```python
import numpy as np
import os
import PIL
import PIL.Image
import tensorflow as tf
import tensorflow_datasets as tfds
```

```
2025-04-18 18:37:24.321966: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:477] Unable to
register cuFFT factory: Attempting to register factory for plugin
cuFFT when one has already been registered
WARNING: All log messages before absl::InitializeLog() is called are
written to STDERR
E0000 00:00:1745001444.581454      31 cuda_dnn.cc:8310] Unable to
register cuDNN factory: Attempting to register factory for plugin
cuDNN when one has already been registered
E0000 00:00:1745001444.652073      31 cuda_blas.cc:1418] Unable to
register cuBLAS factory: Attempting to register factory for plugin
cuBLAS when one has already been registered
```

```python
import pathlib
dataset_url =
"/kaggle/input/tanishq-jewellery-dataset/Jewellery_Data/ring"
archive = tf.keras.utils.get_file(origin=dataset_url, extract=True)
data_dir = pathlib.Path(archive).with_suffix('')
```

First trial

Just look around the images visualise them and see how might it work.

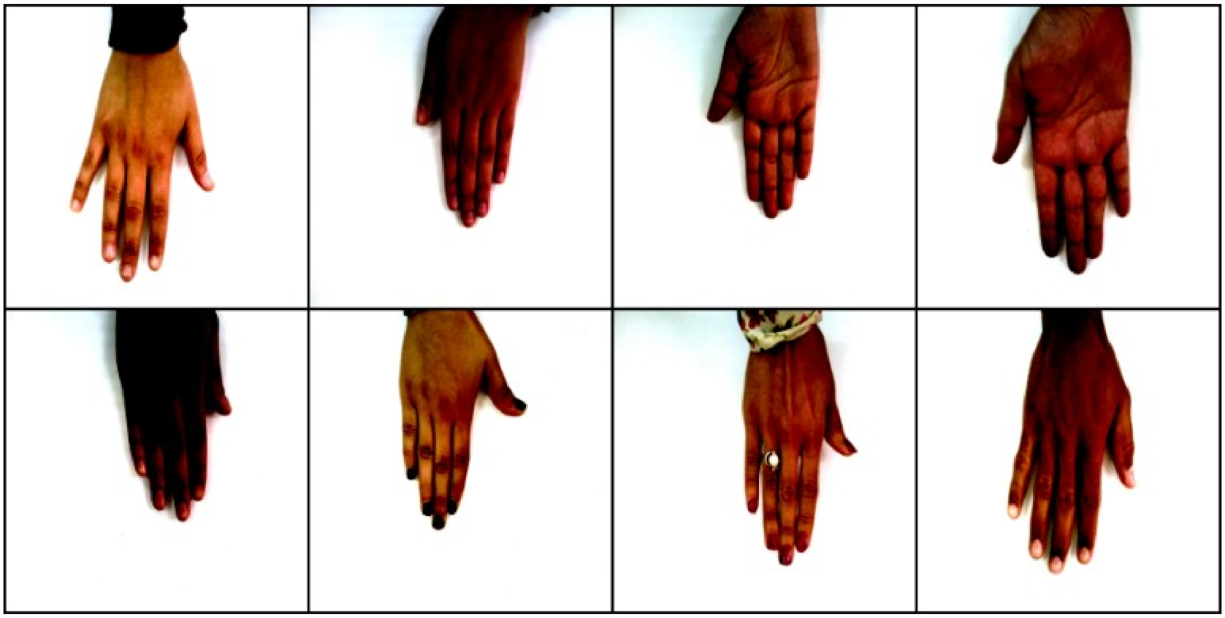
```python
import os
from PIL import Image
from torchvision import transforms
from torch.utils.data import Dataset, DataLoader

transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5], std=[0.5])  # Adjust if RGB
])


class ImageFolderDataset(Dataset):
    def __init__(self, image_folder, transform=None):
        self.image_folder = image_folder
        self.transform = transform
        self.image_paths = [os.path.join(image_folder, img) for img in
os.listdir(image_folder)
```

```
                            if img.lower().endswith(('png', 'jpg',
'jpeg'))]

    def __len__(self):
        return len(self.image_paths)

    def __getitem__(self, idx):
        img_path = self.image_paths[idx]
        image = Image.open(img_path).convert("RGB")
        if self.transform:
            image = self.transform(image)
        return image, img_path  # You can return more info here if
needed

# Load datasets
hand_dataset = ImageFolderDataset("/kaggle/input/hands-and-palm-
images-dataset/Hands/Hands", transform=transform)

# Create DataLoaders
hand_loader = DataLoader(hand_dataset, batch_size=8, shuffle=True)
```

Hand visualsing

Few images from the dataset that I choose

```
import matplotlib.pyplot as plt
import torchvision

def show_batch(loader):
    for images, paths in loader:
        grid = torchvision.utils.make_grid(images, nrow=4)
        plt.figure(figsize=(10, 5))
        plt.imshow(grid.permute(1, 2, 0))
        plt.axis('off')
        plt.show()
        break

show_batch(hand_loader)
```



```
train_transforms = transforms.Compose([
    transforms.Resize((256, 256)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
    transforms.ColorJitter(brightness=0.2, contrast=0.2,
saturation=0.2),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5]*3, std=[0.5]*3)
])
```

Preprocessing

Just a simple preprocessing step such as augmenting images which is not used at all further as this dataset was not completely useful I only selected few images.

```
class PreprocessedImageDataset(Dataset):
    def __init__(self, folder_path, transform=None):
        self.image_paths = [os.path.join(folder_path, f) for f in
os.listdir(folder_path)
                            if f.lower().endswith(('jpg', 'png',
'jpeg'))]
        self.transform = transform

    def __len__(self):
        return len(self.image_paths)

    def __getitem__(self, idx):
        img = Image.open(self.image_paths[idx]).convert("RGB")
```

```
        if self.transform:
            img = self.transform(img)
        return img

hand_dataset_1 = PreprocessedImageDataset("/kaggle/input/hands-and-
palm-images-dataset/Hands/Hands", transform=train_transforms)
```

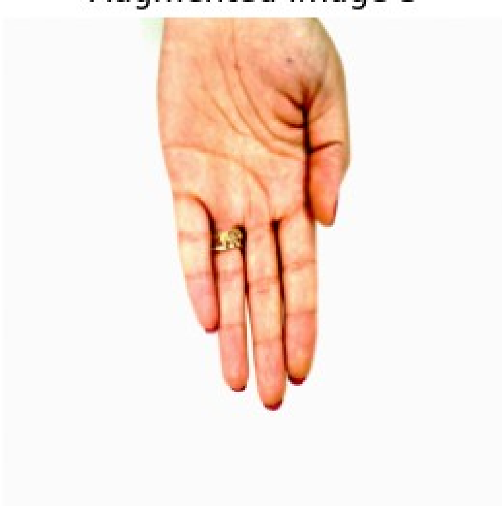
Visualize Augmented Images

```
import random
def visualize_augmented_images(folder_path, transform, num_images=5):
    image_paths = [os.path.join(folder_path, img) for img in
os.listdir(folder_path)
                   if img.lower().endswith(('jpg', 'jpeg', 'png'))]
    random.shuffle(image_paths)

    plt.figure(figsize=(15, 3 * num_images))
    for i in range(num_images):
        img_path = image_paths[i]
        img = Image.open(img_path).convert("RGB")
        transformed_img = transform(img)

        # Convert tensor to numpy image
        np_img = transformed_img.permute(1, 2, 0).numpy()
        plt.subplot(num_images, 1, i + 1)
        plt.imshow(np_img)
        plt.title(f"Augmented Image {i + 1}")
        plt.axis('off')
    plt.tight_layout()
    plt.show()

# Example usage:
visualize_augmented_images("/kaggle/input/hands-and-palm-images-
dataset/Hands/Hands", train_transforms, num_images=5)
```

Augmented Image 1

Augmented Image 2

Augmented Image 3



Segmentation

Then I modved to segmentation step I have skipped here some steps that involve selecting images from the dataset which were around 3179 images in total having some accesories, then I annotated them to train a UNet model.

