CS441: Applied ML - HW 5

Part 1: Applications of AI

Nothing to code for this part.

Part 2: Fine-Tune for Pets Image Classification

Include all the code for Part 2 in this section

2.1 Prepare Data

In [ ]:
```
import torch
import torch.nn as nn
import torch.optim.lr_scheduler as lrs
from torch.utils.data import DataLoader
import torchvision
from torchvision import datasets
from torchvision import transforms
import matplotlib.pyplot as plt
from tqdm import tqdm

import os
from pathlib import Path
import numpy as np
```

In [ ]:
```
datadir = "."
save_dir = "."  # change to your directory
```

In [ ]:
```
def load_pet_dataset(train_transform = None, test_transform = None):
    OxfordIIITPet = datasets.OxfordIIITPet
    if os.path.isdir(datadir+ "oxford-iiit-pet"):
      do_download = False
    else:
      do_download = True
    training_set = OxfordIIITPet(root = datadir,
                                 split = 'trainval',
                                 transform = train_transform,
```

```
                               download = do_download)

    test_set = OxfordIIITPet(root = datadir,
                             split = 'test',
                             transform = test_transform,
                             download = do_download)
    return training_set, test_set
```

In [ ]:
```
train_set, test_set = load_pet_dataset()

# Display a sample in OxfordIIIPet dataset
sample_idx = 3000 # Choose an image index that you want to display
print("Label:", train_set.classes[train_set[sample_idx][1]])
train_set[sample_idx][0]
```

```
Label: Persian
```

Out[ ]:

2.2 Data Preprocess

In [ ]:
```
from torchvision import transforms
from torch.utils.data import DataLoader
```

In [ ]:
```
# Feel free to add augmentation choices

# Apply data augmentation
train_transform = transforms.Compose([
```

```
            transforms.Resize(224),
            transforms.CenterCrop(224),
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                 std= [0.229, 0.224, 0.225]),
        ])

test_transform = transforms.Compose([
            transforms.Resize(224),  # resize to 224x224 because that's the
            transforms.CenterCrop(224),
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                 std= [0.229, 0.224, 0.225]),
        ])
```

In [ ]:

```
# Feel free to change
train_set, test_set = load_pet_dataset(train_transform, test_transform)
train_loader = DataLoader(dataset=train_set,
                          batch_size=64,
                          shuffle=True,
                          num_workers=2)

test_loader = DataLoader(dataset=test_set,
                          batch_size=64,
                          shuffle=False,
                          num_workers=2)
```

2.3 Helper Functions

In [ ]:

```
# Display the number of parameters and model structure
def display_model(model):
  # Check number of parameters
  summary_dict = {}
  num_params = 0
  summary_str = ['='*80]

  for module_name, module in model.named_children():
      summary_count = 0
      for name, param in module.named_parameters():
          if(param.requires_grad):
              summary_count += param.numel()
              num_params += param.numel()
      summary_dict[module_name] = [summary_count]
      summary_str+= [f'- {module_name: <40} : {str(summary_count):^34s}']

  summary_dict['total'] = [num_params]

  # print summary string
```

```
    summary_str += ['='*80]
    summary_str += ['--' +  f'{"Total":<40} : {str(num_params) + " params":^34
    print('\n'.join(summary_str))

    # print model structure
    print(model)
```

In [ ]:
```
# Plot loss or accuracy
def plot_losses(train, val, test_frequency, num_epochs):
    plt.plot(train, label="train")
    indices = [i for i in range(num_epochs) if ((i+1)%test_frequency == 0 or
    plt.plot(indices, val, label="val")
    plt.title("Loss Plot")
    plt.ylabel("Loss")
    plt.xlabel("Epoch")
    plt.legend()
    plt.show()

def plot_accuracy(train, val, test_frequency, num_epochs):
    indices = [i for i in range(num_epochs) if ((i+1)%test_frequency == 0 or
    plt.plot(indices, train, label="train")
    plt.plot(indices, val, label="val")
    plt.title("Accuracy Plot")
    plt.ylabel("Accuracy")
    plt.xlabel("Epoch")
    plt.legend()
    plt.show()

def save_checkpoint(save_dir, model, save_name = 'best_model.pth'):
    save_path = os.path.join(save_dir, save_name)
    torch.save(model.state_dict(), save_path)

def load_model(model, save_dir, save_name = 'best_model.pth'):
    save_path = os.path.join(save_dir, save_name)
    model.load_state_dict(torch.load(save_path))
    return model
```

2.4 YOUR TASK: Fine-Tune Pre-trained Network on Pets

Read and understand the code and then uncomment it. Then, set up your learning rate, learning scheduler, and train/evaluate. Adjust as necessary to reach target performance.

In [ ]:
```
# set device, using GPU 'cuda' will be faster
device = 'cuda'
```

In [ ]:
```
def train(train_loader, model, criterion, optimizer):
```

```
    """
    Train network
    :param train_loader: training dataloader
    :param model: model to be trained
    :param criterion: criterion used to calculate loss (should be CrossEntro
    :param optimizer: optimizer for model's params (Adams or SGD)
    :return: mean training loss
    """
    model.train()
    loss_ = 0.0
    losses = []

    # TO DO: read this documentation and then uncomment the line below; http
    it_train = tqdm(enumerate(train_loader), total=len(train_loader), desc="
    for i, (images, labels) in it_train:

        # TO DO: read/understand these lines and then uncomment the code bel

        images, labels = images.to(device), labels.to(device)

        # zero the gradient
        optimizer.zero_grad()

        # predict labels
        prediction = model(images)

        # compute loss
        loss = criterion(prediction, labels)

        # set text to display
        it_train.set_description(f'loss: {loss:.3f}')

        # compute gradients
        loss.backward()

        # update weights
        optimizer.step()

        # keep track of losses
        losses.append(loss)

    return torch.stack(losses).mean().item()

def test(test_loader, model, criterion):
    """
    Test network.
    :param test_loader: testing dataloader
    :param model: model to be tested
    :param criterion: criterion used to calculate loss (should be CrossEntro
    :return: mean_accuracy: mean accuracy of predicted labels
```

```
            test_loss: mean test loss during testing
    """
    model.eval()
    losses = []
    correct = 0
    total = 0

    # TO DO: read this documentation and then uncomment the line below; http
    it_test = tqdm(enumerate(test_loader), total=len(test_loader), desc="Val
    for i, (images, labels) in it_test:

      # TO DO: read/understand and then uncomment these lines

      images, labels = images.to(device), labels.to(device)
      with torch.no_grad():  # https://pytorch.org/docs/stable/generated/tor
       output = model(images) # do not compute gradient when performing pre
      preds = torch.argmax(output, dim=-1)
      loss = criterion(output, labels)
      losses.append(loss.item())
      correct += (preds == labels).sum().item()
      total += len(labels)

    mean_accuracy = correct / total
    test_loss = np.mean(losses)
    print('Mean Accuracy: {0:.4f}'.format(mean_accuracy))
    print('Avg loss: {}'.format(test_loss))

    return mean_accuracy, test_loss
```

