```
!pip install --upgrade diffusers transformers accelerate torch
```

Show hidden output

```
import torch
from diffusers import StableDiffusionPipeline
import os
from PIL import Image
import matplotlib.pyplot as plt

# --- CONFIGURATION ---
# 1. Setup the storage folder
output_folder = "synthetic_dataset"
os.makedirs(output_folder, exist_ok=True)

# 2. Select the pre-trained model (Stable Diffusion v1.5)
model_id = "runwayml/stable-diffusion-v1-5"

# 3. Check for GPU (CUDA) to speed up generation
device = "cuda" if torch.cuda.is_available() else "cpu"
print(f"Using device: {device}")

# --- LOAD MODEL ---
print("Loading pre-trained model...")
pipe = StableDiffusionPipeline.from_pretrained(model_id, torch_dtype=torch.float16 if device=="cuda" else torch.float32)
pipe = pipe.to(device)

# --- DEFINE PROMPTS ---
# Providing at least 5 input prompts as required
prompts = [
    "A futuristic city with flying cars and neon lights, cyberpunk style",
    "A serene lake surrounded by snowy mountains at sunset",
    "A cute astronaut cat exploring Mars, digital art",
    "A vintage steam train crossing a stone bridge in autumn",
    "A detailed portrait of a robot playing chess in a park"
]

# --- GENERATE & SAVE DATA ---
print(f"Generating synthetic data for {len(prompts)} prompts...")

generated_images = []

for i, prompt in enumerate(prompts):
    print(f"Generating sample {i+1}: {prompt}")

    # Generate the image
    image = pipe(prompt).images[0]

    # Save the output in the dataset folder
    file_name = f"sample_{i+1}.png"
    file_path = os.path.join(output_folder, file_name)
    image.save(file_path)

    generated_images.append((prompt, file_path, image))
    print(f"Saved to {file_path}")

print("\n--- Generation Complete ---\n")

# --- DISPLAY OUTPUTS ---
# Displaying sample outputs as required
plt.figure(figsize=(15, 10))
for i, (prompt, path, img) in enumerate(generated_images):
    plt.subplot(1, 5, i+1)
    plt.imshow(img)
    plt.axis("off")
    plt.title(f"Sample {i+1}")

plt.show()

print(f"All images saved in directory: '{output_folder}'")
```

```
Flax classes are deprecated and will be removed in Diffusers v1.0.0. We recommend migrating to PyTorch classes or pinning yc
Flax classes are deprecated and will be removed in Diffusers v1.0.0. We recommend migrating to PyTorch classes or pinning yc
Using device: cuda
Loading pre-trained model...
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
model_index.json: 100%                                                541/541 [00:00<00:00, 55.3kB/s]
Fetching 15 files: 100%                                               15/15 [01:18<00:00, 12.29s/it]
config.json: 100%                                                   617/617 [00:00<00:00, 4.90kB/s]
scheduler_config.json: 100%                                           308/308 [00:00<00:00, 2.65kB/s]
preprocessor_config.json: 100%                                        342/342 [00:00<00:00, 2.74kB/s]
merges.txt:         525k/? [00:00<00:00, 4.83MB/s]
special_tokens_map.json: 100%                                         472/472 [00:00<00:00, 6.15kB/s]
config.json:         4.72k/? [00:00<00:00, 63.0kB/s]
text_encoder/model.safetensors: 100%                                      492M/492M [01:17<00:00, 4.92MB/s]
safety_checker/model.safetensors: 100%                                    1.22G/1.22G [00:30<00:00, 36.2MB/s]
tokenizer_config.json: 100%                                           806/806 [00:00<00:00, 26.0kB/s]
vocab.json:         1.06M/? [00:00<00:00, 18.8MB/s]
config.json: 100%                                                   743/743 [00:00<00:00, 76.6kB/s]
config.json: 100%                                                   547/547 [00:00<00:00, 29.3kB/s]
vae/diffusion_pytorch_model.safetensors: 100%                                 335M/335M [00:44<00:00, 17.1MB/s]
unet/diffusion_pytorch_model.safetensors: 100%                                3.44G/3.44G [01:16<00:00, 40.6MB/s]
Loading pipeline components...: 100%                                      7/7 [00:24<00:00,  4.18s/it]
`torch_dtype` is deprecated! Use `dtype` instead!
Generating synthetic data for 5 prompts...
Generating sample 1: A futuristic city with flying cars and neon lights, cyberpunk style
100%                                          50/50 [00:07<00:00,  7.66it/s]
Saved to synthetic_dataset/sample_1.png
Generating sample 2: A serene lake surrounded by snowy mountains at sunset
100%                                          50/50 [00:06<00:00,  7.63it/s]
Saved to synthetic_dataset/sample_2.png
Generating sample 3: A cute astronaut cat exploring Mars, digital art
100%                                          50/50 [00:06<00:00,  7.51it/s]
Saved to synthetic_dataset/sample_3.png
Generating sample 4: A vintage steam train crossing a stone bridge in autumn
100%                                          50/50 [00:06<00:00,  7.43it/s]
Saved to synthetic_dataset/sample_4.png
Generating sample 5: A detailed portrait of a robot playing chess in a park
100%                                          50/50 [00:06<00:00,  7.31it/s]
Saved to synthetic_dataset/sample_5.png

--- Generation Complete ---
```

