HOMEWORK 7 - EE541 :

Finetuned model

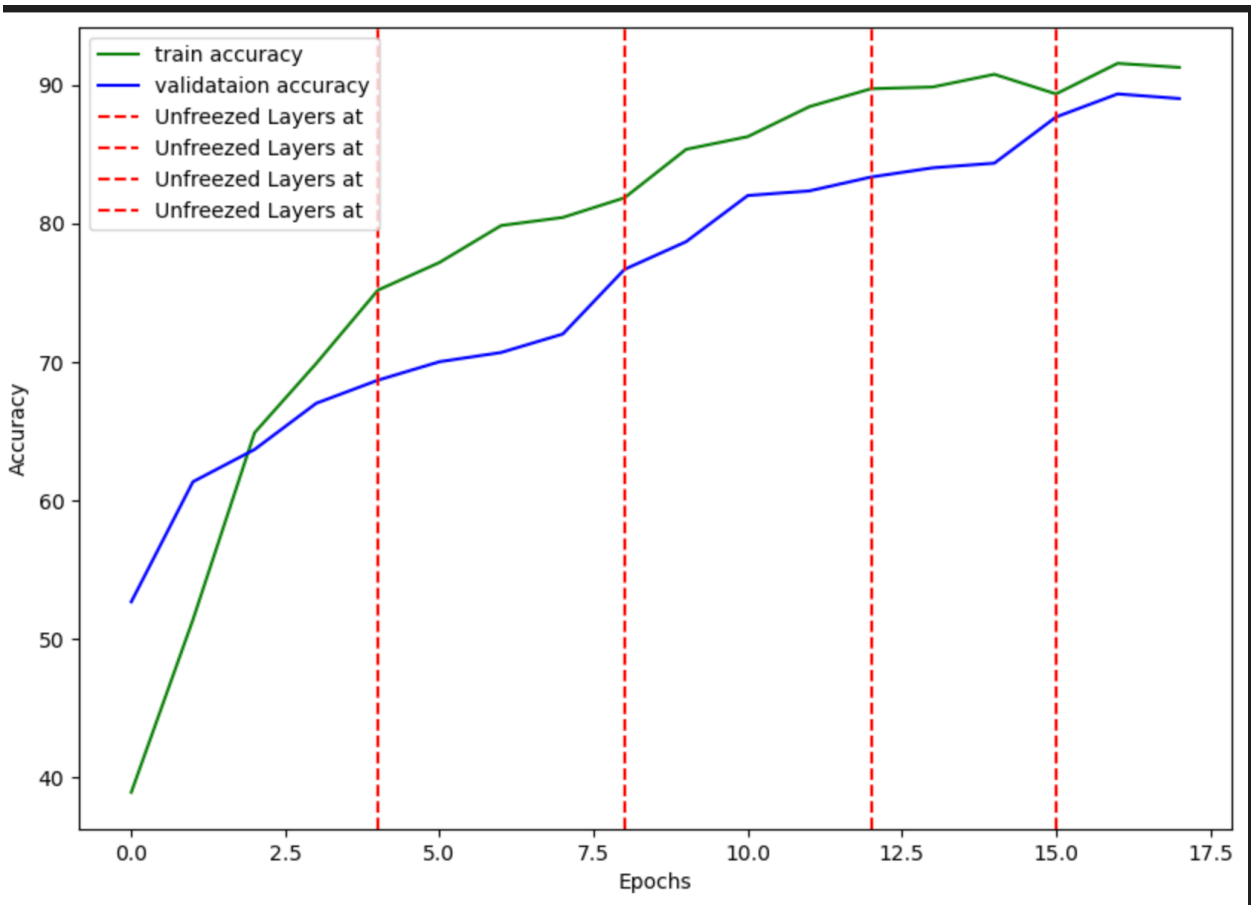
- This model was trained using three learning rates of 1e-1, 1e-2,1e-4 where 1e-4 seemed to have performed the best.