In [ ]:
```
# loads a pre-trained ResNet-34 model
model = torch.hub.load('pytorch/vision:v0.10.0', 'resnet34', pretrained=True
num_target_classes = 37

# TO DO: replace the last layer (classification head) with a new linear laye
in_features = model.fc.in_features
model.fc = torch.nn.Linear(in_features, num_target_classes)

model = model.to(device)
display_model(model) # displays the model structure and parameter count
```

```
Using cache found in /home/janghl/.cache/torch/hub/pytorch_vision_v0.10.0
============================================================================
====
- conv1                                       :              9408
- bn1                                         :              128
- relu                                        :               0
- maxpool                                     :               0
- layer1                                      :             221952
- layer2                                      :            1116416
- layer3                                      :            6822400
```

```
- layer4                                    :             13114368
- avgpool                                   :                0
- fc                                        :              18981
==========================================================================
====
--Total                                     :          21303653 params
--
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3),
bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running
_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_
mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
    )
    (2): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
    )
  )
```

```
    (layer2): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(
1, 1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
        (downsample): Sequential(
          (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      )
      (2): BasicBlock(
        (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      )
      (3): BasicBlock(
        (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      )
```

```
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(
1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
    )
    (2): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
    )
    (3): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
```

```
    )
    (4): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
    )
    (5): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(
1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_ru
```

```
nning_stats=True)
    )
    (2): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=37, bias=True)
)
```

In [ ]:

```
# Training Setting. Feel free to change.
num_epochs = 10
test_interval = 1

# TO DO: set initial learning rate
learn_rate = 1e-4
optimizer = torch.optim.Adam(model.parameters(), lr=learn_rate)

# TO DO: define your learning rate scheduler, e.g. StepLR
# https://pytorch.org/docs/stable/optim.html#module-torch.optim.lr_scheduler
lr_scheduler =  torch.optim.lr_scheduler.StepLR(optimizer, step_size=1, gamm

criterion = torch.nn.CrossEntropyLoss()

train_losses = []
train_accuracy_list = []
test_losses = []
test_accuracy_list = []


# Iterate over the DataLoader for training data
for epoch in tqdm(range(num_epochs), total=num_epochs, desc="Training ...",
    # Train the network for one epoch
    train_loss = train(train_loader, model, criterion, optimizer)

    # TO DO: uncomment the line below. It should be called each epoch to app
    lr_scheduler.step()

    train_losses.append(train_loss)
    print(f'Loss for Training on epoch {str(epoch)} is {str(train_loss)} \n'

    # Get the train accuracy and test loss/accuracy
```

```
    if(epoch%test_interval==0 or epoch==1 or epoch==num_epochs-1):
        print('Evaluating Network')

        train_accuracy, _ = test(train_loader, model, criterion) # Get trair
        train_accuracy_list.append(train_accuracy)

        print(f'Training accuracy on epoch {str(epoch)} is {str(train_accura

        test_accuracy, test_loss = test(test_loader, model, criterion) # Get
        test_losses.append(test_loss)
        test_accuracy_list.append(test_accuracy)
        print(f'Test (val) accuracy on epoch {str(epoch)} is {str(test_accur

        # Checkpoints are used to save the model with best validation accura
        if test_accuracy >= max(test_accuracy_list):
          print("Saving Model")
          save_checkpoint(save_dir, model, save_name = 'best_model.pth') # S
```

```
loss: 0.388: 100%|██████████| 58/58 [00:15<00:00,  3.76it/s]
Loss for Training on epoch 0 is 1.3497225046157837

Evaluating Network
Validating ...: 100%|██████████| 58/58 [00:06<00:00,  9.33it/s]
Mean Accuracy: 0.9747
Avg loss: 0.24443417993085137
Training accuracy on epoch 0 is 0.9747282608695652

Validating ...: 100%|██████████| 58/58 [00:07<00:00,  7.77it/s]
Mean Accuracy: 0.8913
Avg loss: 0.505778176003489
Test (val) accuracy on epoch 0 is 0.8912510220768601

Saving Model
loss: 0.256: 100%|██████████| 58/58 [00:15<00:00,  3.84it/s]
Loss for Training on epoch 1 is 0.19841448962688446

Evaluating Network
Validating ...: 100%|██████████| 58/58 [00:06<00:00,  9.16it/s]
Mean Accuracy: 0.9962
Avg loss: 0.05935830410955281
Training accuracy on epoch 1 is 0.996195652173913

Validating ...: 100%|██████████| 58/58 [00:06<00:00,  8.72it/s]
Mean Accuracy: 0.8972
Avg loss: 0.387282551491055
Test (val) accuracy on epoch 1 is 0.8972472063232488

Saving Model
```

```
loss: 0.045: 100%|██████████| 58/58 [00:15<00:00,  3.84it/s]
Loss for Training on epoch 2 is 0.057415686547756195

Evaluating Network
Validating ...: 100%|██████████| 58/58 [00:06<00:00,  9.44it/s]
Mean Accuracy: 1.0000
Avg loss: 0.015908844752 94372
Training accuracy on epoch 2 is 1.0

Validating ...: 100%|██████████| 58/58 [00:06<00:00,  9.03it/s]
Mean Accuracy: 0.9122
Avg loss: 0.31880737738362674
Test (val) accuracy on epoch 2 is 0.9122376669392205

Saving Model
loss: 0.032: 100%|██████████| 58/58 [00:15<00:00,  3.82it/s]
Loss for Training on epoch 3 is 0.02456776797771454

Evaluating Network
Validating ...: 100%|██████████| 58/58 [00:06<00:00,  9.32it/s]
Mean Accuracy: 1.0000
Avg loss: 0.00957316209176748
Training accuracy on epoch 3 is 1.0

Validating ...: 100%|██████████| 58/58 [00:06<00:00,  8.53it/s]
Mean Accuracy: 0.9106
Avg loss: 0.31288707840802343
Test (val) accuracy on epoch 3 is 0.9106023439629327

loss: 0.027: 100%|██████████| 58/58 [00:14<00:00,  3.89it/s]
Loss for Training on epoch 4 is 0.016316721215844154

Evaluating Network
Validating ...: 100%|██████████| 58/58 [00:06<00:00,  8.88it/s]
Mean Accuracy: 1.0000
Avg loss: 0.005977000045622217 5
Training accuracy on epoch 4 is 1.0

Validating ...: 100%|██████████| 58/58 [00:06<00:00,  8.81it/s]
Mean Accuracy: 0.9207
Avg loss: 0.2894534972849591
Test (val) accuracy on epoch 4 is 0.9206868356500408

Saving Model
loss: 0.012: 100%|██████████| 58/58 [00:15<00:00,  3.80it/s]
```

