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]:

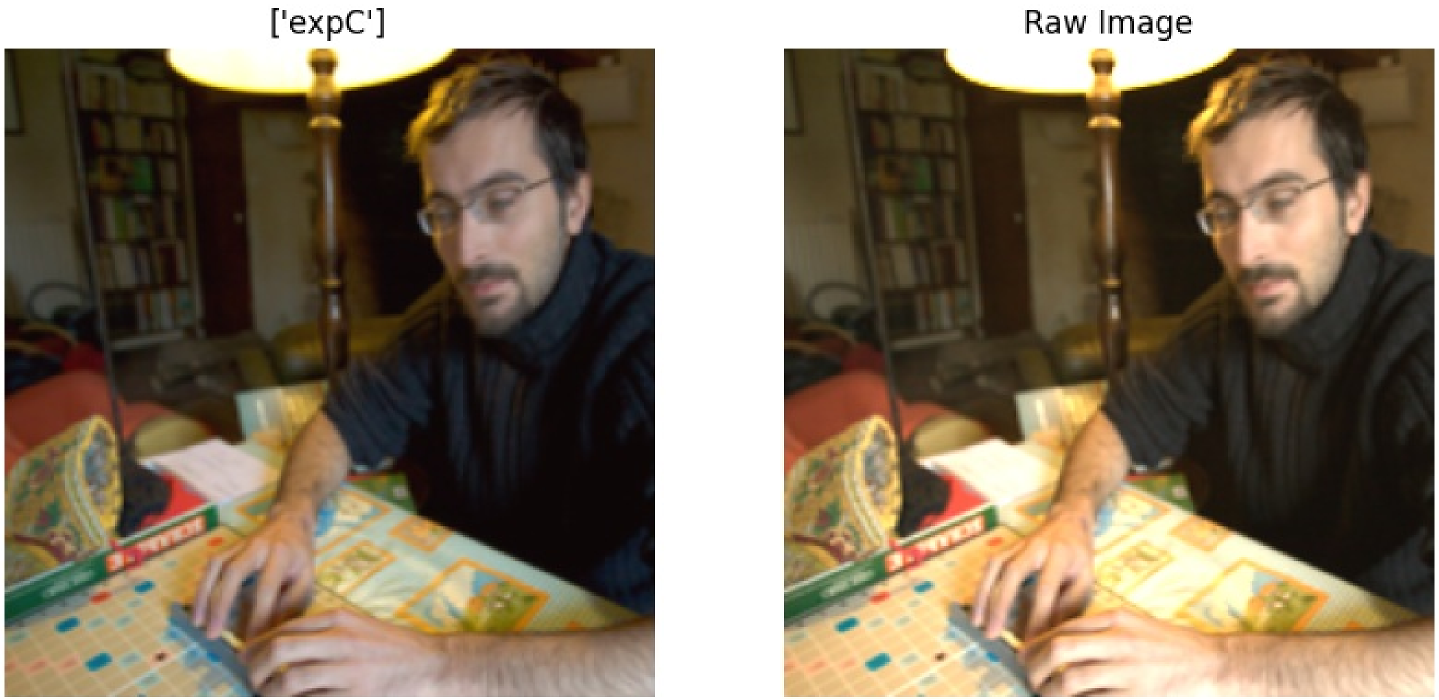
```
train_features, train_labels = next(iter(train_dataloader))
```

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:]
    mean = np.array([0.485, 0.456, 0.406])
    std = np.array([0.229, 0.224, 0.225])
    org_img = std * org_img + mean
    raw_img = std * raw_img + mean
    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]:

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/resetnet34_scratch_raw/resetnet34_scratch_raw_1.pth', '/content/drive/MyDrive/th
esis/model/checkpoints/resetnet34_scratch_raw/resetnet34_scratch_raw_2.pth', '/content/drive/MyDrive/thesis/model/checkpoints/rese
tnet34_scratch_raw/resetnet34_scratch_raw_3.pth', '/content/drive/MyDrive/thesis/model/checkpoints/resetnet34_scratch_raw/resetnet
34_scratch_raw_4.pth', '/content/drive/MyDrive/thesis/model/checkpoints/resetnet34_scratch_raw/resetnet34_scratch_raw_5.pth', '/co
ntent/drive/MyDrive/thesis/model/checkpoints/resetnet34_scratch_raw/resetnet34_scratch_raw_6.pth', '/content/drive/MyDrive/thesis/
model/checkpoints/resetnet34_scratch_raw/resetnet34_scratch_raw_7.pth', '/content/drive/MyDrive/thesis/model/checkpoints/resetnet3
4_scratch_raw/resetnet34_scratch_raw_8.pth', '/content/drive/MyDrive/thesis/model/checkpoints/resetnet34_scratch_raw/resetnet34_sc
ratch_raw_9.pth', '/content/drive/MyDrive/thesis/model/checkpoints/resetnet34_scratch_raw/resetnet34_scratch_raw_10.pth', '/conten
t/drive/MyDrive/thesis/model/checkpoints/resetnet34_scratch_raw/resetnet34_scratch_raw_11.pth', '/content/drive/MyDrive/thesis/mod
el/checkpoints/resetnet34_scratch_raw/resetnet34_scratch_raw_12.pth', '/content/drive/MyDrive/thesis/model/checkpoints/resetnet34_
scratch_raw/resetnet34_scratch_raw_13.pth', '/content/drive/MyDrive/thesis/model/checkpoints/resetnet34_scratch_raw/resetnet34_scr
atch_raw_14.pth', '/content/drive/MyDrive/thesis/model/checkpoints/resetnet34_scratch_raw/resetnet34_scratch_raw_15.pth', '/conten
atch_raw_14.pth', '/content/drive/MyDrive/thesis/model/checkpoints/resetnet34_scratch_raw/resetnet34_scratch_raw_15.pth', '/conten
t/drive/MyDrive/thesis/model/checkpoints/resetnet34_scratch_raw/resetnet34_scratch_raw_16.pth', '/content/drive/MyDrive/thesis/mod
el/checkpoints/resetnet34_scratch_raw/resetnet34_scratch_raw_17.pth', '/content/drive/MyDrive/thesis/model/checkpoints/resetnet34_
scratch_raw/resetnet34_scratch_raw_18.pth', '/content/drive/MyDrive/thesis/model/checkpoints/resetnet34_scratch_raw/resetnet34_scr
atch_raw_19.pth', '/content/drive/MyDrive/thesis/model/checkpoints/resetnet34_scratch_raw/resetnet34_scratch_raw_20.pth', '/conten
t/drive/MyDrive/thesis/model/checkpoints/resetnet34_scratch_raw/resetnet34_scratch_raw_21.pth', '/content/drive/MyDrive/thesis/mod
el/checkpoints/resetnet34_scratch_raw/resetnet34_scratch_raw_22.pth', '/content/drive/MyDrive/thesis/model/checkpoints/resetnet34_
scratch_raw/resetnet34_scratch_raw_23.pth', '/content/drive/MyDrive/thesis/model/checkpoints/resetnet34_scratch_raw/resetnet34_scr
atch_raw_24.pth', '/content/drive/MyDrive/thesis/model/checkpoints/resetnet34_scratch_raw/resetnet34_scratch_raw_25.pth', '/conten
t/drive/MyDrive/thesis/model/checkpoints/resetnet34_scratch_raw/resetnet34_scratch_raw_26.pth', '/content/drive/MyDrive/thesis/mod
el/checkpoints/resetnet34_scratch_raw/resetnet34_scratch_raw_27.pth', '/content/drive/MyDrive/thesis/model/checkpoints/resetnet34_
scratch_raw/resetnet34_scratch_raw_28.pth', '/content/drive/MyDrive/thesis/model/checkpoints/resetnet34_scratch_raw/resetnet34_scr
atch_raw_29.pth', '/content/drive/MyDrive/thesis/model/checkpoints/resetnet34_scratch_raw/resetnet34_scratch_raw_30.pth', '/conten
t/drive/MyDrive/thesis/model/checkpoints/resetnet34_scratch_raw/resetnet34_scratch_raw_31.pth', '/content/drive/MyDrive/thesis/mod
el/checkpoints/resetnet34_scratch_raw/resetnet34_scratch_raw_32.pth', '/content/drive/MyDrive/thesis/model/checkpoints/resetnet34_
scratch_raw/resetnet34_scratch_raw_33.pth', '/content/drive/MyDrive/thesis/model/checkpoints/resetnet34_scratch_raw/resetnet34_scr
atch_raw_34.pth', '/content/drive/MyDrive/thesis/model/checkpoints/resetnet34_scratch_raw/resetnet34_scratch_raw_35.pth', '/conten
t/drive/MyDrive/thesis/model/checkpoints/resetnet34_scratch_raw/resetnet34_scratch_raw_36.pth', '/content/drive/MyDrive/thesis/mod
el/checkpoints/resetnet34_scratch_raw/resetnet34_scratch_raw_37.pth', '/content/drive/MyDrive/thesis/model/checkpoints/resetnet34_
scratch_raw/resetnet34_scratch_raw_38.pth', '/content/drive/MyDrive/thesis/model/checkpoints/resetnet34_scratch_raw/resetnet34_scr
atch_raw_39.pth', '/content/drive/MyDrive/thesis/model/checkpoints/resetnet34_scratch_raw/resetnet34_scratch_raw_40.pth', '/conten
t/drive/MyDrive/thesis/model/checkpoints/resetnet34_scratch_raw/resetnet34_scratch_raw_41.pth', '/content/drive/MyDrive/thesis/mod
el/checkpoints/resetnet34_scratch_raw/resetnet34_scratch_raw_42.pth', '/content/drive/MyDrive/thesis/model/checkpoints/resetnet34_
scratch_raw/resetnet34_scratch_raw_43.pth', '/content/drive/MyDrive/thesis/model/checkpoints/resetnet34_scratch_raw/resetnet34_scr
atch_raw_44.pth', '/content/drive/MyDrive/thesis/model/checkpoints/resetnet34_scratch_raw/resetnet34_scratch_raw_45.pth', '/conten
t/drive/MyDrive/thesis/model/checkpoints/resetnet34_scratch_raw/resetnet34_scratch_raw_46.pth', '/content/drive/MyDrive/thesis/mod
el/checkpoints/resetnet34_scratch_raw/resetnet34_scratch_raw_47.pth']
tensor(0.5895, device='cuda:0', dtype=torch.float64)
Loaded model from checkpoint: /content/drive/MyDrive/thesis/model/checkpoints/resetnet34_scratch_raw/resetnet34_scratch_raw_47.pth
Resuming training from epoch 47
```

In [27]:

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