Sample 1 | Sample 2 | Sample 3 | Sample 4 | Sample 5

```
All images saved in directory: 'synthetic_dataset'
```

```
prompt = 'hello world in japan'
image = pipe(prompt).images[0]
image
```

```
100%                                                 50/50 [00:07<00:00,  7.37it/s]
```


```
x_ray_labels = ["Normal anatomy – healthy lungs, age & gender variations","Infectious patterns – bacterial/viral pneumonia,
# now we will create human x ray images for these specific locations with diseases meantioned, ans store them in a directory
```

```
output_folder = "x_rays"
os.makedirs(output_folder, exist_ok=True)
```

```
condition_list = [
    "Normal anatomy – healthy lungs, age & gender variations",
    "Infectious patterns – bacterial/viral pneumonia, COVID-like opacities",
    "Lung opacities – focal, diffuse, ground-glass, consolidations",
    "Pleural conditions – pleural effusion, pneumothorax",
    "Structural lesions – nodules, masses, fibrosis",
    "Cardiac findings – cardiomegaly, vascular congestion",
    "Medical devices – tubes, catheters, pacemakers",
    "Imaging artifacts – noise, motion blur, exposure issues",
    "View & positioning – PA/AP views, rotation, supine/erect",
    "Domain shift – scanner, hospital, and resolution variations"
]

# --- 2. THE BASE PROMPT (Constant Part) ---
# This ensures every image looks like an X-ray
base_prompt = "A high quality medical Chest X-Ray radiograph showing "

# --- GENERATION LOOP ---
generated_data = []

print(f"Starting generation for {len(condition_list)} conditions...\n")

for item in condition_list:
    # A. Create the dynamic prompt
    # Logic: Base Prompt + Specific Condition + Style Keywords
    final_prompt = f"{base_prompt} {item}, grayscale, black and white, bones, high contrast"

    # B. Create a clean filename from the list item
    # We take the part before the "-" to make the filename short (e.g., "Normal anatomy.png")
    short_name = item.split("–")[0].strip().replace(" ", "_").lower()
    filename = f"{short_name}.png"
```

```
    print(f"Prompt: {final_prompt}")

    # C. Generate
    image = pipe(
        final_prompt,
        negative_prompt="color, cartoon, 3d render, realistic skin, text, watermark, label"
    ).images[0] #

    # D. Save
    save_path = os.path.join(output_folder, filename)
    image.save(save_path) #
    generated_data.append((short_name, image))
    print(f"Saved: {save_path}\n")

print("--- Processing Complete ---")
```

```
Starting generation for 10 conditions...

Prompt: A high quality medical Chest X-Ray radiograph showing  Normal anatomy – healthy lungs, age & gender variations, gray
100%                                          50/50 [00:07<00:00,  6.89it/s]
Saved: x_rays/normal_anatomy.png

Prompt: A high quality medical Chest X-Ray radiograph showing  Infectious patterns – bacterial/viral pneumonia, COVID-like c
