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 set_seed(seed):
    torch.manual_seed(seed)
    np.random.seed(seed)

    # for cuda
    torch.cuda.manual_seed_all(seed)
    torch.backends.cudnn.deterministic = True
```

In [3]:

```
set_seed(0)
```

In [4]:

```
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 [5]:

```
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 [6]:

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

```
cuda:0
```

In [7]:

```
# List of class directories
class_directories = ['expA', 'expB', 'expC', 'expD', 'expE']
# raw data directory
raw_dir = "raw"
```

In [8]:

```
class CustomDataset(Dataset):
    def __init__(self, data_dir, raw_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:
            img_paths= []
            raw_img_paths = []
            for line in f:
                line = line.strip()
                img_paths.append(os.path.join(data_dir, line))
                raw_img_paths.append(os.path.join(raw_data_dir, line))

            self.image_paths = img_paths
            self.raw_image_paths = raw_img_paths

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

    def __getitem__(self, index):
        image_path = self.image_paths[index]
        raw_image_path = self.raw_image_paths[index]
        image = Image.open(image_path)
        raw_image = Image.open(raw_image_path)
        image = np.dstack((np.array(image), np.array(raw_image)))
        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 [9]:

```
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 [10]:

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

In [11]:

```
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),
            raw_data_dir=os.path.join(data_folder, raw_dir),
            filename=os.path.join(txt_file),
            transform=trasform
        )
        datasets.append(class_train_dataset)
    return datasets
```

In [12]:

```
training_tr = transforms.Compose([
        transforms.ToTensor(),
        transforms.RandomResizedCrop(128, antialias=True),
        transforms.RandomHorizontalFlip(),
        transforms.Normalize([0.4397, 0.4234, 0.3911, 0.2279, 0.2017, 0.1825], [0.2306, 0.2201, 0.2327, 0.1191, 0.1092, 0.1088])
    ])
```

In [13]:

```
# Combine datasets if needed (e.g., for training)
train_dataset = torch.utils.data.ConcatDataset(read_dataset(data_folder, train_file, training_tr))
```

In [14]:

```
bs = 128
```

In [15]:

```
train_dataloader = DataLoader(train_dataset, batch_size=bs, shuffle=True)
```

In [16]:

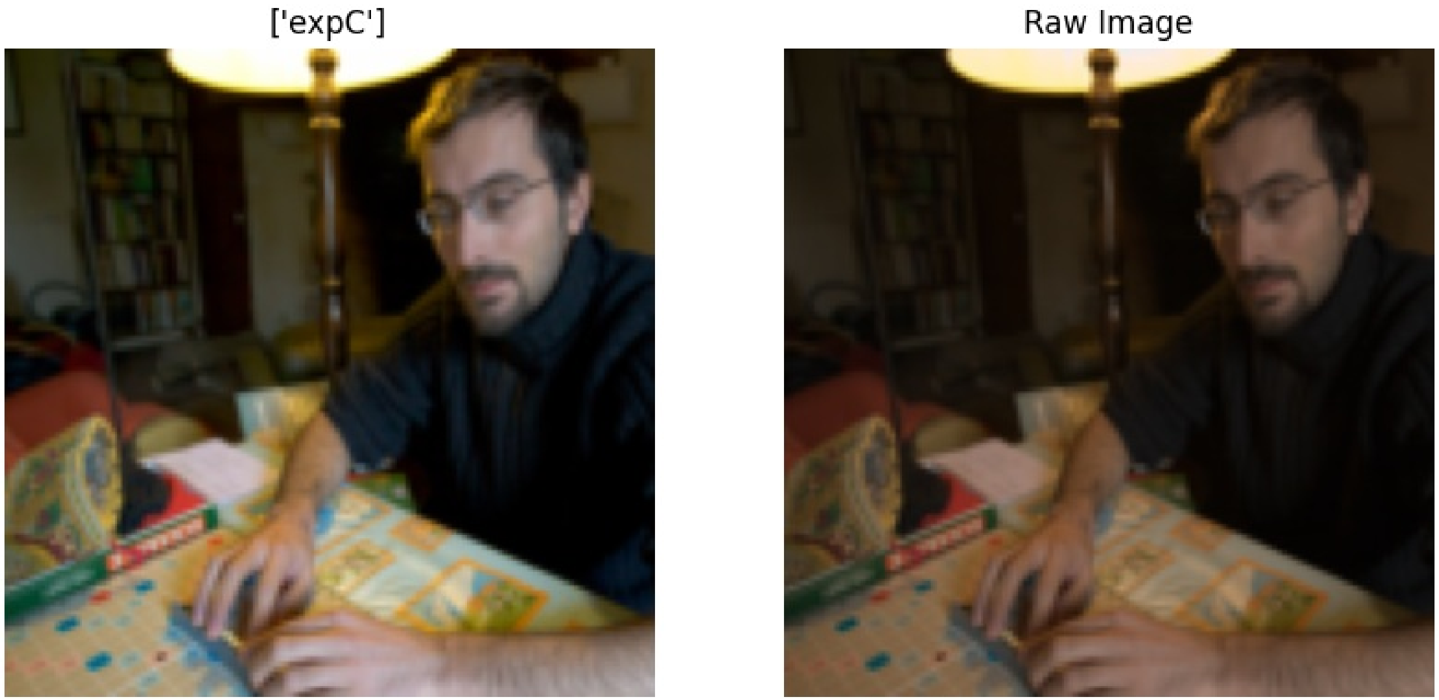
```
train_features, train_labels = next(iter(train_dataloader))
```

In [17]:

```
def imshow(inp, title=None):
    """Display image for Tensor."""
    inp = inp.numpy().transpose((1, 2, 0))
    org_img = inp[:, :, :3]
    raw_img = inp[:, :, 3:]
    mean1 = np.array([0.4397, 0.4234, 0.3911])
    mean2 = np.array([0.2279, 0.2017, 0.1825])
    std1 = np.array([0.2306, 0.2201, 0.2327])
    std2 = np.array([0.1191, 0.1092, 0.1088])
    org_img = std1 * org_img + mean1
    raw_img = std2 * raw_img + mean2
    org_img = np.clip(org_img, 0, 1)
    raw_img = np.clip(raw_img, 0, 1)

    # Create a figure with two subplots
    _, axes = plt.subplots(1, 2, figsize=(10, 5))

    # Plot original image on the first subplot
    axes[0].imshow(org_img)
    if title is not None:
        axes[0].set_title(title)
    axes[0].axis('off')

    # Plot raw image on the second subplot
    axes[1].imshow(raw_img)
    axes[1].set_title('Raw Image')
    axes[1].axis('off')

    plt.pause(0.001)  # pause a bit so that plots are updated
    plt.show()
```

In [18]:

```
# Get a batch of training data
inputs, labels = next(iter(train_dataloader))
out = inputs[:1].squeeze()

imshow(out, title=[class_directories[x] for x in labels[:1]])
```



In [19]:

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

```
20000
```

In [20]:

```
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 [21]:

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

In [22]:

```
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_path)}_{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 [23]:

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

In [24]:

```
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'])

# change the first convolution to accept 6 channels
model.conv1 = nn.Conv2d(6, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)

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

criterion = nn.CrossEntropyLoss()

optimizer = optim.SGD(model.parameters(), lr=config['model']['lr'], momentum=config['model']['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 [25]:

```
# 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(checkpoint_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 [26]:

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

```
['/content/drive/MyDrive/thesis/model/checkpoints/resetnet18_scratch_raw/resetnet18_scratch_raw_1.pth', '/content/drive/MyDrive/th
esis/model/checkpoints/resetnet18_scratch_raw/resetnet18_scratch_raw_2.pth', '/content/drive/MyDrive/thesis/model/checkpoints/rese
tnet18_scratch_raw/resetnet18_scratch_raw_3.pth']
tensor(0.4052, device='cuda:0', dtype=torch.float64)
Loaded model from checkpoint: /content/drive/MyDrive/thesis/model/checkpoints/resetnet18_scratch_raw/resetnet18_scratch_raw_3.pth
Resuming training from epoch 3
```

In [27]:

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

