Vision transformer

March 14, 2025

[1]:
```
import os
import random
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
import numpy as np
import matplotlib.pyplot as plt
from torchvision import transforms
from sklearn.metrics import confusion_matrix, classification_report, roc_curve,␣
 ↪auc
import seaborn as sns
import timm


seed = 42
random.seed(seed)
np.random.seed(seed)
torch.manual_seed(seed)
if torch.cuda.is_available():
    torch.cuda.manual_seed_all(seed)

# Set CUDA device
os.environ["CUDA_VISIBLE_DEVICES"] = "0"
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print("Using device:", device)


torch.cuda.empty_cache()
print("GPU Memory after clearing:", torch.cuda.memory_allocated(device) /␣
 ↪1024**2, "MiB")
```

```
Using device: cuda
GPU Memory after clearing: 0.0 MiB
```

[2]:
```
class PneumoniaDataset(Dataset):
    def __init__(self, images, labels, transform=None):
        self.images = images
        self.labels = labels
        self.transform = transform

    def __len__(self):
        return self.images.shape[0]

    def __getitem__(self, idx):
        img = self.images[idx]
        if img.ndim == 2:
            img = np.expand_dims(img, axis=-1)
        img = np.repeat(img, 3, axis=-1)  # Grayscale to RGB
        img = np.transpose(img, (2, 0, 1))
        img = torch.tensor(img, dtype=torch.float32)
        if self.transform:
            img_pil = transforms.ToPILImage()(img)
            img_pil = self.transform(img_pil)
            img = transforms.ToTensor()(img_pil)

        label = torch.tensor(self.labels[idx].item(), dtype=torch.float32)  #␣
 ↪Convert [1] to scalar
        return img, label


data_path = '/home/rkalyanakumar/.medmnist/pneumoniamnist_224.npz'
data = np.load(data_path)

# Extract datasets
X_train, y_train = data['train_images'], data['train_labels']
X_val, y_val     = data['val_images'], data['val_labels']
X_test, y_test   = data['test_images'], data['test_labels']

# Normalize images
X_train = X_train.astype('float32') / 255.0
X_val   = X_val.astype('float32')   / 255.0
X_test  = X_test.astype('float32')  / 255.0

# Data augmentation
train_transform = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
    transforms.RandomResizedCrop(224, scale=(0.9, 1.1)),
])


train_dataset_basic = PneumoniaDataset(X_train, y_train, transform=None)
train_dataset_aug = PneumoniaDataset(X_train, y_train,␣
 ↪transform=train_transform)
```

```
val_dataset = PneumoniaDataset(X_val, y_val, transform=None)
test_dataset = PneumoniaDataset(X_test, y_test, transform=None)


batch_size = 8
train_loader_basic = DataLoader(train_dataset_basic, batch_size=batch_size,␣
 ↪shuffle=True)
train_loader_aug = DataLoader(train_dataset_aug, batch_size=batch_size,␣
 ↪shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

print("Train dataset size:", len(train_dataset_basic))
print("Validation dataset size:", len(val_dataset))
print("Test dataset size:", len(test_dataset))
```

```
Train dataset size: 4708
Validation dataset size: 524
Test dataset size: 624
```

[3]:

```
# Load  ViT-Small
model = timm.create_model('vit_small_patch16_224', pretrained=True,␣
 ↪num_classes=1)


model = model.to(device)

# Define optimizer and loss function
optimizer = optim.Adam(model.parameters(), lr=3e-5)
criterion = nn.BCEWithLogitsLoss()

print("GPU Memory after model load:", torch.cuda.memory_allocated(device) /␣
 ↪1024**2, "MiB")
```

```
GPU Memory after model load: 82.96240234375 MiB
```