```
Epoch : 0/100
    Training loss : 0.0643329918384552 | Training accuracy : 98.83333587646484
    Validation loss : 0.0009710222366265953| Validation accuracy : 0.9833333492279053
Epoch : 1/100
    Training loss : 0.07109515368938446 | Training accuracy : 98.33333587646484
    Validation loss : 0.0016050420235842466| Validation accuracy : 0.9666666388511658
Epoch : 2/100
    Training loss : 0.04839983582496643 | Training accuracy : 99.16666412353516
    Validation loss : 0.0007674589869566262| Validation accuracy : 0.9833333492279053
Epoch : 3/100
    Training loss : 0.048701170831918716 | Training accuracy : 99.125
    Validation loss : 0.0008434580522589386| Validation accuracy : 0.9866666793823242
Epoch : 4/100
    Training loss : 0.03597068414092064 | Training accuracy : 99.29166412353516
    Validation loss : 0.000856335274875164| Validation accuracy : 0.9866666793823242
Epoch : 5/100
    Training loss : 0.03702739626169205 | Training accuracy : 99.625
    Validation loss : 0.0010632060002535582| Validation accuracy : 0.9833333492279053
```

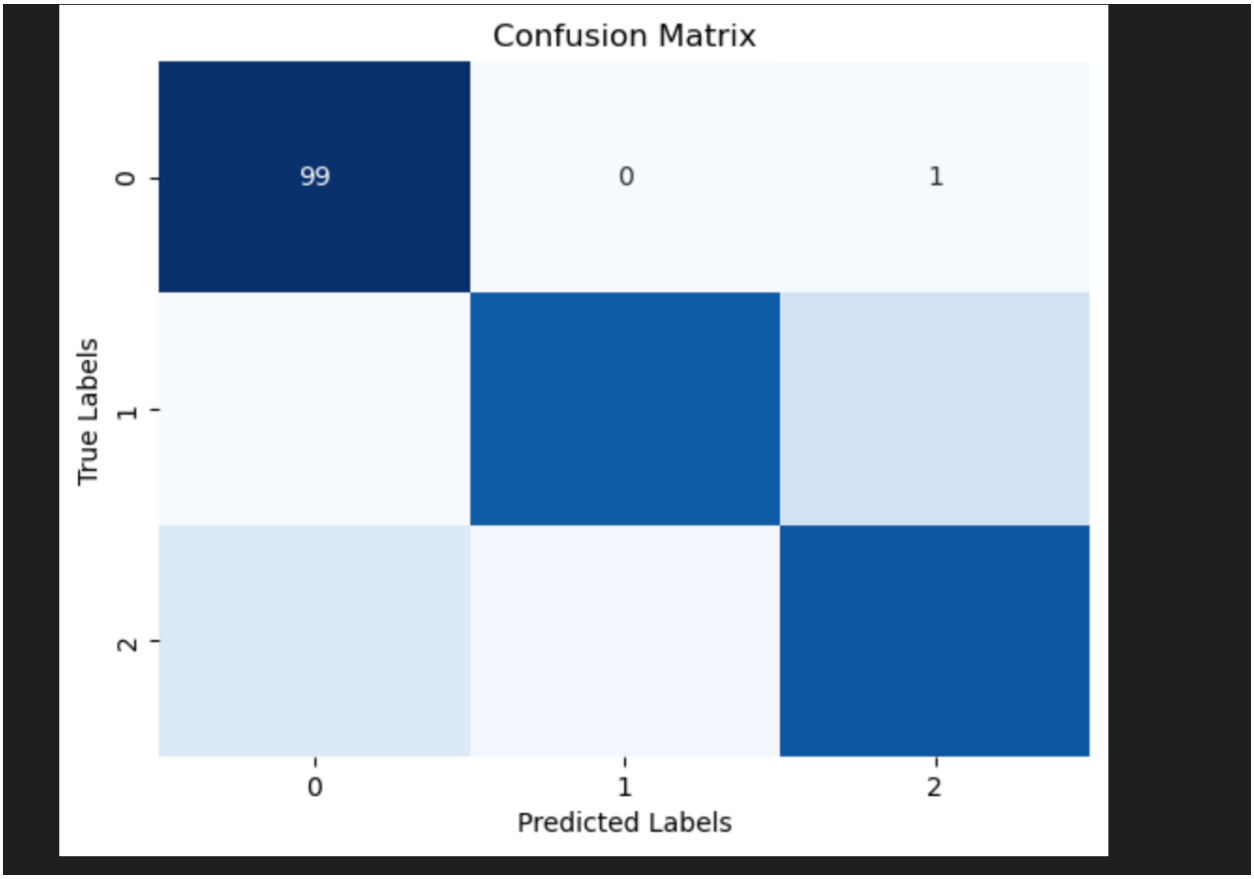
This model using 1e-4 lr also overfits just at epoch 4

Freezed model

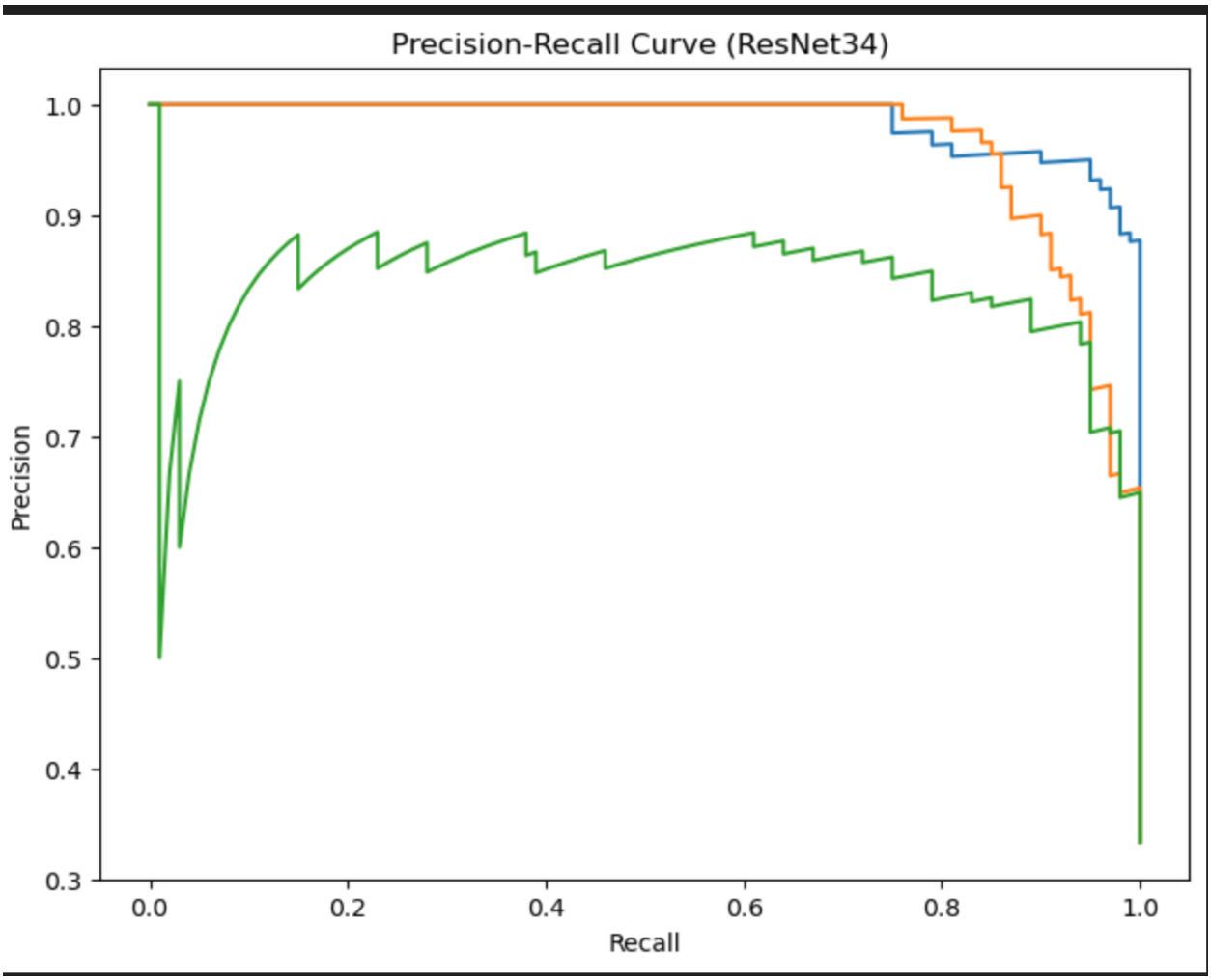
This model best runs with lr=0.0001



Generate a confusion matrix to show inter-class error rates.



