HW7

In [1]:
```
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision.models import resnet34
from torchvision.datasets import ImageFolder
import torchvision.transforms as transforms
from torch.utils.data import random_split, DataLoader, RandomSampler

torch.manual_seed(15)

transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])
])

dataset = ImageFolder(root="data/", transform=transform)

training_ratio=0.7
validation_ratio=0.15
testing_ratio=0.15

dataset_size = len(dataset)

train_size = int(training_ratio * dataset_size)
val_size = int(validation_ratio * dataset_size)
test_size = int(dataset_size-train_size-val_size)

train_set, val_set, test_set = random_split(dataset, [train_size, val_size, test

train_loader = DataLoader(train_set, batch_size=64, shuffle=True, num_workers=6)
val_loader = DataLoader(val_set, batch_size=64, shuffle=False, num_workers=6)
test_loader = DataLoader(test_set, batch_size=64, shuffle=False, num_workers=6)
```

In [2]:
```
device = 'cpu'
device = torch.device("cuda:0" if torch.cuda.is_available() else device)
device = torch.device("mps" if torch.backends.mps.is_available() else device)
```

Model

In [3]:
```
# https://github.com/pytorch/vision/blob/main/torchvision/models/resnet.py
# find, ResNet::forward, self.fc
num_classes = 3
model = resnet34(pretrained=True).to(device)
model.fc = nn.Linear(model.fc.in_features, num_classes)

#print(f"Device: {device}")
#print(f"Model: {model}")
```

```
d:\xy\app\anaconda\Anaconda\envs\ee541_work\Lib\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(
d:\xy\app\anaconda\Anaconda\envs\ee541_work\Lib\site-packages\torchvision\models
\_utils.py:223: UserWarning: Arguments other than a weight enum or `None` for 'we
ights' are deprecated since 0.13 and may be removed in the future. The current be
havior is equivalent to passing `weights=ResNet34_Weights.IMAGENET1K_V1`. You can
also use `weights=ResNet34_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
```

In [4]:

```
#test with baseline model
model.eval()
correct_test = 0
total_test = 0

with torch.no_grad():
    for inputs, labels in test_loader:
        inputs, labels = inputs.to(device), labels.to(device)
        outputs = model(inputs)
        _, predicted = outputs.max(1)
        total_test += labels.size(0)
        correct_test += predicted.eq(labels).sum().item()

baseline_accuracy = correct_test / total_test
print("Pretrained ResNet-34 accuracy on testing set:", baseline_accuracy)
```

```
Pretrained ResNet-34 accuracy on testing set: 0.49777777777777776
```

Training

In [5]:

```
def training(optimizer, criterion):
    total_loss = 0
    correct_train = 0
    total_train = 0
    for inputs, labels in train_loader: # iterate through the train loader
        inputs, labels = inputs.to(device), labels.to(device)  # move data to de
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        total_loss += loss.item() * inputs.size(0)

        _, predicted = torch.max(outputs, 1)
        total_train += labels.size(0)
        correct_train += (predicted == labels).sum().item()

    return total_loss, correct_train, total_train

def validation(criterion):
    total_val_loss = 0
    correct_val = 0
    total_val = 0

    with torch.no_grad():
        for inputs, labels in val_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
```

```
            val_loss = criterion(outputs, labels)
            total_val_loss += val_loss.item() * inputs.size(0)

            _, predicted = torch.max(outputs, 1)
            total_val += labels.size(0)
            correct_val += (predicted == labels).sum().item()

    epoch_val_loss = total_val_loss / len(val_loader.dataset)
    val_accuracy = correct_val / total_val

    return epoch_val_loss, val_accuracy
```

Step 1: Only unfreeze fully connected layer

In [6]:
```
#trainable_layers = [name for name, param in model.named_parameters() if param.r
#print("Trainable layers:")
#print(trainable_layers)

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

trainable_layers = [name for name, param in model.named_parameters() if param.re
print("Trainable layers:")
print(trainable_layers)
```

```
Trainable layers:
['fc.weight', 'fc.bias']
```