100%                                          50/50 [00:07<00:00,  6.82it/s]
Saved: x_rays/infectious_patterns.png

Prompt: A high quality medical Chest X-Ray radiograph showing  Lung opacities – focal, diffuse, ground-glass, consolidations
100%                                          50/50 [00:07<00:00,  6.67it/s]
Potential NSFW content was detected in one or more images. A black image will be returned instead. Try again with a differer
Saved: x_rays/lung_opacities.png

Prompt: A high quality medical Chest X-Ray radiograph showing  Pleural conditions – pleural effusion, pneumothorax, grayscal
100%                                          50/50 [00:07<00:00,  6.72it/s]
Potential NSFW content was detected in one or more images. A black image will be returned instead. Try again with a differer
Saved: x_rays/pleural_conditions.png

Prompt: A high quality medical Chest X-Ray radiograph showing  Structural lesions – nodules, masses, fibrosis, grayscale, bl
100%                                          50/50 [00:07<00:00,  6.82it/s]
Potential NSFW content was detected in one or more images. A black image will be returned instead. Try again with a differer
Saved: x_rays/structural_lesions.png

Prompt: A high quality medical Chest X-Ray radiograph showing  Cardiac findings – cardiomegaly, vascular congestion, graysca
100%                                          50/50 [00:07<00:00,  6.83it/s]
Potential NSFW content was detected in one or more images. A black image will be returned instead. Try again with a differer
Saved: x_rays/cardiac_findings.png

Prompt: A high quality medical Chest X-Ray radiograph showing  Medical devices – tubes, catheters, pacemakers, grayscale, bl
100%                                          50/50 [00:07<00:00,  6.93it/s]
Potential NSFW content was detected in one or more images. A black image will be returned instead. Try again with a differer
Saved: x_rays/medical_devices.png

Prompt: A high quality medical Chest X-Ray radiograph showing  Imaging artifacts – noise, motion blur, exposure issues, gray
100%                                          50/50 [00:07<00:00,  6.95it/s]
Potential NSFW content was detected in one or more images. A black image will be returned instead. Try again with a differer
Saved: x_rays/imaging_artifacts.png

Prompt: A high quality medical Chest X-Ray radiograph showing  View & positioning – PA/AP views, rotation, supine/erect, gra
100%                                          50/50 [00:07<00:00,  6.95it/s]
Potential NSFW content was detected in one or more images. A black image will be returned instead. Try again with a differer
Saved: x_rays/view_&_positioning.png

Prompt: A high quality medical Chest X-Ray radiograph showing  Domain shift – scanner, hospital, and resolution variations,
100%                                          50/50 [00:07<00:00,  6.99it/s]
Saved: x_rays/domain_shift.png

--- Processing Complete ---
```

