```
!mkdir -p /content/decathlon_data
!wget http://www.robots.ox.ac.uk/~vgg/share/decathlon-1.0-data.tar.gz -O /content/decathlon_data/decathlon-1.0-data.tar.gz
```

```
--2024-11-13 16:14:48--  http://www.robots.ox.ac.uk/~vgg/share/decathlon-1.0-data.tar.gz
Resolving www.robots.ox.ac.uk (www.robots.ox.ac.uk)... 129.67.94.2
Connecting to www.robots.ox.ac.uk (www.robots.ox.ac.uk)|129.67.94.2|:80... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://www.robots.ox.ac.uk/~vgg/share/decathlon-1.0-data.tar.gz [following]
--2024-11-13 16:14:48--  https://www.robots.ox.ac.uk/~vgg/share/decathlon-1.0-data.tar.gz
Connecting to www.robots.ox.ac.uk (www.robots.ox.ac.uk)|129.67.94.2|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 406351554 (388M) [application/x-gzip]
Saving to: ‘/content/decathlon_data/decathlon-1.0-data.tar.gz’

/content/decathlon_ 100%[===================>] 387.53M  21.3MB/s    in 20s

2024-11-13 16:15:09 (19.7 MB/s) - ‘/content/decathlon_data/decathlon-1.0-data.tar.gz’ saved [406351554/406351554]
```

```
import tarfile

with tarfile.open('/content/decathlon_data/decathlon-1.0-data.tar.gz') as tar:
    tar.extractall(path='/content/decathlon_data')
```

```
with tarfile.open('/content/decathlon_data/aircraft.tar') as tar:
    tar.extractall(path='/content/decathlon_data/')
```

```
# Cell 1: Import necessary libraries and set up device
import torch
from torch.optim.lr_scheduler import MultiStepLR
import torch.nn as nn
from torchvision import models, transforms
from torch.utils.data import DataLoader
from torchvision.datasets import ImageFolder

# Set up device (GPU if available)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")
```

```
Using device: cuda
```

```
import os
from torchvision import datasets, transforms
from torch.utils.data import DataLoader

train_dir = '/content/decathlon_data/aircraft/train'
val_dir = '/content/decathlon_data/aircraft/val'

transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

train_dataset = datasets.ImageFolder(train_dir, transform=transform)
val_dataset = datasets.ImageFolder(val_dir, transform=transform)

train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=64, shuffle=False)
```

```
import torch
import torch.nn as nn
import torchvision.models as models

# resnet50
model = models.resnet50(pretrained=True)

# final fc layer
```

```
num_classes = len(train_dataset.classes)
model.fc = nn.Linear(model.fc.in_features, num_classes)
model = model.to('cuda')
```

```
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is depreca
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet50-0676ba61.pth" to /root/.cache/torch/hub/checkpoints/resnet50-0676
100%|██████████| 97.8M/97.8M [00:00<00:00, 191MB/s]
```

```
import torch.optim as optim

from torch.optim.lr_scheduler import MultiStepLR

num_epochs = 20  # 150
initial_lr = 0.001
milestones = [5, 10, 15]
gamma = 0.1

optimizer = torch.optim.SGD(model.parameters(), lr=initial_lr, momentum=0.9)
scheduler = MultiStepLR(optimizer, milestones=milestones, gamma=gamma)
criterion = torch.nn.CrossEntropyLoss()



import torch.cuda.amp as amp
scaler = amp.GradScaler()  # mixed-precision


def train_one_epoch(epoch):
    model.train()
    running_loss = 0.0
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()

        with torch.cuda.amp.autocast():  # Mixed precision
            outputs = model(images)
            loss = criterion(outputs, labels)

        loss.backward()
        optimizer.step()
        running_loss += loss.item()

    print(f"Epoch {epoch} - Training loss: {running_loss / len(train_loader)}")

def validate():
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for images, labels in val_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    accuracy = 100 * correct / total