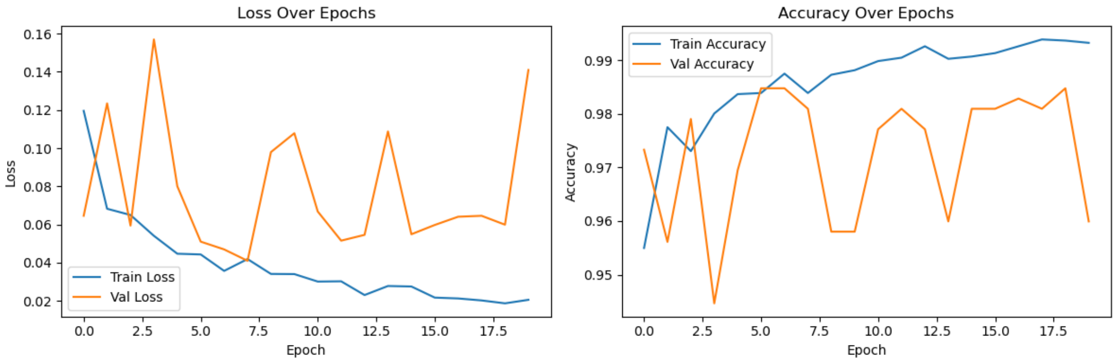
[4]:

```
def train_model(model, train_loader, val_loader, criterion, optimizer,␣
 ↪num_epochs=5):
    train_losses, val_losses = [], []
    train_accs, val_accs = [], []

    for epoch in range(num_epochs):
        # Training phase
        model.train()
        running_loss, correct, total = 0.0, 0, 0
        for inputs, labels in train_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(inputs).squeeze()
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item() * inputs.size(0)
            preds = (torch.sigmoid(outputs) > 0.5).float()
            correct += (preds == labels).sum().item()
            total += labels.size(0)

        epoch_loss = running_loss / total
        epoch_acc = correct / total
        train_losses.append(epoch_loss)
        train_accs.append(epoch_acc)

        # Validation phase
        model.eval()
        val_loss, correct, total = 0.0, 0, 0
        with torch.no_grad():
            for inputs, labels in val_loader:
                inputs, labels = inputs.to(device), labels.to(device)
                outputs = model(inputs).squeeze()
                loss = criterion(outputs, labels)
                val_loss += loss.item() * inputs.size(0)
                preds = (torch.sigmoid(outputs) > 0.5).float()
                correct += (preds == labels).sum().item()
                total += labels.size(0)

        val_loss = val_loss / total
        val_acc = correct / total
        val_losses.append(val_loss)
        val_accs.append(val_acc)

        print(f"Epoch {epoch+1}/{num_epochs} - "
              f"Train Loss: {epoch_loss:.4f}, Train Acc: {epoch_acc:.4f}, "
              f"Val Loss: {val_loss:.4f}, Val Acc: {val_acc:.4f}")

        torch.cuda.empty_cache()

    return train_losses, val_losses, train_accs, val_accs

# Train the model
num_epochs = 20
train_losses, val_losses, train_accs, val_accs = train_model(
    model, train_loader_aug, val_loader, criterion, optimizer, num_epochs
)
```

```python
# Plot results
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(train_losses, label='Train Loss')
plt.plot(val_losses, label='Val Loss')
plt.title('Loss Over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.subplot(1, 2, 2)
plt.plot(train_accs, label='Train Accuracy')
plt.plot(val_accs, label='Val Accuracy')
plt.title('Accuracy Over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.tight_layout()
plt.show()
```

