:

```
Warning: Could not read /path/to/image.jpg, skipping.
```

Solution:

i. **Verify File Paths**:

   - Ensure that `input_dir` is correctly set to the directory containing your images.

ii. **Check Image Integrity**:

   - Make sure images are not corrupted and are in supported formats.

iii. **Permissions**:

   - If using Google Drive, ensure Colab has access to the Drive and the files are not restricted.

8.3 c. MoveNet Model Not Loading or Running Out of Memory

Issue:

```
ResourceExhaustedError: OOM when allocating tensor with shape...
```

Solution:

i. **Batch Size**:

   - The provided script processes images one by one, which should generally prevent memory issues.

ii. **Image Resolution**:

   - High-resolution images consume more memory. Consider resizing images to a lower resolution before processing.

iii. **Colab Resources**:

   - Ensure that you're using a GPU runtime.
   - If issues persist, consider restarting the runtime or reducing the number of concurrent processes.

8.4 d. Keypoints Not Detected Properly

Issue:

- Keypoints are missing or inaccurately placed.

Solution:

i. **Image Quality**:

   - Ensure images are clear, well-lit, and have the full body visible.

ii. **Confidence Threshold**:

   - Adjust the confidence threshold in the `render_keypoints` function if necessary.

iii. **Model Variants**:

   - MoveNet offers different models (e.g., `singlepose/lightning` and `singlepose/thunder`). You can try switching models for better accuracy.

   Example: Switching to a different MoveNet model

```
# Using the Thunder variant
model_url = 'https://tfhub.dev/google/movenet/singlepose/thunder/4'
process_images(input_dir, output_json_dir, output_rendered_dir, model_url=mode
```

8.5 e. Errors During File Operations

Issue:

- Errors related to file reading, writing, or directory creation.

Solution:

i. **Check Directory Paths**:

   - Ensure that all directories (`input_dir`, `output_json_dir`, `output_rendered_dir`) exist and are correctly specified.

ii. **Permissions**:

   - If writing to Google Drive, ensure you have write permissions to the target folders.

iii. **Filename Conflicts**:

   - Ensure that filenames do not contain invalid characters and that no two files have the same name (to prevent overwriting).

9 FAQs

Q1. Can I process multiple images at once?

A1. Yes, the script is designed to handle batch processing. It iterates through all supported image files in the `input_dir` and processes them sequentially.

Q2. How do I change the input and output directories?

A2. Modify the `input_dir`, `output_json_dir`, and `output_rendered_dir` variables in your Colab notebook to point to your desired directories. Ensure that the paths correspond to actual folders in your Google Drive or Colab's local storage.

Q3. What if my images are not being processed?

A3. Ensure that:

- Images are in supported formats.
- The `input_dir` path is correct.
- Images are not corrupted.
- You have sufficient permissions to access the files.

Q4. How can I improve keypoint detection accuracy?

A4.

- Use high-resolution, clear images with minimal obstructions.
- Ensure full-body visibility in the images.
- Experiment with different MoveNet model variants (`lightning` vs. `thunder`).

Q5. Can I integrate this script with other systems?

A5. Yes, the JSON keypoints can be utilized by other models or systems, such as SD-VITON, for virtual try-on functionalities.

10 Contact Support

If you encounter issues not covered in this guide or need further assistance:

- **Email**: support@virtualtryon.com
- **Support Forum**: Virtual Try-On Support
- **GitHub Repository**: Virtual Try-On GitHub

Please provide detailed information about your issue, including error messages and steps to reproduce, to facilitate prompt assistance.