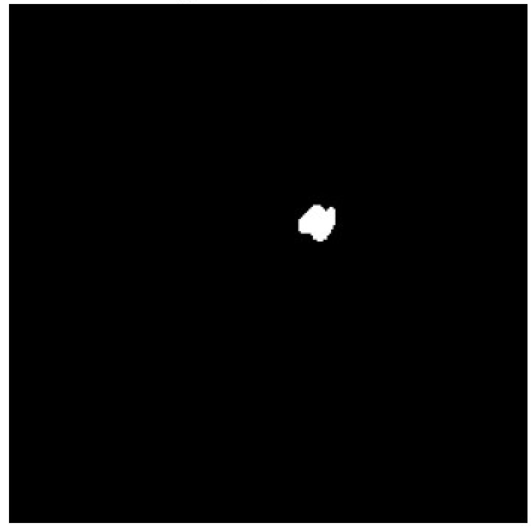
```
transform = transforms.Compose([
    transforms.Resize((256, 256)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5]*3, std=[0.5]*3)
])

dataset = PascalVOCDataset(

image_dir="/kaggle/input/hands-and-palm-images-dataset/Hands/Hands",
    mask_dir="/kaggle/input/segmented-masks/SegmentationClass",
    image_list_file="ImageSets/Segmentation/train.txt",
    transform=transform
)
```

Visulasing Annotations vs Images

```
import matplotlib.pyplot as plt

# Load a few samples
for i in range(3):
    img, mask = dataset[i]  # Load image and mask
    img_np = img.permute(1, 2, 0).numpy()  # Convert to HWC for
plotting
    mask_np = mask.squeeze().numpy()

    plt.figure(figsize=(10, 4))

    # Show image
    plt.subplot(1, 2, 1)
    plt.imshow((img_np * 0.5 + 0.5))  # Unnormalize
    plt.title("Hand Image")
    plt.axis("off")

    # Show mask
    plt.subplot(1, 2, 2)
    plt.imshow(mask_np, cmap="gray")
    plt.title("Jewelry Mask")
    plt.axis("off")

    plt.tight_layout()
    plt.show()
```

Hand Image

Jewelry Mask



Hand Image

Jewelry Mask

UNet

Here is the UNet Encoder-Decoder architectures using PyTorch. Model contains 4 convolution layers in the Encoder part, one bottleneck layer and 4 transpose convolution layers in decoder part. Finally, a sigmoid based output layer

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class UNet(nn.Module):
    # RGB images so channel = 3
    def __init__(self, in_channels=3, out_channels=1):
        super(UNet, self).__init__()

        def conv_block(in_c, out_c):
            return nn.Sequential(
                nn.Conv2d(in_c, out_c, 3, padding=1),
                nn.BatchNorm2d(out_c),
                nn.ReLU(inplace=True),
                nn.Conv2d(out_c, out_c, 3, padding=1),
                nn.BatchNorm2d(out_c),
                nn.ReLU(inplace=True),
            )

        self.enc1 = conv_block(3, 64)
        self.enc2 = conv_block(64, 128)
        self.enc3 = conv_block(128, 256)
        self.enc4 = conv_block(256, 512)
```

```
        self.pool = nn.MaxPool2d(2)

        self.bottleneck = conv_block(512, 1024)

        self.up4 = nn.ConvTranspose2d(1024, 512, kernel_size=2,
stride=2)
        self.dec4 = conv_block(1024, 512)

        self.up3 = nn.ConvTranspose2d(512, 256, kernel_size=2,
stride=2)
        self.dec3 = conv_block(512, 256)

        self.up2 = nn.ConvTranspose2d(256, 128, kernel_size=2,
stride=2)
        self.dec2 = conv_block(256, 128)

        self.up1 = nn.ConvTranspose2d(128, 64, kernel_size=2,
stride=2)
        self.dec1 = conv_block(128, 64)

        self.final = nn.Conv2d(64, 1, kernel_size=1)

    def forward(self, x):
        e1 = self.enc1(x)
        e2 = self.enc2(self.pool(e1))
        e3 = self.enc3(self.pool(e2))
        e4 = self.enc4(self.pool(e3))

        b = self.bottleneck(self.pool(e4))

        d4 = self.up4(b)
        d4 = torch.cat([d4, e4], dim=1)
        d4 = self.dec4(d4)

        d3 = self.up3(d4)
        d3 = torch.cat([d3, e3], dim=1)
        d3 = self.dec3(d3)

        d2 = self.up2(d3)
        d2 = torch.cat([d2, e2], dim=1)
        d2 = self.dec2(d2)

        d1 = self.up1(d2)
        d1 = torch.cat([d1, e1], dim=1)
        d1 = self.dec1(d1)

        return torch.sigmoid(self.final(d1))
```

Loss function

I used here DiceLoss which works well with the segmentation tasks

```
class DiceLoss(nn.Module):
    def __init__(self, smooth=1.0):
        super(DiceLoss, self).__init__()
        self.smooth = smooth

    def forward(self, preds, targets):
        preds = preds.view(-1)
        targets = targets.view(-1)
        intersection = (preds * targets).sum()
        dice = (2. * intersection + self.smooth) / (preds.sum() +
targets.sum() + self.smooth)
        return 1 - dice  # because we minimize the loss

bce = nn.BCELoss()
dice = DiceLoss()

def combined_loss(preds, targets):
    return bce(preds, targets) + dice(preds, targets)
```

Load the data in a pytorch dataloader

```
from torch.utils.data import DataLoader, random_split

# Optional: split into train/val
train_size = int(0.8 * len(dataset))
val_size = len(dataset) - train_size
train_dataset, val_dataset = random_split(dataset, [train_size,
val_size])

train_loader = DataLoader(train_dataset, batch_size=8, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=8)

model = UNet().cuda()
optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)

def train(model, loader, optimizer, epoch):
    model.train()
    total_loss = 0
    for images, masks in loader:
        images = images.cuda()
        masks = masks.cuda()

        preds = model(images)
        loss = combined_loss(preds, masks)
```

```
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        total_loss += loss.item()

    avg_loss = total_loss / len(loader)
    print(f"Epoch {epoch} - Training Loss: {avg_loss:.4f}")

def validate(model, loader):
    model.eval()
    total_loss = 0
    with torch.no_grad():
        for images, masks in loader:
            images = images.cuda()
            masks = masks.cuda()
            preds = model(images)
            loss = combined_loss(preds, masks)
            total_loss += loss.item()

    avg_loss = total_loss / len(loader)
    print(f"Validation Loss: {avg_loss:.4f}")
```

Model training,

I just did it for 10 epochs so that it can be possible in short amount of time with limited rerources however around 50 is recomendded.