    print(f"Validation Accuracy: {accuracy}%")
    return accuracy
```

```
<ipython-input-19-42204bd81e0f>:2: FutureWarning: `torch.cuda.amp.GradScaler(args...)` is deprecated. Please use `torch.amp.
  scaler = amp.GradScaler()  # mixed-precision scaler
```

```
best_accuracy = 0
for epoch in range(num_epochs):
    train_one_epoch(epoch)
    accuracy = validate()
    scheduler.step()  # Update learning rate
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        torch.save(model.state_dict(), "best_finetuned_model.pth")

    print(f"Epoch {epoch+1}/{num_epochs} completed. Best Accuracy so far: {best_accuracy}%")
```

```
Validation Accuracy: 5.6705670567056705%
Epoch 2/20 completed. Best Accuracy so far: 5.6705670567056705%
Epoch 2 - Training loss: 4.260826641658567
Validation Accuracy: 8.4008400840084%
Epoch 3/20 completed. Best Accuracy so far: 8.4008400840084%
Epoch 3 - Training loss: 4.010825202150165
Validation Accuracy: 12.211221122112212%
Epoch 4/20 completed. Best Accuracy so far: 12.211221122112212%
Epoch 4 - Training loss: 3.7188505991449894
Validation Accuracy: 14.641464146414641%
Epoch 5/20 completed. Best Accuracy so far: 14.641464146414641%
Epoch 5 - Training loss: 3.494002382710295
Validation Accuracy: 15.661566156615661%
Epoch 6/20 completed. Best Accuracy so far: 15.661566156615661%
Epoch 6 - Training loss: 3.4613864556798397
Validation Accuracy: 15.931593159315932%
Epoch 7/20 completed. Best Accuracy so far: 15.931593159315932%
Epoch 7 - Training loss: 3.4242069091436997
Validation Accuracy: 16.32163216321632%
Epoch 8/20 completed. Best Accuracy so far: 16.32163216321632%
Epoch 8 - Training loss: 3.3968107835301815
Validation Accuracy: 17.13171317131713%
Epoch 9/20 completed. Best Accuracy so far: 17.13171317131713%
Epoch 9 - Training loss: 3.363909860826888
Validation Accuracy: 17.76177617761776%
Epoch 10/20 completed. Best Accuracy so far: 17.76177617761776%
Epoch 10 - Training loss: 3.351577012044079
Validation Accuracy: 17.671767176717672%
Epoch 11/20 completed. Best Accuracy so far: 17.76177617761776%
Epoch 11 - Training loss: 3.3303668409023643
Validation Accuracy: 17.4017401740174%
Epoch 12/20 completed. Best Accuracy so far: 17.76177617761776%
Epoch 12 - Training loss: 3.3286257105053596
Validation Accuracy: 17.431743174317432%
Epoch 13/20 completed. Best Accuracy so far: 17.76177617761776%
Epoch 13 - Training loss: 3.3340248521768823
Validation Accuracy: 17.64176417641764%
Epoch 14/20 completed. Best Accuracy so far: 17.76177617761776%
Epoch 14 - Training loss: 3.3198551456883267
Validation Accuracy: 17.791779177917793%
Epoch 15/20 completed. Best Accuracy so far: 17.791779177917793%
Epoch 15 - Training loss: 3.3208403092510297
Validation Accuracy: 17.52175217521752%
Epoch 16/20 completed. Best Accuracy so far: 17.791779177917793%
Epoch 16 - Training loss: 3.318177493113392
Validation Accuracy: 17.791779177917793%
Epoch 17/20 completed. Best Accuracy so far: 17.791779177917793%
Epoch 17 - Training loss: 3.3214211239005036
Validation Accuracy: 18.061806180618063%
Epoch 18/20 completed. Best Accuracy so far: 18.061806180618063%
Epoch 18 - Training loss: 3.3247753944037095
Validation Accuracy: 17.581758175817583%
Epoch 19/20 completed. Best Accuracy so far: 18.061806180618063%
Epoch 19 - Training loss: 3.3232493355589092
Validation Accuracy: 17.64176417641764%
Epoch 20/20 completed. Best Accuracy so far: 18.061806180618063%
```

```
# new lr
learning_rates = [0.01, 0.1]
results = {}
num_epochs = 20




def reset_model():
    model = models.resnet50(pretrained=True)
    model.fc = nn.Linear(model.fc.in_features, num_classes)
    return model.to(device)

for lr in learning_rates:
    print(f"\nStarting experiment with learning rate = {lr}")