```
1e-05
```

In [28]:

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

```
Epoch 48/80
----------
Loss: 1.0299 Acc: 0.5907 LR: 0.00001000

Epoch 49/80
----------
Loss: 1.0244 Acc: 0.5954 LR: 0.00001000

Epoch 50/80
----------
```

```
Loss: 1.0284 Acc: 0.5897 LR: 0.00000100

Epoch 51/80
----------
Loss: 1.0294 Acc: 0.5896 LR: 0.00000100

Epoch 52/80
----------
Loss: 1.0244 Acc: 0.5959 LR: 0.00000100

Epoch 53/80
----------
Loss: 1.0207 Acc: 0.5947 LR: 0.00000100

Epoch 54/80
----------
Loss: 1.0141 Acc: 0.5974 LR: 0.00000100

Epoch 55/80
----------
Loss: 1.0095 Acc: 0.5988 LR: 0.00000100

Epoch 56/80
----------
Loss: 1.0259 Acc: 0.5949 LR: 0.00000100

Epoch 57/80
----------
Loss: 1.0223 Acc: 0.5952 LR: 0.00000100

Epoch 58/80
----------
Loss: 1.0236 Acc: 0.5935 LR: 0.00000100

Epoch 59/80
----------
Loss: 1.0258 Acc: 0.5933 LR: 0.00000100

Epoch 60/80
----------
Loss: 1.0249 Acc: 0.5938 LR: 0.00000100

Epoch 61/80
----------
Loss: 1.0206 Acc: 0.5981 LR: 0.00000100

Epoch 62/80
----------
Loss: 1.0195 Acc: 0.6016 LR: 0.00000100

Epoch 63/80
Epoch 63/80
----------
Loss: 1.0191 Acc: 0.5959 LR: 0.00000100

Epoch 64/80
----------
Loss: 1.0159 Acc: 0.6002 LR: 0.00000100

Epoch 65/80
----------
Loss: 1.0216 Acc: 0.5922 LR: 0.00000100

Epoch 66/80
----------
Loss: 1.0295 Acc: 0.5947 LR: 0.00000100

Epoch 67/80
----------
Loss: 1.0224 Acc: 0.5887 LR: 0.00000100

Epoch 68/80
----------
Loss: 1.0266 Acc: 0.5967 LR: 0.00000100

Epoch 69/80
----------
Loss: 1.0191 Acc: 0.5958 LR: 0.00000100

Epoch 70/80
----------
Loss: 1.0256 Acc: 0.5965 LR: 0.00000010

Epoch 71/80
----------
Loss: 1.0221 Acc: 0.5939 LR: 0.00000010

Epoch 72/80
----------
Loss: 1.0272 Acc: 0.5965 LR: 0.00000010

Epoch 73/80
----------
Loss: 1.0319 Acc: 0.5923 LR: 0.00000010

Epoch 74/80
----------
Loss: 1.0187 Acc: 0.5964 LR: 0.00000010

Epoch 75/80
----------
Loss: 1.0277 Acc: 0.5917 LR: 0.00000010
```

```
Epoch 76/80
----------
Loss: 1.0277 Acc: 0.5947 LR: 0.00000010

Epoch 77/80
----------
Loss: 1.0276 Acc: 0.5922 LR: 0.00000010

Epoch 78/80
----------
Loss: 1.0221 Acc: 0.5936 LR: 0.00000010

Epoch 79/80
----------
Loss: 1.0314 Acc: 0.5885 LR: 0.00000010

Epoch 80/80
----------
Loss: 1.0280 Acc: 0.5948 LR: 0.00000010

Training complete in 165m 32s
Best Acc: 0.601600
```

In [ ]:

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