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(224, 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)
```

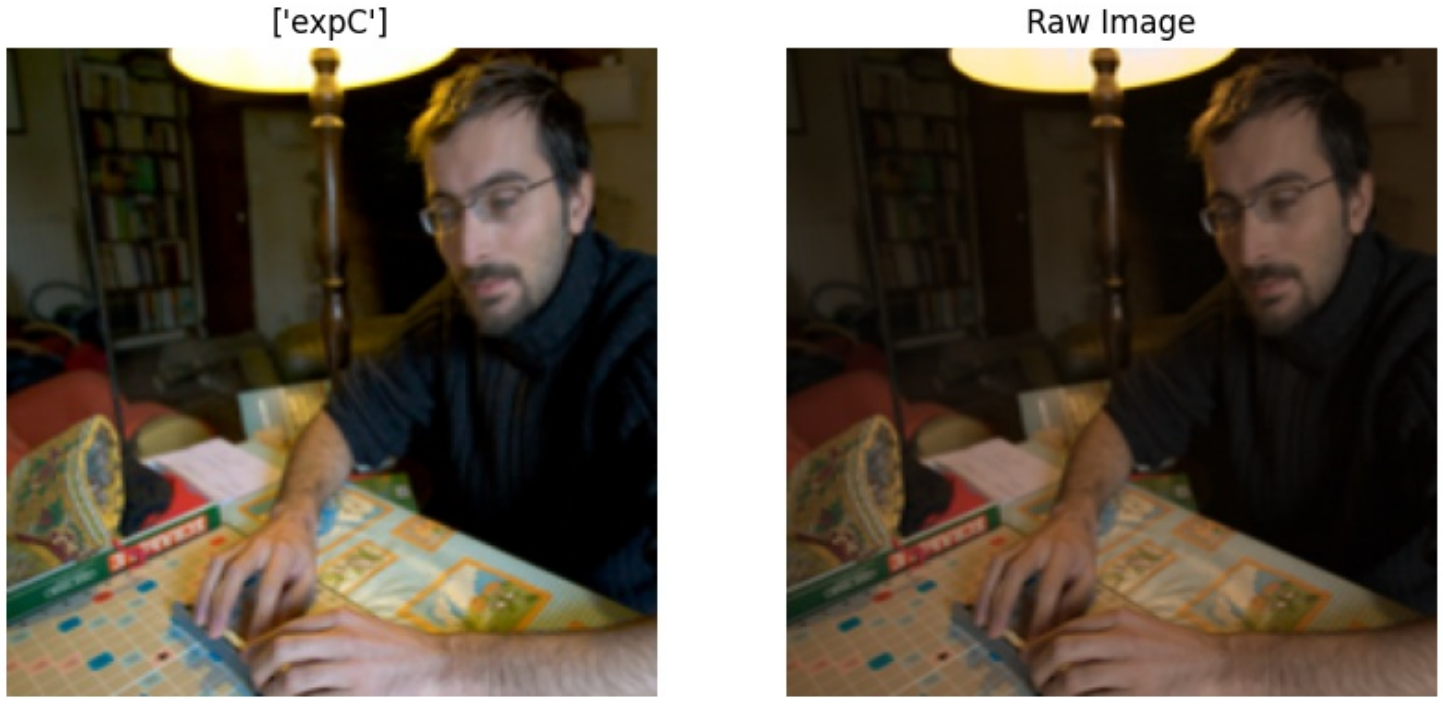
```
    axes[1].set_title('Raw Image')
    axes[1].axis('off')

    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 = [19, 36, 45, 54, 63, 72]
```

```
# 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'))
    # 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)
```

```
No checkpoints found in the directory: /content/drive/MyDrive/thesis/model/checkpoints/resetnet18_scratch_raw
```

In [27]:

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

