In [1]:

```
import torch
from torch.utils.data import Dataset
import torchvision.transforms as transforms
import os
from PIL import Image
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt
import numpy as np
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
import time
import glob
import yaml
import torchvision
```

In [2]:

```
def extract_files():
    import google.colab
    import zipfile

    google.colab.drive.mount('/content/drive')
    PROJECT_DIR = "/content/drive/MyDrive/thesis/data/"

    zip_ref = zipfile.ZipFile(PROJECT_DIR + "fiveK.zip", 'r')
    zip_ref.extractall(".")
    zip_ref.close()
```

In [3]:

```
if 'google.colab' in str(get_ipython()):
  extract_files()
  config_path = "/content/drive/MyDrive/thesis/config.yaml"
else:
  config_path = "../../config.yaml"
```

```
Mounted at /content/drive
```

In [4]:

```
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
print(device)
```

```
cuda:0
```

In [5]:

```
# List of class directories
class_directories = ['expA', 'expB', 'expC', 'expD', 'expE']
```

In [6]:

```
class CustomDataset(Dataset):
    def __init__(self, data_dir, filename, transform=None):
        super().__init__()
        self.filename = filename
        self.transform = transform
        self.classname = self._extract_class_name(data_dir)
        self.encode = {k: i for i, k in enumerate(class_directories)}

        # Read the train.txt file and store the image paths
        with open(self.filename) as f:
            self.image_paths = [os.path.join(data_dir, line.strip()) for line in f]

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

    def __getitem__(self, index):
        image_path = self.image_paths[index]
        image = Image.open(image_path)
        label = self.encode[self.classname]
        if self.transform is not None:
            image = self.transform(image)
        return image, label

    def _extract_class_name(self, root_dir):
        # Extract the class name from the root directory
        class_name = os.path.basename(root_dir)
        return class_name
```

In [7]:

```
try:
    # Load configuration
    with open(config_path, 'r') as config_file:
        config = yaml.safe_load(config_file)
except:
    raise FileNotFoundError(f"Config file not found at path: {config_path}")
```

In [8]:

```
data_folder = config['paths']['data']
train_file = config['paths']['train']
test_file = config['paths']['test']
```

In [9]:

```
def read_dataset(data_folder, txt_file, trasform=None):
    # Create separate datasets for each class
    datasets = []

    for class_dir in class_directories:
        class_train_dataset = CustomDataset(
            data_dir=os.path.join(data_folder, class_dir),
            filename=os.path.join(txt_file),
            transform=trasform
        )
        datasets.append(class_train_dataset)
    return datasets
```

In [10]:

```
training_tr = transforms.Compose([
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ])

test_tr = transforms.Compose([
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ])
```

In [11]:

```
# Combine datasets if needed (e.g., for training)
train_dataset = torch.utils.data.ConcatDataset(read_dataset(data_folder, train_file, trai
ning_tr))
test_dataset = torch.utils.data.ConcatDataset(read_dataset(data_folder, test_file, test_
tr))
```

In [12]:

```
bs = 128
```

In [13]:

```
train_dataloader = DataLoader(train_dataset, batch_size=bs, shuffle=True)
test_dataloader = DataLoader(test_dataset, batch_size=bs*2, shuffle=False)
```

In [14]:

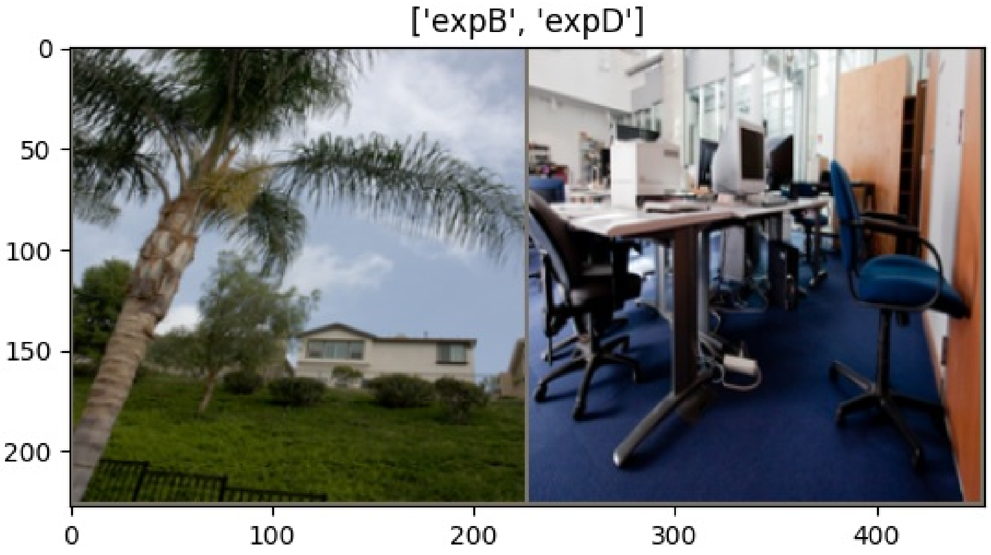
```
train_features, train_labels = next(iter(train_dataloader))
```

In [15]:

```
def imshow(inp, title=None):
    """Display image for Tensor."""
    inp = inp.numpy().transpose((1, 2, 0))
    mean = np.array([0.485, 0.456, 0.406])
    std = np.array([0.229, 0.224, 0.225])
    inp = std * inp + mean
    inp = np.clip(inp, 0, 1)
    plt.imshow(inp)
    if title is not None:
        plt.title(title)
    plt.pause(0.001)  # pause a bit so that plots are updated
```

In [16]:

```
# Get a batch of training data
inputs, labels = next(iter(train_dataloader))
# Make a grid from first 2 images in the batch
out = torchvision.utils.make_grid(inputs[:2])
imshow(out, title=[class_directories[x] for x in labels[:2]])
```



In [17]:

```
print(len(train_dataset))
```

```
20000
```

In [18]:

```
base_checkpoint_path = config['paths']['checkpoints']
# Create the directory if it does not exist
if not os.path.exists(base_checkpoint_path):
    os.makedirs(base_checkpoint_path)
```

In [19]:

```
def get_lr(optimizer):
    for param_group in optimizer.param_groups:
        return param_group['lr']
```

In [20]:

```
def train_model(model, criterion, optimizer, scheduler, current_epoch, num_epochs=25):
    since = time.time()
    best_acc = 0.0
    model.train()
    for epoch in range(current_epoch, num_epochs):
            # formatted string to append epoch number to checkpoint filename
        print(f'Epoch {epoch + 1}/{num_epochs}')
        print('-' * 10)
        running_loss = 0.0
        running_corrects = 0
        # Iterate over data.
        for inputs, labels in train_dataloader:
            inputs = inputs.to(device)
            labels = labels.to(device)

            # zero the parameter gradients
            optimizer.zero_grad()
            outputs = model(inputs)
            _, preds = torch.max(outputs, 1)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            # statistics
            running_loss += loss.item() * inputs.size(0)
            running_corrects += torch.sum(preds == labels.data)

        scheduler.step()

        epoch_loss = running_loss / len(train_dataset)
        epoch_acc = running_corrects.double() / len(train_dataset)

        print(f'Loss: {epoch_loss:.4f} Acc: {epoch_acc:.4f} LR: {get_lr(optimizer):.8f}'
)
        print()

        PATH = os.path.join(base_checkpoint_path, f'{os.path.basename(base_checkpoint_pa
th)}_{epoch+1}.pth')
        # save checkpoint
        state = {
            'epoch': epoch + 1,
            'state_dict': model.state_dict(),
            'optimizer': optimizer.state_dict(),
            'loss': epoch_loss,
            'scheduler': scheduler.state_dict(),
            'accuracy': epoch_acc
        }
        # save the best model parameters
        torch.save(state, PATH)
        # deep copy the model
        if epoch_acc > best_acc:
            best_acc = epoch_acc

    time_elapsed = time.time() - since
    print(f'Training complete in {time_elapsed // 60:.0f}m {time_elapsed % 60:.0f}s')
    print(f'Best Acc: {best_acc:4f}')
```