Start coding or generate with AI.

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

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

```
dataset_root = "medical_xray_dataset_resnet"  # Main dataset folder
num_images_per_class = 2  # How many images to generate per disease
os.makedirs(dataset_root, exist_ok=True)
```

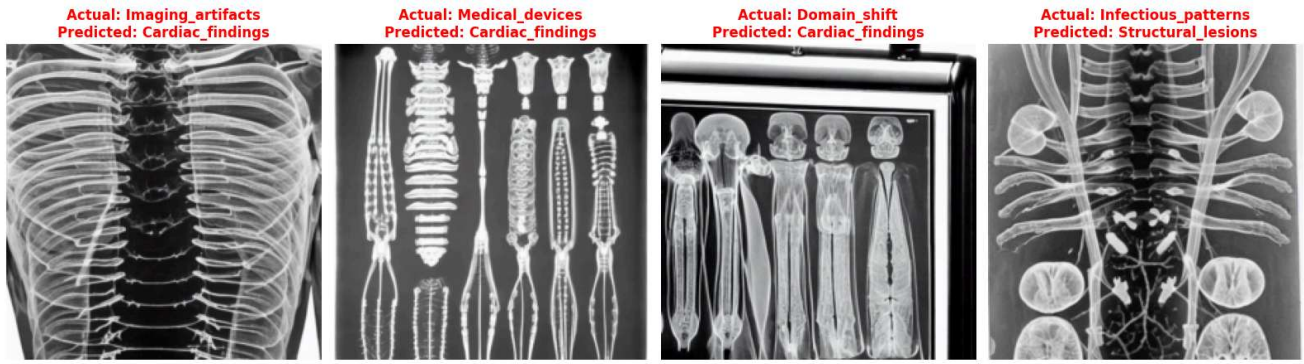
```
condition_list = [
    "Normal anatomy – healthy lungs, age & gender variations",
    "Infectious patterns – bacterial/viral pneumonia, COVID-like opacities",
    "Lung opacities – focal, diffuse, ground-glass, consolidations",
    "Pleural conditions – pleural effusion, pneumothorax",
    "Structural lesions – nodules, masses, fibrosis",
    "Cardiac findings – cardiomegaly, vascular congestion",
    "Medical devices – tubes, catheters, pacemakers",
    "Imaging artifacts – noise, motion blur, exposure issues",
    "View & positioning – PA/AP views, rotation, supine/erect",
    "Domain shift – scanner, hospital, and resolution variations"
]

# --- GENERATION LOOP ---
print(f"Starting generation: {len(condition_list)} classes, {num_images_per_class} images each.\n")

base_prompt = "A high quality , not in NSFW category, medical Chest X-Ray radiograph showing "

for item in condition_list:
    # 1. Extract Class Name (e.g., "Normal anatomy" from the full string)
    # We strip whitespace and replace spaces with underscores for safe folder names
    raw_class_name = item.split("–")[0].strip()
    class_folder_name = raw_class_name.replace(" ", "_")

    # 2. Create the Class Sub-Directory
    class_path = os.path.join(dataset_root, class_folder_name)
    os.makedirs(class_path, exist_ok=True)

    print(f"Processing Class: {class_folder_name}...")

    # 3. Generate Multiple Images per Class
    for i in range(num_images_per_class):
        # Construct dynamic prompt
        final_prompt = f"{base_prompt} {item}, grayscale, high contrast, bones, anatomical detail"

        # Generate
        image = pipe(
            final_prompt,
            negative_prompt="color, cartoon, 3d render, realistic skin, text, watermark, label"
        ).images[0]

        # Save with a unique name inside the class folder
        # e.g., medical_xray_dataset_resnet/Normal_anatomy/img_0.png
        filename = f"img_{i}.png"
        save_path = os.path.join(class_path, filename)
        image.save(save_path)

    print(f"   -> Saved {num_images_per_class} images in /{class_folder_name}")

print("\n--- Dataset Generation Complete ---")
print(f"Dataset is ready for ResNet at: {os.path.abspath(dataset_root)}")
```

```
Starting generation: 10 classes, 2 images each.

Processing Class: Normal_anatomy...
100%                                          50/50 [00:07<00:00,  6.86it/s]
100%                                          50/50 [00:07<00:00,  6.73it/s]
Potential NSFW content was detected in one or more images. A black image will be returned instead. Try again with a differer
   -> Saved 2 images in /Normal_anatomy
Processing Class: Infectious_patterns...
100%                                          50/50 [00:07<00:00,  6.58it/s]
100%                                          50/50 [00:07<00:00,  6.75it/s]
Potential NSFW content was detected in one or more images. A black image will be returned instead. Try again with a differer
   -> Saved 2 images in /Infectious_patterns
Processing Class: Lung_opacities...
100%                                          50/50 [00:07<00:00,  6.84it/s]
100%                                          50/50 [00:07<00:00,  6.83it/s]
   -> Saved 2 images in /Lung_opacities
Processing Class: Pleural_conditions...
100%                                          50/50 [00:07<00:00,  6.93it/s]