```
for epoch in range(1, 11):  # 10 epochs to start
    train(model, train_loader, optimizer, epoch)
    validate(model, val_loader)
```

```
Epoch 1 - Training Loss: 1.1968
Validation Loss: 1.0631
Epoch 2 - Training Loss: 0.9938
Validation Loss: 0.9243
Epoch 3 - Training Loss: 0.8160
Validation Loss: 0.6954
Epoch 5 - Training Loss: 0.3208
Validation Loss: 0.2337
Epoch 6 - Training Loss: 0.2006
Validation Loss: 0.1695
Epoch 7 - Training Loss: 0.1360
Validation Loss: 0.1416
Epoch 8 - Training Loss: 0.0965
Validation Loss: 0.1292
Epoch 9 - Training Loss: 0.0897
Validation Loss: 0.0863
```

```
Epoch 10 - Training Loss: 0.0712
Validation Loss: 0.0619

torch.save(model.state_dict(), "/kaggle/working/jewelry_unet.pth")
```

Save and load the model

```
model = UNet().cuda()
model.load_state_dict(torch.load("/kaggle/working/jewelry_unet.pth"))
model.eval()
```

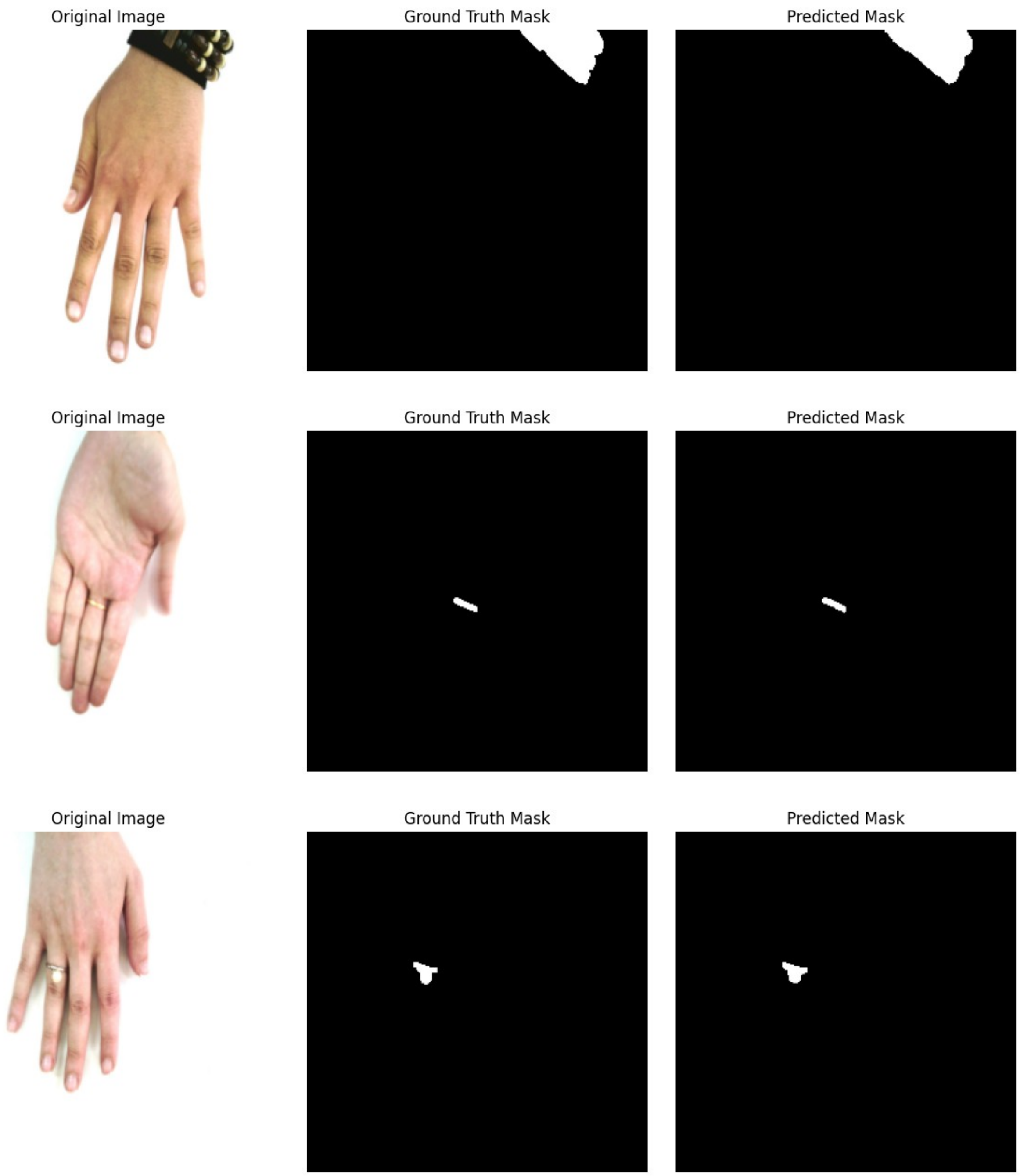
```
/tmp/ipykernel_31/4166248601.py:2: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.

model.load_state_dict(torch.load("/kaggle/working/jewelry_unet.pth"))