In [21]:

```
model_name = config['model']['name']
if not model_name.startswith('resnet'):
    raise ValueError("Model name must start with 'resnet'")
```

In [22]:

```
if config['model']['type'] == 'FEATURE_EXTRACTOR':
    model = torchvision.models.__dict__[model_name](weights='IMAGENET1K_V1')
    # Freeze all layers except the fully connected layers
    for param in model.parameters():
        param.requires_grad = False
elif config['model']['type'] == 'FINE_TUNING':
    model = torchvision.models.__dict__[model_name](weights='IMAGENET1K_V1')
elif config['model']['type'] == 'TRAIN_FROM_SCRATCH':
    model = torchvision.models.__dict__[model_name](weights=None)
else:
    raise ValueError(f"Unknown model type: {config['model']['type']}")

# Parameters of newly constructed modules have requires_grad=True by default
num_ftrs = model.fc.in_features
model.fc =  nn.Linear(num_ftrs, config['model']['num_classes'])

# move the model to GPU/CPU
model = model.to(device)

criterion = nn.CrossEntropyLoss()

optimizer = optim.SGD(model.parameters(), lr=config['model']['lr'], momentum=config['mod
el']['momentum'])

milestones = [9, 18, 34, 50, 70]

# Decay LR by a factor of 0.1 every 7 epochs
scheduler = lr_scheduler.MultiStepLR(optimizer, milestones, gamma=0.1)
```

In [23]:

```
# load the last model saved if there is any
def load_latest_model(model, optimizer, scheduler, checkpoint_dir):
    # Check if the directory exists
    if not os.path.exists(base_checkpoint_path):
        print(f"No directory found: {checkpoint_dir}")
        return model, optimizer, scheduler, 0, None

    # Get a list of all checkpoint files in the directory
    checkpoint_files = glob.glob(os.path.join(checkpoint_dir, f'{os.path.basename(checkp
oint_dir)}_*.pth'))
    print(checkpoint_files)
    # Check if any checkpoint files are present
    if not checkpoint_files:
        print(f"No checkpoints found in the directory: {checkpoint_dir}")
        return model, optimizer, scheduler, 0, None

    # Find the latest checkpoint file based on the epoch number in the filename
    latest_checkpoint = max(checkpoint_files, key=os.path.getctime)

    # Load the latest checkpoint
    checkpoint = torch.load(latest_checkpoint, map_location=torch.device(device))
    model.load_state_dict(checkpoint['state_dict'])
    optimizer.load_state_dict(checkpoint['optimizer'])
    scheduler.load_state_dict(checkpoint['scheduler'])
    epoch = checkpoint['epoch']
    loss = checkpoint['loss']
    print(checkpoint['accuracy'])

    print(f"Loaded model from checkpoint: {latest_checkpoint}")
    print(f"Resuming training from epoch {epoch}")

    return model, optimizer,scheduler, epoch, loss
```

In [24]:

```
model, optimizer, scheduler, current_epoch, loss = load_latest_model(model, optimizer, s
cheduler, base_checkpoint_path)
```

```
[]
No checkpoints found in the directory: /content/drive/MyDrive/thesis/model/checkpoints/re
setnet50_scratch
```