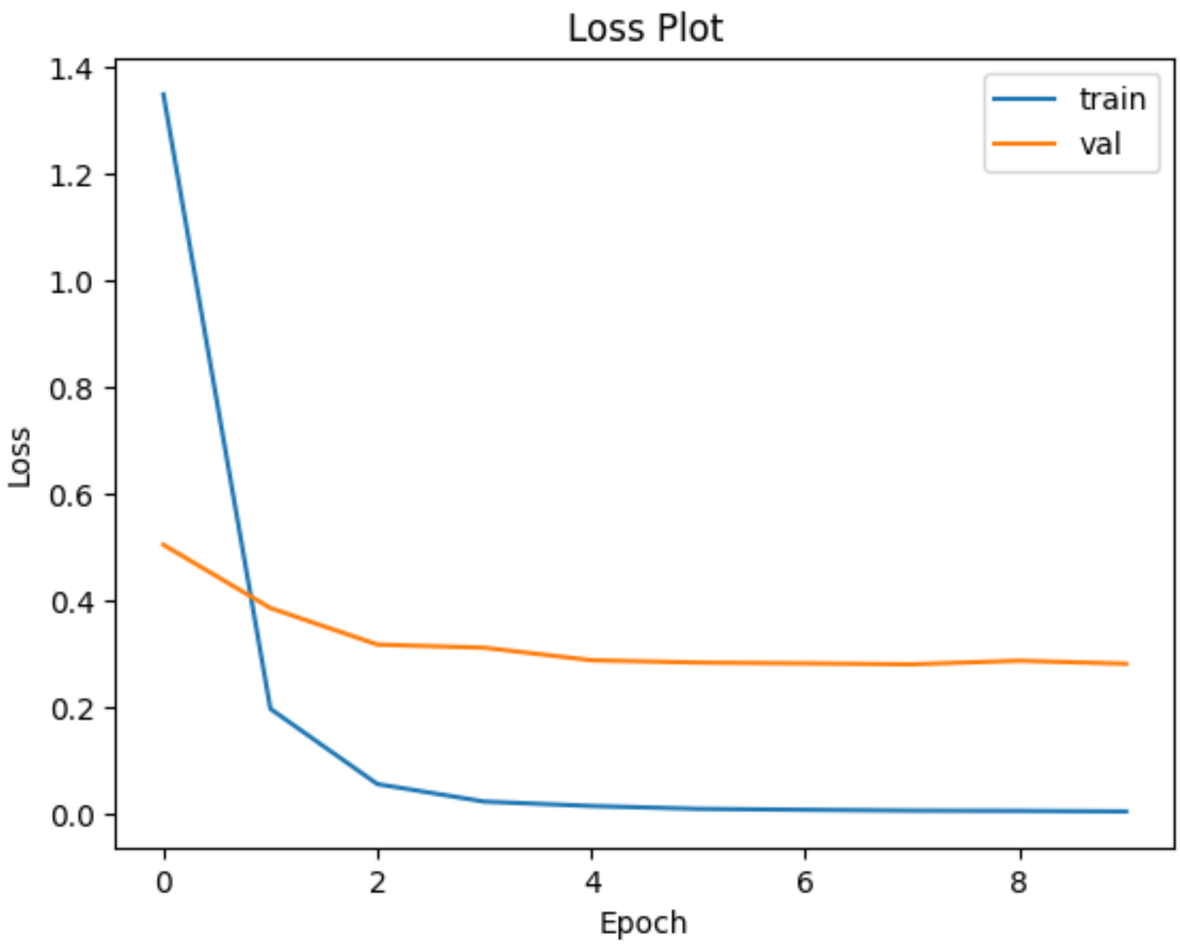
```
Loss for Training on epoch 5 is 0.010935946367681026

Evaluating Network
Validating ...: 100%|██████████| 58/58 [00:06<00:00,  9.03it/s]
Mean Accuracy: 1.0000
Avg loss: 0.004317048186404181
Training accuracy on epoch 5 is 1.0

Validating ...: 100%|██████████| 58/58 [00:06<00:00,  8.93it/s]
Mean Accuracy: 0.9210
Avg loss: 0.2850372875109315
Test (val) accuracy on epoch 5 is 0.9209593894794222

Saving Model
loss: 0.010: 100%|██████████| 58/58 [00:15<00:00,  3.80it/s]
Loss for Training on epoch 6 is 0.00921674631536007

Evaluating Network
Validating ...: 100%|██████████| 58/58 [00:06<00:00,  9.26it/s]
Mean Accuracy: 1.0000
Avg loss: 0.0036559286166046715
Training accuracy on epoch 6 is 1.0

Validating ...: 100%|██████████| 58/58 [00:06<00:00,  9.21it/s]
Mean Accuracy: 0.9204
Avg loss: 0.2835886925780054
Test (val) accuracy on epoch 6 is 0.9204142818206595

loss: 0.008: 100%|██████████| 58/58 [00:15<00:00,  3.84it/s]
Loss for Training on epoch 7 is 0.007689479738473892

Evaluating Network
Validating ...: 100%|██████████| 58/58 [00:06<00:00,  9.33it/s]
Mean Accuracy: 1.0000
Avg loss: 0.0029499575511777194
Training accuracy on epoch 7 is 1.0

Validating ...: 100%|██████████| 58/58 [00:06<00:00,  8.87it/s]
Mean Accuracy: 0.9199
Avg loss: 0.28181350989074544
Test (val) accuracy on epoch 7 is 0.919869174161897

loss: 0.005: 100%|██████████| 58/58 [00:15<00:00,  3.84it/s]
Loss for Training on epoch 8 is 0.00715823145583272

Evaluating Network
Validating ...: 100%|██████████| 58/58 [00:06<00:00,  9.24it/s]
```