Create a precision-recall curve for each class. Calculate the precision and recall for each class by treating it prediciton as a binary classification (i.e., one-vs-many). Then plot the P-R curves on the same plot. You may use preprocessing.label binarize and metrics.precision recall curve from sklearn.



CODE:

```
import numpy as np
import os
from torchvision import transforms
import shutil
import torch, torchvision
from torchvision import datasets
from torch.utils.data import DataLoader, random_split
import torchvision.models as models
import torch.optim as optim
import torch.nn as nn
import random
from torchvision import transforms
from torch.optim.lr_scheduler import ReduceLROnPlateau
```

```
!pip  install split-folders
import splitfolders
# os.makedirs('Validate')
all_files = os.listdir('S1_Raw_Photographs_Full_Study')

initial_data_path=r'/Users/rakshekarajakumar/Documents/DEEP
LEARNING/EE541_HW7/S1_Raw_Photographs_Full_Study'
folder_path=r'/Users/rakshekarajakumar/Documents/DEEP
LEARNING/EE541_HW7/segregated_images'

random.shuffle(all_files)
# for file in all_files:
#     if file.startswith("Ethanol"):
#         label=file.split('_')[0]
#         image_path= os.path.join(data_path, label)
#         os.makedirs(image_path, exist_ok=True)
#         shutil.copy(os.path.join(initial_data_path, file), image_path)

#     if file.startswith("Propanol"):
#         label=file.split('_')[0]
#         image_path= os.path.join(data_path, label)
#         os.makedirs(image_path, exist_ok=True)
#         shutil.copy(os.path.join(initial_data_path, file), image_path)

#     if file.startswith("Pentane"):
#         label=file.split('_')[0]
#         image_path= os.path.join(data_path, label)
#         os.makedirs(image_path, exist_ok=True)
#         shutil.copy(os.path.join(initial_data_path, file), image_path)

for file in all_files:
    if file.endswith('.JPG'):
        label= file.split('_')[0]
        image_new_path= os.path.join(folder_path,label)
        os.makedirs(image_new_path, exist_ok=True)
        image_old_path= os.path.join(initial_data_path,file)
        shutil.copy(image_old_path, image_new_path)
splitfolders.ratio('segregated_images',seed=1337, output="TrainTestVal-Splitted",
ratio=(0.8, 0.1, 0.1))
train_transforms = transforms.Compose([
    transforms.Resize(size=(224,224)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomVerticalFlip(),
    transforms.RandomRotation(degrees=30),
    transforms.ToTensor(),
    transforms.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225)),
])
```

```
val_transforms = transforms.Compose([
    transforms.Resize(size=(224,224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225)),
])
# img = transforms(img)
TrainTestVal= os.listdir('TrainTestVal-Splitted')
# print(TrainTestVal)

# print(os.listdir(os.path.join(os.path.join('TrainTestVal-Splitted', 'test'),
'ethanol')))

# #accesssing all the images
# for eachset in TrainTestVal:
#     chemical_names= os.listdir(os.path.join('TrainTestVal-Splitted', eachset))
#     # print(eachset,chemicalname)
#     for chemical_name in chemical_names:
#         chem_images= os.listdir(os.path.join(os.path.join('TrainTestVal-Splitted',
eachset), chemical_name))
#         for chem_image in chem_images:
#             #print(eachset,chemical_name,chem_image)
#             chem_image=transforms(chem_image)
train_dataset = datasets.ImageFolder(root='TrainTestVal-Splitted/train',
transform=train_transforms)
test_dataset = datasets.ImageFolder(root='TrainTestVal-Splitted/test',