In [7]:
```
learning_rate=1e-4
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.fc.parameters(), lr=learning_rate)

num_epochs = 5

train_losses = []
val_losses = []
train_accuracies = []
val_accuracies = []

for epoch in range(num_epochs):
    #train
    model.train()
    total_loss, correct_train, total_train = training(optimizer, criterion)
    epoch_train_loss = total_loss / len(train_loader.dataset)
    train_losses.append(epoch_train_loss)
    train_accuracy = correct_train / total_train
    train_accuracies.append(train_accuracy)

    #validation
    model.eval()
    epoch_val_loss, val_accuracy = validation(criterion)
    val_losses.append(epoch_val_loss)
    val_accuracies.append(val_accuracy)

    # Print epoch statistics
    print(f"Epoch {epoch+1}, Train Loss: {epoch_train_loss}, Train Accuracy: {tr
```

```
Epoch 1, Train Loss: 1.1750801726749964, Train Accuracy: 0.28809523809523807, Val
Loss: 1.0031107298533122, Val Accuracy: 0.48444444444444446
Epoch 2, Train Loss: 0.9969805021513076, Train Accuracy: 0.5052380952380953, Val
Loss: 0.8848531362745496, Val Accuracy: 0.6355555555555555
Epoch 3, Train Loss: 0.8588830276897975, Train Accuracy: 0.6795238095238095, Val
Loss: 0.769407070212894, Val Accuracy: 0.7733333333333333
Epoch 4, Train Loss: 0.7537943951288859, Train Accuracy: 0.7652380952380953, Val
Loss: 0.6797376097573175, Val Accuracy: 0.8355555555555556
Epoch 5, Train Loss: 0.6759690221150716, Train Accuracy: 0.8019047619047619, Val
Loss: 0.609767620033688, Val Accuracy: 0.8622222222222222
```

Step 2: unfreeze layer4 and fully connected layer

In [8]:
```
for param in model.parameters():
    param.requires_grad = False

for param in model.layer4.parameters():
    param.requires_grad = True
for param in model.fc.parameters():
    param.requires_grad = True


trainable_layers = [name for name, param in model.named_parameters() if param.re
print("Trainable layers:")
print(trainable_layers)
```

```
Trainable layers:
['layer4.0.conv1.weight', 'layer4.0.bn1.weight', 'layer4.0.bn1.bias', 'layer4.0.c
onv2.weight', 'layer4.0.bn2.weight', 'layer4.0.bn2.bias', 'layer4.0.downsample.0.
weight', 'layer4.0.downsample.1.weight', 'layer4.0.downsample.1.bias', 'layer4.1.
conv1.weight', 'layer4.1.bn1.weight', 'layer4.1.bn1.bias', 'layer4.1.conv2.weigh
t', 'layer4.1.bn2.weight', 'layer4.1.bn2.bias', 'layer4.2.conv1.weight', 'layer4.
2.bn1.weight', 'layer4.2.bn1.bias', 'layer4.2.conv2.weight', 'layer4.2.bn2.weigh
t', 'layer4.2.bn2.bias', 'fc.weight', 'fc.bias']
```

In [9]:
```
learning_rate=1e-5
optimizer = optim.Adam(filter(lambda p: p.requires_grad, model.parameters()), lr

num_epochs = 5
for epoch in range(num_epochs):
    #train
    model.train()
    total_loss, correct_train, total_train = training(optimizer, criterion)
    epoch_train_loss = total_loss / len(train_loader.dataset)
    train_losses.append(epoch_train_loss)
    train_accuracy = correct_train / total_train
    train_accuracies.append(train_accuracy)

    #validation
    model.eval()
    epoch_val_loss, val_accuracy = validation(criterion)
    val_losses.append(epoch_val_loss)
    val_accuracies.append(val_accuracy)

    # Print epoch statistics
    print(f"Epoch {epoch+1}, Train Loss: {epoch_train_loss}, Train Accuracy: {tr
```

