code3_VIT_RESNET-Copy4_valid_sep

November 4, 2024

[1]:
```
# Imports here
from __future__ import print_function, division
import numpy as np
import matplotlib.pyplot as plt
from torch.utils import data
import torch
from torch import nn, optim
import torchvision
import torch.nn.functional as F
from torchvision import datasets, transforms, models
from torch.utils.data.sampler import SubsetRandomSampler
from torch.utils.data import Dataset, DataLoader
from skimage import io, transform
from PIL import Image, ImageFile
import json
from torch.optim import lr_scheduler
import seaborn as sns
import cv2
import os
import pandas as pd
from sklearn.metrics import confusion_matrix
from transformers import ViTModel, ViTFeatureExtractor
```

```
2024-11-02 13:12:49.404512: I tensorflow/core/util/port.cc:153] oneDNN custom
operations are on. You may see slightly different numerical results due to
floating-point round-off errors from different computation orders. To turn them
off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2024-11-02 13:12:49.648136: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:485] Unable to register
cuFFT factory: Attempting to register factory for plugin cuFFT when one has
already been registered
2024-11-02 13:12:49.752811: E
external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:8454] Unable to register
cuDNN factory: Attempting to register factory for plugin cuDNN when one has
already been registered
2024-11-02 13:12:49.782413: E
external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1452] Unable to
register cuBLAS factory: Attempting to register factory for plugin cuBLAS when
```

```
one has already been registered
2024-11-02 13:12:49.990208: I tensorflow/core/platform/cpu_feature_guard.cc:210]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 AVX512F AVX512_VNNI FMA, in other
operations, rebuild TensorFlow with the appropriate compiler flags.
2024-11-02 13:12:52.563624: W
tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not
find TensorRT
```

[2]:
```
import os
os.environ['KMP_DUPLICATE_LIB_OK'] = 'True'
```

[3]:
```
# Paths and dataset loading
train_csv = pd.read_csv('/home/gcekcse/Documents/ML_Project_hk/data/train.csv')
test_csv = pd.read_csv('/home/gcekcse/Documents/ML_Project_hk/data/test.csv')
valid_csv = pd.read_csv('/home/gcekcse/Documents/ML_Project_hk/data/valid.csv')

train_path = "/home/gcekcse/Documents/ML_Project_hk/data/train_images/"
test_path = "/home/gcekcse/Documents/ML_Project_hk/data/test_images/"
val_path = "/home/gcekcse/Documents/ML_Project_hk/data/val_images/"
```

[4]:
```
print(train_csv.head())
print(test_csv.head())
```

```
                             id_code  diagnosis
0  005dcc1569054efb94f5d28a07d896cb          0
1  0069f61db3e24b3b995b7fe95ae7fc54          0
2  016454c0db3d4f6c8a17f432a5e820a7          0
3  020d594c4f8b431692b81eadb5386375          0
4  026c3394f9a845af84a9b3e3293b30bb          0
        id_code  diagnosis
0  e4dcca36ceb4          0
1  e50b0174690d          0
2  e5197d77ec68          0
3  e529c5757d64          0
4  e582e56e7942          0
```

[5]:
```
# Check image distribution
counts = train_csv['diagnosis'].value_counts()
class_list = ['No DR', 'Mild', 'Moderate', 'Severe', 'Proliferative']
for i, x in enumerate(class_list):
    counts[x] = counts.pop(i)
plt.figure(figsize=(10, 5))
sns.barplot(x=counts.index, y=counts.values, alpha=0.8, palette='bright')
plt.title('Distribution of Output Classes')
plt.ylabel('Number of Occurrences', fontsize=12)
```

```
plt.xlabel('Target Classes', fontsize=12)
plt.show()
```

```
/tmp/ipykernel_1392160/3989213185.py:7: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=counts.index, y=counts.values, alpha=0.8, palette='bright')
```