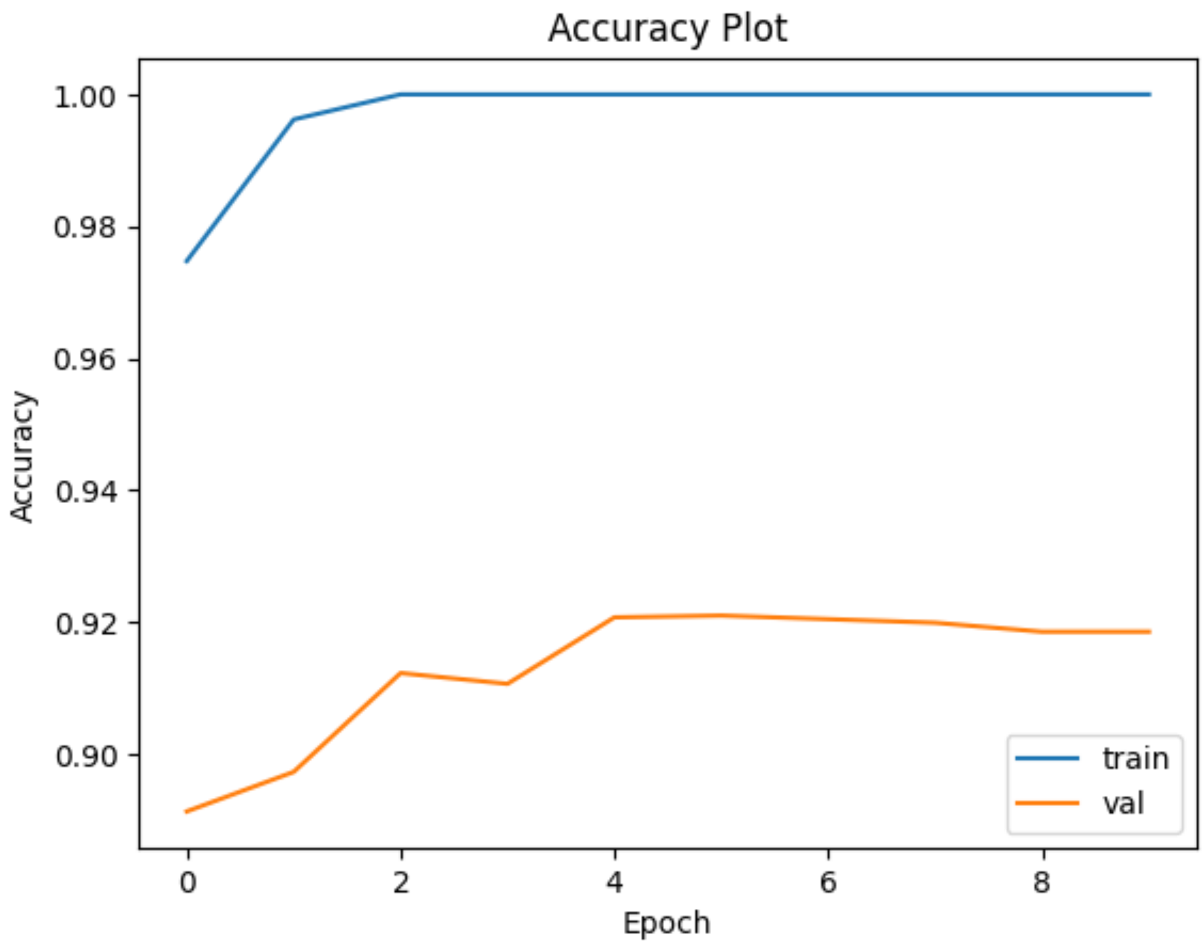
```
Mean Accuracy: 1.0000
Avg loss: 0.0027045900692585214
Training accuracy on epoch 8 is 1.0

Validating ...: 100%|██████████| 58/58 [00:06<00:00,  8.88it/s]
Mean Accuracy: 0.9185
Avg loss: 0.28851110982740746
Test (val) accuracy on epoch 8 is 0.9185064050149905

loss: 0.009: 100%|██████████| 58/58 [00:15<00:00,  3.84it/s]
Loss for Training on epoch 9 is 0.006114728283137083

Evaluating Network
Validating ...: 100%|██████████| 58/58 [00:06<00:00,  9.39it/s]
Mean Accuracy: 1.0000
Avg loss: 0.0022216173077548116
Training accuracy on epoch 9 is 1.0

Validating ...: 100%|██████████| 58/58 [00:06<00:00,  8.89it/s]
Training ...: 100%|██████████| 10/10 [04:43<00:00, 28.36s/it]
Mean Accuracy: 0.9185
Avg loss: 0.282880000994894
Test (val) accuracy on epoch 9 is 0.9185064050149905
```

2.5 Plotting of losses and accuracy

In [ ]:

```
plot_losses(train_losses, test_losses, test_interval, num_epochs)
plot_accuracy(train_accuracy_list, test_accuracy_list, test_interval, num_ep
```


Loss Plot
train
val
Loss
Epoch
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8



2.6 Evaluating trained model

In [ ]:

```
# TO DO: initialize your trained model as you did before so that you can loa

model = torch.hub.load('pytorch/vision:v0.10.0', 'resnet34', pretrained=True
# replace last layer
in_features = model.fc.in_features
model.fc = torch.nn.Linear(in_features, num_target_classes)

model = model.to(device)
load_model(model, save_dir) # Load the trained weight

test_accuracy, test_loss= test(test_loader, model, criterion)
print(f"Validation accuracy is {str(test_accuracy)} \n")
```

```
Using cache found in /home/janghl/.cache/torch/hub/pytorch_vision_v0.10.0
Validating ...: 100%|██████████| 58/58 [00:12<00:00,  4.59it/s]
Mean Accuracy: 0.9210
Avg loss: 0.2850372875109315
Validation accuracy is 0.9209593894794222
```

Part 3: CLIP: Contrastive Language-Image Pretraining

Include all the code for Part 3 in this section

3.1 Prepare data

Here is the json file you need for labels of flowers 102

In [ ]:
```
import json
import os
import os.path as osp
import numpy as np
import torch
from torchvision.datasets import Flowers102
%matplotlib inline
from matplotlib import pyplot as plt
```

In [ ]:
```
datadir = "."
```

In [ ]:
```
def load_flower_data(img_transform=None):
    if os.path.isdir(datadir+ "flowers-102"):
      do_download = False
    else:
      do_download = True
    train_set = Flowers102(root=datadir, split='train', transform=img_transf
    test_set = Flowers102(root=datadir, split='val', transform=img_transform
    classes = json.load(open(osp.join(datadir, "flowers102_classes.json")))

    return train_set, test_set, classes
```

In [ ]:
```
# READ ME!  This takes some time (a few minutes), so if you are using Colabs
#           first set to use GPU: Edit->Notebook Settings->Hardware Accelera

# Data structure details
#   flower_train[n][0] is the nth train image
#   flower_train[n][1] is the nth train label
#   flower_test[n][0] is the nth test image
#   flower_test[n][1] is the nth test label
#   flower_classes[k] is the name of the kth class
flower_train, flower_test, flower_classes = load_flower_data()
```

In [ ]:
```
len(flower_train), len(flower_test)  # output should be (1020, 1020)
```

Out[ ]: (1020, 1020)

In [ ]:
```
# Display a sample in Flowers 102 dataset
sample_idx = 5 # Choose an image index that you want to display
print("Label:", flower_classes[flower_train[sample_idx][1]])
flower_train[sample_idx][0]
```

Label: hard-leaved pocket orchid

Out[ ]:


3.2 Prepare CLIP model

In [ ]:
```
!pip install git+https://github.com/openai/CLIP.git
```

```
Collecting git+https://github.com/openai/CLIP.git
  Cloning https://github.com/openai/CLIP.git to /tmp/pip-req-build-4_ff0q1l
  Running command git clone --filter=blob:none --quiet https://github.com/op
enai/CLIP.git /tmp/pip-req-build-4_ff0q1l
  Resolved https://github.com/openai/CLIP.git to commit a1d071733d7111c9c014
f024669f959182114e33
  Preparing metadata (setup.py) ... done
Requirement already satisfied: ftfy in /home/janghl/anaconda3/envs/DNN/lib/p
ython3.9/site-packages (from clip==1.0) (6.2.0)
Requirement already satisfied: regex in /home/janghl/anaconda3/envs/DNN/lib/
python3.9/site-packages (from clip==1.0) (2023.12.25)
Requirement already satisfied: tqdm in /home/janghl/anaconda3/envs/DNN/lib/p
ython3.9/site-packages (from clip==1.0) (4.66.2)
Requirement already satisfied: torch in /home/janghl/anaconda3/envs/DNN/lib/
python3.9/site-packages (from clip==1.0) (1.11.0+cu113)
Requirement already satisfied: torchvision in /home/janghl/anaconda3/envs/DN
N/lib/python3.9/site-packages (from clip==1.0) (0.12.0+cu113)
Requirement already satisfied: wcwidth<0.3.0,>=0.2.12 in /home/janghl/anacon
da3/envs/DNN/lib/python3.9/site-packages (from ftfy->clip==1.0) (0.2.13)
Requirement already satisfied: typing-extensions in /home/janghl/anaconda3/e
nvs/DNN/lib/python3.9/site-packages (from torch->clip==1.0) (4.11.0)
Requirement already satisfied: numpy in /home/janghl/anaconda3/envs/DNN/lib/
python3.9/site-packages (from torchvision->clip==1.0) (1.26.4)
Requirement already satisfied: requests in /home/janghl/anaconda3/envs/DNN/l
ib/python3.9/site-packages (from torchvision->clip==1.0) (2.31.0)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /home/janghl/anacond
a3/envs/DNN/lib/python3.9/site-packages (from torchvision->clip==1.0) (10.3.
0)
Requirement already satisfied: charset-normalizer<4,>=2 in /home/janghl/anac
onda3/envs/DNN/lib/python3.9/site-packages (from requests->torchvision->clip
==1.0) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /home/janghl/anaconda3/envs/D
NN/lib/python3.9/site-packages (from requests->torchvision->clip==1.0) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /home/janghl/anaconda3/
envs/DNN/lib/python3.9/site-packages (from requests->torchvision->clip==1.0)
(2.2.1)
Requirement already satisfied: certifi>=2017.4.17 in /home/janghl/anaconda3/
envs/DNN/lib/python3.9/site-packages (from requests->torchvision->clip==1.0)
(2024.2.2)
```