100%                                          50/50 [00:07<00:00,  7.00it/s]
Potential NSFW content was detected in one or more images. A black image will be returned instead. Try again with a differer
   -> Saved 2 images in /Pleural_conditions
Processing Class: Structural_lesions...
100%                                          50/50 [00:07<00:00,  6.98it/s]
100%                                          50/50 [00:07<00:00,  6.93it/s]
   -> Saved 2 images in /Structural_lesions
Processing Class: Cardiac_findings...
100%                                          50/50 [00:07<00:00,  6.93it/s]
100%                                          50/50 [00:07<00:00,  6.88it/s]
   -> Saved 2 images in /Cardiac_findings
Processing Class: Medical_devices...
100%                                          50/50 [00:07<00:00,  6.90it/s]
100%                                          50/50 [00:07<00:00,  6.84it/s]
Potential NSFW content was detected in one or more images. A black image will be returned instead. Try again with a differer
   -> Saved 2 images in /Medical_devices
Processing Class: Imaging_artifacts...
100%                                          50/50 [00:07<00:00,  6.90it/s]
100%                                          50/50 [00:07<00:00,  6.85it/s]
   -> Saved 2 images in /Imaging_artifacts
Processing Class: View_&_positioning...
100%                                          50/50 [00:07<00:00,  6.80it/s]
Potential NSFW content was detected in one or more images. A black image will be returned instead. Try again with a differer
100%                                          50/50 [00:07<00:00,  6.93it/s]
   -> Saved 2 images in /View_&_positioning
Processing Class: Domain_shift...
100%                                          50/50 [00:07<00:00,  6.90it/s]
100%                                          50/50 [00:07<00:00,  6.86it/s]
   -> Saved 2 images in /Domain_shift

--- Dataset Generation Complete ---
Dataset is ready for ResNet at: /content/medical_xray_dataset_resnet
```

```
import matplotlib.pyplot as plt
import numpy as np

# --- HELPER FUNCTION TO UN-NORMALIZE IMAGES ---
# We normalized images for the AI (math), but humans need normal colors to see them.
def imshow(inp, title=None, color='black'):
    inp = inp.numpy().transpose((1, 2, 0)) # Convert from Tensor (C,H,W) to Image (H,W,C)
    mean = np.array([0.485, 0.456, 0.406])
    std = np.array([0.229, 0.224, 0.225])
    inp = std * inp + mean # Undo the normalization
    inp = np.clip(inp, 0, 1) # Ensure pixel values are valid
```

```
    plt.imshow(inp)
    if title is not None:
        plt.title(title, color=color, fontsize=12, fontweight='bold')
    plt.axis('off')

# --- VISUALIZE PREDICTIONS ---
model.eval() # Set to evaluation mode

# Get a batch of training data
inputs, classes = next(iter(test_loader))
inputs, classes = inputs.to(device), classes.to(device)

# Make predictions
outputs = model(inputs)
_, preds = torch.max(outputs, 1)

# Plot the results
fig = plt.figure(figsize=(16, 8))
num_images_to_show = 4 # Adjust depending on your batch size

print("\n--- Visualizing Predictions ---")

for i in range(num_images_to_show):
    ax = fig.add_subplot(1, num_images_to_show, i+1)

    # Get the names
    actual_label = full_dataset.classes[classes[i]]
    predicted_label = full_dataset.classes[preds[i]]

    # Check if correct (Green) or Wrong (Red)
    is_correct = (actual_label == predicted_label)
    color = 'green' if is_correct else 'red'

    # Display
    title = f"Actual: {actual_label}\nPredicted: {predicted_label}"
    imshow(inputs.cpu().data[i], title=title, color=color)

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