transform=val_transforms)
val_dataset = datasets.ImageFolder(root='TrainTestVal-Splitted/val',
transform=val_transforms)

train_dataloader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_dataloader = DataLoader(test_dataset, batch_size=64, shuffle=True)
val_dataloader= DataLoader(val_dataset,batch_size=64, shuffle=True)


# Load the pretrained ResNet-34 model
num_classes=3
r_model = models.resnet34(pretrained=True)

r_model.fc = nn.Linear(r_model.fc.in_features, num_classes)

# for param in r_model.parameters():
#     param.requires_grad = False

# for param in r_model.fc.parameters():
#     param.requires_grad = True


for param in r_model.parameters():
```

```
    param.requires_grad = False
for param in r_model.fc.parameters():
    param.requires_grad = True

# optimizer
params_to_optimize = list({'params' : r_model.fc.parameters()})
# optimizer = optim.SGD(params_to_optimize, lr=LEARNING_RATE, momentum=0.9)
optimizer = optim.SGD([
                {'params': r_model.fc.parameters()}
            ], lr=1e-4, momentum=0.9)
scheduler = ReduceLROnPlateau(optimizer, 'min', factor=0.1, verbose=True)
# loss function
criterion = nn.CrossEntropyLoss()
cpu_or_gpu = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
r_model = r_model.to(cpu_or_gpu)
#training
epochs=100
val_loss=[]
val_acc=[]
train_loss=[]
train_acc=[]

for epoch in range(epochs):

    #train dataset
    r_model.train()

    training_loss=0
    training_correct=0
    for inputs_img,labels in train_dataloader:
        inputs_img.to(cpu_or_gpu)
        labels.to(cpu_or_gpu)

        optimizer.zero_grad()

        outputs_lbl= r_model(inputs_img)
        _,preds= torch.max(outputs_lbl,1)
        loss= criterion(outputs_lbl,labels)

        loss.backward()
        optimizer.step()

        training_loss+= loss
        training_correct+= torch.sum(preds==labels)

    epoch_loss_training= training_loss/train_dataset.__len__() * 100
    epoch_accuracy_training= training_correct/train_dataset.__len__() * 100
```

```
    train_loss.append(epoch_loss_training)
    train_acc.append(epoch_accuracy_training)

    #val dataset
    r_model.eval()

    validation_loss=0
    validation_correct=0
    for inputs,labels in val_dataloader:
        inputs.to(cpu_or_gpu)
        labels.to(cpu_or_gpu)

        optimizer.zero_grad()

        outputs= r_model(inputs)
        _,preds= torch.max(outputs,1)
        loss= criterion(outputs,labels)

        validation_loss+=loss
        validation_correct+=torch.sum(preds==labels)

    epoch_loss_validation= validation_loss/val_dataset.__len__()
    epoch_accuracy_validation= validation_correct/val_dataset.__len__()

    val_loss.append(epoch_loss_validation)
    val_acc.append(epoch_accuracy_validation)

    print(f"Epoch : {epoch}/{epochs}")
    print(f"    Training loss : {epoch_loss_training} | Training accuracy :
{epoch_accuracy_training}")
    print(f"    Validation loss : {epoch_loss_validation}| Validation accuracy :
{epoch_accuracy_validation}")
#training
epochs=40
val_loss=[]
val_acc=[]
train_loss=[]
train_acc=[]

for epoch in range(epochs):

    #----freezing and unfreezing layers----#
    if epoch == 8:
        for param in r_model.layer4.parameters():
            param.requires_grad = True
    optimizer = optim.SGD([
                    {'params': r_model.fc.parameters()},
                    {'params': r_model.layer4.parameters(), 'lr': 1e-4}
```

```
                ], lr=1e-4, momentum=0.9)

    if epoch == 16:
        for param in r_model.layer3.parameters():