In [ ]:

```
import clip
```

In [ ]:

```
# Sets device to "cuda" if a GPU is available
device = "cuda" if torch.cuda.is_available() else 'cpu'
print(device)
# If this takes a really long time, stop and then restart the download
clip_model, clip_preprocess = clip.load("ViT-B/32", device=device)
```

```
cuda
```

3.3 CLIP zero-shot prediction

In [ ]:
```
"""The following is an example of using CLIP pre-trained model for zero-shot
# Prepare the inputs
n = 500  # image index to use
image, class_id = flower_train[n]
image_input = clip_preprocess(image).unsqueeze(0).to(device) # extract image
text_inputs = torch.cat([clip.tokenize(f"a photo of a {c}, a type of flower.

# Calculate features
with torch.no_grad():
    image_features = clip_model.encode_image(image_input) # compute image fe
    text_features = clip_model.encode_text(text_inputs) # compute text featu
image_features /= image_features.norm(dim=-1, keepdim=True) # unit-normalize
text_features /= text_features.norm(dim=-1, keepdim=True) # unit-normalize t

# Pick the top 5 most similar labels for the image
similarity = (100.0 * image_features @ text_features.T) # score is cosine si
p_class_given_image= similarity.softmax(dim=-1)  # P(y|x) is score through s
values, indices = p_class_given_image[0].topk(5) # gets the top 5 labels

# Print the probability of the top five labels
print("Ground truth:", flower_classes[class_id])
print("\nTop predictions:\n")
for value, index in zip(values, indices):
    print(f"{flower_classes[index]:>16s}: {100 * value.item():.2f}%")
image
```

```
Ground truth: wild pansy

Top predictions:

 mexican petunia: 28.22%
         petunia: 27.34%
        geranium: 9.75%
  balloon flower: 6.81%
          mallow: 5.39%
```

Out[ ]:

3.4 YOUR TASK: Test CLIP zero-shot performance on Flowers 102

Use pre-trained text and image representations to classify images. For zero-shot recognition, text features are computed from the CLIP model for phrases such as “An image of [flower_name], a type of flower” for varying [flower_name] inserts. Then, image features are computed using the CLIP model for an image, and the cosine similarity between each text and image is computed. The label corresponding to the most similar text is assigned to the image. You'll get that working using a data loader, which enables faster batch processing; then, compute the accuracy over the test set. You should see top-1 accuracy in the 60-70% range.

For zero-shot, you do not use the training set at all. You should only have to compute the text vectors once and re-use them for all test images.

Basic steps:

1. Create the normalized CLIP text vectors for each class label.

2. For each batch:
    - Create normalized CLIP image vectors
    - Compute similarity between text and image vectors
    - Get index of most likely class label and check whether it matches the ground truth
    - Keep a count of number correct and number total
3. Return accuracy = # correct / # total

In [ ]:
```
from tqdm import tqdm
from torch.utils.data import DataLoader
```

In [ ]:
```
# Load flowers dataset again. This time, with clip_preprocess as transform (
flower_train_trans, flower_test_trans, flower_classes = load_flower_data(img
```

In [ ]:
```
def clip_zero_shot(data_set, classes):
    # data_loader = DataLoader(data_set, batch_size=32, shuffle=False)  # da
    # TO DO: Needs code here
    correct = 0
    total = 0
    for batch in tqdm(data_set):
        image_input = batch[0].unsqueeze(0).to(device)
        text_inputs = torch.cat([clip.tokenize(f"a photo of a {c}, a type of
        with torch.no_grad():
            image_features = clip_model.encode_image(image_input)
            text_features = clip_model.encode_text(text_inputs)
        image_features /= image_features.norm(dim=-1, keepdim=True)
        text_features /= text_features.norm(dim=-1, keepdim=True)
        similarity = (100.0 * image_features @ text_features.T).softmax(dim=
        values, indices = similarity[0].topk(5)
        # print(f"{indices[0]}, {batch[1]}")
        if batch[1] == indices[0]:
            correct += 1
        total += 1


    accuracy = correct / total

    return accuracy
```

In [ ]:
```
accuracy = clip_zero_shot(data_set=flower_test_trans, classes=flower_classes
print(f"\nAccuracy = {100*accuracy:.3f}")
```

```
100%|██████████| 1020/1020 [01:50<00:00,  9.24it/s]
Accuracy = 0.686
```

3.5 YOUR TASK: Test CLIP linear probe performance on Flowers

102

We do not use text features for the linear probe method. Train on the train set, and evaluate on the test set and report your performance. You can get top-1 accuracy in the 90-95% range. If you're getting in the 80's, try both normalizing and not normalizing the features.

In [ ]:

```python
from sklearn.linear_model import LogisticRegression
```

In [ ]:

```python
"""
Returns image features and labels in numpy format.
The labels should just be integers representing class index, not text vector
"""
def get_features(data_set):
    # TO DO: Needs code here to extract features and labels
    all_features = []
    all_labels = []

    with torch.no_grad():
        for images, labels in tqdm(DataLoader(data_set, batch_size=100)):
            features = clip_model.encode_image(images.to(device))

            all_features.append(features)
            all_labels.append(labels)

    return torch.cat(all_features).cpu().numpy(), torch.cat(all_labels).cpu(
```

In [ ]:

```python
# Calculate the image features
train_features, train_labels = get_features(flower_train_trans)
test_features, test_labels = get_features(flower_test_trans)

# TO DO: Needs code here
# Train logistic regression model with train_features, train_labels
classifier = LogisticRegression(random_state=0, C=0.316, max_iter=1000, verb
classifier.fit(train_features, train_labels)

# Evaluate accuracy on test_features, test_labels
predictions = classifier.predict(test_features)
accuracy = np.mean((test_labels == predictions).astype(float)) * 100.
print(f"Accuracy = {accuracy:.3f}")
```

```
100%|██████████| 11/11 [00:05<00:00,  1.90it/s]
100%|██████████| 11/11 [00:05<00:00,  1.96it/s]
 This problem is unconstrained.
```

```
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =        52326     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f=  4.62497D+00    |proj g|=  1.33024D-02

At iterate   50    f=  9.06565D-01    |proj g|=  9.95379D-04

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
52326     84     90      1     0     0   7.270D-05   9.039D-01
  F =  0.90393596410274291

CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL
Accuracy = 93.431
```

3.6 YOUR TASK: Evaluate a nearest-neighbor classifier on CLIP features

Extract features based on the pre-trained model (can be the same features as 3.5) and apply a nearest neighbor classifier. You can use your own implementation of nearest neighbor or a library like sklearn or FAISS for this. Try K={1, 3, 5, 7, 11, 21}. If using sklearn, you can also experiment with 'uniform' and 'distance' weighting. Report performance for best K on the test set. You can also experiment with using unnormalized or normalized features. You should see top-1 accuracy in the 80-90% range.