```
--- Visualizing Predictions ---
```




```
import torch.nn as nn
import torchvision.models as models

# 1. Determine device
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

# 2. Load pre-trained DenseNet121
model = models.densenet121(pretrained=True)
print("Loaded pre-trained DenseNet121 model.")

# 3. Move model to device
model = model.to(device)

# 4. Modify the final classification layer
num_ftrs = model.classifier.in_features
num_classes = 10 # As specified in the instructions
```

```
num_classes = 10 # As specified in the instructions
model.classifier = nn.Linear(num_ftrs, num_classes)
model = model.to(device) # Ensure the new layer is also on the correct device

print(f"Modified classifier layer to output {num_classes} classes.")
print("Model loaded and adapted successfully.")
```

```
Using device: cuda:0
Loaded pre-trained DenseNet121 model.
Modified classifier layer to output 10 classes.
Model loaded and adapted successfully.
/usr/local/lib/python3.12/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecat
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or
  warnings.warn(msg)
```

```
import torch.nn as nn
import torchvision.models as models
from torchvision.models import DenseNet121_Weights

# 1. Determine device
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

# 2. Load pre-trained DenseNet121
model = models.densenet121(weights=DenseNet121_Weights.DEFAULT)
print("Loaded pre-trained DenseNet121 model.")

# 3. Move model to device
model = model.to(device);

# 4. Modify the final classification layer
num_ftrs = model.classifier.in_features
num_classes = 10 # As specified in the instructions
model.classifier = nn.Linear(num_ftrs, num_classes)
model = model.to(device) # Ensure the new layer is also on the correct device

print(f"Modified classifier layer to output {num_classes} classes.")
print("Model loaded and adapted successfully.")
```

```
Using device: cuda:0
Loaded pre-trained DenseNet121 model.
Modified classifier layer to output 10 classes.
Model loaded and adapted successfully.
```

```
import torchvision.transforms as transforms
import torchvision.datasets as datasets
import torch.utils.data as data

# 1. Define image transformations
# ImageNet statistics for normalization
mean = [0.485, 0.456, 0.406]
std = [0.229, 0.224, 0.225]

train_transforms = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean, std)
])

val_transforms = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean, std)
])

print("Image transformations defined.")

# 2. Create ImageFolder dataset
# dataset_root is defined in a previous cell as "medical_xray_dataset_resnet"
full_dataset = datasets.ImageFolder(root=dataset_root, transform=train_transforms)
print(f"Full dataset created with {len(full_dataset)} images and {len(full_dataset.classes)} classes.")

# 3. Split dataset into training and validation sets
torch.manual_seed(42) # For reproducibility
train_size = int(0.8 * len(full_dataset))
```

```
val_size = len(full_dataset) - train_size
train_dataset, val_dataset = data.random_split(full_dataset, [train_size, val_size])

print(f"Dataset split: Training set size = {len(train_dataset)}, Validation set size = {len(val_dataset)}.")

# 4. Create DataLoader instances
batch_size = 4
num_workers = 2 # Set based on available CPU cores

train_loader = data.DataLoader(train_dataset, batch_size=batch_size, shuffle=True, num_workers=num_workers)
val_loader = data.DataLoader(val_dataset, batch_size=batch_size, shuffle=False, num_workers=num_workers)

print(f"DataLoaders created with batch size {batch_size} and {num_workers} workers.")
print("Dataset preparation complete.")
```

