Problem 1 - Learning Rate, Batch Size, FashionMNIST

[ ] ↳ 已隐藏 21 个单元格

Problem 2 - Convolutional Neural Networks Architectures

↳ 已隐藏 21 个单元格

Problem 3 - Transfer learning: Shallow learning vs Finetuning, Pytorch

Q1(a).

```
# check is gpu working
import tensorflow as tf
tf.test.gpu_device_name()
```

```
'/device:GPU:0'
```

```
# get the entire compressed folder
!wget http://www.robots.ox.ac.uk/~vgg/share/decathlon-1.0-data.tar.gz
```

```
--2022-11-06 02:23:13--  http://www.robots.ox.ac.uk/~vgg/share/decathlon-1.0-data
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... conne
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://www.robots.ox.ac.uk/~vgg/share/decathlon-1.0-data.tar.gz [foll
--2022-11-06 02:23:13--  https://www.robots.ox.ac.uk/~vgg/share/decathlon-1.0-da
Connecting to www.robots.ox.ac.uk (www.robots.ox.ac.uk)|129.67.94.2|:443... conn
HTTP request sent, awaiting response... 200 OK
Length: 406351554 (388M) [application/x-gzip]
Saving to: ‘decathlon-1.0-data.tar.gz’

decathlon-1.0-data. 100%[===================>] 387.53M  35.3MB/s    in 12s

2022-11-06 02:23:25 (33.5 MB/s) - ‘decathlon-1.0-data.tar.gz’ saved [406351554/4
```

```
# extract the compressed folder
!tar -xzvf "/content/decathlon-1.0-data.tar.gz"
```

```
aircraft.tar
cifar100.tar
daimlerpedcls.tar
dtd.tar
gtsrb.tar
omniglot.tar
svhn.tar
ucf101.tar
vgg-flowers.tar
```

```
# extract the compressed folder for the dataset
! tar -xvf /content/gtsrb.tar
```

```
gtsrb/train/0020/000087.jpg
gtsrb/train/0020/000131.jpg
gtsrb/train/0020/000066.jpg
gtsrb/train/0020/000182.jpg
gtsrb/train/0020/000051.jpg
gtsrb/train/0020/000181.jpg
gtsrb/train/0020/000163.jpg
gtsrb/train/0020/000118.jpg
gtsrb/train/0020/000185.jpg
gtsrb/train/0020/000080.jpg
gtsrb/train/0020/000128.jpg
gtsrb/train/0020/000079.jpg
gtsrb/train/0020/000006.jpg
gtsrb/train/0020/000169.jpg
gtsrb/train/0020/000158.jpg
gtsrb/train/0020/000038.jpg
gtsrb/train/0020/000186.jpg
gtsrb/train/0020/000085.jpg
gtsrb/train/0020/000102.jpg
gtsrb/train/0020/000001.jpg
gtsrb/train/0020/000069.jpg
gtsrb/train/0020/000023.jpg
gtsrb/train/0020/000155.jpg
gtsrb/train/0020/000167.jpg
gtsrb/train/0020/000067.jpg
gtsrb/train/0020/000052.jpg

gtsrb/train/0020/000177.jpg
gtsrb/train/0020/000095.jpg
gtsrb/train/0020/000126.jpg
gtsrb/train/0020/000119.jpg
gtsrb/train/0020/000123.jpg
gtsrb/train/0020/000074.jpg
gtsrb/train/0020/000121.jpg
gtsrb/train/0020/000170.jpg
gtsrb/train/0020/000183.jpg
gtsrb/train/0020/000156.jpg
gtsrb/train/0020/000205.jpg
gtsrb/train/0020/000064.jpg
gtsrb/train/0020/000193.jpg
gtsrb/train/0020/000018.jpg
gtsrb/train/0020/000024.jpg
gtsrb/train/0020/000192.jpg
```

```
gtsrb/train/0020/000148.jpg
gtsrb/train/0020/000168.jpg
gtsrb/train/0020/000091.jpg
gtsrb/train/0020/000111.jpg
gtsrb/train/0020/000133.jpg
gtsrb/train/0020/000210.jpg
gtsrb/train/0020/000017.jpg
gtsrb/train/0020/000124.jpg
gtsrb/train/0020/000020.jpg
gtsrb/train/0020/000176.jpg
gtsrb/train/0020/000150.jpg
gtsrb/train/0020/000178.jpg
gtsrb/train/0020/000013.jpg
gtsrb/train/0020/000135.jpg
gtsrb/train/0020/000014.jpg
gtsrb/train/0020/000012.jpg
```

```
# License: BSD
# Author: Sasank Chilamkurthy

from __future__ import print_function, division

import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
import torch.backends.cudnn as cudnn
import numpy as np
import torchvision
from torchvision import datasets, models, transforms
import matplotlib.pyplot as plt
import time
import os
import copy

cudnn.benchmark = True
plt.ion()   # interactive mode
```

```
# Data augmentation and normalization for training
# Just normalization for validation
data_transforms = {
    'train': transforms.Compose([
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
    'val': transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
```

```
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
}

data_dir = '/content/gtsrb'
image_datasets = {x: datasets.ImageFolder(os.path.join(data_dir, x),
                                          data_transforms[x])
                  for x in ['train', 'val']}
dataloaders = {x: torch.utils.data.DataLoader(image_datasets[x], batch_size=64,
                                             shuffle=True, num_workers=4)
              for x in ['train', 'val']}
dataset_sizes = {x: len(image_datasets[x]) for x in ['train', 'val']}
class_names = image_datasets['train'].classes

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