In [25]:

```
print(get_lr(optimizer))
print(get_lr(optimizer))
```

```
0.1
```

In [26]:

```
train_model(model, criterion, optimizer, scheduler,current_epoch, num_epochs=config['model']['num_epochs'])
```

```
Epoch 1/80
----------
Loss: 3.4457 Acc: 0.2042 LR: 0.10000000

Epoch 2/80
----------
Loss: 1.6515 Acc: 0.2142 LR: 0.10000000

Epoch 3/80
----------
Loss: 1.6327 Acc: 0.2339 LR: 0.10000000

Epoch 4/80
----------
Loss: 1.6090 Acc: 0.2464 LR: 0.10000000

Epoch 5/80
----------
Loss: 1.5961 Acc: 0.2645 LR: 0.10000000

Epoch 6/80
----------
Loss: 1.5864 Acc: 0.2767 LR: 0.10000000

Epoch 7/80
----------
Loss: 1.5600 Acc: 0.2865 LR: 0.10000000

Epoch 8/80
----------
Loss: 1.5520 Acc: 0.3065 LR: 0.10000000

Epoch 9/80
----------
Loss: 1.5829 Acc: 0.2647 LR: 0.01000000

Epoch 10/80
----------
Loss: 1.5303 Acc: 0.3209 LR: 0.01000000

Epoch 11/80
----------
Loss: 1.5227 Acc: 0.3229 LR: 0.01000000

Epoch 12/80
----------
Loss: 1.5210 Acc: 0.3226 LR: 0.01000000

Epoch 13/80
----------
Loss: 1.5138 Acc: 0.3339 LR: 0.01000000

Epoch 14/80
----------
Loss: 1.5047 Acc: 0.3411 LR: 0.01000000

Epoch 15/80
----------
Loss: 1.5006 Acc: 0.3431 LR: 0.01000000

Epoch 16/80
----------
Loss: 1.4982 Acc: 0.3404 LR: 0.01000000
```

```
Epoch 17/80
----------
Loss: 1.4890 Acc: 0.3480 LR: 0.01000000

Epoch 18/80
----------
Loss: 1.4853 Acc: 0.3533 LR: 0.00100000

Epoch 19/80
----------
Loss: 1.4812 Acc: 0.3528 LR: 0.00100000

Epoch 20/80
----------
Loss: 1.4791 Acc: 0.3535 LR: 0.00100000

Epoch 21/80
----------
Loss: 1.4774 Acc: 0.3614 LR: 0.00100000

Epoch 22/80
----------
Loss: 1.4749 Acc: 0.3593 LR: 0.00100000

Epoch 23/80
----------
Loss: 1.4751 Acc: 0.3573 LR: 0.00100000

Epoch 24/80
----------
Loss: 1.4704 Acc: 0.3634 LR: 0.00100000

Epoch 25/80
----------
Loss: 1.4766 Acc: 0.3563 LR: 0.00100000

Epoch 26/80
----------
Loss: 1.4748 Acc: 0.3594 LR: 0.00100000

Epoch 27/80
----------
Loss: 1.4737 Acc: 0.3597 LR: 0.00100000

Epoch 28/80
----------
Loss: 1.4715 Acc: 0.3616 LR: 0.00100000

Epoch 29/80
----------
Loss: 1.4703 Acc: 0.3674 LR: 0.00100000

Epoch 30/80
----------
Loss: 1.4711 Acc: 0.3602 LR: 0.00100000

Epoch 31/80
----------
Loss: 1.4726 Acc: 0.3599 LR: 0.00100000

Epoch 32/80
----------
Loss: 1.4710 Acc: 0.3591 LR: 0.00100000

Epoch 33/80
----------
Loss: 1.4684 Acc: 0.3629 LR: 0.00100000

Epoch 34/80
----------
Loss: 1.4684 Acc: 0.3671 LR: 0.00010000
```

```
Epoch 35/80
----------
Loss: 1.4664 Acc: 0.3639 LR: 0.00010000

Epoch 36/80
----------
Loss: 1.4700 Acc: 0.3636 LR: 0.00010000

Epoch 37/80
----------
Loss: 1.4668 Acc: 0.3657 LR: 0.00010000

Epoch 38/80
----------
Loss: 1.4668 Acc: 0.3663 LR: 0.00010000

Epoch 39/80
----------
Loss: 1.4672 Acc: 0.3626 LR: 0.00010000

Epoch 40/80
----------
Loss: 1.4643 Acc: 0.3658 LR: 0.00010000