```
Image transformations defined.
Full dataset created with 20 images and 10 classes.
Dataset split: Training set size = 16, Validation set size = 4.
DataLoaders created with batch size 4 and 2 workers.
Dataset preparation complete.
```

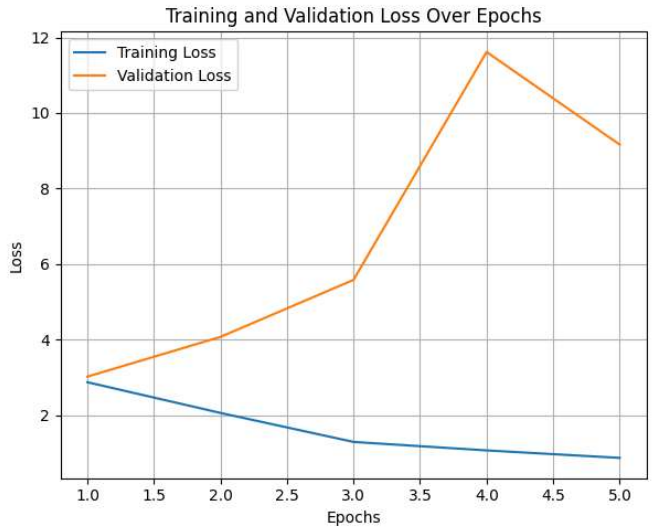
```
import torch.optim as optim

# 1. Define the loss function
criterion = nn.CrossEntropyLoss()

# 2. Define the optimizer
learning_rate = 0.001
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

# 3. Set the number of training epochs
num_epochs = 5

# 4. Initialize lists to store training and validation losses and accuracies
train_losses = []
train_accuracies = []
val_losses = []
val_accuracies = []

print(f"Starting training for {num_epochs} epochs...")

# 5. Implement the training loop
for epoch in range(num_epochs):
    model.train()  # Set model to training mode
    running_loss = 0.0
    correct_train = 0
    total_train = 0

    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.to(device)

        optimizer.zero_grad()  # Zero the parameter gradients

        outputs = model(inputs)  # Forward pass
        loss = criterion(outputs, labels)  # Calculate loss
        loss.backward()  # Backward pass
        optimizer.step()  # Optimize

        running_loss += loss.item() * inputs.size(0)
        _, predicted = torch.max(outputs.data, 1)
        total_train += labels.size(0)
        correct_train += (predicted == labels).sum().item()

    epoch_train_loss = running_loss / len(train_dataset)
    epoch_train_acc = (correct_train / total_train) * 100
    train_losses.append(epoch_train_loss)
    train_accuracies.append(epoch_train_acc)

    # 6. Implement the validation step
    model.eval()  # Set model to evaluation mode
    running_val_loss = 0.0
    correct_val = 0
    total_val = 0

    with torch.no_grad():  # Disable gradient calculations during validation
```

```
        for inputs, labels in val_loader:
            inputs, labels = inputs.to(device), labels.to(device)

            outputs = model(inputs)
            loss = criterion(outputs, labels)

            running_val_loss += loss.item() * inputs.size(0)
            _, predicted = torch.max(outputs.data, 1)
            total_val += labels.size(0)
            correct_val += (predicted == labels).sum().item()

    epoch_val_loss = running_val_loss / len(val_dataset)
    epoch_val_acc = (correct_val / total_val) * 100
    val_losses.append(epoch_val_loss)
    val_accuracies.append(epoch_val_acc)

    # 7. Print the training and validation loss and accuracy for each epoch
    print(f"Epoch {epoch+1}/{num_epochs} - "
          f"Train Loss: {epoch_train_loss:.4f}, Train Acc: {epoch_train_acc:.2f}% | "
          f"Val Loss: {epoch_val_loss:.4f}, Val Acc: {epoch_val_acc:.2f}%")

print("Training complete.")
```

```
Starting training for 5 epochs...
Epoch 1/5 - Train Loss: 2.8722, Train Acc: 6.25% | Val Loss: 3.0194, Val Acc: 0.00%
Epoch 2/5 - Train Loss: 2.0595, Train Acc: 31.25% | Val Loss: 4.0719, Val Acc: 0.00%
Epoch 3/5 - Train Loss: 1.2915, Train Acc: 68.75% | Val Loss: 5.5815, Val Acc: 0.00%
Epoch 4/5 - Train Loss: 1.0668, Train Acc: 75.00% | Val Loss: 11.6144, Val Acc: 0.00%
Epoch 5/5 - Train Loss: 0.8692, Train Acc: 68.75% | Val Loss: 9.1628, Val Acc: 0.00%
Training complete.
```

Reasoning: The training process has completed, and the losses and accuracies for both training and validation sets have been stored. To analyze the model's performance and identify trends like overfitting or underfitting, it's crucial to visualize these metrics over the epochs. This plot will clearly show how the model's performance evolved during training.