UNet(
  (enc1): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (5): ReLU(inplace=True)
  )
  (enc2): Sequential(
    (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU(inplace=True)
```

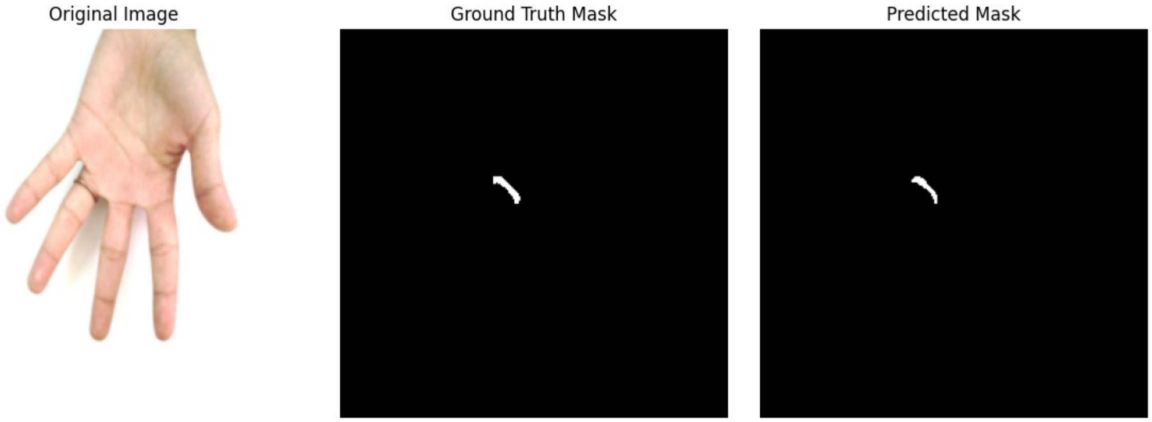
```
    (3): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (4): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (5): ReLU(inplace=True)
  )
  (enc3): Sequential(
    (0): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (5): ReLU(inplace=True)
  )
  (enc4): Sequential(
    (0): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (5): ReLU(inplace=True)
  )
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  (bottleneck): Sequential(
    (0): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (4): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (5): ReLU(inplace=True)
  )
  (up4): ConvTranspose2d(1024, 512, kernel_size=(2, 2), stride=(2, 2))
  (dec4): Sequential(
    (0): Conv2d(1024, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
```

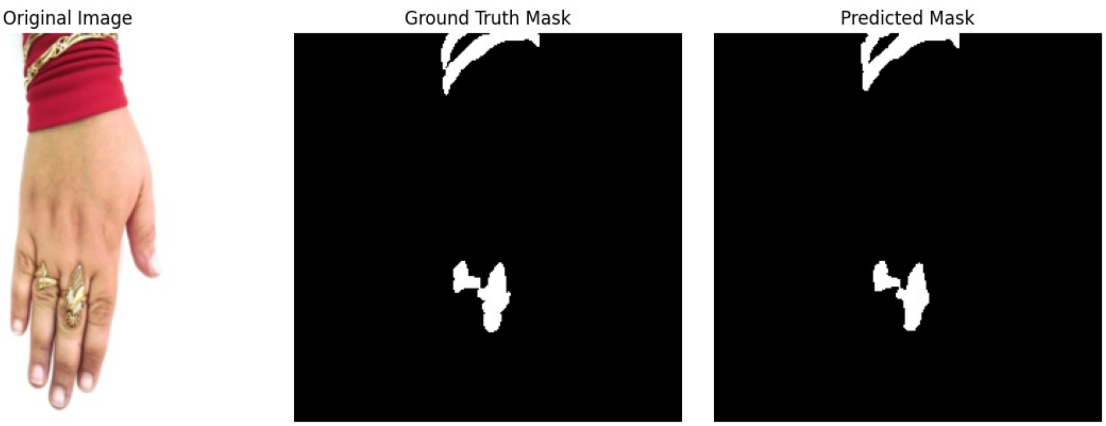
```
track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (5): ReLU(inplace=True)
  )
  (up3): ConvTranspose2d(512, 256, kernel_size=(2, 2), stride=(2, 2))
  (dec3): Sequential(
    (0): Conv2d(512, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (5): ReLU(inplace=True)
  )
  (up2): ConvTranspose2d(256, 128, kernel_size=(2, 2), stride=(2, 2))
  (dec2): Sequential(
    (0): Conv2d(256, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (4): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (5): ReLU(inplace=True)
  )
  (up1): ConvTranspose2d(128, 64, kernel_size=(2, 2), stride=(2, 2))
  (dec1): Sequential(
    (0): Conv2d(128, 64, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (5): ReLU(inplace=True)
  )
  (final): Conv2d(64, 1, kernel_size=(1, 1), stride=(1, 1))
)
```

Model Inference

```
import matplotlib.pyplot as plt

def visualize_predictions(model, loader, num_samples=3):
    model.eval()
    count = 0
    with torch.no_grad():
        for images, masks in loader:
            images = images.cuda()
            preds = model(images)
            preds = (preds > 0.5).float()  # thresholding

            for i in range(images.size(0)):
                if count >= num_samples:
                    return
                img = images[i].cpu().permute(1, 2, 0).numpy()
                true_mask = masks[i].cpu().squeeze().numpy()
                pred_mask = preds[i].cpu().squeeze().numpy()

                plt.figure(figsize=(12, 4))

                plt.subplot(1, 3, 1)
                plt.imshow((img * 0.5 + 0.5))  # unnormalize
                plt.title("Original Image")
                plt.axis("off")

                plt.subplot(1, 3, 2)
                plt.imshow(true_mask, cmap="gray")
                plt.title("Ground Truth Mask")
                plt.axis("off")

                plt.subplot(1, 3, 3)
                plt.imshow(pred_mask, cmap="gray")
                plt.title("Predicted Mask")
                plt.axis("off")

                plt.tight_layout()
                plt.show()
                count += 1

visualize_predictions(model, val_loader, num_samples=5)
```


Original Image
Ground Truth Mask
Predicted Mask
Original Image
Ground Truth Mask
Predicted Mask
Original Image
Ground Truth Mask
Predicted Mask



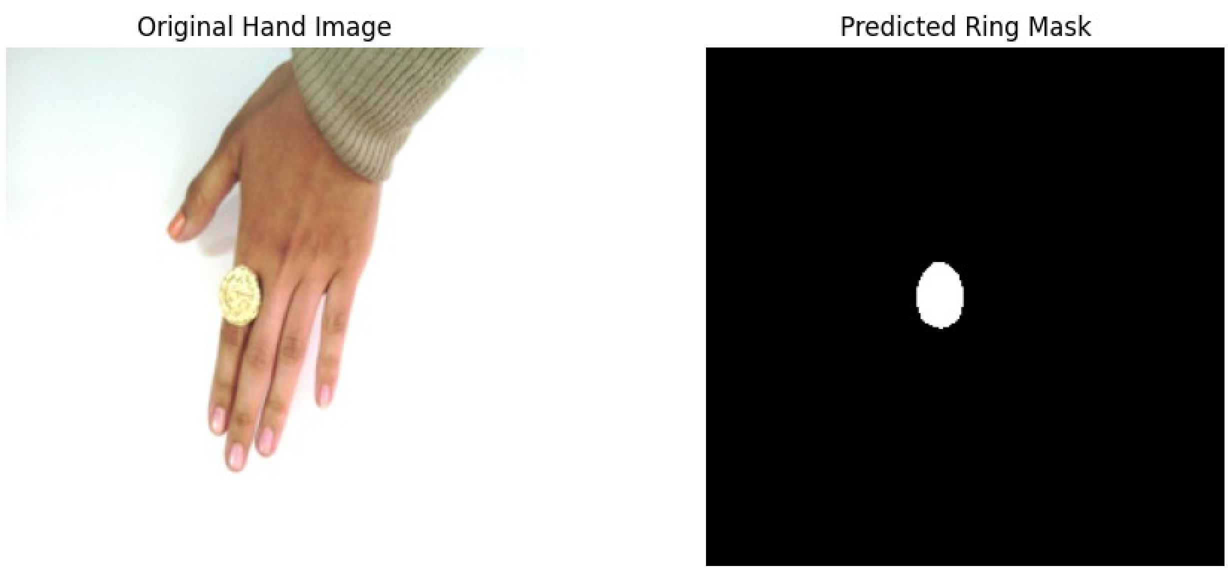


Ring detection

Now this model i tried to use to detect the rings and did the segmentation mask and image overlapping as well so that I can detect the ring.

