X-Ray Image Classification with Vision Transformer

Introduction

This notebook demonstrates how to classify chest X-ray images using three different models:

1. Vision Transformer (ViT) without freezing parameters.
2. Vision Transformer (ViT) with freezing parameters.
3. Convolutional Neural Network (CNN) using ResNet50.

Dataset

The dataset used is the Chest X-Ray Images dataset, available on Kaggle. The notebook

covers the process of data preparation, model training, and result comparison.

This dataset contains chest X-ray images categorized into four distinct classes:

1. **COVID-19**: X-ray images from patients diagnosed with COVID-19.
2. **Normal**: X-ray images from healthy individuals without any respiratory disease.
3. **Pneumonia-Bacterial**: X-ray images from patients diagnosed with bacterial pneumonia.
4. **Pneumonia-Viral**: X-ray images from patients diagnosed with viral pneumonia.




Dataset Summary

- **Train Set**: 6,902 images belonging to 4 classes.
- **Validation Set**: 923 images belonging to 4 classes.
- **Test Set**: 1,384 images belonging to 4 classes.

Getting Started 1. Opening the Notebook

To begin, follow these steps:

1. **Accessing Google Colab**:
    - Open Google Colab in your web browser: Google Colab.
    - Click on "File" > "Open notebook". ○ Select the "GitHub" tab or upload your notebook directly if you have it saved on your local machine.
2. **Mounting Google Drive** (if using Colab):
    - Ensure your dataset is uploaded to Google Drive. ○ Run the following code snippet to mount your Google Drive to Colab:

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

2. Importing Requirements

Start by importing the necessary libraries and setting up the environment:

```
import torch
from torch import nn
import os
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
from pathlib import Path
import zipfile
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import precision_score, recall_score, f1_score, confusion_matrix
from tqdm.auto import tqdm
import torchvision
from torchinfo import summary
```

3. Dataset Overview

Download and Upload the Dataset

- Download the dataset from Kaggle: Chest X-Ray Images Dataset.
- Upload the dataset zip file to your Google Drive.

Extracting the Dataset

Run the following code to extract the dataset:

```
zip_ref = zipfile.ZipFile("/content/drive/MyDrive/archive.zip", 'r')
zip_ref.extractall("tmp/")
zip_ref.close()
```

Update the data_paths and dataset directories as follows:

```
data_paths = Path('/content/tmp')
train_dir = data_paths / 'data_split/train'
valid_dir = data_paths / 'data_split/validation'
test_dir = data_paths / 'data_split/test'
```

4. Data Preparation

```
IMG_SIZE = 224
BATCH_SIZE = 32

def create_dataloaders(train_dir, test_dir, transform, batch_size):
    train_data = datasets.ImageFolder(train_dir, transform=transform)
    test_data = datasets.ImageFolder(test_dir, transform=transform)
    class_names = train_data.classes
    train_dataloader = DataLoader(train_data, batch_size=batch_size, shuffle=True, pin_memory=True)
    test_dataloader = DataLoader(test_data, batch_size=batch_size, shuffle=False, pin_memory=True)
    return train_dataloader, test_dataloader, class_names

transforms_train = transforms.Compose([
    transforms.Resize((IMG_SIZE, IMG_SIZE)),
    transforms.RandomHorizontalFlip(p=0.3),
    transforms.RandomResizedCrop(IMG_SIZE),
    transforms.ToTensor(),
    transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225)),
])

transforms_test = transforms.Compose([
    transforms.Resize((IMG_SIZE, IMG_SIZE)),
    transforms.ToTensor(),
    transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225)),
])

train_dataloader, _, _ = create_dataloaders(train_dir=train_dir, test_dir=test_dir, batch_size=BATCH_SIZE, transform=transforms_
_, test_dataloader, _ = create_dataloaders(train_dir=train_dir, test_dir=test_dir, batch_size=BATCH_SIZE, transform=transforms_t
_, valid_dataloader, class_names = create_dataloaders(train_dir=train_dir, test_dir=valid_dir, batch_size=BATCH_SIZE, transform
```