    model = reset_model()
    optimizer = torch.optim.SGD(model.parameters(), lr=lr, momentum=0.9)
    scheduler = MultiStepLR(optimizer, milestones=milestones, gamma=gamma)

    best_accuracy = 0

    for epoch in range(num_epochs):
        train_one_epoch(epoch)
        accuracy = validate()
        scheduler.step()

        if accuracy > best_accuracy:
            best_accuracy = accuracy
            torch.save(model.state_dict(), f"best_finetuned_model_lr_{lr}.pth")

        print(f"Epoch {epoch+1}/{num_epochs} completed. Best Accuracy so far with LR {lr}: {best_accuracy}%")

    results[lr] = best_accuracy

for lr, accuracy in results.items():
    print(f"Final best accuracy with learning rate {lr}: {accuracy}%")
```

```
Epoch 2 - Training loss: 4.500230177393499
Validation Accuracy: 1.3501350135013501%
Epoch 3/20 completed. Best Accuracy so far with LR 0.1: 2.1002100210021%
Epoch 3 - Training loss: 4.428632187393476
Validation Accuracy: 2.6102610261026102%
Epoch 4/20 completed. Best Accuracy so far with LR 0.1: 2.6102610261026102%
Epoch 4 - Training loss: 4.351501140954359
Validation Accuracy: 3.6903690369036903%
Epoch 5/20 completed. Best Accuracy so far with LR 0.1: 3.6903690369036903%
Epoch 5 - Training loss: 4.202538652240105
Validation Accuracy: 4.2004200420042%
Epoch 6/20 completed. Best Accuracy so far with LR 0.1: 4.2004200420042%
Epoch 6 - Training loss: 4.101372619844833
Validation Accuracy: 4.71047104710471%
Epoch 7/20 completed. Best Accuracy so far with LR 0.1: 4.71047104710471%
Epoch 7 - Training loss: 4.0063172331396135
Validation Accuracy: 4.53045304530453%
Epoch 8/20 completed. Best Accuracy so far with LR 0.1: 4.71047104710471%
Epoch 8 - Training loss: 3.9393026603842682
Validation Accuracy: 4.59045904590459%
Epoch 9/20 completed. Best Accuracy so far with LR 0.1: 4.71047104710471%
Epoch 9 - Training loss: 3.870732118498604
Validation Accuracy: 6.0606060606060606%
Epoch 10/20 completed. Best Accuracy so far with LR 0.1: 6.0606060606060606%
Epoch 10 - Training loss: 3.695266350260321
Validation Accuracy: 7.170717071707171%
Epoch 11/20 completed. Best Accuracy so far with LR 0.1: 7.170717071707171%
Epoch 11 - Training loss: 3.6369820855698496
Validation Accuracy: 7.260726072607261%
Epoch 12/20 completed. Best Accuracy so far with LR 0.1: 7.260726072607261%
Epoch 12 - Training loss: 3.5877696343188017
Validation Accuracy: 7.050705070507051%
Epoch 13/20 completed. Best Accuracy so far with LR 0.1: 7.260726072607261%
Epoch 13 - Training loss: 3.559072831891618
Validation Accuracy: 7.650765076507651%
Epoch 14/20 completed. Best Accuracy so far with LR 0.1: 7.650765076507651%
Epoch 14 - Training loss: 3.5189842368071935
Validation Accuracy: 7.650765076507651%
Epoch 15/20 completed. Best Accuracy so far with LR 0.1: 7.650765076507651%
Epoch 15 - Training loss: 3.4759910466536037
Validation Accuracy: 7.650765076507651%
Epoch 16/20 completed. Best Accuracy so far with LR 0.1: 7.650765076507651%
Epoch 16 - Training loss: 3.4782895322115914
Validation Accuracy: 7.320732073207321%
Epoch 17/20 completed. Best Accuracy so far with LR 0.1: 7.650765076507651%
Epoch 17 - Training loss: 3.4875368802052624
Validation Accuracy: 7.590759075907591%
Epoch 18/20 completed. Best Accuracy so far with LR 0.1: 7.650765076507651%
Epoch 18 - Training loss: 3.4543918258738966
Validation Accuracy: 7.620762076207621%
Epoch 19/20 completed. Best Accuracy so far with LR 0.1: 7.650765076507651%
Epoch 19 - Training loss: 3.459229586259374
Validation Accuracy: 7.440744074407441%
Epoch 20/20 completed. Best Accuracy so far with LR 0.1: 7.650765076507651%
Final best accuracy with learning rate 0.01: 57.605760576057605%
Final best accuracy with learning rate 0.1: 7.650765076507651%
```