```
0.01
```

In [ ]:

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

```
Epoch 1/80
----------
Loss: 1.5286 Acc: 0.3311 LR: 0.01000000

Epoch 2/80
----------
Loss: 1.4262 Acc: 0.3908 LR: 0.01000000

Epoch 3/80
----------
Loss: 1.3762 Acc: 0.4279 LR: 0.01000000

Epoch 4/80
----------
Loss: 1.3433 Acc: 0.4417 LR: 0.01000000

Epoch 5/80
----------
Loss: 1.3086 Acc: 0.4625 LR: 0.01000000

Epoch 6/80
----------
Loss: 1.2660 Acc: 0.4806 LR: 0.01000000

Epoch 7/80
----------
Loss: 1.2338 Acc: 0.4999 LR: 0.01000000

Epoch 8/80
----------
Loss: 1.2238 Acc: 0.5042 LR: 0.01000000

Epoch 9/80
----------
Loss: 1.2059 Acc: 0.5097 LR: 0.01000000

Epoch 10/80
----------
Loss: 1.1901 Acc: 0.5232 LR: 0.01000000

Epoch 11/80
----------
Loss: 1.1660 Acc: 0.5259 LR: 0.01000000

Epoch 12/80
----------
Loss: 1.1606 Acc: 0.5346 LR: 0.01000000
```

```
Epoch 13/80
----------
Loss: 1.1469 Acc: 0.5406 LR: 0.01000000

Epoch 14/80
----------
Loss: 1.1243 Acc: 0.5503 LR: 0.01000000

Epoch 15/80
----------
Loss: 1.1134 Acc: 0.5572 LR: 0.01000000

Epoch 16/80
----------
Loss: 1.1039 Acc: 0.5534 LR: 0.01000000

Epoch 17/80
----------
Loss: 1.0970 Acc: 0.5578 LR: 0.01000000

Epoch 18/80
----------
Loss: 1.0878 Acc: 0.5649 LR: 0.01000000

Epoch 19/80
----------
Loss: 1.0777 Acc: 0.5689 LR: 0.00100000

Epoch 20/80
----------
Loss: 0.9976 Acc: 0.6069 LR: 0.00100000

Epoch 21/80
----------
Loss: 0.9727 Acc: 0.6122 LR: 0.00100000

Epoch 22/80
----------
Loss: 0.9684 Acc: 0.6163 LR: 0.00100000

Epoch 23/80
----------
Loss: 0.9583 Acc: 0.6217 LR: 0.00100000

Epoch 24/80
----------
Loss: 0.9571 Acc: 0.6234 LR: 0.00100000

Epoch 25/80
----------
```

```
Loss: 0.9533 Acc: 0.6230 LR: 0.00100000

Epoch 26/80
----------
Loss: 0.9441 Acc: 0.6286 LR: 0.00100000

Epoch 27/80
----------
Loss: 0.9411 Acc: 0.6280 LR: 0.00100000

Epoch 28/80
----------
Loss: 0.9394 Acc: 0.6280 LR: 0.00100000

Epoch 29/80
----------
Loss: 0.9415 Acc: 0.6267 LR: 0.00100000

Epoch 30/80
----------
Loss: 0.9294 Acc: 0.6344 LR: 0.00100000

Epoch 31/80
----------
Loss: 0.9273 Acc: 0.6323 LR: 0.00100000

Epoch 32/80
----------
Loss: 0.9280 Acc: 0.6323 LR: 0.00100000

Epoch 33/80
----------
Loss: 0.9188 Acc: 0.6351 LR: 0.00100000

Epoch 34/80
----------
Loss: 0.9238 Acc: 0.6321 LR: 0.00100000

Epoch 35/80
----------
Loss: 0.9225 Acc: 0.6371 LR: 0.00100000

Epoch 36/80
----------
Loss: 0.9111 Acc: 0.6411 LR: 0.00010000

Epoch 37/80
----------
Loss: 0.8939 Acc: 0.6510 LR: 0.00010000

Epoch 38/80
```

```
----------
Loss: 0.8900 Acc: 0.6517 LR: 0.00010000

Epoch 39/80
----------
Loss: 0.8896 Acc: 0.6517 LR: 0.00010000

Epoch 40/80
----------
Loss: 0.8909 Acc: 0.6522 LR: 0.00010000