            param.requires_grad = True
        params_to_optimize.append({'params': r_model.layer3.parameters(), 'lr':
0.0001, 'weight_decay': 1e-4})
        optimizer = optim.SGD([
                        {'params': r_model.fc.parameters()},
                        {'params': r_model.layer4.parameters(), 'lr': 1e-4,
'weight_decay' : 1e-4},
                        {'params': r_model.layer3.parameters(), 'lr': 0.0001,
'weight_decay': 1e-4}
                    ], lr=1e-4, momentum=0.9)

    if epoch == 24:
        for param in r_model.layer2.parameters():
            param.requires_grad = True
        optimizer = optim.SGD([
                        {'params': r_model.fc.parameters()},
                        {'params': r_model.layer4.parameters(), 'lr': 1e-4,
'weight_decay' : 1e-4},
                        {'params': r_model.layer3.parameters(), 'lr': 0.0001,
'weight_decay': 1e-4},
                        {'params': r_model.layer2.parameters(), 'lr': 0.0001,
'weight_decay': 1e-4}
                    ], lr=1e-4, momentum=0.9)

    if epoch == 50:
        for param in r_model.layer1.parameters():
            param.requires_grad = True
        params_to_optimize.append({'params': r_model.layer1.parameters(), 'lr':
0.0001, 'weight_decay': 1e-4})
        optimizer = optim.SGD([
                        {'params': r_model.fc.parameters()},
                        {'params': r_model.layer4.parameters(), 'lr': 1e-4,
'weight_decay' : 1e-4},
                        {'params': r_model.layer3.parameters(), 'lr': 0.0001,
'weight_decay': 1e-4},
                        {'params': r_model.layer2.parameters(), 'lr': 0.0001,
'weight_decay': 1e-4},
                        {'params': r_model.layer1.parameters(), 'lr': 0.0001,
'weight_decay': 1e-4}
                    ], lr=1e-4, momentum=0.9)


    #----freezing and unfreezing layers----#
```

```
    #train dataset
    r_model.train()

    training_loss=0
    training_correct=0
    for inputs_img,labels in train_dataloader:
        inputs_img.to(cpu_or_gpu)
        labels.to(cpu_or_gpu)

        optimizer.zero_grad()

        outputs_lbl= r_model(inputs_img)
        _,preds= torch.max(outputs_lbl,1)
        loss= criterion(outputs_lbl,labels)

        loss.backward()
        optimizer.step()

        training_loss+= loss
        training_correct+= torch.sum(preds==labels)

    epoch_loss_training= training_loss/train_dataset.__len__()
    epoch_accuracy_training= training_correct/train_dataset.__len__() * 100

    train_loss.append(epoch_loss_training)
    train_acc.append(epoch_accuracy_training)

    #val dataset
    r_model.eval()

    validation_loss=0
    validation_correct=0
    for inputs,labels in val_dataloader:
        inputs.to(cpu_or_gpu)
        labels.to(cpu_or_gpu)

        optimizer.zero_grad()

        outputs= r_model(inputs)
        _,preds= torch.max(outputs,1)
        loss= criterion(outputs,labels)

        validation_loss+=loss
        validation_correct+=torch.sum(preds==labels)

    epoch_loss_validation= validation_loss/val_dataset.__len__()
    epoch_accuracy_validation= validation_correct/val_dataset.__len__() * 100
```

```
    val_loss.append(epoch_loss_validation)
    val_acc.append(epoch_accuracy_validation)

    print(f"Epoch : {epoch}/{epochs}")
    print(f"    Training loss : {epoch_loss_training} | Training accuracy :
{epoch_accuracy_training}")
    print(f"    Validation loss : {epoch_loss_validation}| Validation accuracy :
{epoch_accuracy_validation}")
import matplotlib.pyplot as plt