```
from PIL import Image
import torchvision.transforms as transforms
import torch

# Replace with your image path
image_path = "/kaggle/input/hands-and-palm-images-dataset/Hands/Hands/Hand_0000403.jpg"

# Preprocessing (same as training)
transform = transforms.Compose([
    transforms.Resize((256, 256)),
    transforms.ToTensor(),
    transforms.Normalize([0.5]*3, [0.5]*3)
])
```

```python
image = Image.open(image_path).convert("RGB")
input_tensor = transform(image).unsqueeze(0).cuda()  # Add batch dim

model.eval()
with torch.no_grad():
    pred = model(input_tensor)
    pred_mask = (pred > 0.5).float().squeeze().cpu().numpy()

import matplotlib.pyplot as plt
import numpy as np

img_np = np.array(image.resize((256, 256)))

plt.figure(figsize=(10, 4))

# Original Image
plt.subplot(1, 2, 1)
plt.imshow(img_np)
plt.title("Original Hand Image")
plt.axis("off")

# Predicted Mask
plt.subplot(1, 2, 2)
plt.imshow(pred_mask, cmap="gray")
plt.title("Predicted Ring Mask")
plt.axis("off")

plt.tight_layout()
plt.show()
```



Ring tracking

But not detection just overlapping the segmentation mask and image

```
overlay = img_np.copy()
overlay[pred_mask > 0.5] = [200, 0, 0]  # Red overlay on detected regions

plt.imshow(overlay)
plt.title("Ring Detected Overlay")
plt.axis("off")
plt.show()
```

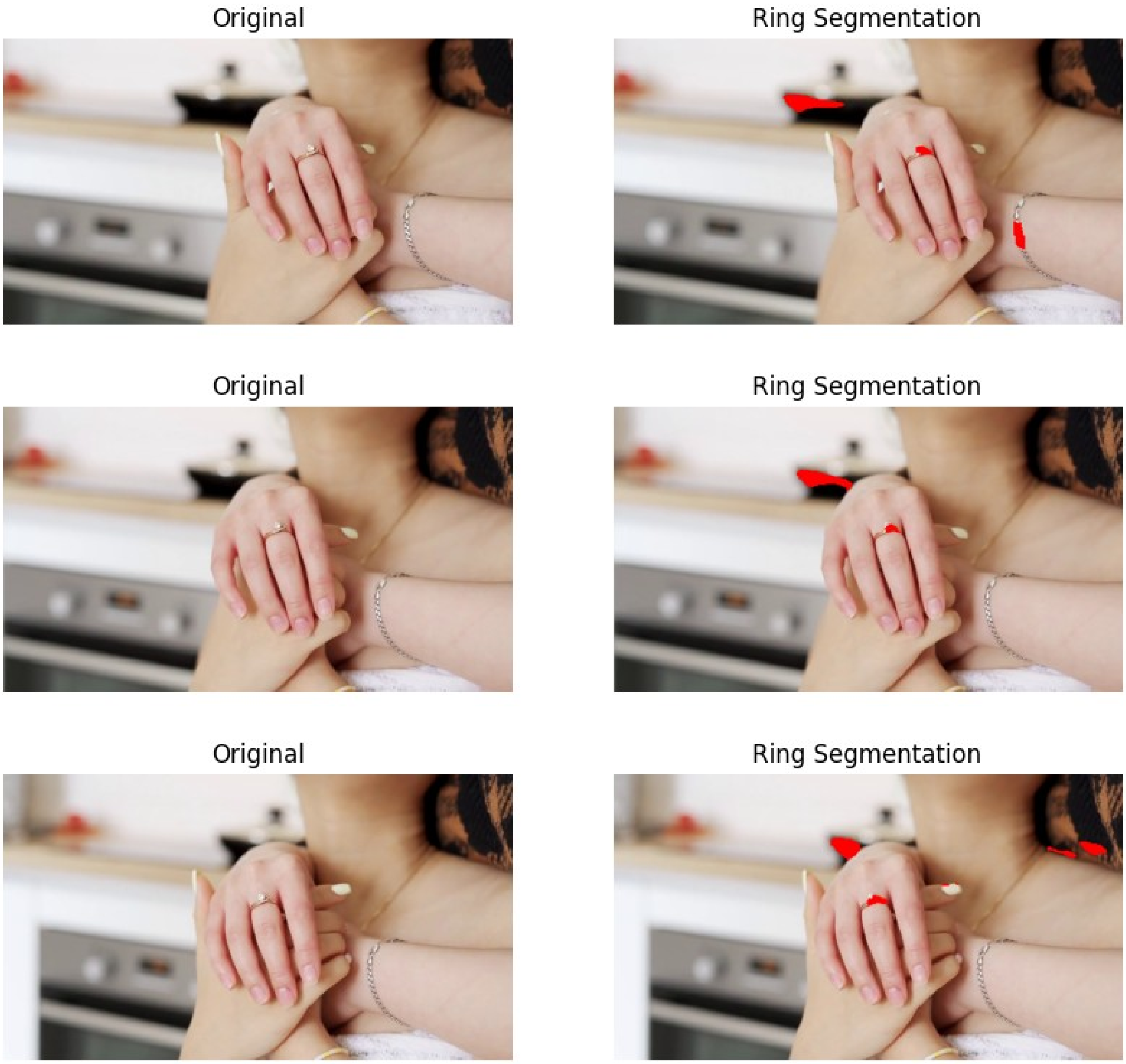
Video Data

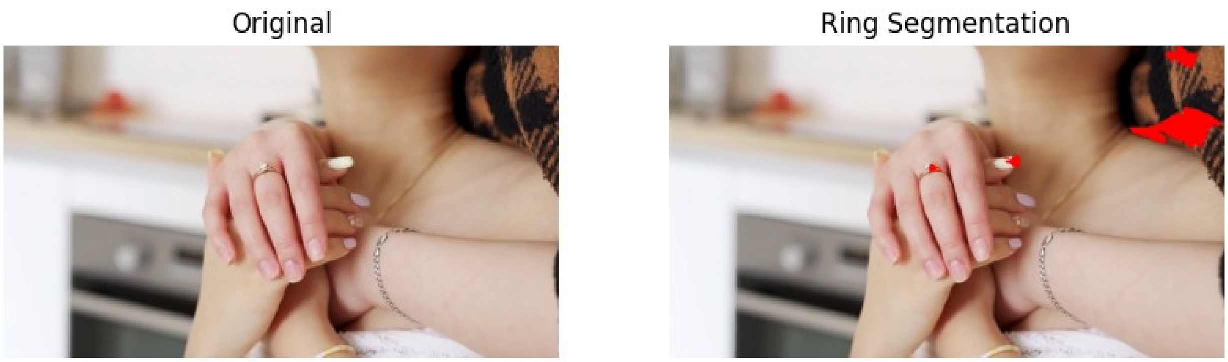
Also tried to use the model on image data but failed even to segment the video data which is reasonable because of the data on which I trained the model

```
import cv2
import torch
import numpy as np
from torchvision import transforms
from PIL import Image
```

```python
import matplotlib.pyplot as plt


transform = transforms.Compose([
    transforms.Resize((256, 256)),
    transforms.ToTensor(),
    transforms.Normalize([0.5]*3, [0.5]*3)
])

video_path = "/kaggle/input/short-video/219228_small.mp4"
cap = cv2.VideoCapture(video_path)

fourcc = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter("/kaggle/working/ring_output.avi", fourcc, 20.0,
(int(cap.get(3)), int(cap.get(4))))

frame_count = 0

while cap.isOpened() and frame_count < 100:  # Limit frames to speed
up Kaggle preview
    ret, frame = cap.read()
    if not ret:
        break

    img_pil = Image.fromarray(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
    input_tensor = transform(img_pil).unsqueeze(0).cuda()

    with torch.no_grad():
        pred = model(input_tensor)
        mask = (pred > 0.5).float().squeeze().cpu().numpy()

    mask_resized = cv2.resize(mask, (frame.shape[1], frame.shape[0]))
    overlay = frame.copy()
    overlay[mask_resized > 0.5] = [0, 0, 255]  # Red overlay

    out.write(overlay)

    if frame_count % 30 == 0:
        plt.figure(figsize=(10, 4))
        plt.subplot(1, 2, 1)
        plt.imshow(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
        plt.title("Original")
        plt.axis("off")

        plt.subplot(1, 2, 2)
        plt.imshow(cv2.cvtColor(overlay, cv2.COLOR_BGR2RGB))
        plt.title("Ring Segmentation")
        plt.axis("off")
        plt.show()
```