5. Helper Functions

Define training, testing, and result plotting functions:

```python
def train_step(model, dataloader, loss_fn, optimizer, device):
    model.train()
    train_loss, train_acc = 0, 0
    all_labels, all_preds = [], []
    for batch, (X, y) in enumerate(dataloader):
        X, y = X.to(device), y.to(device)
        y_pred = model(X)
        loss = loss_fn(y_pred, y)
        train_loss += loss.item()
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        y_pred_class = torch.argmax(torch.softmax(y_pred, dim=1), dim=1)
        train_acc += (y_pred_class == y).sum().item() / len(y_pred)
        all_labels.extend(y.cpu().numpy())
        all_preds.extend(y_pred_class.cpu().numpy())
    train_loss /= len(dataloader)
    train_acc /= len(dataloader)
    train_precision = precision_score(all_labels, all_preds, average='weighted')
    train_recall = recall_score(all_labels, all_preds, average='weighted')
    train_f1 = f1_score(all_labels, all_preds, average='weighted')
    return {"loss": train_loss, "accuracy": train_acc, "precision": train_precision, "recall": train_recall, "f1": train_f1}
```

```python
def test_step(model, dataloader, loss_fn, device):
    model.eval()
    test_loss, test_acc = 0, 0
    all_labels, all_preds = [], []
    with torch.no_grad():
        for batch, (X, y) in enumerate(dataloader):
            X, y = X.to(device), y.to(device)
            test_pred_logits = model(X)
            loss = loss_fn(test_pred_logits, y)
            test_loss += loss.item()
            test_pred_labels = test_pred_logits.argmax(dim=1)
            test_acc += ((test_pred_labels == y).sum().item() / len(test_pred_labels))
            all_labels.extend(y.cpu().numpy())
            all_preds.extend(test_pred_labels.cpu().numpy())
    test_loss /= len(dataloader)
    test_acc /= len(dataloader)
    test_precision = precision_score(all_labels, all_preds, average='weighted')
    test_recall = recall_score(all_labels, all_preds, average='weighted')
    test_f1 = f1_score(all_labels, all_preds, average='weighted')
    return {"loss": test_loss, "accuracy": test_acc, "precision": test_precision, "recall": test_recall, "f1": test_f1}, all_lal
```

```python
def train(model, train_dataloader, test_dataloader, optimizer, loss_fn, epochs, device):
    results = {"train_loss": [], "train_acc": [], "train_precision": [], "train_recall": [], "train_f1": [],
               "test_loss": [], "test_acc": [], "test_precision": [], "test_recall": [], "test_f1": []}
    model.to(device)
    final_all_labels, final_all_preds = [], []
    for epoch in tqdm(range(epochs)):
        train_metrics = train_step(model, train_dataloader, loss_fn, optimizer, device)
        test_metrics, all_labels, all_preds = test_step(model, test_dataloader, loss_fn, device)
        if epoch == epochs - 1:
            final_all_labels = all_labels
            final_all_preds = all_preds
        print(f"Epoch: {epoch + 1} | "
              f"train_loss: {train_metrics['loss']:.4f} | "
              f"train_acc: {train_metrics['accuracy']:.4f} | "
              f"train_precision: {train_metrics['precision']:.4f} | "
              f"train_recall: {train_metrics['recall']:.4f} | "
              f"train_f1: {train_metrics['f1']:.4f} | "
              f"test_loss: {test_metrics['loss']:.4f} | "
              f"test_acc: {test_metrics['accuracy']:.4f} | "
              f"test_precision: {test_metrics['precision']:.4f} | "
              f"test_recall: {test_metrics['recall']:.4f} | "
              f"test_f1: {test_metrics['f1']:.4f}")
        results["train_loss"].append(train_metrics['loss'])
        results["train_acc"].append(train_metrics['accuracy'])
        results["train_precision"].append(train_metrics['precision'])
        results["train_recall"].append(train_metrics['recall'])
        results["train_f1"].append(train_metrics['f1'])
        results["test_loss"].append(test_metrics['loss'])
        results["test_acc"].append(test_metrics['accuracy'])
        results["test_precision"].append(test_metrics['precision'])
        results["test_recall"].append(test_metrics['recall'])
        results["test_f1"].append(test_metrics['f1'])
    conf_matrix = confusion_matrix(final_all_labels, final_all_preds)
    plt.figure(figsize=(10, 7))
    sns.heatmap(conf_matrix, annot=True, fmt='d

', cmap='Blues', xticklabels=class_names, yticklabels=class_names)
    plt.xlabel('Predicted labels')
    plt.ylabel('True labels')
    plt.show()
    return results
```

```python
def plot_results(results):
    epochs = range(len(results['train_loss']))
    plt.figure(figsize=(12, 8))
    plt.subplot(2, 2, 1)
    plt.plot(epochs, results['train_loss'], label='Train Loss')
    plt.plot(epochs, results['test_loss'], label='Test Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.title('Train and Test Loss')
    plt.legend()

    plt.subplot(2, 2, 2)
    plt.plot(epochs, results['train_acc'], label='Train Accuracy')
    plt.plot(epochs, results['test_acc'], label='Test Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.title('Train and Test Accuracy')
    plt.legend()

    plt.subplot(2, 2, 3)
    plt.plot(epochs, results['train_precision'], label='Train Precision')
    plt.plot(epochs, results['test_precision'], label='Test Precision')
    plt.xlabel('Epoch')
    plt.ylabel('Precision')
    plt.title('Train and Test Precision')
    plt.legend()

    plt.subplot(2, 2, 4)
    plt.plot(epochs, results['train_recall'], label='Train Recall')
    plt.plot(epochs, results['test_recall'], label='Test Recall')
    plt.xlabel('Epoch')
    plt.ylabel('Recall')
    plt.title('Train and Test Recall')
    plt.legend()

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

6. Model Training and Evaluation

Vision Transformer (ViT) without Freezing Parameters:

```python
from torchvision.models import vit_b_16
import torch.optim as optim

vit_model = vit_b_16(pretrained=True)
num_ftrs = vit_model.heads[0].in_features
vit_model.heads = nn.Linear(num_ftrs, len(class_names))

optimizer = optim.AdamW(vit_model.parameters(), lr=0.0001)
loss_fn = nn.CrossEntropyLoss()
results_vit_no_freeze = train(vit_model, train_dataloader, test_dataloader, optimizer, loss_fn, epochs=10, device='cuda')
plot_results(results_vit_no_freeze)
```

Vision Transformer (ViT) with Freezing Parameters:

```python
from torchvision.models import vit_b_16
import torch.optim as optim

vit_model = vit_b_16(pretrained=True)
num_ftrs = vit_model.heads[0].in_features
vit_model.heads = nn.Linear(num_ftrs, len(class_names))

for param in vit_model.parameters():
    param.requires_grad = False
vit_model.heads.requires_grad = True

optimizer = optim.AdamW(vit_model.parameters(), lr=0.0001)
loss_fn = nn.CrossEntropyLoss()
results_vit_freeze = train(vit_model, train_dataloader, test_dataloader, optimizer, loss_fn, epochs=10, device='cuda')
plot_results(results_vit_freeze)
```

ResNet50:

```python
[ ]  pre_trained_model = ResNet50(include_top=False,
                                  weights=None,
                                  input_shape=(224,224,3))
```

7. Results Comparison

Model Comparison Table

Metric	ViT (No Freeze)	ViT (Freeze)	ResNet50
Training Loss	0.8531	0.4897	1.3127
Test Loss	0.7471	0.4047	1.3157
Training Accuracy	0.6584	0.8009	0.3633
Test Accuracy	0.7038	0.8361	0.3554
Training Precision	0.6439	0.7909	[Value]
Test Precision	0.7119	0.8300	[Value]
Training Recall	0.6584	0.8008	[Value]
Test Recall	0.7047	0.8373	[Value]
Training F1 Score	0.6350	0.7928	[Value]
Test F1 Score	0.6943	0.8266	[Value]

In this case, the Vision Transformer (ViT) outperformed the ResNet50 model. The ViT demonstrated superior accuracy and generalization on the test set, likely due to its ability to capture more complex patterns and global context in the X-ray images, which are crucial for accurate classification. The performance gap suggests that the ViT's self-attention mechanism provides a significant advantage over the convolutional approach of ResNet50 in this specific task.

---

8. Conclusion

This notebook provides a detailed guide to setting up and evaluating different image classification models. By following these steps, you can gain insights into the performance of Vision Transformer models with and without parameter freezing, as well as ResNet50.

Feel free to modify and adapt the notebook to fit your specific needs and experiments.