```
import matplotlib.pyplot as plt

# Plotting training and validation loss
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(range(1, num_epochs + 1), train_losses, label='Training Loss')
plt.plot(range(1, num_epochs + 1), val_losses, label='Validation Loss')
plt.title('Training and Validation Loss Over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)

# Plotting training and validation accuracy
plt.subplot(1, 2, 2)
plt.plot(range(1, num_epochs + 1), train_accuracies, label='Training Accuracy')
plt.plot(range(1, num_epochs + 1), val_accuracies, label='Validation Accuracy')
plt.title('Training and Validation Accuracy Over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy (%)')
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()

print("Plotted training and validation loss and accuracy.")
```




```
Plotted training and validation loss and accuracy.
```

```
import sklearn.metrics as metrics
import seaborn as sns
import numpy as np

# 1. Set the model to evaluation mode
model.eval()

# 2. Initialize variables to track correct predictions and total predictions
# and lists to store true labels and predicted labels for the confusion matrix.
correct_predictions = 0
total_predictions = 0
all_labels = []
all_predictions = []

print("Evaluating model on the validation set...")

# 3. Iterate through the val_loader (validation data).
with torch.no_grad():  # Disable gradient calculations during evaluation
    for inputs, labels in val_loader:
        inputs, labels = inputs.to(device), labels.to(device)

        outputs = model(inputs)
        _, predicted = torch.max(outputs.data, 1)

        total_predictions += labels.size(0)
        correct_predictions += (predicted == labels).sum().item()

        all_labels.extend(labels.cpu().numpy())
        all_predictions.extend(predicted.cpu().numpy())

# 5. Calculate the overall accuracy on the validation set and print it.
accuracy = (correct_predictions / total_predictions) * 100
print(f"Validation Accuracy: {accuracy:.2f}%")

# Get class names from the full dataset
class_names = full_dataset.classes

# Define all possible labels (class indices) for the classification report and confusion matrix
# This addresses the ValueError when not all classes are present in the validation set
all_possible_labels = list(range(num_classes))

# 6. Generate a classification report
print("\nClassification Report:")
print(metrics.classification_report(all_labels, all_predictions, target_names=class_names, labels=all_possible_labels, zero_

# 7. Compute the confusion matrix
confusion_mtx = metrics.confusion_matrix(all_labels, all_predictions, labels=all_possible_labels)

# 8. Plot the confusion matrix
plt.figure(figsize=(10, 8))
sns.heatmap(confusion_mtx, annot=True, fmt='d', cmap='Blues', xticklabels=class_names, yticklabels=class_names)
```

```
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()

print("Model evaluation complete.")
```

```
Evaluating model on the validation set...
Validation Accuracy: 0.00%

Classification Report:
                    precision    recall  f1-score   support

   Cardiac_findings       0.00      0.00      0.00       0.0
       Domain_shift       0.00      0.00      0.00       0.0
  Imaging_artifacts       0.00      0.00      0.00       1.0
Infectious_patterns       0.00      0.00      0.00       0.0
     Lung_opacities       0.00      0.00      0.00       1.0
    Medical_devices       0.00      0.00      0.00       0.0
     Normal_anatomy       0.00      0.00      0.00       0.0
 Pleural_conditions       0.00      0.00      0.00       0.0
 Structural_lesions       0.00      0.00      0.00       1.0
 View_&_positioning       0.00      0.00      0.00       1.0

           accuracy                           0.00       4.0
          macro avg       0.00      0.00      0.00       4.0
       weighted avg       0.00      0.00      0.00       4.0
```