```
    frame_count += 1

cap.release()
out.release()
```





YOLO object detection

Then I moved to the yolo object detection option, in this part I just create bounding boxes around the segmentation masks of images to create a labels directory for YOLO object detection model

```
import cv2
import numpy as np
import os

image_path =
"/kaggle/input/hands-and-palm-images-dataset/Hands/Hands/Hand_0000021.
jpg"
mask_path =
"/kaggle/input/segmented-masks/SegmentationClass/Hand_0000021.png"

output_label_path = "/kaggle/working/labels/Hand_0000021.txt"
os.makedirs("/kaggle/working/labels", exist_ok=True)

def create_yolo_bbox_from_mask(mask_path, image_shape):
    mask = cv2.imread(mask_path, cv2.IMREAD_GRAYSCALE)
    if mask is None:
        print(f"Could not load mask: {mask_path}")
        return None

    contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    if contours:
        x, y, w, h = cv2.boundingRect(contours[0])
        img_h, img_w = image_shape[:2]

        x_center = (x + w / 2) / img_w
        y_center = (y + h / 2) / img_h
        norm_w = w / img_w
        norm_h = h / img_h

        return (x_center, y_center, norm_w, norm_h)
```

```python
    else:
        print(f"No contours found in mask")
    return None

image = cv2.imread(image_path)
if image is None:
    print(" Failed to read image.")
else:
    bbox = create_yolo_bbox_from_mask(mask_path, image.shape)
    if bbox:
        with open(output_label_path, "w") as f:
            f.write(f"0 {bbox[0]:.6f} {bbox[1]:.6f} {bbox[2]:.6f}
{bbox[3]:.6f}\n")
        print("Label file created:", output_label_path)
    else:
        print("No bounding box generated.")
```

```
 Label file created: /kaggle/working/labels/Hand_0000021.txt
```

A simple example

I first tried on single image if it is working well, and finally yeah it did create a fine bounding box on the ring.

```python
x1 = int((bbox[0] - bbox[2]/2) * image.shape[1])
y1 = int((bbox[1] - bbox[3]/2) * image.shape[0])
x2 = int((bbox[0] + bbox[2]/2) * image.shape[1])
y2 = int((bbox[1] + bbox[3]/2) * image.shape[0])

image_with_box = image.copy()
cv2.rectangle(image_with_box, (x1, y1), (x2, y2), (0, 255, 0), 2)

from matplotlib import pyplot as plt
plt.imshow(cv2.cvtColor(image_with_box, cv2.COLOR_BGR2RGB))
plt.title("Bounding Box from Mask")
plt.axis('off')
plt.show()
```

Bouding box for all images

```
import cv2
import numpy as np
import os


image_dir = "/kaggle/input/selected-hand-images/Hands"
mask_dir = "/kaggle/input/segmented-masks/SegmentationClass"
output_label_dir = "/kaggle/working/labels"
os.makedirs(output_label_dir, exist_ok=True)

def create_yolo_bbox_from_mask(mask_path, image_shape):
    mask = cv2.imread(mask_path, cv2.IMREAD_GRAYSCALE)
    if mask is None:
        print(f" Could not load mask: {mask_path}")
        return None

    contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    if contours:
        x, y, w, h = cv2.boundingRect(contours[0])
        img_h, img_w = image_shape[:2]
```

```
        x_center = (x + w / 2) / img_w
        y_center = (y + h / 2) / img_h
        norm_w = w / img_w
        norm_h = h / img_h

        return (x_center, y_center, norm_w, norm_h)
    else:
        print(f" No contours found in mask: {mask_path}")
    return None

for img_file in os.listdir(image_dir):
    img_name, _ = os.path.splitext(img_file)
    mask_path = os.path.join(mask_dir, img_name + ".png")
    image_path = os.path.join(image_dir, img_file)

    if not os.path.exists(mask_path):
        print(f" No mask for {img_file}")
        continue

    image = cv2.imread(image_path)
    if image is None:
        print(f" Could not read image: {image_path}")
        continue

    bbox = create_yolo_bbox_from_mask(mask_path, image.shape)
    if bbox:
        label_path = os.path.join(output_label_dir, img_name + ".txt")
        with open(label_path, "w") as f:
            f.write(f"0 {bbox[0]:.6f} {bbox[1]:.6f} {bbox[2]:.6f}
{bbox[3]:.6f}\n")
    else:
        print(f" No bbox for {img_file}")

!pip install ultralytics
```

```
Collecting ultralytics
  Downloading ultralytics-8.3.112-py3-none-any.whl.metadata (37 kB)
Requirement already satisfied: numpy<=2.1.1,>=1.23.0 in
/usr/local/lib/python3.11/dist-packages (from ultralytics) (1.26.4)
Requirement already satisfied: matplotlib>=3.3.0 in
/usr/local/lib/python3.11/dist-packages (from ultralytics) (3.7.5)
Requirement already satisfied: opencv-python>=4.6.0 in
/usr/local/lib/python3.11/dist-packages (from ultralytics) (4.11.0.86)
Requirement already satisfied: pillow>=7.1.2 in
/usr/local/lib/python3.11/dist-packages (from ultralytics) (11.1.0)
Requirement already satisfied: pyyaml>=5.3.1 in
/usr/local/lib/python3.11/dist-packages (from ultralytics) (6.0.2)
Requirement already satisfied: requests>=2.23.0 in
/usr/local/lib/python3.11/dist-packages (from ultralytics) (2.32.3)
Requirement already satisfied: scipy>=1.4.1 in
```

```
/usr/local/lib/python3.11/dist-packages (from ultralytics) (1.15.2)
Requirement already satisfied: torch>=1.8.0 in
/usr/local/lib/python3.11/dist-packages (from ultralytics)
(2.5.1+cu124)
Requirement already satisfied: torchvision>=0.9.0 in
/usr/local/lib/python3.11/dist-packages (from ultralytics)
(0.20.1+cu124)
Requirement already satisfied: tqdm>=4.64.0 in
/usr/local/lib/python3.11/dist-packages (from ultralytics) (4.67.1)
Requirement already satisfied: psutil in
/usr/local/lib/python3.11/dist-packages (from ultralytics) (7.0.0)
Requirement already satisfied: py-cpuinfo in
/usr/local/lib/python3.11/dist-packages (from ultralytics) (9.0.0)
Requirement already satisfied: pandas>=1.1.4 in
/usr/local/lib/python3.11/dist-packages (from ultralytics) (2.2.3)
Requirement already satisfied: seaborn>=0.11.0 in
/usr/local/lib/python3.11/dist-packages (from ultralytics) (0.12.2)
Collecting ultralytics-thop>=2.0.0 (from ultralytics)
  Downloading ultralytics_thop-2.0.14-py3-none-any.whl.metadata (9.4
kB)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.11/dist-packages (from matplotlib>=3.3.0-
>ultralytics) (1.3.1)
Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.11/dist-packages (from matplotlib>=3.3.0-
>ultralytics) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.11/dist-packages (from matplotlib>=3.3.0-
>ultralytics) (4.56.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.11/dist-packages (from matplotlib>=3.3.0-
>ultralytics) (1.4.8)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.11/dist-packages (from matplotlib>=3.3.0-
>ultralytics) (24.2)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.11/dist-packages (from matplotlib>=3.3.0-
>ultralytics) (3.2.1)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.11/dist-packages (from matplotlib>=3.3.0-
>ultralytics) (2.9.0.post0)
Requirement already satisfied: mkl_fft in
/usr/local/lib/python3.11/dist-packages (from numpy<=2.1.1,>=1.23.0-
>ultralytics) (1.3.8)
Requirement already satisfied: mkl_random in
/usr/local/lib/python3.11/dist-packages (from numpy<=2.1.1,>=1.23.0-
>ultralytics) (1.2.4)
Requirement already satisfied: mkl_umath in
/usr/local/lib/python3.11/dist-packages (from numpy<=2.1.1,>=1.23.0-
```