Epoch 41/80
----------
Loss: 1.4654 Acc: 0.3661 LR: 0.00010000

Epoch 42/80
----------
Loss: 1.4680 Acc: 0.3641 LR: 0.00010000

Epoch 43/80
----------
Loss: 1.4660 Acc: 0.3644 LR: 0.00010000

Epoch 44/80
----------
Loss: 1.4648 Acc: 0.3661 LR: 0.00010000

Epoch 45/80
----------
Loss: 1.4702 Acc: 0.3608 LR: 0.00010000

Epoch 46/80
----------
Loss: 1.4652 Acc: 0.3657 LR: 0.00010000

Epoch 47/80
----------
Loss: 1.4678 Acc: 0.3661 LR: 0.00010000

Epoch 48/80
----------
Loss: 1.4641 Acc: 0.3681 LR: 0.00010000

Epoch 49/80
----------
Loss: 1.4680 Acc: 0.3627 LR: 0.00010000

Epoch 50/80
----------
Loss: 1.4661 Acc: 0.3670 LR: 0.00001000

Epoch 51/80
----------
Loss: 1.4657 Acc: 0.3657 LR: 0.00001000

Epoch 52/80
----------
Loss: 1.4705 Acc: 0.3657 LR: 0.00001000
```

```
Epoch 53/80
----------
Loss: 1.4675 Acc: 0.3635 LR: 0.00001000

Epoch 54/80
----------
Loss: 1.4658 Acc: 0.3654 LR: 0.00001000

Epoch 55/80
----------
Loss: 1.4662 Acc: 0.3654 LR: 0.00001000

Epoch 56/80
----------
Loss: 1.4668 Acc: 0.3663 LR: 0.00001000

Epoch 57/80
----------
Loss: 1.4663 Acc: 0.3651 LR: 0.00001000

Epoch 58/80
----------
Loss: 1.4654 Acc: 0.3644 LR: 0.00001000

Epoch 59/80
----------
Loss: 1.4644 Acc: 0.3661 LR: 0.00001000

Epoch 60/80
----------
Loss: 1.4643 Acc: 0.3629 LR: 0.00001000

Epoch 61/80
----------
Loss: 1.4671 Acc: 0.3652 LR: 0.00001000

Epoch 62/80
----------
Loss: 1.4645 Acc: 0.3634 LR: 0.00001000

Epoch 63/80
----------
Loss: 1.4678 Acc: 0.3629 LR: 0.00001000

Epoch 64/80
----------
Loss: 1.4661 Acc: 0.3677 LR: 0.00001000

Epoch 65/80
----------
Loss: 1.4664 Acc: 0.3650 LR: 0.00001000

Epoch 66/80
----------
Loss: 1.4630 Acc: 0.3686 LR: 0.00001000

Epoch 67/80
----------
Loss: 1.4642 Acc: 0.3641 LR: 0.00001000

Epoch 68/80
----------
Loss: 1.4658 Acc: 0.3695 LR: 0.00001000

Epoch 69/80
----------
Loss: 1.4618 Acc: 0.3680 LR: 0.00001000

Epoch 70/80
----------
Loss: 1.4645 Acc: 0.3661 LR: 0.00000100
```

```
Epoch 71/80
----------
Loss: 1.4676 Acc: 0.3663 LR: 0.00000100

Epoch 72/80
----------
Loss: 1.4687 Acc: 0.3661 LR: 0.00000100

Epoch 73/80
----------
Loss: 1.4637 Acc: 0.3671 LR: 0.00000100

Epoch 74/80
----------
Loss: 1.4694 Acc: 0.3656 LR: 0.00000100

Epoch 75/80
----------
Loss: 1.4659 Acc: 0.3702 LR: 0.00000100

Epoch 76/80
----------
Loss: 1.4677 Acc: 0.3639 LR: 0.00000100

Epoch 77/80
----------
Loss: 1.4651 Acc: 0.3647 LR: 0.00000100

Epoch 78/80
----------
Loss: 1.4645 Acc: 0.3683 LR: 0.00000100

Epoch 79/80
----------
Loss: 1.4662 Acc: 0.3672 LR: 0.00000100

Epoch 80/80
----------
Loss: 1.4680 Acc: 0.3668 LR: 0.00000100

Training complete in 377m 48s
Best Acc: 0.370200
```

In [ ]:

```python
time.sleep(5)  # Sleep for 5 seconds to let the system cool down
from google.colab import runtime
runtime.unassign()
```