[6]:
```
# Dataset class
class CreateDataset(Dataset):
    def __init__(self, df_data, data_dir, transform=None):
        super().__init__()
        self.df = df_data.values
        self.data_dir = data_dir
        self.transform = transform

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

    def __getitem__(self, index):
        img_name, label = self.df[index]
        img_path = os.path.join(self.data_dir, img_name)

        # Check if the file has the correct extension
```

```
        if not img_path.endswith('.png'):
            img_path += '.png'

        # Load the image
        image = cv2.imread(img_path)
        if image is None:
            raise FileNotFoundError(f"Image not found: {img_path}")

        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)  # Convert to RGB if␣
 ↪needed
        if self.transform is not None:
            image = self.transform(image)

        return image, label

# Transforms
train_transforms = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Resize((224, 224)),
    transforms.RandomHorizontalFlip(p=0.4),
    transforms.ToTensor(),
    transforms.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225))
])

test_transforms = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Resize((224, 224)),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

# Validation transforms
val_transforms = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Resize((224, 224)),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])
```

[7]:
```
# Datasets
train_data = CreateDataset(df_data=train_csv, data_dir=train_path,␣
 ↪transform=train_transforms)
valid_data = CreateDataset(df_data=valid_csv, data_dir=val_path,␣
 ↪transform=val_transforms)
```

```
test_data = CreateDataset(df_data=test_csv, data_dir=test_path,␣
 ↪transform=test_transforms)

# Data loaders
trainloader = DataLoader(train_data, batch_size=64, shuffle=True)
validloader = DataLoader(valid_data, batch_size=64, shuffle=False)
testloader = DataLoader(test_data, batch_size=64, shuffle=False)
```

[ ]:

[8]:
```
# Hybrid Model with ResNet152 and ViT
class HybridModel(nn.Module):
    def __init__(self):
        super(HybridModel, self).__init__()

        # ResNet152 model
        self.resnet = models.resnet152(pretrained=True)
        num_ftrs_resnet = self.resnet.fc.in_features
        self.resnet.fc = nn.Identity()  # Remove the ResNet's classifier

        # Vision Transformer (ViT) model
        self.vit = ViTModel.from_pretrained("google/vit-base-patch16-224-in21k")

        # Combined classifier
        self.fc = nn.Sequential(
            nn.Linear(num_ftrs_resnet + 768, 512),  # Combining ResNet and ViT␣
 ↪outputs
            nn.ReLU(),
            nn.Linear(512, 5),  # 5 output classes for diabetic retinopathy
            nn.LogSoftmax(dim=1)
        )

    def forward(self, x):
        resnet_features = self.resnet(x)  # ResNet output
        vit_features = self.vit(pixel_values=x).last_hidden_state[:, 0]  # ViT␣
 ↪output (CLS token)

        combined = torch.cat((resnet_features, vit_features), dim=1)  #␣
 ↪Concatenate ResNet and ViT features
        output = self.fc(combined)  # Classifier
        return output

# Initialize the model
model = HybridModel()

# Define device (CPU or GPU)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```
# Move model to the device (GPU or CPU)
model = model.to(device)

# Loss, optimizer, scheduler
criterion = nn.NLLLoss()
optimizer = torch.optim.Adam(filter(lambda p: p.requires_grad, model.
 ↪parameters()), lr=0.00001)
scheduler = lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.1)
```

```
/home/gcekcse/hkenv/lib/python3.12/site-
packages/torchvision/models/_utils.py:208: UserWarning: The parameter
'pretrained' is deprecated since 0.13 and may be removed in the future, please
use 'weights' instead.
  warnings.warn(
/home/gcekcse/hkenv/lib/python3.12/site-
packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a
weight enum or `None` for 'weights' are deprecated since 0.13 and may be removed
in the future. The current behavior is equivalent to passing
`weights=ResNet152_Weights.IMAGENET1K_V1`. You can also use
`weights=ResNet152_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
```

