```
Start coding or generate with AI.
```

```
!pip install -q diffusers transformers accelerate peft
!pip install -q pycocotools opencv-python pillow tqdm
```

```
import os, requests
from pycocotools.coco import COCO
from tqdm import tqdm

# download annotations
if not os.path.exists("annotations"):
    !wget -q http://images.cocodataset.org/annotations/annotations_trainval2017.zip
    !unzip -q annotations_trainval2017.zip

coco = COCO("annotations/instances_train2017.json")

person_cat = coco.getCatIds(catNms=['person'])
img_ids = coco.getImgIds(catIds=person_cat)

os.makedirs("dataset/images", exist_ok=True)

# only 200 images (fast training)
selected = img_ids[:200]

for img_id in tqdm(selected):
    img = coco.loadImgs(img_id)[0]
    url = img["coco_url"]
    filename = f"dataset/images/{img_id}.jpg"

    try:
        r = requests.get(url, timeout=10)
        with open(filename, "wb") as f:
            f.write(r.content)
    except:
        pass
```

```
loading annotations into memory...
Done (t=21.47s)
creating index...
index created!
100%|██████████| 200/200 [01:29<00:00,  2.24it/s]
```

```
!pip install -q torchvision --no-deps
```

```
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.1/8.1 MB 50.8 MB/s eta 0:00:00
```

```
import torch
import torchvision.transforms as T
from torchvision.models.segmentation import deeplabv3_resnet50
from PIL import Image
import os
import numpy as np
from tqdm import tqdm

model = deeplabv3_resnet50(pretrained=True).eval().cuda()

transform = T.Compose([
    T.Resize(520),
    T.ToTensor(),
])

os.makedirs("dataset/masks", exist_ok=True)

for img_name in tqdm(os.listdir("dataset/images")):
    path = f"dataset/images/{img_name}"
    image = Image.open(path).convert("RGB")

    inp = transform(image).unsqueeze(0).cuda()
    with torch.no_grad():
        output = model(inp)["out"][0]

    mask = output.argmax(0).byte().cpu().numpy()

    # person class
    person_mask = (mask == 15).astype(np.uint8)*255

    Image.fromarray(person_mask).save(f"dataset/masks/{img_name.replace('.jpg','.png')}")
```

```
/usr/local/lib/python3.12/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or `No
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/deeplabv3_resnet50_coco-cd0a2569.pth" to /root/.cache/torch/hub/checkpoints/de
100%|██████████| 161M/161M [00:03<00:00, 48.1MB/s]
100%|██████████| 200/200 [00:43<00:00,  4.63it/s]
```

```
!pip uninstall -y huggingface_hub
!pip uninstall -y huggingface_hub

!pip install -q huggingface_hub==0.20.3
```

```
Found existing installation: huggingface-hub 0.20.3
Uninstalling huggingface-hub-0.20.3:
  Successfully uninstalled huggingface-hub-0.20.3
WARNING: Skipping huggingface_hub as it is not installed.
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the
sentence-transformers 5.2.2 requires transformers<6.0.0,>=4.41.0, but you have transformers 4.36.2 which is incompatible.
datasets 4.0.0 requires huggingface-hub>=0.24.0, but you have huggingface-hub 0.20.3 which is incompatible.
gradio 5.50.0 requires huggingface-hub<2.0,>=0.33.5, but you have huggingface-hub 0.20.3 which is incompatible.
```

```
import torch
from diffusers import StableDiffusionInpaintPipeline, DDPMScheduler
from peft import LoraConfig, get_peft_model
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms
from PIL import Image
import os
from tqdm import tqdm
import torch.nn.functional as F

device = "cuda"

# ---------------- DATASET ----------------
class BgDataset(Dataset):
    def __init__(self):
        self.files = os.listdir("dataset/images")

        self.img_t = transforms.Compose([
            transforms.Resize((512,512)),
            transforms.ToTensor(),
            transforms.Normalize([0.5],[0.5])
        ])

        self.mask_t = transforms.Compose([
            transforms.Resize((512,512)),
            transforms.ToTensor()
        ])

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

    def __getitem__(self, idx):
        name = self.files[idx]

        img = Image.open(f"dataset/images/{name}").convert("RGB")
        mask = Image.open(f"dataset/masks/{name.replace('.jpg','.png')}").convert("L")

        return self.img_t(img), self.mask_t(mask)

dataset = BgDataset()
loader = DataLoader(dataset, batch_size=1, shuffle=True)
```

```
# ---------------- MODEL ----------------
pipe = StableDiffusionInpaintPipeline.from_pretrained(
    "runwayml/stable-diffusion-inpainting",
    torch_dtype=torch.float16
).to(device)

pipe.enable_attention_slicing()

# keep VAE on CPU (VERY IMPORTANT FOR COLAB)
pipe.vae.to("cpu")

vae = pipe.vae
unet = pipe.unet