```
    /usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:566: UserWa
      cpuset_checked))
```

```
# visualize a few training images so as to understand the data augmentations
def imshow(inp, title=None):
    """Imshow 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


# Get a batch of training data
inputs, classes = next(iter(dataloaders['train']))

# Make a grid from batch
out = torchvision.utils.make_grid(inputs[:4])

imshow(out, title=[class_names[x] for x in classes[:4]])
```


```
from collections import Counter
```

```
train_class = image_datasets['train'].classes
train_class_label = [label for pic, label in image_datasets['train']]
count_train_label = Counter(train_class_label)

val_class = image_datasets['val'].classes
val_class_label = [label for pic, label in image_datasets['val']]
count_val_label = Counter(val_class_label)

plt.style.use('seaborn')
fig, axs = plt.subplots(2, 1, figsize=(10, 10))

axs[0].bar(count_train_label.keys(), count_train_label.values())
axs[0].set_xlabel('class')
axs[0].set_ylabel('num of images')
axs[0].set_title('distribution of images per class in train dataset')

axs[1].bar(count_val_label.keys(),count_val_label.values())
axs[1].set_xlabel('class')
axs[1].set_ylabel('num of images')
axs[1].set_title('distribution of images per class in val dataset')
```

```
Text(0.5, 1.0, 'distribution of images per class in val dataset')
```

Q1(b).

```
# Training the model
def train_model(model, criterion, optimizer, scheduler, num_epochs=25):
    since = time.time()

    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0

    for epoch in range(num_epochs):
        print(f'Epoch {epoch}/{num_epochs - 1}')
        print('-' * 10)

        # Each epoch has a training and validation phase
        for phase in ['train', 'val']:
            if phase == 'train':
                model.train()  # Set model to training mode
            else:
                model.eval()   # Set model to evaluate mode

            running_loss = 0.0
            running_corrects = 0

            # Iterate over data.
            for inputs, labels in dataloaders[phase]:
                inputs = inputs.to(device)
                labels = labels.to(device)

                # zero the parameter gradients
                optimizer.zero_grad()

                # forward
                # track history if only in train
                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
                    _, preds = torch.max(outputs, 1)
                    loss = criterion(outputs, labels)

                    # backward + optimize only if in training phase
                    if phase == 'train':
                        loss.backward()
                        optimizer.step()