In [ ]:

```
# TO DO: code for KNN prediction and evaluation (may use sklearn.neighbors.N
from sklearn.neighbors import KNeighborsClassifier

for k in [1, 3, 5, 7, 11, 21]:
    classifier = KNeighborsClassifier(n_neighbors=k)
```

```
    classifier.fit(train_features, train_labels)
    predictions = classifier.predict(test_features)
    accuracy = np.mean((test_labels == predictions).astype(float)) * 100.
    print(f"k={k}\nAccuracy = {accuracy:.3f}%")
```

```
k=1
Accuracy = 85.000%
k=3
Accuracy = 84.216%
k=5
Accuracy = 84.706%
k=7
Accuracy = 85.196%
k=11
Accuracy = 83.627%
k=21
Accuracy = 78.725%
```

Part 4: Stretch Goals

Include any new code needed for Part 4 here.

4.a Compare word tokenizers

Train at least two 8K token word tokenizers (e.g. BPE, WordPiece, SentencePiece) on the WikiText-2, and compare their encodings. You can use existing libraries, such as those linked below to train and encode/decode. Report the encodings for "I am learning about word tokenizers. They are not very complicated, and they are a good way to convert natural text into tokens." E.g. "I am the fastest planet" may end up being tokenized as [I, _am, _the, _fast, est, _plan, et]. Also, report the tokenizations of an additional sentence of your choice that results in different encodings by the two models.

https://github.com/huggingface/tokenizers

In [ ]:
```
# Choose your model between Byte-Pair Encoding, WordPiece or Unigram and ins

from tokenizers import Tokenizer
from tokenizers.models import BPE
from datasets import load_dataset

dataset = load_dataset(path="wikitext", name="wikitext-2-raw-v1")
tokenizer = Tokenizer(BPE())
# You can customize how pre-tokenization (e.g., splitting into words) is dor

from tokenizers.pre_tokenizers import Whitespace
```

```
tokenizer.pre_tokenizer = Whitespace()
# Then training your tokenizer on a set of files just takes two lines of coc

from tokenizers.trainers import BpeTrainer

trainer = BpeTrainer(special_tokens=["[UNK]", "[CLS]", "[SEP]", "[PAD]", "[M
tokenizer.train(files=[], trainer)
# Once your tokenizer is trained, encode any text with just one line:

output = tokenizer.encode("I am learning about word tokenizers. They are not
print(output.tokens)
# ["Hello", ",", "y", "'", "all", "!", "How", "are", "you", "[UNK]","?"]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[2], line 17
     14 from tokenizers.trainers import BpeTrainer
     16 trainer = BpeTrainer(special_tokens=["[UNK]", "[CLS]", "[SEP]", "[PA
D]", "[MASK]"])
---> 17 tokenizer.train(dataset, trainer)
     18 # Once your tokenizer is trained, encode any text with just one lin
e:
     20 output = tokenizer.encode("I am learning about word tokenizers. They
are not very complicated, and they are a good way to convert natural text in
to tokens.")

TypeError: argument 'files': 'DatasetDict' object cannot be converted to 'Se
quence'
```

4.b Implement your own network

For the Oxford Pets dataset, try to write the network by yourself. You can get ideas from existing works, but you cannot directly import them using packages, and the parameter number should be lower than 20M. Train your network from scratch. You would get points if your network can reach an accuracy of 35% (15 pts), and another 15 pts if it reaches 45%. You would want to pay more attention to data augmentation and other hyper-parameters during this part. Feel free to re-use any functions defined in Part 2.

In [ ]:

```
# example network definition that needs to be modified for custom network st
from torch import nn

class ResBlock(nn.Module):
    def __init__(self, in_channels, out_channels, downsample):
        super().__init__()
        if downsample:
            self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3,
```

```
            self.shortcut = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=2),
            nn.BatchNorm2d(out_channels)
            )
        else:
            self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3,
            self.shortcut = nn.Sequential()
            self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3
            self.bn1 = nn.BatchNorm2d(out_channels)
            self.bn2 = nn.BatchNorm2d(out_channels)

    def forward(self, input):
        shortcut = self.shortcut(input)
        input = nn.ReLU()(self.bn1(self.conv1(input)))
        input = nn.ReLU()(self.bn2(self.conv2(input)))
        input = input + shortcut
        return nn.ReLU()(input)

class Network(nn.Module):
    def __init__(self, num_classes=37):
        super().__init__()
        resblock = ResBlock
        self.layer0 = nn.Sequential(
        nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3),
        nn.MaxPool2d(kernel_size=3, stride=2, padding=1),
        nn.BatchNorm2d(64),
        nn.ReLU()
        )
        self.layer1 = nn.Sequential(
        resblock(64, 64, downsample=False),
        resblock(64, 64, downsample=False)
        )
        self.layer2 = nn.Sequential(
        resblock(64, 128, downsample=True),
        resblock(128, 128, downsample=False)
        )
        self.layer3 = nn.Sequential(
        resblock(128, 256, downsample=True),
        resblock(256, 256, downsample=False)
        )
        self.layer4 = nn.Sequential(
        resblock(256, 512, downsample=True),
        resblock(512, 512, downsample=False)
        )
        self.gap = torch.nn.AdaptiveAvgPool2d(1)
        self.fc = torch.nn.Linear(512, num_classes)

    def forward(self, input):
        input = self.layer0(input)
        input = self.layer1(input)
```

```
        input = self.layer2(input)
        input = self.layer3(input)
        input = self.layer4(input)
        input = self.gap(input)
        input = torch.flatten(input, 1)
        input = self.fc(input)
        return input
```

In [ ]:

```
model = Network()

# Feel free to change
train_set, test_set = load_pet_dataset(train_transform, test_transform)
train_loader = DataLoader(dataset=train_set,
                          batch_size=64,
                          shuffle=True,
                          num_workers=2)

test_loader = DataLoader(dataset=test_set,
                          batch_size=64,
                          shuffle=False,
                          num_workers=2)

# TO DO: replace the last layer (classification head) with a new linear laye
# in_features = model.fc.in_features
# model.fc = torch.nn.Linear(in_features, num_target_classes)
device = 'cuda'
model = model.to(device)
display_model(model) # displays the model structure and parameter count
```

```
================================================================================
====
- layer0                                    :              9600
- layer1                                    :            148224
- layer2                                    :            378112
- layer3                                    :           1509888
- layer4                                    :           6034432
- gap                                       :                 0
- fc                                        :             18981
================================================================================
====
--Total                                     :           8099237 params
--
Network(
  (layer0): Sequential(
    (0): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3))
    (1): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode
=False)
    (2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running
_stats=True)
    (3): ReLU()
```

```
    )
    (layer1): Sequential(
      (0): ResBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
        (shortcut): Sequential()
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      )
      (1): ResBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
        (shortcut): Sequential()
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      )
    )
    (layer2): Sequential(
      (0): ResBlock(
        (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(
1, 1))
        (shortcut): Sequential(
          (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2))
          (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
        )
      )
      (1): ResBlock(
        (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1))
        (shortcut): Sequential()
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1))
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      )
    )
    (layer3): Sequential(
      (0): ResBlock(
        (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(
```

```
1, 1))
      (shortcut): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2))
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      )
    )
    (1): ResBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1))
      (shortcut): Sequential()
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1))
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
    )
  )
  (layer4): Sequential(
    (0): ResBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(
1, 1))
      (shortcut): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2))
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      )
    )
    (1): ResBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1))
      (shortcut): Sequential()
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1))
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
    )
  )
  (gap): AdaptiveAvgPool2d(output_size=1)
  (fc): Linear(in_features=512, out_features=37, bias=True)
)
```