[9]:
```
# Define model file name and checkpoint path
model_name = "classifier.pt"
checkpoint_path = "/home/gcekcse/Documents/ML_Project_hk/data/models/"
model_file_path = os.path.join(checkpoint_path, model_name)
checkpoint_file = os.path.join(checkpoint_path, 'checkpoint_4.pt')

# Train and test loop with checkpointing and early stopping
def train_and_test(e, patience=5):
    # Check if there's a checkpoint to load
    if os.path.exists(checkpoint_file):
        print("Loading checkpoint...")
        checkpoint = torch.load(checkpoint_file)
        start_epoch = checkpoint['epoch'] + 1
        train_losses = checkpoint['train_losses']
        test_losses = checkpoint['test_losses']
        acc = checkpoint['acc']
        model.load_state_dict(checkpoint['model_state_dict'])
        optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
        valid_loss_min = checkpoint['valid_loss_min']
    else:
        start_epoch = 0
        train_losses, test_losses, acc = [], [], []
        valid_loss_min = np.Inf
```

```
model.train()
print("Model Training started.....")

epochs_no_improve = 0  # Track epochs with no improvement for early stopping

for epoch in range(start_epoch, e):
    running_loss = 0
    batch = 0
    for images, labels in trainloader:
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
        batch += 1
        if batch % 10 == 0:
            print(f" epoch {epoch + 1} batch {batch} completed")

    test_loss, accuracy = 0, 0
    with torch.no_grad():
        model.eval()
        for images, labels in validloader:
            images, labels = images.to(device), labels.to(device)
            logps = model(images)
            test_loss += criterion(logps, labels).item()
            ps = torch.exp(logps)
            top_p, top_class = ps.topk(1, dim=1)
            equals = top_class == labels.view(*top_class.shape)
            accuracy += torch.mean(equals.type(torch.FloatTensor))

    train_losses.append(running_loss / len(trainloader))
    test_losses.append(test_loss / len(validloader))
    acc.append(accuracy / len(validloader))
    scheduler.step()

    print(f"Epoch: {epoch + 1}/{e}.. "
          f"Training Loss: {running_loss/len(trainloader):.3f}.. "
          f"Valid Loss: {test_loss/len(validloader):.3f}.. "
          f"Valid Accuracy: {accuracy/len(validloader):.3f}")

    # Save checkpoint after each epoch
    checkpoint = {
        'epoch': epoch,
        'model_state_dict': model.state_dict(),
        'optimizer_state_dict': optimizer.state_dict(),
```

```
            'train_losses': train_losses,
            'test_losses': test_losses,
            'acc': acc,
            'valid_loss_min': valid_loss_min
        }
        torch.save(checkpoint, checkpoint_file)

        # Check if validation loss has improved
        if test_loss / len(validloader) < valid_loss_min:
            torch.save({'model_state_dict': model.state_dict(),␣
↪'optimizer_state_dict': optimizer.state_dict()},
                       model_file_path)
            valid_loss_min = test_loss / len(validloader)
            epochs_no_improve = 0  # Reset early stopping counter if validation␣
↪loss improved
        else:
            epochs_no_improve += 1  # Increment early stopping counter

        # Early stopping check
        if epochs_no_improve >= patience:
            print("Early stopping triggered.")
            break

    print('Training Completed Successfully!')
    return train_losses, test_losses, acc
```