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

```
                running_corrects += torch.sum(preds == labels.data)
            if phase == 'train':
                scheduler.step()

            epoch_loss = running_loss / dataset_sizes[phase]
            epoch_acc = running_corrects.double() / dataset_sizes[phase]

            print(f'{phase} Loss: {epoch_loss:.4f} Acc: {epoch_acc:.4f}')

            # deep copy the model
            if phase == 'val' and epoch_acc > best_acc:
                best_acc = epoch_acc
                best_model_wts = copy.deepcopy(model.state_dict())

        print()

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

    # load best model weights
    model.load_state_dict(best_model_wts)
    return model


# Visualizing the model predictions
def visualize_model(model, num_images=6):
    was_training = model.training
    model.eval()
    images_so_far = 0
    fig = plt.figure()

    with torch.no_grad():
        for i, (inputs, labels) in enumerate(dataloaders['val']):
            inputs = inputs.to(device)
            labels = labels.to(device)

            outputs = model(inputs)
            _, preds = torch.max(outputs, 1)

            for j in range(inputs.size()[0]):
                images_so_far += 1
                ax = plt.subplot(num_images//2, 2, images_so_far)
                ax.axis('off')
                ax.set_title(f'predicted: {class_names[preds[j]]}')
                imshow(inputs.cpu().data[j])

                if images_so_far == num_images:
                    model.train(mode=was_training)
                    return
        model.train(mode=was_training)
```

```
# Finetuning the convnet
model_ft = models.resnet50(pretrained=True)
num_ftrs = model_ft.fc.in_features
# Here the size of each output sample is generalized to nn.Linear(num_ftrs, len(class_
model_ft.fc = nn.Linear(num_ftrs, len(class_names))

model_ft = model_ft.to(device)

criterion = nn.CrossEntropyLoss()

# Observe that all parameters are being optimized
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)

# Decay LR by a factor of 0.1 every 3 epochs
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=3, gamma=0.1)

# Train and evaluate
# reduce the number of epochs, but just make sure you decay the learning rate 3 times
model_ft = train_model(model_ft, criterion, optimizer_ft, exp_lr_scheduler, num_epochs
visualize_model(model_ft)
```

```
/usr/local/lib/python3.7/dist-packages/torchvision/models/_utils.py:209: UserWar
  f"The parameter '{pretrained_param}' is deprecated since 0.13 and will be remo
/usr/local/lib/python3.7/dist-packages/torchvision/models/_utils.py:223: UserWar
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet50-0676ba61.pth" to /roo
```

100% 97.8M/97.8M [00:00<00:00, 243MB/s]

```
Epoch 0/11
----------
/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:566: UserW
  cpuset_checked))
train Loss: 1.5457 Acc: 0.5985
val Loss: 0.3089 Acc: 0.9181

Epoch 1/11
----------
train Loss: 0.4405 Acc: 0.8753
val Loss: 0.1026 Acc: 0.9727

Epoch 2/11
----------
train Loss: 0.3122 Acc: 0.9070
val Loss: 0.0674 Acc: 0.9770

Epoch 3/11
----------
train Loss: 0.2626 Acc: 0.9210
val Loss: 0.0531 Acc: 0.9857

Epoch 4/11
----------
train Loss: 0.2373 Acc: 0.9295
val Loss: 0.0505 Acc: 0.9871

Epoch 5/11
----------
train Loss: 0.2401 Acc: 0.9292
val Loss: 0.0476 Acc: 0.9872

Epoch 6/11
----------
train Loss: 0.2451 Acc: 0.9263
val Loss: 0.0454 Acc: 0.9874

Epoch 7/11
----------
train Loss: 0.2346 Acc: 0.9293
val Loss: 0.0461 Acc: 0.9878

Epoch 8/11
----------
train Loss: 0.2384 Acc: 0.9285
val Loss: 0.0438 Acc: 0.9871

Epoch 9/11
----------
```

```
train Loss: 0.2337 Acc: 0.9303
val Loss: 0.0469 Acc: 0.9871

Epoch 10/11
----------
train Loss: 0.2350 Acc: 0.9283
val Loss: 0.0461 Acc: 0.9883

Epoch 11/11
----------
train Loss: 0.2357 Acc: 0.9294
val Loss: 0.0450 Acc: 0.9888

Training complete in 66m 30s
Best val Acc: 0.988778
```

predicted: 0019

Q1(c).