Start coding or generate with AI.

Three learning rates were compared: 0.001, 0.01, and 0.1. At 0.001, the model converged steadily, achieving a best accuracy of 18.06%, indicating slow but stable learning. With 0.01, accuracy improved significantly to 57.61%, showing that a higher rate accelerated convergence and improved performance, However, with 0.1, accuracy dropped to 7.17%, likely due to excessive updates that hindered convergence. Overall, 0.01 provided the best results, balancing faster learning with stable convergence.

```
import torch
import torch.nn as nn
import torchvision.models as models

model = models.resnet50(pretrained=True)

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

num_classes = len(train_dataset.classes)
model.fc = nn.Linear(model.fc.in_features, num_classes)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = model.to(device)



import torch.optim as optim
from torch.optim.lr_scheduler import MultiStepLR

def train_one_epoch(epoch, dataloader, criterion, optimizer):
    model.train()
    running_loss = 0.0
    for inputs, labels in dataloader:
        inputs, labels = inputs.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item() * inputs.size(0)

    epoch_loss = running_loss / len(dataloader.dataset)
    print(f"Epoch {epoch} - Training loss: {epoch_loss}")


def validate(dataloader, criterion):
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for inputs, labels in dataloader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            _, preds = torch.max(outputs, 1)
            correct += (preds == labels).sum().item()
            total += labels.size(0)

    accuracy = 100 * correct / total
    print(f"Validation Accuracy: {accuracy}%")
    return accuracy




learning_rates = [0.01, 0.001] #[0.1,0.01,0.001]
num_epochs = 20
milestones = [5, 10, 15]

for lr in learning_rates:
    print(f"\nStarting experiment with learning rate = {lr}")
    optimizer = optim.SGD(model.fc.parameters(), lr=lr, momentum=0.9)
    scheduler = MultiStepLR(optimizer, milestones=milestones, gamma=0.1)
    criterion = nn.CrossEntropyLoss()

    best_accuracy = 0
    for epoch in range(num_epochs):
        train_one_epoch(epoch, train_loader, criterion, optimizer)
        accuracy = validate(val_loader, criterion)

        if accuracy > best_accuracy:
            best_accuracy = accuracy
            torch.save(model.state_dict(), f"best_feature_extractor_lr_{lr}.pth")

        scheduler.step()
    print(f"Best Accuracy with LR={lr}: {best_accuracy}%")
```

```
Epoch 13 - Training loss: 2.6465751937426845
Validation Accuracy: 18.81188118811881%
Epoch 14 - Training loss: 2.6419636473801584
Validation Accuracy: 18.391839183918393%
Epoch 15 - Training loss: 2.6458321075347917
Validation Accuracy: 18.421842184218423%
Epoch 16 - Training loss: 2.6482593464007547
Validation Accuracy: 18.21182118211821%
Epoch 17 - Training loss: 2.647386990411595
Validation Accuracy: 18.6018601860186%
Epoch 18 - Training loss: 2.634377986997205
Validation Accuracy: 18.421842184218423%
Epoch 19 - Training loss: 2.635482817977649
Validation Accuracy: 18.421842184218423%
Best Accuracy with LR=0.01: 18.81188118811881%

Starting experiment with learning rate = 0.001
Epoch 0 - Training loss: 2.656406985547776
Validation Accuracy: 18.69186918691869%
Epoch 1 - Training loss: 2.6462399905215643
Validation Accuracy: 18.72187218721872%
Epoch 2 - Training loss: 2.6255331062312317
Validation Accuracy: 18.781878187818783%
Epoch 3 - Training loss: 2.6133554242082035
Validation Accuracy: 19.141914191419144%
Epoch 4 - Training loss: 2.5943562279365415
Validation Accuracy: 18.93189318931893%
Epoch 5 - Training loss: 2.5581023102401614
Validation Accuracy: 18.81188118811881%
Epoch 6 - Training loss: 2.5547952250084194
Validation Accuracy: 19.08190819081908%
Epoch 7 - Training loss: 2.557178331551326
Validation Accuracy: 18.631863186318633%
Epoch 8 - Training loss: 2.5486749276903193
Validation Accuracy: 18.72187218721872%
Epoch 9 - Training loss: 2.546762874760024
Validation Accuracy: 18.751875187518753%
Epoch 10 - Training loss: 2.545011272098608
Validation Accuracy: 18.901890189018903%
Epoch 11 - Training loss: 2.551009082765585
Validation Accuracy: 18.781878187818783%
Epoch 12 - Training loss: 2.5469813667233288
Validation Accuracy: 18.84188418841884%
Epoch 13 - Training loss: 2.5422537564230168
Validation Accuracy: 19.021902190219024%
Epoch 14 - Training loss: 2.5458513687811144
Validation Accuracy: 18.661866186618663%
Epoch 15 - Training loss: 2.5418577438782415
Validation Accuracy: 19.05190519051905%
Epoch 16 - Training loss: 2.550052585041158
Validation Accuracy: 18.661866186618663%
Epoch 17 - Training loss: 2.545046376552708
Validation Accuracy: 18.84188418841884%
Epoch 18 - Training loss: 2.5437792268568837
Validation Accuracy: 18.72187218721872%
Epoch 19 - Training loss: 2.552525713119095
Validation Accuracy: 19.17191719171917%
Best Accuracy with LR=0.001: 19.17191719171917%
```