In [ ]:

```
# Training Setting. Feel free to change.
num_epochs = 10
test_interval = 1

# TO DO: set initial learning rate
```

```
learn_rate = 1e-3
optimizer = torch.optim.Adam(model.parameters(), lr=learn_rate)

# TO DO: define your learning rate scheduler, e.g. StepLR
# https://pytorch.org/docs/stable/optim.html#module-torch.optim.lr_scheduler
lr_scheduler =  torch.optim.lr_scheduler.StepLR(optimizer, step_size=1, gamm

criterion = torch.nn.CrossEntropyLoss()

train_losses = []
train_accuracy_list = []
test_losses = []
test_accuracy_list = []


# Iterate over the DataLoader for training data
for epoch in tqdm(range(num_epochs), total=num_epochs, desc="Training ...",
    # Train the network for one epoch
    train_loss = train(train_loader, model, criterion, optimizer)

    # TO DO: uncomment the line below. It should be called each epoch to app
    lr_scheduler.step()

    train_losses.append(train_loss)
    print(f'Loss for Training on epoch {str(epoch)} is {str(train_loss)} \n'

    # Get the train accuracy and test loss/accuracy
    if(epoch%test_interval==0 or epoch==1 or epoch==num_epochs-1):
        print('Evaluating Network')

        train_accuracy, _ = test(train_loader, model, criterion) # Get train
        train_accuracy_list.append(train_accuracy)

        print(f'Training accuracy on epoch {str(epoch)} is {str(train_accura

        test_accuracy, test_loss = test(test_loader, model, criterion) # Get
        test_losses.append(test_loss)
        test_accuracy_list.append(test_accuracy)
        print(f'Test (val) accuracy on epoch Test (val) accuracy on epoch {s

        # Checkpoints are used to save the model with best validation accura
        if test_accuracy >= max(test_accuracy_list):
          print("Saving Model")
          save_checkpoint(save_dir, model, save_name = 'own_best_model.pth')
```

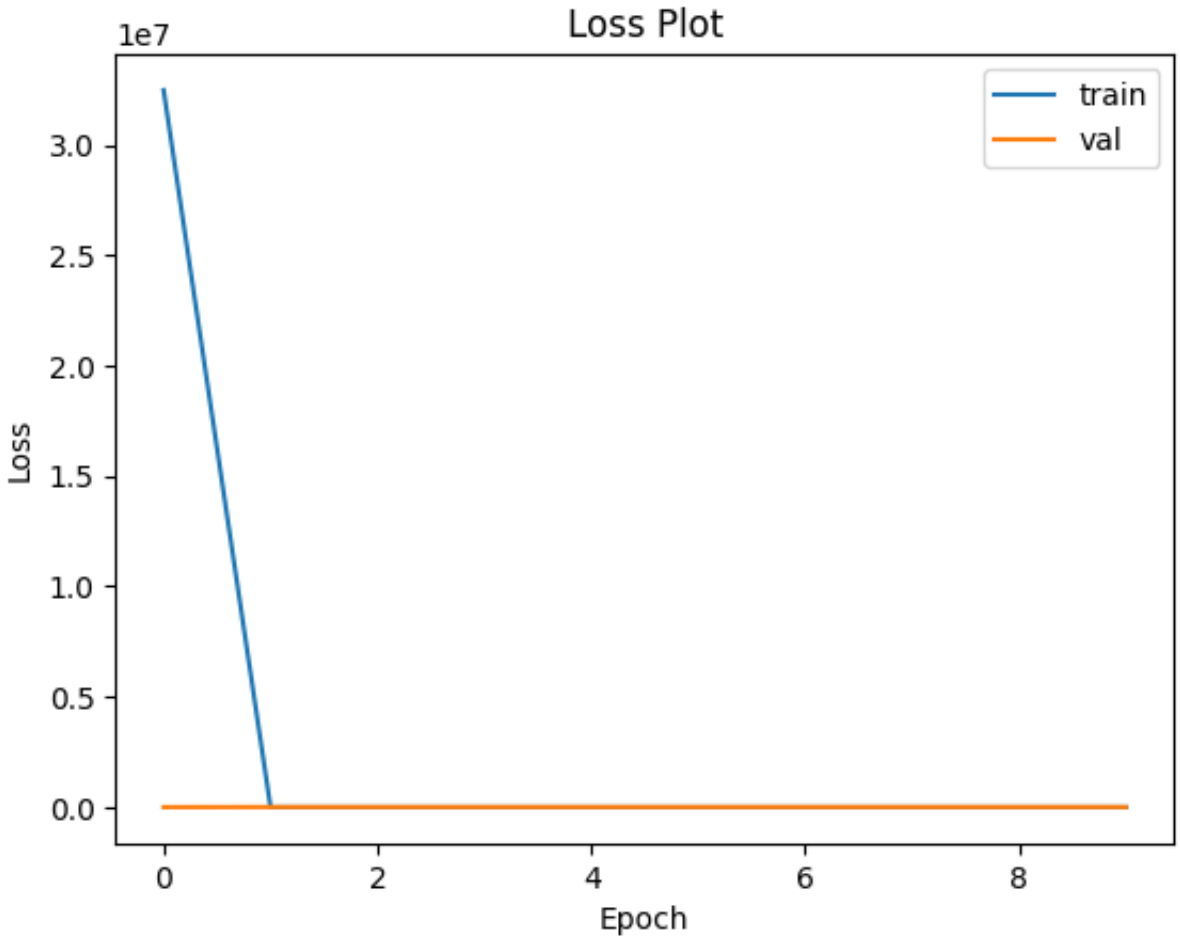
```
Training ...:   0%|          | 0/58 [00:00<?, ?it/s]
Training ...:   0%|          | 0/10 [00:01<?, ?it/s]

---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[17], line 24
```