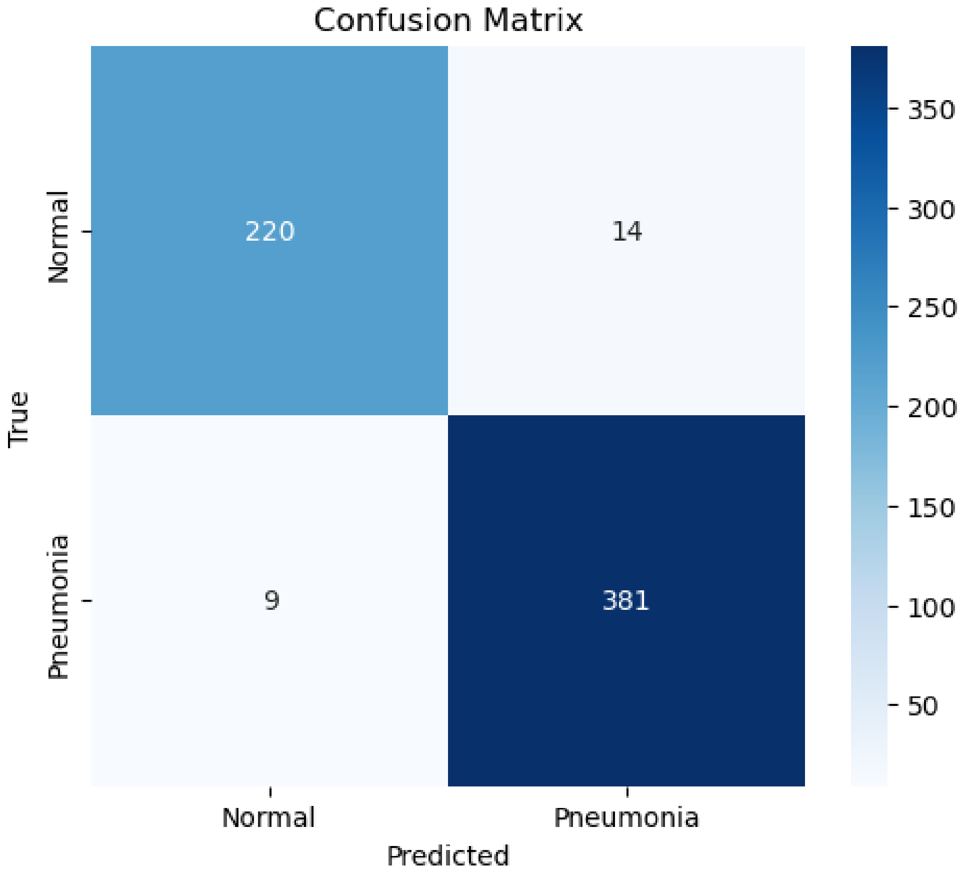
```
Epoch 1/20 - Train Loss: 0.1195, Train Acc: 0.9550, Val Loss: 0.0646, Val Acc:
0.9733
Epoch 2/20 - Train Loss: 0.0682, Train Acc: 0.9775, Val Loss: 0.1235, Val Acc:
0.9561
Epoch 3/20 - Train Loss: 0.0650, Train Acc: 0.9730, Val Loss: 0.0594, Val Acc:
0.9790
Epoch 4/20 - Train Loss: 0.0541, Train Acc: 0.9800, Val Loss: 0.1570, Val Acc:
0.9447
Epoch 5/20 - Train Loss: 0.0447, Train Acc: 0.9836, Val Loss: 0.0800, Val Acc:
0.9695
Epoch 6/20 - Train Loss: 0.0443, Train Acc: 0.9839, Val Loss: 0.0510, Val Acc:
0.9847
Epoch 7/20 - Train Loss: 0.0357, Train Acc: 0.9875, Val Loss: 0.0469, Val Acc:
0.9847
Epoch 8/20 - Train Loss: 0.0419, Train Acc: 0.9839, Val Loss: 0.0410, Val Acc:
0.9809
Epoch 9/20 - Train Loss: 0.0341, Train Acc: 0.9873, Val Loss: 0.0979, Val Acc:
0.9580
Epoch 10/20 - Train Loss: 0.0340, Train Acc: 0.9881, Val Loss: 0.1078, Val Acc:
0.9580
Epoch 11/20 - Train Loss: 0.0301, Train Acc: 0.9898, Val Loss: 0.0668, Val Acc:
0.9771
Epoch 12/20 - Train Loss: 0.0302, Train Acc: 0.9904, Val Loss: 0.0515, Val Acc:
0.9809
Epoch 13/20 - Train Loss: 0.0230, Train Acc: 0.9926, Val Loss: 0.0546, Val Acc:
0.9771
Epoch 14/20 - Train Loss: 0.0278, Train Acc: 0.9902, Val Loss: 0.1088, Val Acc:
```