```
0.01
```

In [28]:

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

```
Epoch 4/80
----------
Loss: 1.3634 Acc: 0.4289 LR: 0.01000000

Epoch 5/80
----------
Loss: 1.3233 Acc: 0.4539 LR: 0.01000000

Epoch 6/80
----------
Loss: 1.2796 Acc: 0.4774 LR: 0.01000000

Epoch 7/80
----------
Loss: 1.2828 Acc: 0.4726 LR: 0.01000000

Epoch 8/80
----------
Loss: 1.2632 Acc: 0.4844 LR: 0.01000000

Epoch 9/80
----------
Loss: 1.2284 Acc: 0.4976 LR: 0.00100000

Epoch 10/80
----------
Loss: 1.1512 Acc: 0.5377 LR: 0.00100000

Epoch 11/80
----------
Loss: 1.1347 Acc: 0.5450 LR: 0.00100000

Epoch 12/80
----------
Loss: 1.1222 Acc: 0.5466 LR: 0.00100000

Epoch 13/80
----------
Loss: 1.1170 Acc: 0.5528 LR: 0.00100000

Epoch 14/80
----------
```

```
Loss: 1.1108 Acc: 0.5502 LR: 0.00100000

Epoch 15/80
----------
Loss: 1.1082 Acc: 0.5570 LR: 0.00100000

Epoch 16/80
----------
Loss: 1.1044 Acc: 0.5558 LR: 0.00100000

Epoch 17/80
----------
Loss: 1.0959 Acc: 0.5601 LR: 0.00100000

Epoch 18/80
----------
Loss: 1.0988 Acc: 0.5574 LR: 0.00010000

Epoch 19/80
----------
Loss: 1.0774 Acc: 0.5696 LR: 0.00010000

Epoch 20/80
----------
Loss: 1.0755 Acc: 0.5690 LR: 0.00010000

Epoch 21/80
----------
Loss: 1.0779 Acc: 0.5690 LR: 0.00010000

Epoch 22/80
----------
Loss: 1.0729 Acc: 0.5719 LR: 0.00010000

Epoch 23/80
----------
Loss: 1.0753 Acc: 0.5678 LR: 0.00010000

Epoch 24/80
----------
Loss: 1.0679 Acc: 0.5732 LR: 0.00010000

Epoch 25/80
----------
Loss: 1.0757 Acc: 0.5690 LR: 0.00010000

Epoch 26/80
----------
Loss: 1.0675 Acc: 0.5739 LR: 0.00010000

Epoch 27/80
Epoch 27/80
----------
Loss: 1.0703 Acc: 0.5760 LR: 0.00010000

Epoch 28/80
----------
Loss: 1.0711 Acc: 0.5716 LR: 0.00010000

Epoch 29/80
----------
Loss: 1.0697 Acc: 0.5729 LR: 0.00010000

Epoch 30/80
----------
Loss: 1.0668 Acc: 0.5744 LR: 0.00010000

Epoch 31/80
----------
Loss: 1.0685 Acc: 0.5725 LR: 0.00010000

Epoch 32/80
----------
Loss: 1.0698 Acc: 0.5719 LR: 0.00010000

Epoch 33/80
----------
Loss: 1.0642 Acc: 0.5766 LR: 0.00010000

Epoch 34/80
----------
Loss: 1.0652 Acc: 0.5736 LR: 0.00001000

Epoch 35/80
----------
Loss: 1.0637 Acc: 0.5756 LR: 0.00001000

Epoch 36/80
----------
Loss: 1.0629 Acc: 0.5753 LR: 0.00001000

Epoch 37/80
----------
Loss: 1.0693 Acc: 0.5712 LR: 0.00001000

Epoch 38/80
----------
Loss: 1.0695 Acc: 0.5725 LR: 0.00001000

Epoch 39/80
----------
Loss: 1.0569 Acc: 0.5771 LR: 0.00001000
```

```
Epoch 40/80
----------
Loss: 1.0657 Acc: 0.5743 LR: 0.00001000

Epoch 41/80
----------
Loss: 1.0619 Acc: 0.5742 LR: 0.00001000

Epoch 42/80
----------
Loss: 1.0654 Acc: 0.5738 LR: 0.00001000

Epoch 43/80