def save_plots(train_acc, valid_acc, train_loss, valid_loss):
    unfreeze = [4, 8, 12, 15]
    # accuracy plots
    plt.figure(figsize=(10, 7))
    plt.plot(train_acc, color='green', linestyle='-', label='train accuracy')
    plt.plot(valid_acc, color='blue', linestyle='-', label='validataion accuracy')
    for point in unfreeze:
        plt.axvline(x=point, color='red', linestyle='--', label='Unfreezed Layers at')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.savefig('accuracy.png')

    # loss plots
    plt.figure(figsize=(10, 7))
    plt.plot(
        train_loss, color='orange', linestyle='-', label='train loss')
    plt.plot(
        valid_loss, color='red', linestyle='-', label='validataion loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.savefig('loss.png')
save_plots(train_acc=train_acc, valid_acc=val_acc, train_loss=train_loss,
valid_loss=val_loss)
import seaborn as sns
from sklearn.metrics import confusion_matrix, precision_recall_curve

true_labels = []
predicted_labels = []

all_labels = []
probabilities = []
#calculating visualizations
r_model.eval()
with torch.no_grad():
    for inputs, labels in test_dataloader:
        outputs = r_model(inputs)
        probs = torch.nn.functional.softmax(outputs, dim=1)
```

```
        _, preds = torch.max(outputs, 1)
        probabilities.extend(probs.numpy())
        true_labels.extend(labels.numpy())
        predicted_labels.extend(preds.numpy())
conf_matrix = confusion_matrix(true_labels, predicted_labels)

# confusion matrix
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('Confusion Matrix')
plt.ylabel('True Labels')
plt.xlabel('Predicted Labels')

plt.show()


probabilities = np.array(probabilities)
true_labels = np.array(true_labels)
print(probabilities.shape, true_labels.shape)
true_labels_one_hot = np.eye(3)[true_labels]

plt.figure(figsize=(8, 6))

for class_index in range(3):
    precision, recall, _ = precision_recall_curve(
        true_labels_one_hot[:, class_index],
        probabilities[:, class_index]
    )
    plt.plot(recall, precision, label=f'Class {class_index}')

plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve (ResNet34) ')
plt.show()
from PIL import Image
print(r_model)
def hook_visualization(module, input, output):
    plt.figure(figsize=(15, 15))
    for i in range(output.size(1)):
        plt.subplot(8, 8, i + 1)
        plt.imshow(output[0, i].detach().cpu().numpy(), cmap="viridis")
        plt.axis("off")
    plt.show()


visual_layer = r_model.layer2[0].conv1
hook = visual_layer.register_forward_hook(hook_visualization)
image_path = 'TrainTestVal-Splitted/val/Ethanol/Ethanol_Full_0026.JPG'
image = Image.open(image_path)
```

```
input_image = val_transforms(image).unsqueeze(0)
xyz= r_model(input_image)
hook.remove()
epochs= 10

test_loss=[]
test_acc=[]
test_correct=0

for epoch in range(epochs):
    r_model.eval()

    epoch_loss_test=0
    epoch_acc_test=0

    for inputs,labels in test_dataloader:
        inputs.to(cpu_or_gpu)
        labels.to(cpu_or_gpu)

        optimizer.zero_grad()

        outputs= r_model(inputs)
        _,preds= torch.max(outputs,1)
        loss= criterion(outputs,labels)

        test_loss+=loss
        test_correct+=torch.sum(preds==labels)

    epoch_loss_test= test_loss/test_dataset.__len__()
    epoch_accuracy_test= test_correct/test_dataset.__len__() * 100

    test_loss.append(epoch_loss_test)
    test_acc.append(epoch_accuracy_test)

    print(f"epoch : {epoch}/{epochs}")
    print(f"test loss : {test_loss} | test accuracy : {test_acc}")
```