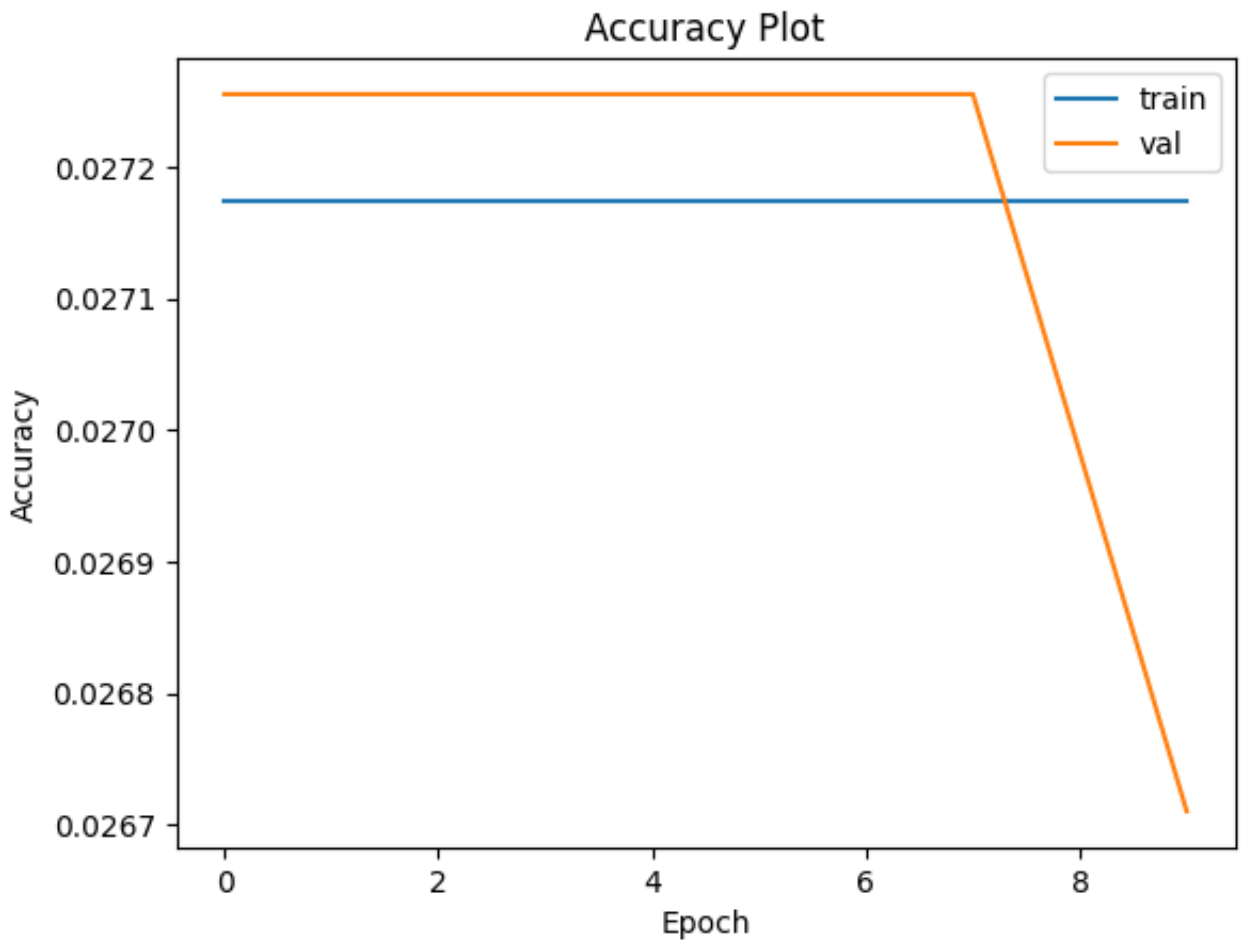
```
     21 # Iterate over the DataLoader for training data
     22 for epoch in tqdm(range(num_epochs), total=num_epochs, desc="Trainin
g ...", position=1):
     23     # Train the network for one epoch
---> 24     train_loss = train(train_loader, model, criterion, optimizer)
     26     # TO DO: uncomment the line below. It should be called each epoc
h to apply the lr_scheduler
     27     lr_scheduler.step()

Cell In[16], line 26, in train(train_loader, model, criterion, optimizer)
     23 optimizer.zero_grad()
     25 # predict labels
---> 26 prediction = model(images)
     28 # compute loss
     29 loss = criterion(prediction, labels)

File ~/anaconda3/envs/DNN/lib/python3.9/site-packages/torch/nn/modules/modul
e.py:1110, in Module._call_impl(self, *input, **kwargs)
   1106 # If we don't have any hooks, we want to skip the rest of the logic
in
   1107 # this function, and just call forward.
   1108 if not (self._backward_hooks or self._forward_hooks or self._forward
_pre_hooks or _global_backward_hooks
   1109         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1110     return forward_call(*input, **kwargs)
   1111 # Do not call functions when jit is used
   1112 full_backward_hooks, non_full_backward_hooks = [], []

Cell In[13], line 59, in Network.forward(self, input)
     57 input = self.layer0(input)
     58 input = self.layer1(input)
---> 59 input = self.layer2(input)
     60 input = self.layer3(input)
     61 input = self.layer4(input)

File ~/anaconda3/envs/DNN/lib/python3.9/site-packages/torch/nn/modules/modul
e.py:1110, in Module._call_impl(self, *input, **kwargs)
   1106 # If we don't have any hooks, we want to skip the rest of the logic
in
   1107 # this function, and just call forward.
   1108 if not (self._backward_hooks or self._forward_hooks or self._forward
_pre_hooks or _global_backward_hooks
   1109         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1110     return forward_call(*input, **kwargs)
   1111 # Do not call functions when jit is used
   1112 full_backward_hooks, non_full_backward_hooks = [], []

File ~/anaconda3/envs/DNN/lib/python3.9/site-packages/torch/nn/modules/conta
iner.py:141, in Sequential.forward(self, input)
    139 def forward(self, input):
```

```
    140     for module in self:
--> 141         input = module(input)
    142     return input

File ~/anaconda3/envs/DNN/lib/python3.9/site-packages/torch/nn/modules/modul
e.py:1110, in Module._call_impl(self, *input, **kwargs)
   1106 # If we don't have any hooks, we want to skip the rest of the logic
in
   1107 # this function, and just call forward.
   1108 if not (self._backward_hooks or self._forward_hooks or self._forward
_pre_hooks or _global_backward_hooks
   1109         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1110     return forward_call(*input, **kwargs)
   1111 # Do not call functions when jit is used
   1112 full_backward_hooks, non_full_backward_hooks = [], []

Cell In[13], line 22, in ResBlock.forward(self, input)
     20 def forward(self, input):
     21     shortcut = self.shortcut(input)
---> 22     input = nn.ReLU()(self.bn1(self.conv1(input)))
     23     input = nn.ReLU()(self.bn2(self.conv2(input)))
     24     input = input + shortcut

File ~/anaconda3/envs/DNN/lib/python3.9/site-packages/torch/nn/modules/modul
e.py:1185, in Module.__getattr__(self, name)
   1183     if name in modules:
   1184         return modules[name]
-> 1185 raise AttributeError("'{}' object has no attribute '{}'".format(
   1186     type(self).__name__, name))

AttributeError: 'ResBlock' object has no attribute 'bn1'
```

In [ ]:

```
plot_losses(train_losses, test_losses, test_interval, num_epochs)
plot_accuracy(train_accuracy_list, test_accuracy_list, test_interval, num_ep
```


Loss Plot
1e7
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Loss
0
2
4
6
8
Epoch
train
val


Accuracy Plot
train
val
Accuracy
0.0272
0.0271
0.0270
0.0269
0.0268
0.0267
Epoch
0
2
4
6
8