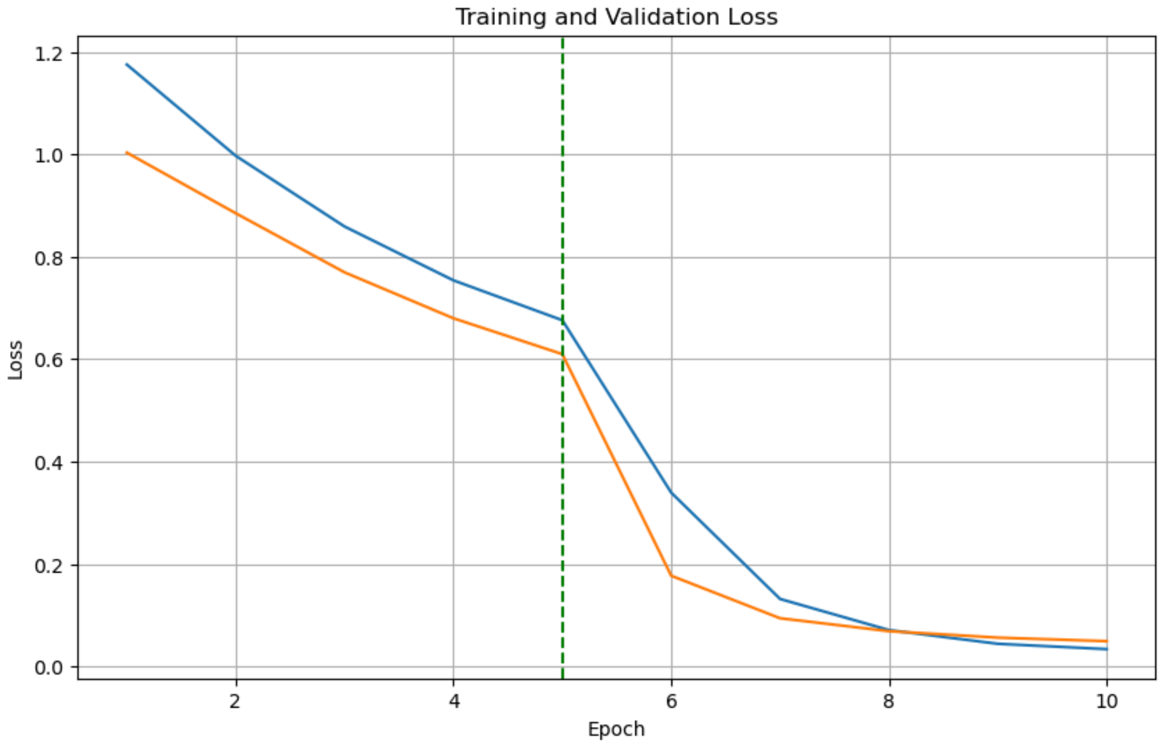
```
Epoch 1, Train Loss: 0.33939536310377577, Train Accuracy: 0.9161904761904762, Val
Loss: 0.17712022736668587, Val Accuracy: 0.9311111111111111
Epoch 2, Train Loss: 0.13176500215416864, Train Accuracy: 0.9628571428571429, Val
Loss: 0.09409621429526144, Val Accuracy: 0.9711111111111111
Epoch 3, Train Loss: 0.07100835974727358, Train Accuracy: 0.981904761904762, Val
Loss: 0.06883041349136167, Val Accuracy: 0.9777777777777777
Epoch 4, Train Loss: 0.04420292901850882, Train Accuracy: 0.9890476190476191, Val
Loss: 0.05624884502755271, Val Accuracy: 0.98
Epoch 5, Train Loss: 0.033807739452237175, Train Accuracy: 0.991904761904762, Val
Loss: 0.049334891597843836, Val Accuracy: 0.9844444444444445
```