Start coding or generate with AI.

In feature extraction with a frozen ResNet50 model, learning rates of 0.1, 0.01, and 0.001 yielded distinct results in validation accuracy and training dynamics. Learning rate 0.1 achieved the highest validation accuracy at 19.53%, though training loss fluctuated. Learning rate 0.01 reached a peak accuracy of 18.81% with more consistent loss. The lowest rate, 0.001, produced a peak accuracy of 19.14% with slow, steady reduction in loss over epochs, showing stable but gradual progress. Higher learning rates accelerated convergence due to updates in only the final layer.

Start coding or generate with AI.

Start coding or generate with AI.

full finetuning yielded a much higher final accuracy (~59%) compared to feature extraction (where the best accuracy reached around 19.53%) across all tested learning rates. The winning approach was full finetuning with a learning rate of 0.01, which showed strong performance, achieving the highest validation accuracy due to the model adjusting weights in all layers.

The main reason full finetuning outperformed feature extraction is that updating all layers allows the model to adapt more to the nuances of the new dataset, especially when the target dataset is distinct from the source dataset (ImageNet). By freezing most of the layers, feature extraction restricts adaptation, resulting in lower accuracy as it relies only on adjustments to the final layer. Full finetuning takes advantage of all layers' ability to learn dataset-specific features.

Start coding or generate with AI.