tokenizer = pipe.tokenizer
text_encoder = pipe.text_encoder

# freeze non-trained parts
vae.requires_grad_(False)
text_encoder.requires_grad_(False)

# ---------------- LoRA ----------------
lora_config = LoraConfig(
    r=4,
    lora_alpha=8,
    target_modules=["to_q","to_v"],
)

unet = get_peft_model(unet, lora_config)
unet.train()

optimizer = torch.optim.AdamW(unet.parameters(), lr=1e-6)
noise_scheduler = DDPMScheduler(num_train_timesteps=1000)

# text conditioning
text_input = tokenizer(
    ["a person"],
    padding="max_length",
    max_length=tokenizer.model_max_length,
    return_tensors="pt"
)

with torch.no_grad():
    encoder_hidden_states = text_encoder(text_input.input_ids.to(device))[0]

# ---------------- TRAINING ----------------
steps = 700
step = 0

for epoch in range(20):
    for batch in tqdm(loader):

        images, masks = batch
        images = images.to(device, dtype=torch.float16)
        masks = masks.to(device, dtype=torch.float16)

        # ---- KEEP TWO MASKS ----
        image_masks = masks                      # 512x512
        latent_masks = F.interpolate(masks, size=(64,64))   # 64x64

        # ---- VAE ENCODE ----
        vae.to(device)

        with torch.no_grad():
            latents = vae.encode(images).latent_dist.sample()
            latents = latents * 0.18215

        # remove background using IMAGE mask
        masked_images = images * (1 - image_masks)

        with torch.no_grad():
            masked_latents = vae.encode(masked_images).latent_dist.sample()
            masked_latents = masked_latents * 0.18215

        # move VAE back to CPU to free VRAM
        vae.to("cpu")
        torch.cuda.empty_cache()
        # ---- DIFFUSION PROCESS ----
        noise = torch.randn_like(latents)
        timesteps = torch.randint(0, 1000, (latents.shape[0],), device=device).long()
        noisy_latents = noise_scheduler.add_noise(latents, noise, timesteps)

        # 9-channel inpainting input
        latent_model_input = torch.cat([noisy_latents, latent_masks, masked_latents], dim=1)

        # UNet prediction
        noise_pred = unet(latent_model_input, timesteps, encoder_hidden_states).sample

        # diffusion loss
        loss = F.mse_loss(noise_pred, noise)

        loss.backward()
        optimizer.step()
        optimizer.zero_grad()

        step += 1

        if step % 50 == 0:
            print("step:", step, "loss:", loss.item())

        if step >= steps:
            break

    if step >= steps:
        break

torch.save(unet.state_dict(), "bg_lora.pt")
print("Training complete")
```

```
safety_checker/model.safetensors not found
Loading pipeline components...: 100%                                    7/7 [00:29<00:00,  4.53s/it]
`text_config_dict` is provided which will be used to initialize `CLIPTextConfig`. The value `text_config["id2label"]` will be ov
`text_config_dict` is provided which will be used to initialize `CLIPTextConfig`. The value `text_config["bos_token_id"]` will b
`text_config_dict` is provided which will be used to initialize `CLIPTextConfig`. The value `text_config["eos_token_id"]` will b
 25%|██▌       | 50/200 [00:36<02:01,  1.23it/s]step: 50 loss: nan
 50%|█████     | 100/200 [01:17<01:21,  1.22it/s]step: 100 loss: nan
 75%|███████▌  | 150/200 [01:59<00:42,  1.18it/s]step: 150 loss: nan
100%|██████████| 200/200 [02:40<00:00,  1.24it/s]
step: 200 loss: nan
 25%|██▌       | 50/200 [00:41<02:07,  1.17it/s]step: 250 loss: nan
 50%|█████     | 100/200 [01:23<01:25,  1.17it/s]step: 300 loss: nan
 75%|███████▌  | 150/200 [02:05<00:39,  1.26it/s]step: 350 loss: nan
100%|██████████| 200/200 [02:43<00:00,  1.22it/s]
step: 400 loss: nan
 25%|██▌       | 50/200 [00:41<02:07,  1.18it/s]step: 450 loss: nan
 50%|█████     | 100/200 [01:23<01:25,  1.18it/s]step: 500 loss: nan
 75%|███████▌  | 150/200 [02:05<00:42,  1.19it/s]step: 550 loss: nan
100%|██████████| 200/200 [02:47<00:00,  1.19it/s]
step: 600 loss: nan
 25%|██▌       | 50/200 [00:42<02:07,  1.18it/s]step: 650 loss: nan
 50%|████▉     | 99/200 [01:25<01:27,  1.16it/s]step: 700 loss: nan

Training complete
```

```
# ---------- Upload Image ----------
from google.colab import files
uploaded = files.upload()

import os
for name in uploaded.keys():
    os.rename(name, "test.jpg")

print("Image uploaded as test.jpg")

# ---------- Create Mask Automatically ----------
import torch
import torchvision.transforms as T
from torchvision.models.segmentation import deeplabv3_resnet50
from PIL import Image
import numpy as np

device = "cuda" if torch.cuda.is_available() else "cpu"

print("Loading segmentation model...")