----------
Loss: 1.0702 Acc: 0.5695 LR: 0.00001000

Epoch 44/80
----------
Loss: 1.0668 Acc: 0.5761 LR: 0.00001000

Epoch 45/80
----------
Loss: 1.0633 Acc: 0.5742 LR: 0.00001000

Epoch 46/80
----------
Loss: 1.0617 Acc: 0.5766 LR: 0.00001000

Epoch 47/80
----------
Loss: 1.0643 Acc: 0.5762 LR: 0.00001000

Epoch 48/80
----------
Loss: 1.0657 Acc: 0.5738 LR: 0.00001000

Epoch 49/80
----------
Loss: 1.0660 Acc: 0.5736 LR: 0.00001000

Epoch 50/80
----------
Loss: 1.0639 Acc: 0.5728 LR: 0.00000100

Epoch 51/80
----------
Loss: 1.0624 Acc: 0.5758 LR: 0.00000100

Epoch 52/80
----------
```

```
----------
Loss: 1.0630 Acc: 0.5757 LR: 0.00000100

Epoch 53/80
----------
Loss: 1.0674 Acc: 0.5745 LR: 0.00000100

Epoch 54/80
----------
Loss: 1.0630 Acc: 0.5742 LR: 0.00000100

Epoch 55/80
----------
Loss: 1.0633 Acc: 0.5768 LR: 0.00000100

Epoch 56/80
----------
Loss: 1.0608 Acc: 0.5772 LR: 0.00000100

Epoch 57/80
----------
Loss: 1.0618 Acc: 0.5764 LR: 0.00000100

Epoch 58/80
----------
Loss: 1.0651 Acc: 0.5781 LR: 0.00000100

Epoch 59/80
----------
Loss: 1.0584 Acc: 0.5777 LR: 0.00000100

Epoch 60/80
----------
Loss: 1.0642 Acc: 0.5730 LR: 0.00000100

Epoch 61/80
----------
Loss: 1.0629 Acc: 0.5740 LR: 0.00000100

Epoch 62/80
----------
Loss: 1.0608 Acc: 0.5768 LR: 0.00000100

Epoch 63/80
----------
Loss: 1.0604 Acc: 0.5752 LR: 0.00000100

Epoch 64/80
----------
Loss: 1.0667 Acc: 0.5744 LR: 0.00000100
```

```
Epoch 65/80
----------
Loss: 1.0624 Acc: 0.5733 LR: 0.00000100

Epoch 66/80
----------
Loss: 1.0616 Acc: 0.5745 LR: 0.00000100

Epoch 67/80
----------
Loss: 1.0629 Acc: 0.5746 LR: 0.00000100

Epoch 68/80
----------
Loss: 1.0633 Acc: 0.5751 LR: 0.00000100

Epoch 69/80
----------
Loss: 1.0647 Acc: 0.5708 LR: 0.00000100

Epoch 70/80
----------
Loss: 1.0624 Acc: 0.5783 LR: 0.00000010

Epoch 71/80
----------
Loss: 1.0634 Acc: 0.5761 LR: 0.00000010

Epoch 72/80
----------
Loss: 1.0624 Acc: 0.5776 LR: 0.00000010

Epoch 73/80
----------
Loss: 1.0583 Acc: 0.5777 LR: 0.00000010

Epoch 74/80
----------
Loss: 1.0623 Acc: 0.5771 LR: 0.00000010

Epoch 75/80
----------
Loss: 1.0624 Acc: 0.5746 LR: 0.00000010

Epoch 76/80
----------
Loss: 1.0640 Acc: 0.5787 LR: 0.00000010

Epoch 77/80
----------
Loss: 1.0655 Acc: 0.5796 LR: 0.00000010
```

```
Loss: 1.0655 Acc: 0.5796 LR: 0.00000010

Epoch 78/80
----------
Loss: 1.0644 Acc: 0.5745 LR: 0.00000010

Epoch 79/80
----------
Loss: 1.0664 Acc: 0.5731 LR: 0.00000010

Epoch 80/80
----------
Loss: 1.0593 Acc: 0.5768 LR: 0.00000010

Training complete in 242m 16s
Best Acc: 0.579650
```

In [ ]:

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