[10]:
```
# Start Training
train_losses, valid_losses, acc = train_and_test(50)
```

```
Model Training started…
 epoch 1 batch 10 completed
 epoch 1 batch 20 completed
 epoch 1 batch 30 completed
 epoch 1 batch 40 completed
 epoch 1 batch 50 completed
 epoch 1 batch 60 completed
 epoch 1 batch 70 completed
 epoch 1 batch 80 completed
 epoch 1 batch 90 completed
 epoch 1 batch 100 completed
 epoch 1 batch 110 completed
 epoch 1 batch 120 completed
 epoch 1 batch 130 completed
 epoch 1 batch 140 completed
 epoch 1 batch 150 completed
Epoch: 1/50.. Training Loss: 1.038.. Valid Loss: 0.519.. Valid Accuracy: 0.807
 epoch 2 batch 10 completed
```

```
 epoch 2 batch 20 completed
 epoch 2 batch 30 completed
 epoch 2 batch 40 completed
 epoch 2 batch 50 completed
 epoch 2 batch 60 completed
 epoch 2 batch 70 completed
 epoch 2 batch 80 completed
 epoch 2 batch 90 completed
 epoch 2 batch 100 completed
 epoch 2 batch 110 completed
 epoch 2 batch 120 completed
 epoch 2 batch 130 completed
 epoch 2 batch 140 completed
 epoch 2 batch 150 completed
Epoch: 2/50.. Training Loss: 0.467.. Valid Loss: 0.598.. Valid Accuracy: 0.802
 epoch 3 batch 10 completed
 epoch 3 batch 20 completed
 epoch 3 batch 30 completed
 epoch 3 batch 40 completed
 epoch 3 batch 50 completed
 epoch 3 batch 60 completed
 epoch 3 batch 70 completed
 epoch 3 batch 80 completed
 epoch 3 batch 90 completed
 epoch 3 batch 100 completed
 epoch 3 batch 110 completed
 epoch 3 batch 120 completed
 epoch 3 batch 130 completed
 epoch 3 batch 140 completed
 epoch 3 batch 150 completed
Epoch: 3/50.. Training Loss: 0.215.. Valid Loss: 0.727.. Valid Accuracy: 0.800
 epoch 4 batch 10 completed
 epoch 4 batch 20 completed
 epoch 4 batch 30 completed
 epoch 4 batch 40 completed
 epoch 4 batch 50 completed
 epoch 4 batch 60 completed
 epoch 4 batch 70 completed
 epoch 4 batch 80 completed
 epoch 4 batch 90 completed
 epoch 4 batch 100 completed
 epoch 4 batch 110 completed
 epoch 4 batch 120 completed
 epoch 4 batch 130 completed
 epoch 4 batch 140 completed
 epoch 4 batch 150 completed
Epoch: 4/50.. Training Loss: 0.111.. Valid Loss: 0.940.. Valid Accuracy: 0.815
 epoch 5 batch 10 completed
```

```
 epoch 5 batch 20 completed
 epoch 5 batch 30 completed
 epoch 5 batch 40 completed
 epoch 5 batch 50 completed
 epoch 5 batch 60 completed
 epoch 5 batch 70 completed
 epoch 5 batch 80 completed
 epoch 5 batch 90 completed
 epoch 5 batch 100 completed
 epoch 5 batch 110 completed
 epoch 5 batch 120 completed
 epoch 5 batch 130 completed
 epoch 5 batch 140 completed
 epoch 5 batch 150 completed
Epoch: 5/50.. Training Loss: 0.082.. Valid Loss: 0.850.. Valid Accuracy: 0.836
 epoch 6 batch 10 completed
 epoch 6 batch 20 completed
 epoch 6 batch 30 completed
 epoch 6 batch 40 completed
 epoch 6 batch 50 completed
 epoch 6 batch 60 completed
 epoch 6 batch 70 completed
 epoch 6 batch 80 completed
 epoch 6 batch 90 completed
 epoch 6 batch 100 completed
 epoch 6 batch 110 completed
 epoch 6 batch 120 completed
 epoch 6 batch 130 completed
 epoch 6 batch 140 completed
 epoch 6 batch 150 completed
Epoch: 6/50.. Training Loss: 0.028.. Valid Loss: 0.913.. Valid Accuracy: 0.843
Early stopping triggered.
Training Completed Successfully!
```

[ ]:

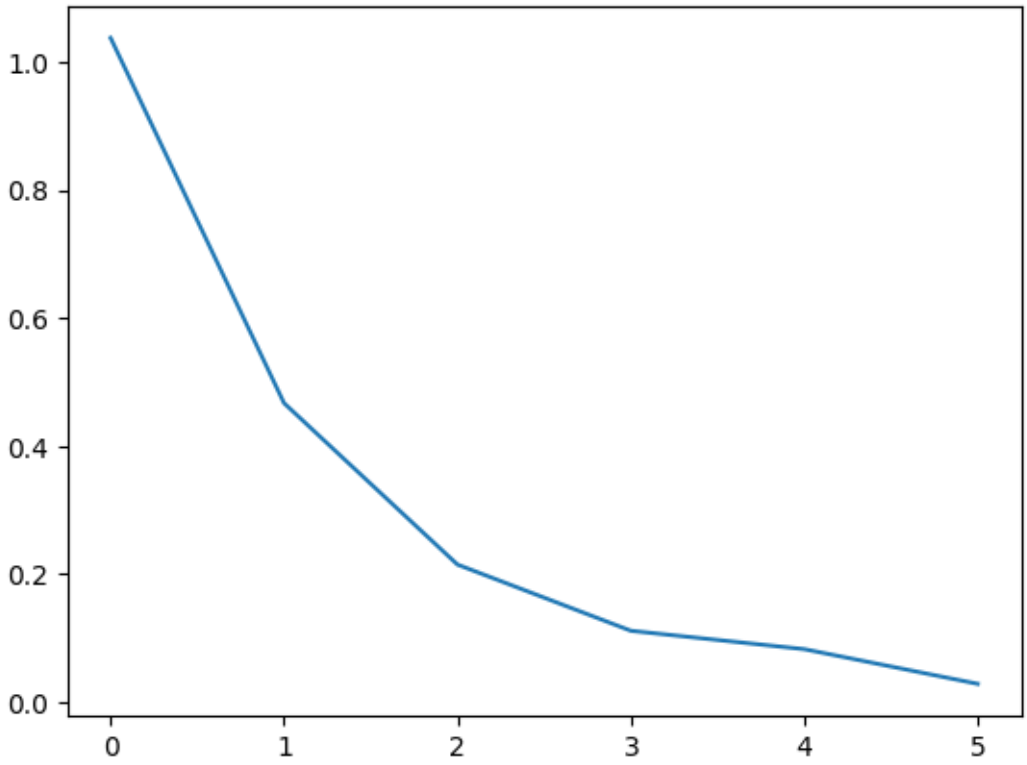
[11]:
```python
# Plotting training curves
plt.plot(train_losses, label='train_loss')
plt.plot(valid_losses, label='valid_loss')  # Directly use valid_losses
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend(frameon=False)
plt.show()

plt.plot(acc, label='accuracy')  # Assuming acc is also in float format
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
```

```
plt.legend(frameon=False)
plt.show()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[11], line 3
      1 # Plotting training curves
      2 plt.plot(train_losses, label='train_loss')
----> 3 plt.plot([x.cpu().numpy() for x in valid_losses], label='valid_loss')
      4 plt.xlabel("Epochs")
      5 plt.ylabel("Loss")

AttributeError: 'float' object has no attribute 'cpu'
```



[ ]:
```
# Define class names
class_list = ['No DR', 'Mild', 'Moderate', 'Severe', 'Proliferative']

# Function to generate graphical confusion matrix
def generate_confusion_matrix(model, dataloader):
    y_true, y_pred = [], []
    with torch.no_grad():
```

```
        model.eval()
        for images, labels in dataloader:
            images, labels = images.to(device), labels.to(device)
            logps = model(images)
            ps = torch.exp(logps)
            top_p, top_class = ps.topk(1, dim=1)
            y_true.extend(labels.cpu().numpy())
            y_pred.extend(top_class.cpu().numpy())

    # Create confusion matrix
    cm = confusion_matrix(y_true, [x[0] for x in y_pred])
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_list,␣
↪yticklabels=class_list)
    plt.title('Confusion Matrix')
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.show()
```

[ ]:
```
# Generate confusion matrix for the validation set
generate_confusion_matrix(model, validloader)
```

[ ]:
```
# Load the model
model_path = "/home/gcekcse/Documents/ML_Project_hk/data/models/classifier_4.pt"
checkpoint = torch.load(model_path)
model.load_state_dict(checkpoint['model_state_dict'])
model.eval()  # Set model to evaluation mode

# Generate confusion matrix for the test set
generate_confusion_matrix(model, validloader)
```

[ ]:
```

```