```

```
seg_model = deeplabv3_resnet50(weights="DEFAULT").to(device).eval()

transform = T.Compose([
    T.Resize((384,384)),
    T.ToTensor(),
])

image = Image.open("test.jpg").convert("RGB")
original_size = image.size

inp = transform(image).unsqueeze(0).to(device)

with torch.no_grad():
    output = seg_model(inp)["out"][0]

mask = output.argmax(0).byte().cpu().numpy()
person_mask = (mask == 15).astype(np.uint8) * 255

mask_img = Image.fromarray(person_mask)
mask_img = mask_img.resize(original_size, Image.NEAREST)
mask_img.save("mask.png")

print("Mask generated!")

# ---------- Load Diffusion Model ----------
from diffusers import StableDiffusionInpaintPipeline

print("Loading inpainting model...")
pipe = StableDiffusionInpaintPipeline.from_pretrained(
    "runwayml/stable-diffusion-inpainting",
    torch_dtype=torch.float16
).to(device)

# load your trained LoRA weights
pipe.unet.load_state_dict(torch.load("bg_lora.pt"), strict=False)

# ---------- Generate New Background ----------
image = Image.open("test.jpg").resize((512,512))
mask = Image.open("mask.png").resize((512,512))

prompt = "a person standing in a luxury modern office interior, cinematic lighting, ultra realistic, DSLR photo"

print("Generating image...")

result = pipe(
    prompt=prompt,
    image=image,
    mask_image=mask,
    guidance_scale=7.5,
    num_inference_steps=30
).images[0]

result.save("output.png")

print("Done! Downloading output...")

files.download("output.png")
```

Choose Files 20240327_173733.jpg

20240327_173733.jpg(image/jpeg) - 3222528 bytes, last modified: 6/28/2025 - 100% done

```
Saving 20240327_173733.jpg to 20240327_173733.jpg
Image uploaded as test.jpg
Loading segmentation model...
Mask generated!
Loading inpainting model...
safety_checker/model.safetensors not found
Loading pipeline components...: 100%                                    7/7 [00:27<00:00,  4.25s/it]
`text_config_dict` is provided which will be used to initialize `CLIPTextConfig`. The value `text_config["id2label"]` will be ov
`text_config_dict` is provided which will be used to initialize `CLIPTextConfig`. The value `text_config["bos_token_id"]` will b
`text_config_dict` is provided which will be used to initialize `CLIPTextConfig`. The value `text_config["eos_token_id"]` will b
Generating image...
100%                                                   30/30 [00:04<00:00,  7.00it/s]
Done! Downloading output...
```

```python
from diffusers import StableDiffusionInpaintPipeline
import torch
from PIL import Image

pipe = StableDiffusionInpaintPipeline.from_pretrained(
    "runwayml/stable-diffusion-inpainting",
    torch_dtype=torch.float16
).to("cuda")

pipe.unet.load_state_dict(torch.load("bg_lora.pt"), strict=False)

image = Image.open("test.jpg").resize((512,512))
mask = Image.open("mask.png").resize((512,512))

prompt = "a person standing in a futuristic city at night, neon lighting, cinematic photography, ultra realistic"

result = pipe(
    prompt=prompt,
    image=image,
    mask_image=mask,
    guidance_scale=7.5,
    num_inference_steps=30
).images[0]

result.save("output.png")
```

```
safety_checker/model.safetensors not found
Loading pipeline components...: 100%                                   7/7 [00:28<00:00,  4.43s/it]
`text_config_dict` is provided which will be used to initialize `CLIPTextConfig`. The value `text_config["id2label"]` will be ov
`text_config_dict` is provided which will be used to initialize `CLIPTextConfig`. The value `text_config["bos_token_id"]` will b
`text_config_dict` is provided which will be used to initialize `CLIPTextConfig`. The value `text_config["eos_token_id"]` will b
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
/tmp/ipython-input-3930125616.py in <cell line: 0>()
     10 pipe.unet.load_state_dict(torch.load("bg_lora.pt"), strict=False)
     11
---> 12 image = Image.open("test.jpg").resize((512,512))
     13 mask = Image.open("mask.png").resize((512,512))
     14

/usr/local/lib/python3.12/dist-packages/PIL/Image.py in open(fp, mode, formats)
   3511     if is_path(fp):
   3512         filename = os.fspath(fp)
-> 3513         fp = builtins.open(filename, "rb")
   3514         exclusive_fp = True
   3515     else:

FileNotFoundError: [Errno 2] No such file or directory: 'test.jpg'
```

Next steps: Explain error