My task

Avatar Creation and Dress Selection

connect with drive

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount
```

Preprocessing methods

step-1

Removed background from Person_image and resize that image

```
pip install rembg
```

```
from rembg import remove
from PIL import Image

input_path = '/content/drive/MyDrive/task/input/input_person.png'
output_path = '/content/drive/MyDrive/task/Preprossed_images/new@=person_front.png'

input = Image.open(input_path)
output = remove(input)
output.save(output_path)
```

```
import numpy as np
import cv2
from PIL import Image

p1 = np.array(Image.open("/content/drive/MyDrive/task/Preprossed_images/new@=person_front.png"))
print(p1.shape)

"""# changed based on our requirements"""

res = cv2.resize(p1, dsize=(320,512), interpolation=cv2.INTER_CUBIC)
#convert 4 channal to 3 channel
img = cv2.cvtColor(res, cv2.COLOR_BGRA2BGR)
from PIL import Image
im = Image.fromarray(img)
im.save("/content/drive/MyDrive/task/mpv3d_example/image/new@=person_whole_front.png",quality=95)
```

```
(512, 320, 4)
```

show the input image

```
import matplotlib.pyplot as plt
from PIL import Image
plt.imshow(im)
plt.axis('off')
plt.show()
```

Create the Keypoints from input_person image

```
import cv2
import numpy as np
import os
import json

class general_pose_model(object):
    def __init__(self, modelpath):
        # Specify the model to be used
        #   Body25: 25 points
        #   COCO:   18 points
        #   MPI:    15 points
        self.inWidth = 368
        self.inHeight = 368
        self.threshold = 0.05
        self.pose_net = self.general_body25_model(modelpath)

    def general_body25_model (self, modelpath):
        self.num_points = 25
        self.point_pairs = [[1, 0], [1, 2], [1, 5],
                            [2, 3], [3, 4], [5, 6],
                            [6, 7], [0, 15], [15, 17],
                            [0, 16], [16, 18], [1, 8],
                            [8, 9], [9, 10], [10, 11],
                            [11, 22], [22, 23], [11, 24],
                            [8, 12], [12, 13], [13, 14],
                            [14, 19], [19, 20], [14, 21]]
        prototxt = os.path.join (
            modelpath,
            "/content/drive/MyDrive/task/Preprossed_images/pose_deploy.prototxt")
        caffemodel = os.path.join (
            modelpath,
            "/content/drive/MyDrive/task/Preprossed_images/pose_iter_584000.caffemodel")
        coco_model = cv2.dnn.readNetFromCaffe (prototxt, caffemodel)

        return coco_model

    def predict(self, imgfile):
        img_cv2 = cv2.imread(imgfile)
        img_height, img_width, _ = img_cv2.shape
        inpBlob = cv2.dnn.blobFromImage(img_cv2,
                                        1.0 / 255,
                                        (self.inWidth, self.inHeight),
                                        (0, 0, 0),
                                        swapRB=False,
                                        crop=False)
        self.pose_net.setInput(inpBlob)
        self.pose_net.setPreferableBackend(cv2.dnn.DNN_BACKEND_OPENCV)
        self.pose_net.setPreferableTarget(cv2.dnn.DNN_TARGET_OPENCL)

        output = self.pose_net.forward()

        H = output.shape[2]
        W = output.shape[3]

        points = []
```

```
            for idx in range(self.num_points):
                probMap = output[0, idx, :, :] # confidence map.

                # Find global maxima of the probMap.
                minVal, prob, minLoc, point = cv2.minMaxLoc(probMap)

                # Scale the point to fit on the original image
                x = (img_width * point[0]) / W
                y = (img_height * point[1]) / H

                if prob > self.threshold:
                    points.append(x)
                    points.append(y)
                    points.append(prob)
                else:
                    points.append(0)
                    points.append(0)
                    points.append(0)

            return points

    def generate_pose_keypoints(img_file, pose_file):

        modelpath = 'pose'
        pose_model = general_pose_model(modelpath)

        res_points = pose_model.predict(img_file)

        pose_data = {"version": 1.3,
                     "people":  [
                                    {'person_id': [-1], 'pose_keypoints_2d': res_points,'face_keypoints_2d': [], 'hand_left_keypoi
                                ]
                    }

        pose_keypoints_path = pose_file

        json_object = json.dumps(pose_data, indent = 4)

        # Writing to sample.json
        with open(pose_keypoints_path, "w") as outfile:
            outfile.write(json_object)
        print('File saved at {}'.format(pose_keypoints_path))
```

