Approach for Fine-Tuning DeepVTO Model from Huggingface

Our Goal of is to fine-tune the DeepVTO model to generate realistic and accurate virtual try-on images. This involves training the model on a dataset of train_images and train_target, enhancing the model's performance for our specific use case.

Model Selection

We have selected the DeepVTO model hosted on Hugging Face, which use deep learning techniques and architectures like as stable diffusion, DreamBooth, EfficientNetB3 for and OpenPose

About DeepVTO Model

DeepVTO model uses multiple techniques to provide a realistic virtual try-on experience.

Components of DeepVTO

- Stable Diffusion -For high-quality image generation.
- DreamBooth -Enhances the model's ability to generate realistic and personalized images.
- EfficientNetB3 -Used for feature extraction.
- OpenPose -Estimates person keypoints to align clothing items accurately.

```
import torch
from torch import nn
from torch.utils.data import DataLoader, Dataset
from torchvision import transforms
from PIL import Image
from transformers import AutoModel, AutoFeatureExtractor
```

Dataset Preparation

- Train_images -Individual images of clothing items.
- Train_targets -Images of models wearing clothing items.

Load Pre-Trained Model

Load DeepVTO model and feature extractor from Hugging Face

```
model_name = "gouthaml/raos-virtual-try-on-model" #DeepVTO Model
model = AutoModel.from_pretrained(model_name)
feature_extractor = AutoFeatureExtractor.from_pretrained(model_name)
```

Data Loading

Define custom dataset class to handle the loading and transformation of images.

Transformation

Transformations to resize, convert to tensors, normalize

Create Dataset and DataLoader

```
class FashionDataset(Dataset):
    def __init__(self, images, targets, transform=None):
        self.images = images
        self.targets = targets
        self.transform = transform

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

    def __getitem__(self, idx):
        image = Image.open(self.images[idx]).convert("RGB")
        target = Image.open(self.targets[idx]).convert("RGB")
        if self.transform:
            image = self.transform(image)
            target = self.transform(target)
        return image, target

# Path
train_images = ["path"]
train_targets = ["path"]
```

```
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])
train_dataset = FashionDataset(train_images, train_targets, transform=transform)
train_loader = DataLoader(train_dataset, batch_size=8, shuffle=True)
```

Model Architecture of Pre-Trained Model

DeepVTO model architecture is used, and a U-Net architecture is added for image generation.

Changes in Model (Fine-Tuning Based on Our Task)

Only architectural change made was the addition of U-Net component to the pre-trained DeepVTO model.

Why U Net added?

- U-Net Architecture is excellent for image-to-image translation, thus it best fit in our project
- Skip connection in U-Net retains fine details in generated image
- U-Net is ideal for generating precise, high-resolution images needed for virtual try-on
- It efficiently fine model by focusing on decoder with help of well trained future extraction

```
class CustomDeepVTOModel(nn.Module):
    def __init__(self, pretrained_model):
        super(CustomDeepVTOModel, self).__init__()
        self.base_model = pretrained_model
        self.unet = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=4, stride=2, padding=1),
            nn.ReLU(),
            nn.Conv2d(64, 128, kernel_size=4, stride=2, padding=1),
            nn.ReLU(),
            nn.Conv2d(128, 256, kernel_size=4, stride=2, padding=1),
            nn.ReLU(),
```

```
            nn.Conv2d(256, 512, kernel_size=4, stride=2, padding=1),
            nn.ReLU(),
            nn.ConvTranspose2d(512, 256, kernel_size=4, stride=2, padding=1),
            nn.ReLU(),
            nn.ConvTranspose2d(256, 128, kernel_size=4, stride=2, padding=1),
            nn.ReLU(),
            nn.ConvTranspose2d(128, 64, kernel_size=4, stride=2, padding=1),
            nn.ReLU(),
            nn.ConvTranspose2d(64, 3, kernel_size=4, stride=2, padding=1),
            nn.Tanh()
        )

    def forward(self, x):
        features = self.base_model(x).last_hidden_state
        features = features.permute(0, 2, 1).contiguous().view(features.size(0), 768, 14, 14)
        x = self.unet(features)
        return x
```

Initialize Custom Model

Custom model is initialized with DeepVTO model.

```
custom_model = CustomDeepVTOModel(model)
```

Training

Training loop with details needed

```
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(custom_model.parameters(), lr=1e-4)
num_epochs = 10

for epoch in range(num_epochs):
    custom_model.train()
    running_loss = 0.0
    for images, targets in train_loader:
        optimizer.zero_grad()
```

```
        inputs = feature_extractor(images, return_tensors="pt").pixel_values
        outputs = custom_model(inputs)
        loss = criterion(outputs, targets)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
    print(f'Epoch {epoch+1}, Loss: {running_loss/len(train_loader)}')
```

Hyperparameters tunning

- Check multiple hyperparameters and select optimal hyperparameters which will improve model performance

Saving Model for Further Use

Save the fine-tuned model for future use.

```
torch.save(custom_model.state_dict(), 'fine_tuned_deepvto_model.pth')
```