Achieved great results, stop unfreezing.

Layer visualisation

In [10]:

```
import matplotlib.pyplot as plt

epochs = np.arange(1, 11)

plt.figure(figsize=(10, 6))
plt.plot(epochs, train_losses, label='Train Loss')
plt.plot(epochs, val_losses, label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.grid(True)
plt.axvline(x=5, color='g', linestyle='--')
plt.show()

# Plot accuracy curves
plt.figure(figsize=(10, 6))
plt.plot(epochs, train_accuracies, label='Train Accuracy')
plt.plot(epochs, val_accuracies, label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.grid(True)
plt.axvline(x=5, color='g', linestyle='--')
plt.show()
plt.tight_layout()
```



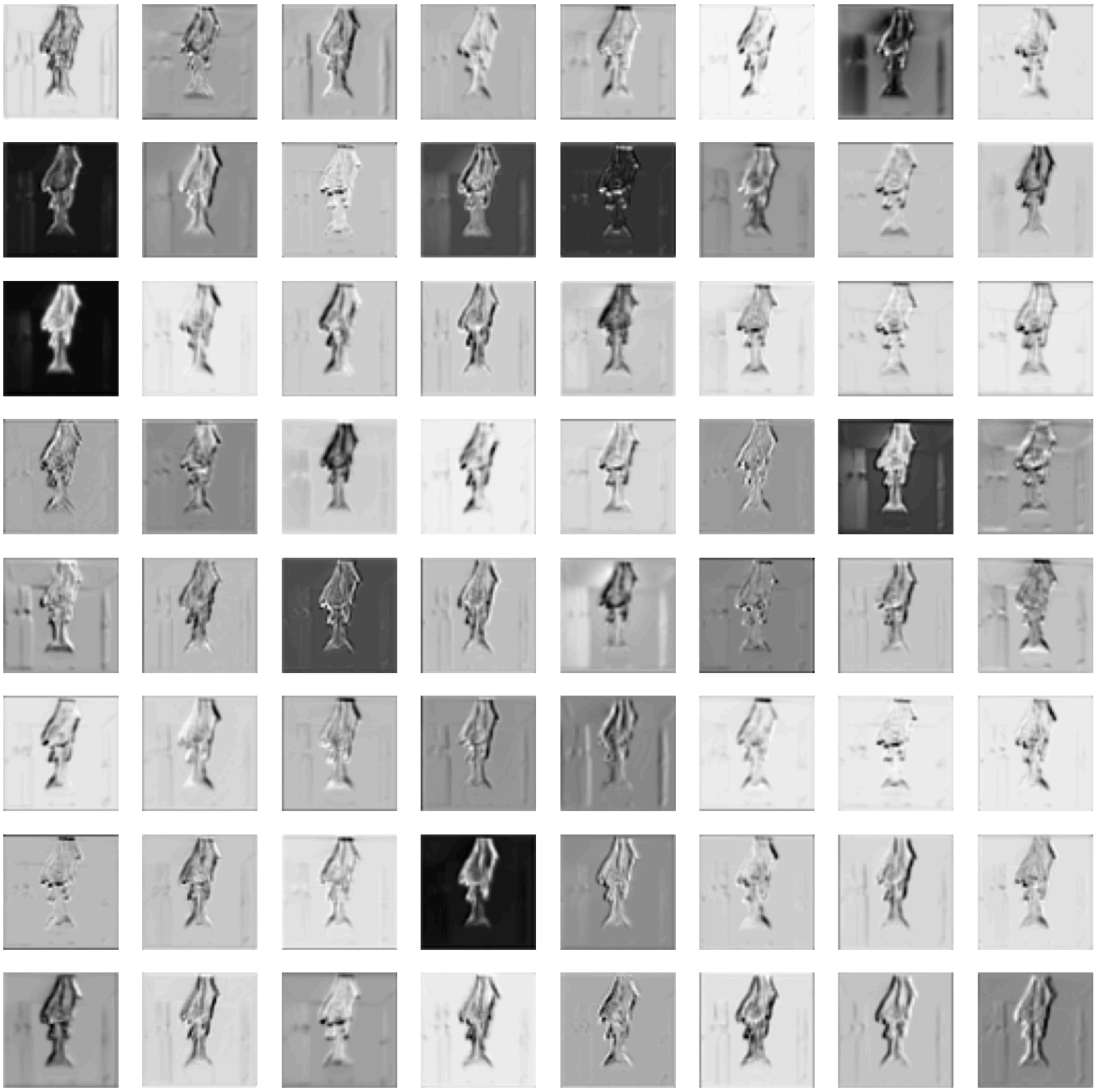

```
<Figure size 640x480 with 0 Axes>
```