```
generate_pose_keypoints("/content/drive/MyDrive/task/mpv3d_example/image/new@=person_whole_front.png",
                        "/content/drive/MyDrive/task/mpv3d_example/pose/new@=person_whole_front_keypoints.json")
```

```
File saved at /content/drive/MyDrive/task/mpv3d_example/pose/new@=person_whole_front_keypoints.json
```

for dress

resize the cloth

```
"""#for dress"""

from PIL import Image
img = Image.open("/content/drive/MyDrive/task/input/input_cloth.jpg")
# Resize cloth image
img = img.resize((320,512), Image.BICUBIC).convert('RGB')
img.save("/content/drive/MyDrive/task/mpv3d_example/cloth/10=cloth_front.jpg")
```

```
import matplotlib.pyplot as plt
from PIL import Image
plt.imshow(img)
```

```
plt.axis('off')
plt.show()
```

mask image

```
import cv2
aa = cv2.imread("/content/drive/MyDrive/task/mpv3d_example/cloth/10=cloth_front.jpg",0)
from matplotlib import pyplot as plt

def showimage(myimage):
    if (myimage.ndim > 2):  # This only applies to RGB or RGBA images (e.g. not to Black and White images)
        myimage = myimage[:, :, ::-1]  # OpenCV follows BGR order, while matplotlib likely follows RGB order

    fig, ax = plt.subplots(figsize=[10, 10])
    ax.imshow(myimage, cmap='gray', interpolation='bicubic')
    plt.xticks([]), plt.yticks([])  # to hide tick values on X and Y axis
    plt.show()

threshold = 225
assignvalue = 255  # Value to assign the pixel if the threshold is met
threshold_method = cv2.THRESH_BINARY


threshold_method = cv2.THRESH_BINARY_INV
_, result = cv2.threshold(aa, threshold, assignvalue, threshold_method)


showimage(result)

cv2.imwrite("/content/drive/MyDrive/task/mpv3d_example/cloth-mask/10=cloth_front_mask.jpg",result)
```

Dress alignment process ---

```
!python /content/drive/MyDrive/task/util/data_preprocessing.py --MPV3D_root /content/drive/MyDrive/task/mpv3d_example

    0it [00:00, ?it/s]
    1it [00:00, 18.63it/s]
    clothes pre-alignment done and saved to /content/drive/MyDrive/task/mpv3d_example/aligned/test_pairs/cloth!
    100% 1/1 [00:00<00:00, 34.69it/s]
    palms segmentaion done and saved to /content/drive/MyDrive/task/mpv3d_example/palm-mask!
    100% 1/1 [00:00<00:00, 16.72it/s]
    Getting image sobel done and saving to /content/drive/MyDrive/task/mpv3d_example/image-sobel!
    ******Data preprocessing done!******
```

add the pretrained model.

```
!python /content/drive/MyDrive/task/test.py --model MTM --name MTM --dataroot /content/drive/MyDrive/task/mpv3d_example --data

    ----------------- Options ---------------
                   add_depth: True
              add_grid_loss: False
                  add_segmt: True
             add_theta_loss: False
                    add_tps: True
               aspect_ratio: 1.0
                 batch_size: 8
            checkpoints_dir: /content/drive/MyDrive/task/pretrained
                   datalist: test_pairs                          [default: /content/drive/MyDrive/task/mpv3d_example/tes
                   datamode: aligned
                   dataroot: /content/drive/MyDrive/task/mpv3d_example
            display_winsize: 512
                      epoch: latest
                       eval: False
                    gpu_ids: 0
                  grid_size: 3
                 img_height: 512
                  img_width: 320
                  init_gain: 0.02
                  init_type: normal
                 input_nc_A: 29
                 input_nc_B: 3
                    isTrain: False                             [default: None]
                lambda_depth: 1.0
                 lambda_grid: 1.0
                lambda_segmt: 1.0
                lambda_theta: 0.1
                 lambda_warp: 1.0
                   load_iter: 0                                [default: 0]
          max_dataset_size: inf
                      model: MTM
                 n_layers_D: 3
       n_layers_feat_extract: 3
                       name: MTM
```

```
                         ndf: 64
                        netD: basic
                         ngf: 64
               no_pin_memory: False
                        norm: instance
                    num_test: 10000
                 num_threads: 8
                       phase: test
                      radius: 5
                 results_dir: /content/drive/MyDrive/task/results
               save_depth_vis: False
              save_normal_vis: False
               save_segmt_vis: False
               serial_batches: False
                      suffix:
                 use_dropout: False
                     verbose: False
----------------- End -------------------
dataset [AlignedMPV3dDataset] was created
initialize network with normal
model [MTMModel] was created
```