predicted: 0013

```
# keeping learning rate of all layers at 0.01

# Finetuning the convnet
model_fixedlr1 = models.resnet50(pretrained=True)
num_ftrs = model_fixedlr1.fc.in_features
# Here the size of each output sample is generalized to nn.Linear(num_ftrs, len(class_
model_fixedlr1.fc = nn.Linear(num_ftrs, len(class_names))

model_fixedlr1 = model_fixedlr1.to(device)

criterion = nn.CrossEntropyLoss()

# Observe that all parameters are being optimized
optimizer_ft = optim.SGD(model_fixedlr1.parameters(), lr=0.01, momentum=0.9)

# Constant LR
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=3, gamma=1)

# Train and evaluate
# reduce the number of epochs, but just make sure you decay the learning rate 3 times
model_fixedlr1 = train_model(model_fixedlr1, criterion, optimizer_ft, exp_lr_schedule
visualize_model(model_fixedlr1)
```

```
/usr/local/lib/python3.7/dist-packages/torchvision/models/_utils.py:209: UserWarn
  f"The parameter '{pretrained_param}' is deprecated since 0.13 and will be remov
/usr/local/lib/python3.7/dist-packages/torchvision/models/_utils.py:223: UserWarn
  warnings.warn(msg)
Epoch 0/11
----------
/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:566: UserWa
  cpuset_checked))
train Loss: 0.6690 Acc: 0.8045
val Loss: 0.2612 Acc: 0.9143

Epoch 1/11
----------
train Loss: 0.2910 Acc: 0.9081
val Loss: 0.0624 Acc: 0.9745

Epoch 2/11
----------
train Loss: 0.2370 Acc: 0.9232
val Loss: 0.0783 Acc: 0.9703

Epoch 3/11
----------
train Loss: 0.2257 Acc: 0.9286
val Loss: 0.0460 Acc: 0.9856

Epoch 4/11
----------
train Loss: 0.1952 Acc: 0.9364
val Loss: 0.0196 Acc: 0.9939

Epoch 5/11
----------
train Loss: 0.1813 Acc: 0.9418
val Loss: 0.0172 Acc: 0.9949

Epoch 6/11
----------
train Loss: 0.1704 Acc: 0.9449
val Loss: 0.0316 Acc: 0.9895

Epoch 7/11
----------
train Loss: 0.1650 Acc: 0.9464
val Loss: 0.0158 Acc: 0.9954

Epoch 8/11
----------
train Loss: 0.1591 Acc: 0.9487
val Loss: 0.0151 Acc: 0.9959

Epoch 9/11
----------
train Loss: 0.1551 Acc: 0.9500
val Loss: 0.0118 Acc: 0.9968
```

```
Epoch 10/11
----------
train Loss: 0.1434 Acc: 0.9536
val Loss: 0.0163 Acc: 0.9959

Epoch 11/11
----------
train Loss: 0.1426 Acc: 0.9535
val Loss: 0.0061 Acc: 0.9978

Training complete in 66m 19s
Best val Acc: 0.997832
```

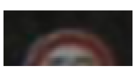
predicted: 0020

predicted: 0002

predicted: 0035

predicted: 0026

predicted: 0003



```
# keeping learning rate of all layers at 0.1

# Finetuning the convnet
model_fixedlr = models.resnet50(pretrained=True)
num_ftrs = model_fixedlr.fc.in_features
# Here the size of each output sample is generalized to nn.Linear(num_ftrs, len(class_
model_fixedlr.fc = nn.Linear(num_ftrs, len(class_names))

model_fixedlr = model_fixedlr.to(device)

criterion = nn.CrossEntropyLoss()

# Observe that all parameters are being optimized
optimizer_ft = optim.SGD(model_fixedlr.parameters(), lr=0.1, momentum=0.9)

# Constant LR
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=3, gamma=1)
```

```
# Train and evaluate
# reduce the number of epochs, but just make sure you decay the learning rate 3 times
model_fixedlr = train_model(model_fixedlr, criterion, optimizer_ft, exp_lr_scheduler,
visualize_model(model_fixedlr)
```

```
Epoch 0/11
----------
train Loss: 3.3074 Acc: 0.1281
val Loss: 2.9758 Acc: 0.1495

Epoch 1/11
----------
train Loss: 2.7759 Acc: 0.2031
val Loss: 2.4119 Acc: 0.2821

Epoch 2/11
----------
train Loss: 2.2107 Acc: 0.3207
val Loss: 2.0204 Acc: 0.3771

Epoch 3/11
----------
train Loss: 1.6819 Acc: 0.4609
val Loss: 1.2516 Acc: 0.5782

Epoch 4/11
----------
train Loss: 1.0689 Acc: 0.6641
val Loss: 0.6353 Acc: 0.7766

Epoch 5/11
----------
train Loss: 0.6852 Acc: 0.7858
val Loss: 0.3068 Acc: 0.8973

Epoch 6/11
----------
train Loss: 0.5295 Acc: 0.8349
val Loss: 0.2369 Acc: 0.9248

Epoch 7/11
----------
train Loss: 0.4343 Acc: 0.8631
val Loss: 0.1511 Acc: 0.9485

Epoch 8/11
----------
train Loss: 0.3815 Acc: 0.8805
val Loss: 0.0943 Acc: 0.9691
```

1. finetune with 0.001 exponential decay learning rate

- Best val Acc: 0.988778

2. finetune with 0.01 constant learning rate

- Best val Acc: 0.997832

3. finetune with 0.1 constant learning rate

- Best val Acc: 0.982147

Thus, 0.01 constant learning rate gives the best accuracy on the target dataset.

```
Training complete in 65m 42s
```