In [16]:
```
random_sampler = RandomSampler(dataset)
random_index = next(iter(random_sampler))
random_image, random_label = dataset[random_index]

def visualize_hook(layer_name, modele, input, output):
    print(layer_name)
    plt.figure(figsize=(15, 15))
    num_subplots = min(output.size(1), 64)
    for i in range(num_subplots):
        plt.subplot(8, 8, i + 1)
        plt.imshow(output[0, i].detach().cpu().numpy(), cmap="gray")
        plt.axis("off")
    plt.show()
```

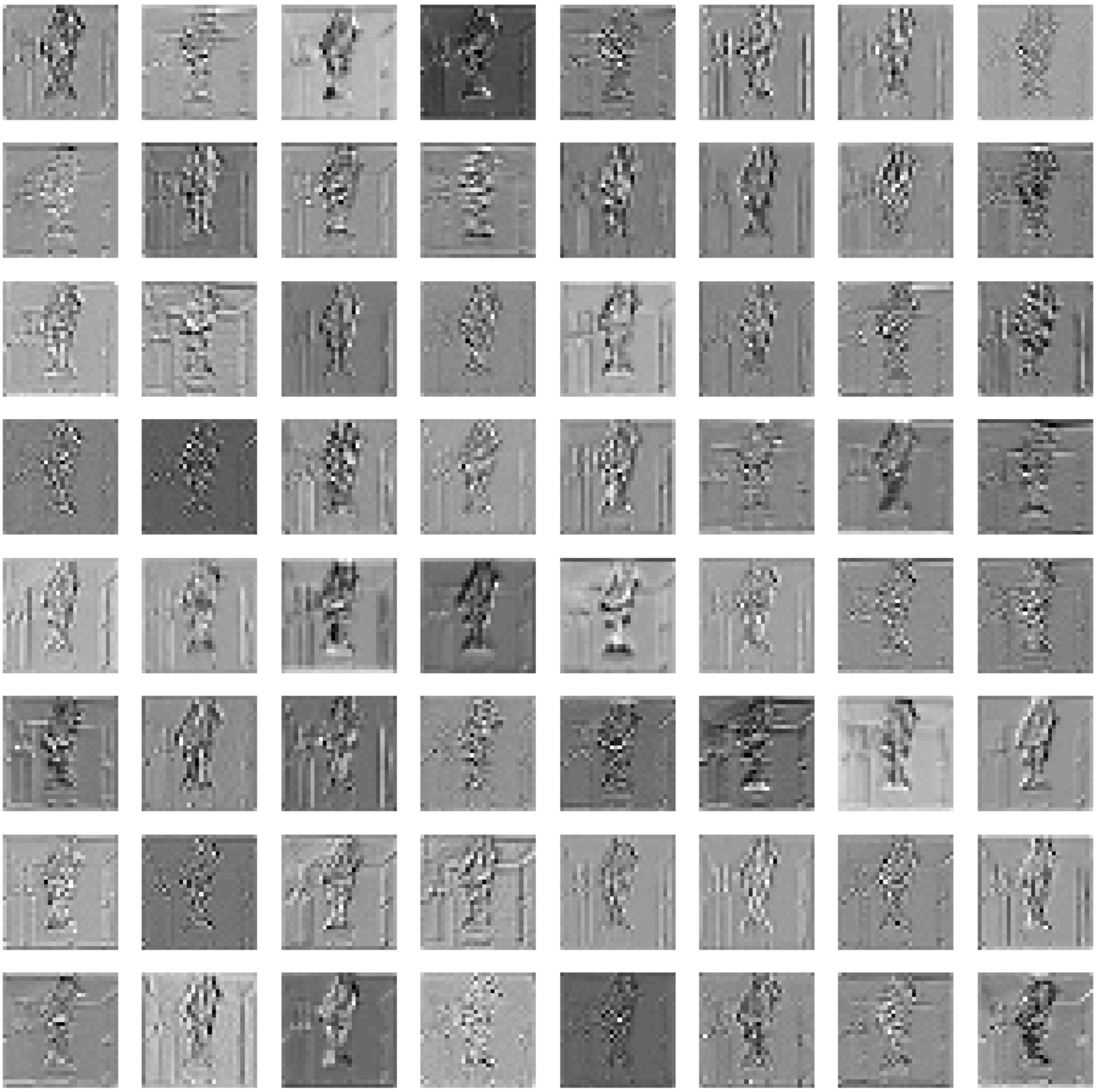
```
layer_to_visualize = [("conv1", model.conv1),
                      ("layer1.0.conv1", model.layer1[0].conv1),
                      ("layer2.0.conv1", model.layer2[0].conv1),
                      ("layer3.0.conv1", model.layer3[0].conv1),
                      ("layer4.0.conv1", model.layer4[0].conv1),
                      ("layer4.2.conv2", model.layer4[2].conv2)]

hooks = []
for layer_name, layer in layer_to_visualize:
    hook = layer.register_forward_hook(lambda module, input, output, name=layer_
    hooks.append(hook)

image = random_image.unsqueeze(0).to(device)
_ = model(image)

for hook in hooks:
    hook.remove()
```