```
!python /content/drive/MyDrive/task/test.py --model DRM --name DRM --dataroot /content/drive/MyDrive/task/mpv3d_example --data
```

```
----------------- Options ---------------
             add_gan_loss: False
            add_grad_loss: True
           add_normal_loss: False
              aspect_ratio: 1.0
                batch_size: 8
           checkpoints_dir: /content/drive/MyDrive/task/pretrained
                 datalist: test_pairs                          [default: /content/drive/MyDrive/task/mpv3d_example/te
                 datamode: aligned
                 dataroot: /content/drive/MyDrive/task/mpv3d_example
            display_ncols: 2
           display_winsize: 512
                    epoch: latest
                     eval: False
                  gpu_ids: 0
               img_height: 512
                img_width: 320
                init_gain: 0.02
                init_type: normal
            input_gradient: True
                  input_nc: 8
               input_nc_D: 4
                  isTrain: False                             [default: None]
              lambda_depth: 1.0
                lambda_gan: 1.0
               lambda_grad: 1.0
             lambda_normal: 1.5
                  load_iter: 0                               [default: 0]
          max_dataset_size: inf
                     model: DRM                              [default: MTM]
                n_layers_D: 3
                      name: DRM                              [default: MTM]
                       ndf: 64
                      netD: basic
                       ngf: 64
              no_pin_memory: False
                       norm: instance
                   num_test: 10000
                num_threads: 8
                  output_nc: 2
                      phase: test
                     radius: 5
                results_dir: /content/drive/MyDrive/task/results
             save_depth_vis: False
            save_normal_vis: False
             save_segmt_vis: False
             serial_batches: False
                      suffix:
                use_dropout: False
                    verbose: False
                   warproot: /content/drive/MyDrive/task/results/aligned/MTM/test_pairs
----------------- End -------------------
dataset [AlignedMPV3dDataset] was created
initialize network with normal
model [DRMModel] was created
loading the model from /content/drive/MyDrive/task/pretrained/aligned/DRM/latest_net_DRM.pth
---------- Networks initialized -------------
```

```
!python /content/drive/MyDrive/task/test.py --model TFM --name TFM --dataroot /content/drive/MyDrive/task/mpv3d_example --data
```

```
                batch_size: 8
            checkpoints_dir: /content/drive/MyDrive/task/pretrained
                   datalist: test_pairs                          [default: /content/drive/MyDrive/task/mpv3d_example/te
                   datamode: aligned
                   dataroot: /content/drive/MyDrive/task/mpv3d_example
            display_winsize: 512
                      epoch: latest
                       eval: False
                    gpu_ids: 0
                 img_height: 512
                  img_width: 320
                  init_gain: 0.02
                  init_type: normal
                input_depth: True
                   input_nc: 7
                 input_nc_D: 12
                input_segmt: True
                    isTrain: False                               [default: None]
                 lambda_gan: 1.0
                  lambda_l1: 1.0
                lambda_mask: 1.0
                 lambda_vgg: 1.0
                  load_iter: 0                                   [default: 0]
           max_dataset_size: inf
                      model: TFM                                 [default: MTM]
                 n_layers_D: 3
                       name: TFM                                 [default: MTM]
                        ndf: 64
                       netD: basic
                        ngf: 64
              no_pin_memory: False
                       norm: instance
                  num_downs: 6
                   num_test: 10000
                num_threads: 8
                  output_nc: 4
                      phase: test
                     radius: 5
                results_dir: /content/drive/MyDrive/task/results
              save_depth_vis: False
             save_normal_vis: False
              save_segmt_vis: False
              serial_batches: False
                      suffix:
                use_dropout: False
                    verbose: False
                   warproot: /content/drive/MyDrive/task/results/aligned/MTM/test_pairs
----------------- End -------------------
dataset [AlignedMPV3dDataset] was created
initialize network with normal
model [TFMModel] was created
loading the model from /content/drive/MyDrive/task/pretrained/aligned/TFM/latest_net_TFM.pth
---------- Networks initialized -------------
[Network TFM] Total number of parameters : 21.333 M
-----------------------------------------------
processing (0001)-th / (0001) image...
Testing TFM finished.
```

output image in 2d_form

```
from PIL import Image

image_path = '/content/drive/MyDrive/task/results/aligned/TFM/test_pairs/tryon/new@=person_whole_front.png'

# Read the image using PIL
image = Image.open(image_path)


import matplotlib.pyplot as plt

# Display the image using matplotlib
plt.imshow(image)
plt.axis('off')  # Remove the axis labels
plt.show()
```

To view 3d model