Epoch 41/80
----------
Loss: 0.8804 Acc: 0.6581 LR: 0.00010000

Epoch 42/80
----------
Loss: 0.8868 Acc: 0.6537 LR: 0.00010000

Epoch 43/80
----------
Loss: 0.8812 Acc: 0.6532 LR: 0.00010000

Epoch 44/80
----------
Loss: 0.8830 Acc: 0.6530 LR: 0.00010000

Epoch 45/80
----------
Loss: 0.8841 Acc: 0.6518 LR: 0.00001000

Epoch 46/80
----------
Loss: 0.8806 Acc: 0.6540 LR: 0.00001000

Epoch 47/80
----------
Loss: 0.8777 Acc: 0.6558 LR: 0.00001000

Epoch 48/80
----------
Loss: 0.8759 Acc: 0.6554 LR: 0.00001000

Epoch 49/80
----------
Loss: 0.8799 Acc: 0.6553 LR: 0.00001000

Epoch 50/80
----------
Loss: 0.8766 Acc: 0.6594 LR: 0.00001000
```

```
Epoch 51/80
----------
Loss: 0.8789 Acc: 0.6573 LR: 0.00001000

Epoch 52/80
----------
Loss: 0.8782 Acc: 0.6592 LR: 0.00001000

Epoch 53/80
----------
Loss: 0.8789 Acc: 0.6563 LR: 0.00001000

Epoch 54/80
----------
Loss: 0.8757 Acc: 0.6552 LR: 0.00000100

Epoch 55/80
----------
Loss: 0.8767 Acc: 0.6566 LR: 0.00000100

Epoch 56/80
----------
Loss: 0.8739 Acc: 0.6554 LR: 0.00000100

Epoch 57/80
----------
Loss: 0.8833 Acc: 0.6553 LR: 0.00000100

Epoch 58/80
----------
Loss: 0.8764 Acc: 0.6561 LR: 0.00000100

Epoch 59/80
----------
Loss: 0.8742 Acc: 0.6546 LR: 0.00000100

Epoch 60/80
----------
Loss: 0.8764 Acc: 0.6549 LR: 0.00000100

Epoch 61/80
----------
Loss: 0.8774 Acc: 0.6582 LR: 0.00000100

Epoch 62/80
----------
Loss: 0.8758 Acc: 0.6576 LR: 0.00000100

Epoch 63/80
----------
```

```
Loss: 0.8756 Acc: 0.6583 LR: 0.00000010

Epoch 64/80
----------
Loss: 0.8745 Acc: 0.6577 LR: 0.00000010

Epoch 65/80
----------
Loss: 0.8739 Acc: 0.6573 LR: 0.00000010

Epoch 66/80
----------
Loss: 0.8790 Acc: 0.6566 LR: 0.00000010

Epoch 67/80
----------
Loss: 0.8753 Acc: 0.6557 LR: 0.00000010

Epoch 68/80
----------
Loss: 0.8807 Acc: 0.6538 LR: 0.00000010

Epoch 69/80
----------
Loss: 0.8816 Acc: 0.6563 LR: 0.00000010

Epoch 70/80
----------
Loss: 0.8752 Acc: 0.6562 LR: 0.00000010

Epoch 71/80
----------
Loss: 0.8765 Acc: 0.6549 LR: 0.00000010

Epoch 72/80
----------
Loss: 0.8773 Acc: 0.6569 LR: 0.00000001

Epoch 73/80
----------
Loss: 0.8813 Acc: 0.6558 LR: 0.00000001

Epoch 74/80
----------
Loss: 0.8826 Acc: 0.6557 LR: 0.00000001

Epoch 75/80
----------
Loss: 0.8832 Acc: 0.6544 LR: 0.00000001

Epoch 76/80
```

```
Epoch 76/80
----------
Loss: 0.8757 Acc: 0.6590 LR: 0.00000001

Epoch 77/80
----------
Loss: 0.8793 Acc: 0.6545 LR: 0.00000001

Epoch 78/80
----------
Loss: 0.8758 Acc: 0.6577 LR: 0.00000001

Epoch 79/80
----------
Loss: 0.8791 Acc: 0.6542 LR: 0.00000001

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

In [ ]:

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