```
>ultralytics) (0.1.1)
Requirement already satisfied: mkl in /usr/local/lib/python3.11/dist-
packages (from numpy<=2.1.1,>=1.23.0->ultralytics) (2025.1.0)
Requirement already satisfied: tbb4py in
/usr/local/lib/python3.11/dist-packages (from numpy<=2.1.1,>=1.23.0-
>ultralytics) (2022.1.0)
Requirement already satisfied: mkl-service in
/usr/local/lib/python3.11/dist-packages (from numpy<=2.1.1,>=1.23.0-
>ultralytics) (2.4.1)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.11/dist-packages (from pandas>=1.1.4-
>ultralytics) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in
/usr/local/lib/python3.11/dist-packages (from pandas>=1.1.4-
>ultralytics) (2025.2)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.11/dist-packages (from requests>=2.23.0-
>ultralytics) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.11/dist-packages (from requests>=2.23.0-
>ultralytics) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.11/dist-packages (from requests>=2.23.0-
>ultralytics) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.11/dist-packages (from requests>=2.23.0-
>ultralytics) (2025.1.31)
Requirement already satisfied: filelock in
/usr/local/lib/python3.11/dist-packages (from torch>=1.8.0-
>ultralytics) (3.18.0)
Requirement already satisfied: typing-extensions>=4.8.0 in
/usr/local/lib/python3.11/dist-packages (from torch>=1.8.0-
>ultralytics) (4.13.1)
Requirement already satisfied: networkx in
/usr/local/lib/python3.11/dist-packages (from torch>=1.8.0-
>ultralytics) (3.4.2)
Requirement already satisfied: jinja2 in
/usr/local/lib/python3.11/dist-packages (from torch>=1.8.0-
>ultralytics) (3.1.6)
Requirement already satisfied: fsspec in
/usr/local/lib/python3.11/dist-packages (from torch>=1.8.0-
>ultralytics) (2025.3.2)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.4.127 in
/usr/local/lib/python3.11/dist-packages (from torch>=1.8.0-
>ultralytics) (12.4.127)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.4.127
in /usr/local/lib/python3.11/dist-packages (from torch>=1.8.0-
>ultralytics) (12.4.127)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.4.127 in
```

```
/usr/local/lib/python3.11/dist-packages (from torch>=1.8.0-
>ultralytics) (12.4.127)
Collecting nvidia-cudnn-cu12==9.1.0.70 (from torch>=1.8.0-
>ultralytics)
  Downloading nvidia_cudnn_cu12-9.1.0.70-py3-none-
manylinux2014_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-cublas-cu12==12.4.5.8 (from torch>=1.8.0-
>ultralytics)
  Downloading nvidia_cublas_cu12-12.4.5.8-py3-none-
manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cufft-cu12==11.2.1.3 (from torch>=1.8.0-
>ultralytics)
  Downloading nvidia_cufft_cu12-11.2.1.3-py3-none-
manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-curand-cu12==10.3.5.147 (from torch>=1.8.0-
>ultralytics)
  Downloading nvidia_curand_cu12-10.3.5.147-py3-none-
manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cusolver-cu12==11.6.1.9 (from torch>=1.8.0-
>ultralytics)
  Downloading nvidia_cusolver_cu12-11.6.1.9-py3-none-
manylinux2014_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-cusparse-cu12==12.3.1.170 (from torch>=1.8.0-
>ultralytics)
  Downloading nvidia_cusparse_cu12-12.3.1.170-py3-none-
manylinux2014_x86_64.whl.metadata (1.6 kB)
Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in
/usr/local/lib/python3.11/dist-packages (from torch>=1.8.0-
>ultralytics) (2.21.5)
Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in
/usr/local/lib/python3.11/dist-packages (from torch>=1.8.0-
>ultralytics) (12.4.127)
Collecting nvidia-nvjitlink-cu12==12.4.127 (from torch>=1.8.0-
>ultralytics)
  Downloading nvidia_nvjitlink_cu12-12.4.127-py3-none-
manylinux2014_x86_64.whl.metadata (1.5 kB)
Requirement already satisfied: triton==3.1.0 in
/usr/local/lib/python3.11/dist-packages (from torch>=1.8.0-
>ultralytics) (3.1.0)
Requirement already satisfied: sympy==1.13.1 in
/usr/local/lib/python3.11/dist-packages (from torch>=1.8.0-
>ultralytics) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in
/usr/local/lib/python3.11/dist-packages (from sympy==1.13.1-
>torch>=1.8.0->ultralytics) (1.3.0)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.7-
>matplotlib>=3.3.0->ultralytics) (1.17.0)
Requirement already satisfied: MarkupSafe>=2.0 in
```

```
/usr/local/lib/python3.11/dist-packages (from jinja2->torch>=1.8.0-
>ultralytics) (3.0.2)
Requirement already satisfied: intel-openmp<2026,>=2024 in
/usr/local/lib/python3.11/dist-packages (from mkl-
>numpy<=2.1.1,>=1.23.0->ultralytics) (2024.2.0)
Requirement already satisfied: tbb==2022.* in
/usr/local/lib/python3.11/dist-packages (from mkl-
>numpy<=2.1.1,>=1.23.0->ultralytics) (2022.1.0)
Requirement already satisfied: tcmlib==1.* in
/usr/local/lib/python3.11/dist-packages (from tbb==2022.*->mkl-
>numpy<=2.1.1,>=1.23.0->ultralytics) (1.2.0)
Requirement already satisfied: intel-cmplr-lib-rt in
/usr/local/lib/python3.11/dist-packages (from mkl_umath-
>numpy<=2.1.1,>=1.23.0->ultralytics) (2024.2.0)
Requirement already satisfied: intel-cmplr-lib-ur==2024.2.0 in
/usr/local/lib/python3.11/dist-packages (from intel-
openmp<2026,>=2024->mkl->numpy<=2.1.1,>=1.23.0->ultralytics)
(2024.2.0)
Downloading ultralytics-8.3.112-py3-none-any.whl (981 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 981.3/981.3 kB 18.2 MB/s eta
0:00:00a 0:00:01
anylinux2014_x86_64.whl (363.4 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 363.4/363.4 MB 4.4 MB/s eta
0:00:000:00:0100:01
anylinux2014_x86_64.whl (664.8 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 664.8/664.8 MB 1.9 MB/s eta
0:00:000:00:0100:01
anylinux2014_x86_64.whl (211.5 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 211.5/211.5 MB 8.0 MB/s eta
0:00:000:00:0100:01
anylinux2014_x86_64.whl (56.3 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.3/56.3 MB 30.0 MB/s eta
0:00:00:00:0100:01
anylinux2014_x86_64.whl (127.9 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 127.9/127.9 MB 13.5 MB/s eta
0:00:00:00:0100:01
anylinux2014_x86_64.whl (207.5 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 207.5/207.5 MB 8.2 MB/s eta
0:00:000:00:0100:01
anylinux2014_x86_64.whl (21.1 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.1/21.1 MB 67.9 MB/s eta
0:00:00:00:0100:01
pting uninstall: nvidia-nvjitlink-cu12
    Found existing installation: nvidia-nvjitlink-cu12 12.8.93
    Uninstalling nvidia-nvjitlink-cu12-12.8.93:
      Successfully uninstalled nvidia-nvjitlink-cu12-12.8.93
  Attempting uninstall: nvidia-curand-cu12
    Found existing installation: nvidia-curand-cu12 10.3.9.90
    Uninstalling nvidia-curand-cu12-10.3.9.90:
```