conv1

layer1.0.conv1

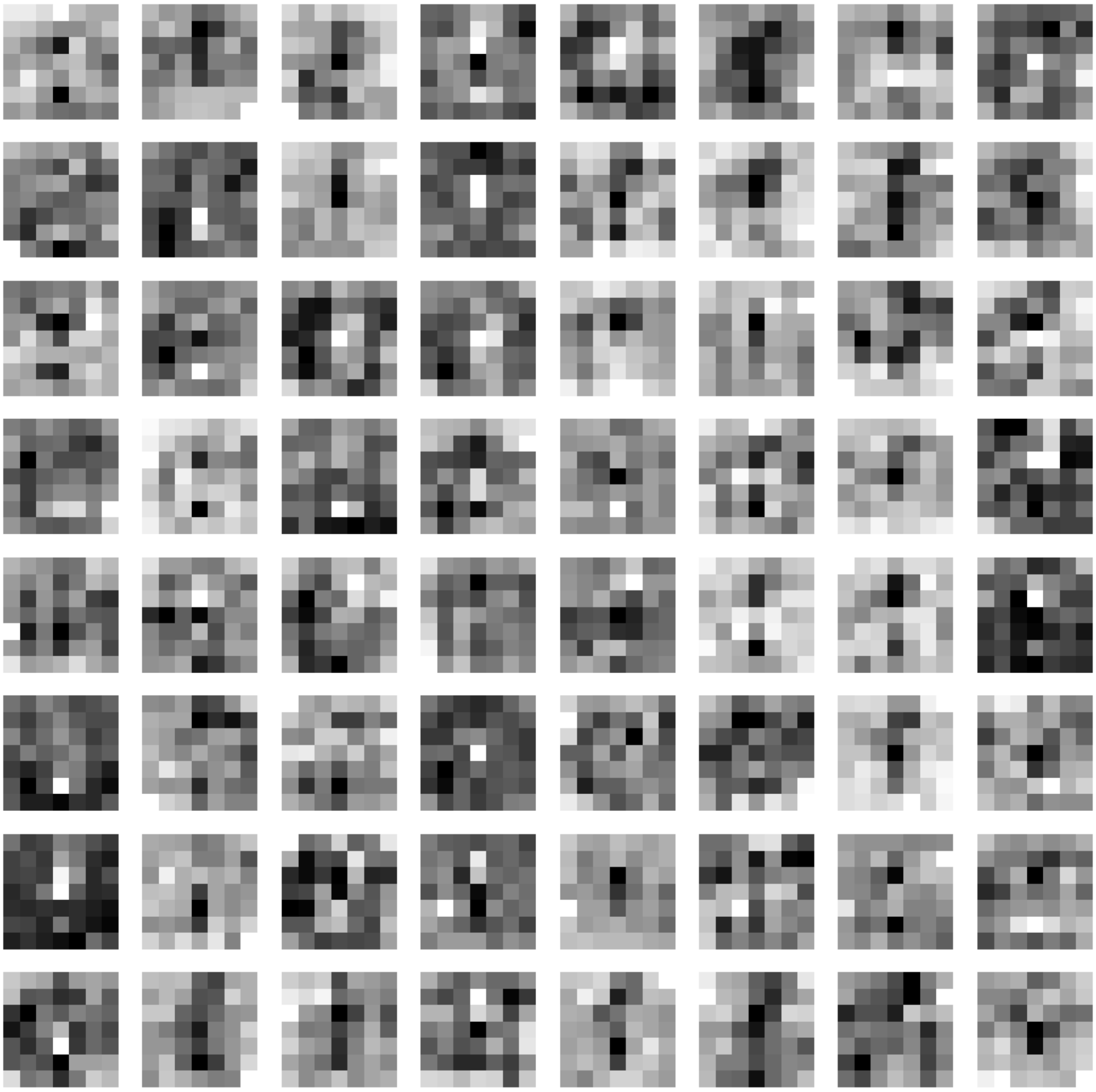


layer2.0.conv1

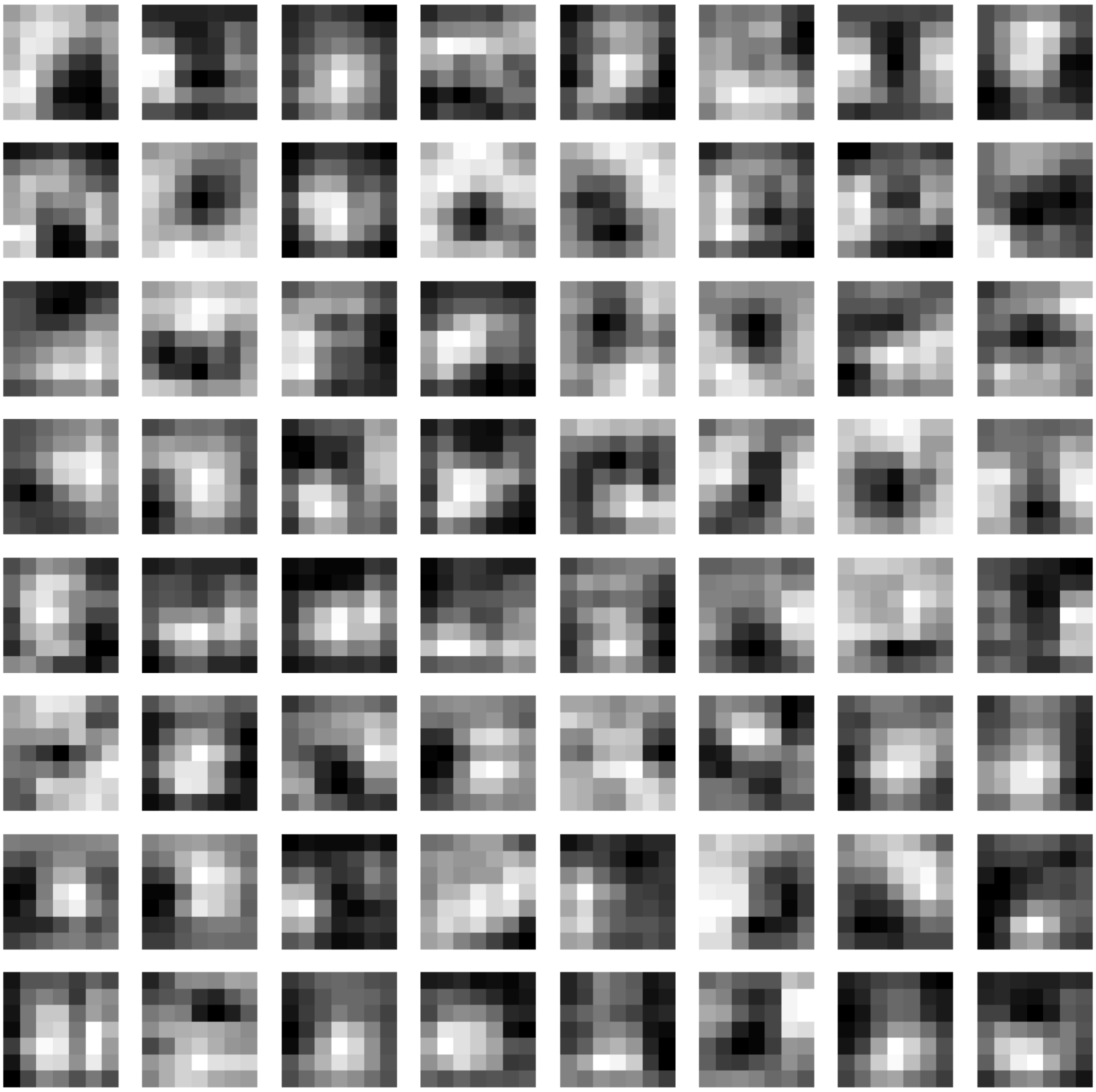


layer3.0.conv1

layer4.0.conv1



layer4.2.conv2



Analysis

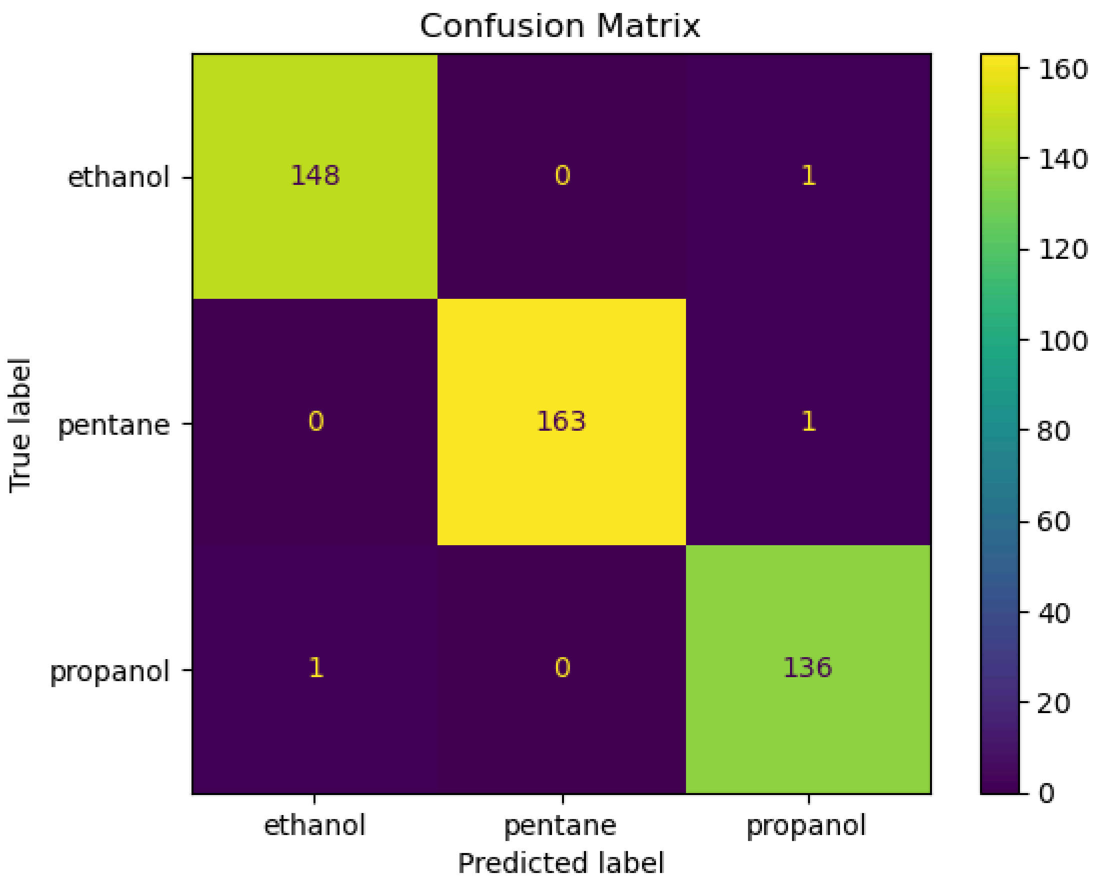
Q1: Compare the accuracy to the baseline vanilla pretrained ResNet-34 model.

In [17]:
```
from sklearn.metrics import accuracy_score

model.eval()
all_preds = []
all_labels = []

with torch.no_grad():
    for inputs, labels in test_loader:
        inputs, labels = inputs.to(device), labels.to(device)
        outputs = model(inputs)
        _, preds = torch.max(outputs, 1)
        all_preds.extend(preds.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())

accuracy = accuracy_score(all_labels, all_preds)

print("Pretrained ResNet-34 accuracy on testing set:", baseline_accuracy)
print("Fine-tuned model accuracy on testing set:", accuracy)
```

```
Pretrained ResNet-34 accuracy on testing set: 0.49777777777777776
Fine-tuned model accuracy on testing set: 0.9933333333333333
```