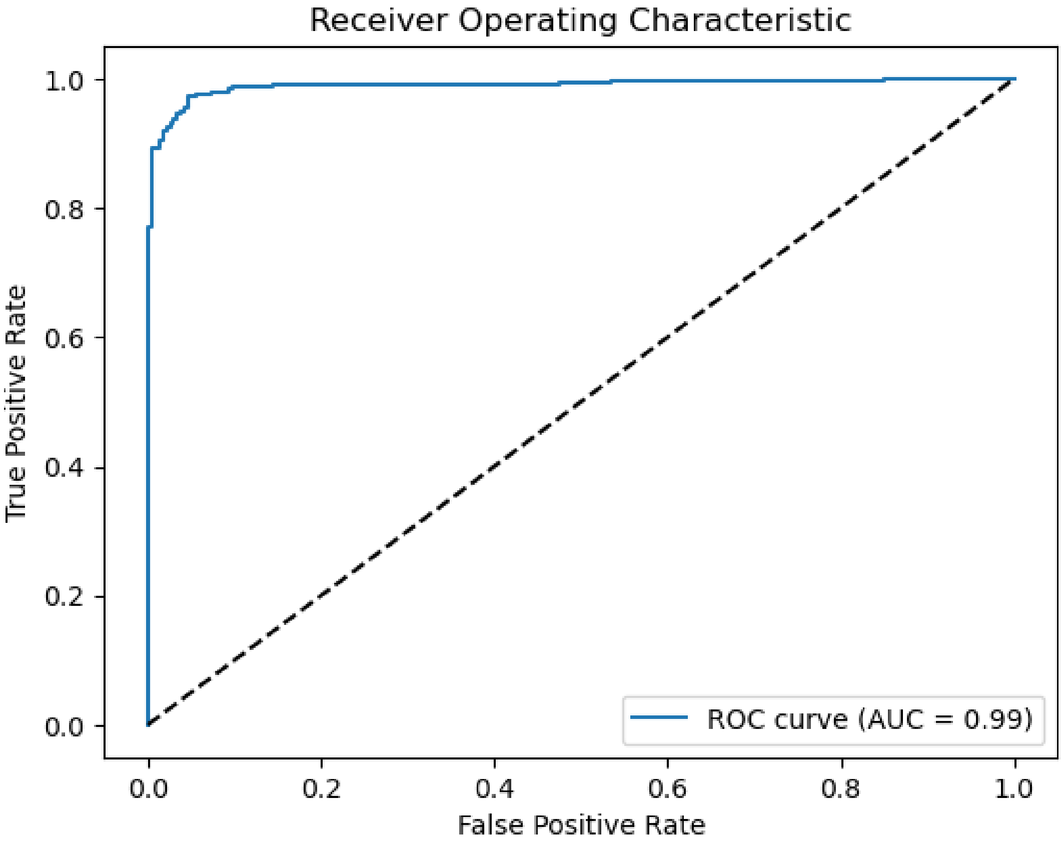
```
0.9599
Epoch 15/20 - Train Loss: 0.0275, Train Acc: 0.9907, Val Loss: 0.0549, Val Acc:
0.9809
Epoch 16/20 - Train Loss: 0.0217, Train Acc: 0.9913, Val Loss: 0.0597, Val Acc:
0.9809
Epoch 17/20 - Train Loss: 0.0212, Train Acc: 0.9926, Val Loss: 0.0641, Val Acc:
0.9828
Epoch 18/20 - Train Loss: 0.0202, Train Acc: 0.9938, Val Loss: 0.0646, Val Acc:
0.9809
Epoch 19/20 - Train Loss: 0.0187, Train Acc: 0.9936, Val Loss: 0.0599, Val Acc:
0.9847
Epoch 20/20 - Train Loss: 0.0205, Train Acc: 0.9932, Val Loss: 0.1410, Val Acc:
0.9599
```