```
!python /content/drive/MyDrive/task/rgbd2pcd.py
```

```
100% 2/2 [00:00<00:00,  6.58it/s]
The unprojected point cloud file(s) are saved to /content/drive/MyDrive/task/results/aligned/pcd/test_pairs
```

```
!pip install trimesh
```

```
Collecting trimesh
  Downloading trimesh-3.22.3-py3-none-any.whl (682 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 682.3/682.3 kB 12.1 MB/s eta 0:00:00
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from trimesh) (1.22.4)
Installing collected packages: trimesh
Successfully installed trimesh-3.22.3
```

```
pip install pyglet==1.5.27
```

```
Collecting pyglet==1.5.27
  Downloading pyglet-1.5.27-py3-none-any.whl (1.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.1/1.1 MB 17.2 MB/s eta 0:00:00
Installing collected packages: pyglet
Successfully installed pyglet-1.5.27
```

```
import trimesh

# Load the .ply file
mesh = trimesh.load_mesh('/content/drive/MyDrive/task/results/aligned/pcd/test_pairs/new@=person.ply')

# Display the mesh
mesh.show()
```

it is visible in pychram and other IDLE,,bec it will lead crush the run time env in colab

bec .ply has huge face values

here i show screenshot images of 3d version output image

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
image_paths = [
```

```
    "/content/drive/MyDrive/task/Screenshot 2023-07-04 230213.png",
    "/content/drive/MyDrive/task/Screenshot 2023-07-04 230237.png",
    "/content/drive/MyDrive/task/Screenshot 2023-07-04 230303.png",
    "/content/drive/MyDrive/task/Screenshot 2023-07-04 230352.png"
]



for i, path in enumerate(image_paths):
    img = mpimg.imread(path)
    plt.subplot(2, 2, i + 1)  # Arrange images in a 2x2 grid
    plt.imshow(img)
    plt.axis("off")

plt.show()
```

Evaluation:

```
from skimage.metrics import structural_similarity as ssim
import cv2

# Load the original and generated avatar images
original_image_path = '/content/drive/MyDrive/task/input/input_person.png'
generated_image_path = '/content/drive/MyDrive/task/results/aligned/TFM/test_pairs/tryon/new@=person_whole_front.png'

original_image = cv2.imread(original_image_path, cv2.IMREAD_GRAYSCALE)
generated_image = cv2.imread(generated_image_path, cv2.IMREAD_GRAYSCALE)

# Calculate the SSIM
ssim_score = ssim(original_image, generated_image)

# Print the SSIM score
print(f"SSIM: {ssim_score}")
```

```
SSIM: 0.9007408958786737
```

```
import cv2

# Load the original and generated avatar images
original_image_path = '/content/drive/MyDrive/task/input/input_person.png'
generated_image_path = '/content/drive/MyDrive/task/results/aligned/TFM/test_pairs/tryon/new@=person_whole_front.png'

original_image = cv2.imread(original_image_path)
generated_image = cv2.imread(generated_image_path)

# Calculate the PSNR
psnr_score = cv2.PSNR(original_image, generated_image)

# Print the PSNR score
```

```
print(f"PSNR: {psnr_score}")
```

```
PSNR: 15.654851383034833
```

```
import torch
from PIL import Image
from torchvision import transforms
from torchvision.models.segmentation import deeplabv3_resnet50

# Load the original and generated avatar images
original_image_path = '/content/drive/MyDrive/task/input/input_person.png'
generated_image_path = '/content/drive/MyDrive/task/results/aligned/TFM/test_pairs/tryon/new@=person_whole_front.png'

# Load the pre-trained DeepLabV3 model
model = deeplabv3_resnet50(pretrained=True)
model.eval()

# Preprocess the images
preprocess = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

original_image = Image.open(original_image_path).convert("RGB")
generated_image = Image.open(generated_image_path).convert("RGB")

original_input = preprocess(original_image).unsqueeze(0)
generated_input = preprocess(generated_image).unsqueeze(0)

# Perform semantic segmentation on the original and generated images
original_output = model(original_input)['out']
generated_output = model(generated_input)['out']

# Calculate pixel-wise accuracy of the segmentation
original_pred = original_output.argmax(dim=1)
generated_pred = generated_output.argmax(dim=1)

correct_pixels = (original_pred == generated_pred).sum().item()
total_pixels = original_pred.numel()

accuracy = correct_pixels / total_pixels

# Print the segmentation accuracy
print(f"Segmentation Accuracy: {accuracy}")
```

```
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is depr
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum
  warnings.warn(msg)
Segmentation Accuracy: 0.976177978515625
```