```
      Successfully uninstalled nvidia-curand-cu12-10.3.9.90
  Attempting uninstall: nvidia-cufft-cu12
    Found existing installation: nvidia-cufft-cu12 11.3.3.83
    Uninstalling nvidia-cufft-cu12-11.3.3.83:
      Successfully uninstalled nvidia-cufft-cu12-11.3.3.83
  Attempting uninstall: nvidia-cublas-cu12
    Found existing installation: nvidia-cublas-cu12 12.8.4.1
    Uninstalling nvidia-cublas-cu12-12.8.4.1:
      Successfully uninstalled nvidia-cublas-cu12-12.8.4.1
  Attempting uninstall: nvidia-cusparse-cu12
    Found existing installation: nvidia-cusparse-cu12 12.5.8.93
    Uninstalling nvidia-cusparse-cu12-12.5.8.93:
      Successfully uninstalled nvidia-cusparse-cu12-12.5.8.93
  Attempting uninstall: nvidia-cudnn-cu12
    Found existing installation: nvidia-cudnn-cu12 9.3.0.75
    Uninstalling nvidia-cudnn-cu12-9.3.0.75:
      Successfully uninstalled nvidia-cudnn-cu12-9.3.0.75
  Attempting uninstall: nvidia-cusolver-cu12
    Found existing installation: nvidia-cusolver-cu12 11.7.3.90
    Uninstalling nvidia-cusolver-cu12-11.7.3.90:
      Successfully uninstalled nvidia-cusolver-cu12-11.7.3.90
ERROR: pip's dependency resolver does not currently take into account
all the packages that are installed. This behaviour is the source of
the following dependency conflicts.
pylibcugraph-cu12 24.12.0 requires pylibraft-cu12==24.12.*, but you
have pylibraft-cu12 25.2.0 which is incompatible.
pylibcugraph-cu12 24.12.0 requires rmm-cu12==24.12.*, but you have
rmm-cu12 25.2.0 which is incompatible.
Successfully installed nvidia-cublas-cu12-12.4.5.8 nvidia-cudnn-cu12-
9.1.0.70 nvidia-cufft-cu12-11.2.1.3 nvidia-curand-cu12-10.3.5.147
nvidia-cusolver-cu12-11.6.1.9 nvidia-cusparse-cu12-12.3.1.170 nvidia-
nvjitlink-cu12-12.4.127 ultralytics-8.3.112 ultralytics-thop-2.0.14
```

I downloaded the labels to use on local machine

```
import shutil

zip_file_path = "/kaggle/working/labels.zip"

shutil.make_archive(zip_file_path.replace('.zip', ''), 'zip',
'/kaggle/working', 'labels')

print(f"Labels folder zipped and saved at: {zip_file_path}")
```

```
 Labels folder zipped and saved at: /kaggle/working/labels.zip
```

YOLO model training,

I did then trained a YOLOv8 model to detect the rings

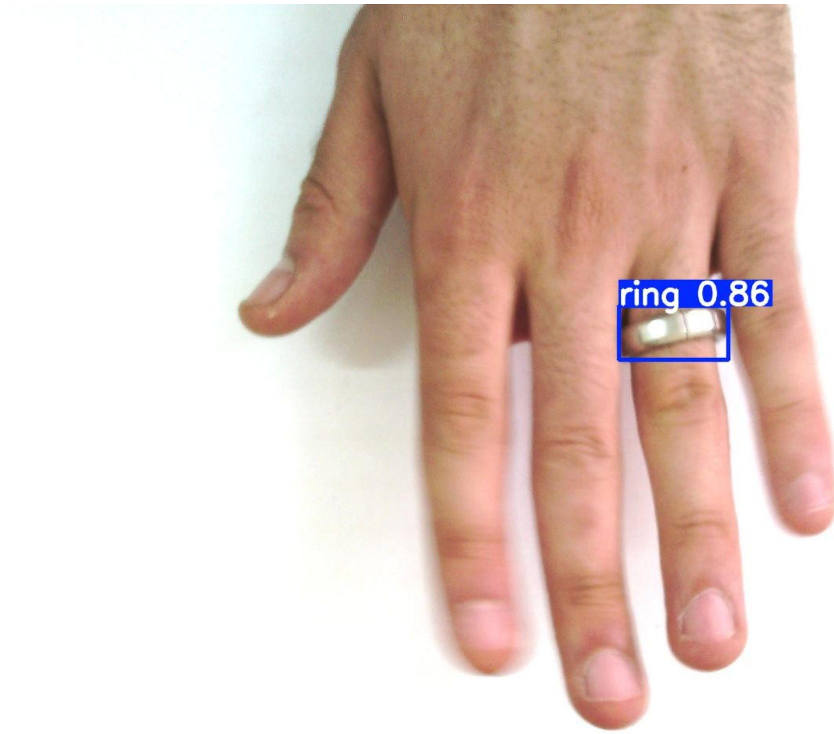
```
 from ultralytics import YOLO

 model = YOLO("yolov8n.yaml")

 # Train the model
 model.train(data="/kaggle/input/yolo-based-data/Hands/data.yaml",
epochs=50, batch=16, imgsz=640)

results = model.val()
print(results)
```

Final results

So, finally we can see some results where I can detect the ring on Anna's hand

```
img_path = '/kaggle/input/selected-hand-images/Hands/Hand_0000027.jpg'
results = model(img_path)

results[0].show()
```

```
image 1/1 /kaggle/input/selected-hand-images/Hands/Hand_0000027.jpg:
480x640 1 ring, 8.1ms
Speed: 3.3ms preprocess, 8.1ms inference, 1.3ms postprocess per image
at shape (1, 3, 480, 640)
```



Download the best model

```
import shutil

source_path = '/kaggle/working/runs/detect/train/weights/best.pt'  # Adjust based on where you found it
destination_path = '/kaggle/working/best.pt'

shutil.copy(source_path, destination_path)
print(" Model copied to:", destination_path)
```

```
 Model copied to: /kaggle/working/best.pt
```