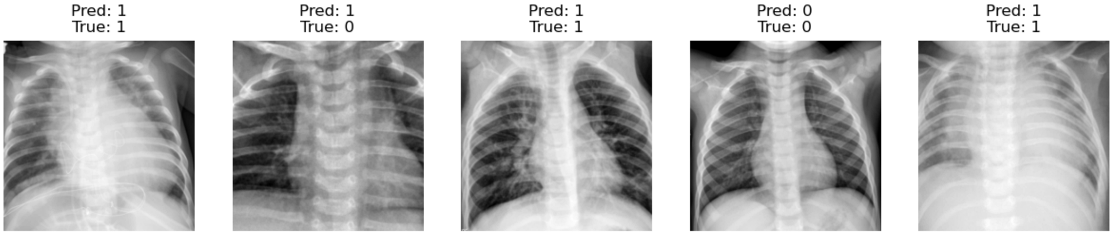


[5]:
```
def evaluate_model(model, test_loader, criterion):
    model.eval()
    test_loss, correct, total = 0.0, 0, 0
    all_preds, all_labels, all_probs = [], [], []

    with torch.no_grad():
        for inputs, labels in test_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs).squeeze()
            loss = criterion(outputs, labels)
            test_loss += loss.item() * inputs.size(0)
            probs = torch.sigmoid(outputs)
            preds = (probs > 0.5).float()
            correct += (preds == labels).sum().item()
            total += labels.size(0)
            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())
            all_probs.extend(probs.cpu().numpy())

    test_loss = test_loss / total
    test_acc = correct / total
    print(f"Test Loss: {test_loss:.4f}, Test Accuracy: {test_acc:.4f}")

    # Confusion matrix
    cm = confusion_matrix(all_labels, all_preds)
    plt.figure(figsize=(6, 5))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=['Normal', 'Pneumonia'], yticklabels=['Normal',␣
↪'Pneumonia'])
    plt.title('Confusion Matrix')
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.show()

    # ROC Curve
    fpr, tpr, _ = roc_curve(all_labels, all_probs)
    roc_auc = auc(fpr, tpr)
    plt.figure()
    plt.plot(fpr, tpr, label=f'ROC curve (AUC = {roc_auc:.2f})')
    plt.plot([0, 1], [0, 1], 'k--')
    plt.title('Receiver Operating Characteristic')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.legend()
    plt.show()

    return all_preds, all_labels


preds, labels = evaluate_model(model, test_loader, criterion)

# Visualize predictions
images, labels = next(iter(test_loader))
images, labels = images.to(device), labels.to(device)
outputs = model(images).squeeze()
preds = (torch.sigmoid(outputs) > 0.5).float()
images, labels, preds = images.cpu(), labels.cpu(), preds.cpu()

plt.figure(figsize=(15, 5))
for i in range(5):
    plt.subplot(1, 5, i+1)
    plt.imshow(images[i][0], cmap='gray')
    plt.title(f"Pred: {int(preds[i])}\nTrue: {int(labels[i])}")
    plt.axis('off')
plt.show()
```

```
Test Loss: 0.1530, Test Accuracy: 0.9631
```


Confusion Matrix
True
Normal
Pneumonia
220
14
9
381
Normal
Pneumonia
Predicted
350
300
250
200
150
100
50


Receiver Operating Characteristic
True Positive Rate
1.0
0.8
0.6
0.4
0.2
0.0
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (AUC = 0.99)


Pred: 1
True: 1
Pred: 1
True: 0
Pred: 1
True: 1
Pred: 0
True: 0
Pred: 1
True: 1