Q2(a).

```
learning_rate = [1, 0.1, 0.01, 0.001]

for lr in learning_rate:
  model_conv = torchvision.models.resnet50(pretrained=True)
  # freeze all the network except the final layer
  for param in model_conv.parameters():
    param.requires_grad = False

  # Parameters of newly constructed modules have requires_grad=True by default
  num_ftrs = model_conv.fc.in_features
  model_conv.fc = nn.Linear(num_ftrs, len(class_names))

  model_conv = model_conv.to(device)

  criterion = nn.CrossEntropyLoss()

  # Observe that only parameters of final layer are being optimized as
  # opposed to before.
  optimizer_conv = optim.SGD(model_conv.fc.parameters(), lr=lr, momentum=0.9)

  # Constant LR
  exp_lr_scheduler = lr_scheduler.StepLR(optimizer_conv, step_size=3, gamma=1)

  # Train and evaluate
  model_conv = train_model(model_conv, criterion, optimizer_conv,
                         exp_lr_scheduler, num_epochs=10)
  visualize_model(model_conv)
  plt.ioff()
  plt.show()
```

```
Epoch 0/9
----------
train Loss: 90.4331 Acc: 0.3776
val Loss: 68.8344 Acc: 0.4638

Epoch 1/9
----------
train Loss: 47.3495 Acc: 0.4600
val Loss: 54.2641 Acc: 0.4971

Epoch 2/9
----------
train Loss: 43.3212 Acc: 0.4935
val Loss: 31.8074 Acc: 0.5955

Epoch 3/9
----------
train Loss: 41.4446 Acc: 0.5108
val Loss: 35.0210 Acc: 0.5899

Epoch 4/9
----------
train Loss: 42.9836 Acc: 0.5153
val Loss: 34.9825 Acc: 0.6187

Epoch 5/9
----------
train Loss: 38.8874 Acc: 0.5350
val Loss: 24.0335 Acc: 0.6524

Epoch 6/9
----------
train Loss: 37.7525 Acc: 0.5393
val Loss: 37.5836 Acc: 0.5950

Epoch 7/9
----------
train Loss: 37.2203 Acc: 0.5469
val Loss: 48.2749 Acc: 0.5819

Epoch 8/9
----------
train Loss: 35.0830 Acc: 0.5624
val Loss: 21.1686 Acc: 0.6878

Epoch 9/9
----------
train Loss: 37.1811 Acc: 0.5534
val Loss: 26.9658 Acc: 0.6612

Training complete in 21m 35s
Best val Acc: 0.687835
```

predicted: 0036

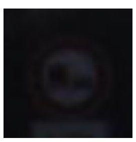
predicted: 0010

predicted: 0008

predicted: 0013

predicted: 0026

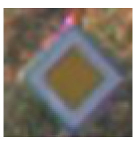
predicted: 0011



```
Epoch 0/9
----------
train Loss: 6.0322 Acc: 0.3974
val Loss: 2.3638 Acc: 0.5954

Epoch 1/9
----------
train Loss: 3.9670 Acc: 0.4804
val Loss: 4.5536 Acc: 0.5214

Epoch 2/9
----------
train Loss: 3.8569 Acc: 0.5070
val Loss: 3.1590 Acc: 0.6168

Epoch 3/9
----------
train Loss: 3.6777 Acc: 0.5252
val Loss: 2.9310 Acc: 0.6137

Epoch 4/9
----------
train Loss: 3.7027 Acc: 0.5390
val Loss: 1.6918 Acc: 0.6950

Epoch 5/9
----------
train Loss: 3.4862 Acc: 0.5460
```

```
val Loss: 2.2259 Acc: 0.6583

Epoch 6/9
----------
train Loss: 3.5571 Acc: 0.5504
val Loss: 3.3054 Acc: 0.6116

Epoch 7/9
----------
train Loss: 3.5547 Acc: 0.5604
val Loss: 1.5734 Acc: 0.7365

Epoch 8/9
----------
train Loss: 3.3064 Acc: 0.5715
val Loss: 2.8493 Acc: 0.6441

Epoch 9/9
----------
train Loss: 3.3607 Acc: 0.5716
val Loss: 2.7948 Acc: 0.6308

Training complete in 21m 37s
Best val Acc: 0.736547
```

predicted: 0013

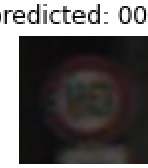


predicted: 0005

predicted: 0041

predicted: 0035

predicted: 0009



predicted: 0030