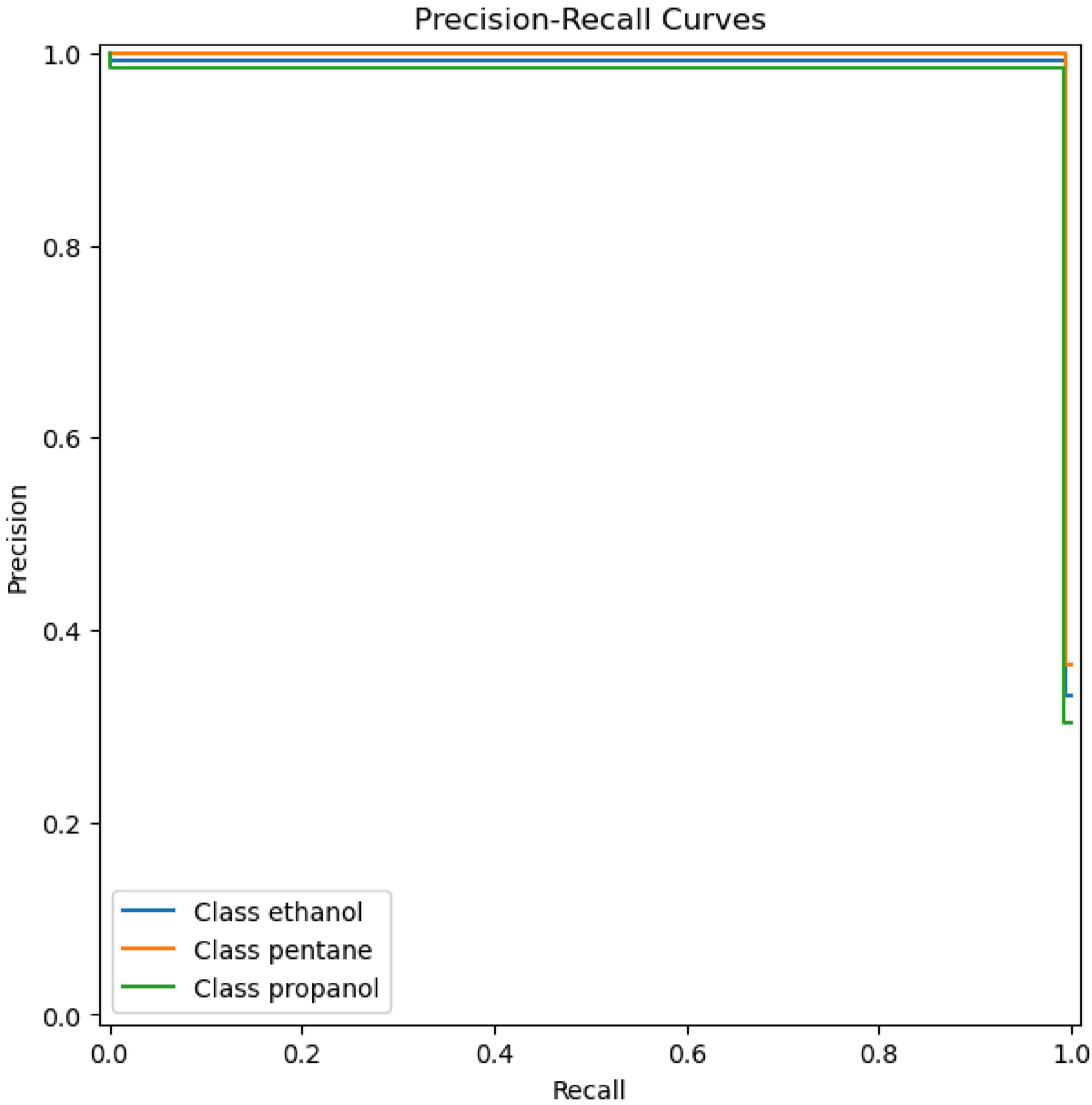
Q2: Generate a confusion matrix to show inter-class error rates.

In [21]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

cm = confusion_matrix(all_labels, all_preds)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=dataset.classe
disp.plot()
plt.title("Confusion Matrix")
plt.show()
```



Q3: Create a precision-recall curve for each class.

In [22]:
```
from sklearn.metrics import precision_recall_curve
from sklearn.preprocessing import label_binarize
from sklearn.metrics import PrecisionRecallDisplay

n_classes = len(dataset.classes)
binarized_labels = label_binarize(all_labels, classes=range(n_classes))
binarized_preds = label_binarize(all_preds, classes=range(n_classes))

plt.figure(figsize=(10, 7))
for i in range(n_classes):
    precision, recall, _ = precision_recall_curve(binarized_labels[:, i], binari
    disp = PrecisionRecallDisplay(precision=precision, recall=recall)
    disp.plot(ax=plt.gca(), name=f"Class {dataset.classes[i]}")

plt.title("Precision-Recall Curves")
plt.legend(loc="best")
plt.show()
```


Precision-Recall Curves
Precision
Recall
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
Class ethanol
Class pentane
Class propanol