CS441: Applied ML - HW 3

Part 1: CLIP: Contrastive Language-Image Pretraining

Include all the code for Part 1 in this section

1.1 Prepare data

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

```
import json
import os
import os.path as osp
import numpy as np
from google.colab import drive
import torch
from torchvision.datasets import Flowers102
%matplotlib inline
from matplotlib import pyplot as plt
```

```
drive.mount('/content/drive')
datadir = "/content/drive/My Drive/CS441/hw3/"
```

```
Mounted at /content/drive
```

```
def load_flower_data(img_transform=None):
    drive.mount('/content/drive')
    datadir = "/content/drive/My Drive/CS441/hw3/" #/content/drive/My Drive/CS441/hw3/"
    if os.path.isdir(datadir+ "flowers-102"):
        do_download = False
    else:
        do_download = True
    train_set = Flowers102(root=datadir, split='train', transform=img_transform, download=do_download)
    test_set = Flowers102(root=datadir, split='val', transform=img_transform, download=do_download)
    classes = json.load(open(osp.join("/content/drive/My Drive/CS441/hw3/"", "flowers102_classes.json")))

    return train_set, test_set, classes
```

```
# READ ME!  This takes some time (a few minutes), so if you are using Colabs,
#           first set to use GPU: Edit->Notebook Settings->Hardware Accelerator=GPU, and restart instance

# 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()
```

```
len(flower_train), len(flower_test)
```

```
(1020, 1020)
```

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

Label: pink primrose

1.2 Prepare CLIP model

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

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting git+https://github.com/openai/CLIP.git
  Cloning https://github.com/openai/CLIP.git to /tmp/pip-req-build-qsft5q7b
  Running command git clone --filter=blob:none --quiet https://github.com/openai/CLIP.git /tmp/pip-req-build-qsft5q7b
  Resolved https://github.com/openai/CLIP.git to commit a9b1bf5920416aaeaec965c25dd9e8f98c864f16
  Preparing metadata (setup.py) ... done
Collecting ftfy
  Downloading ftfy-6.1.1-py3-none-any.whl (53 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 53.1/53.1 KB 7.1 MB/s eta 0:00:00
Requirement already satisfied: regex in /usr/local/lib/python3.9/dist-packages (from clip==1.0) (2022.10.31)
Requirement already satisfied: tqdm in /usr/local/lib/python3.9/dist-packages (from clip==1.0) (4.65.0)
Requirement already satisfied: torch in /usr/local/lib/python3.9/dist-packages (from clip==1.0) (1.13.1+cu116)
Requirement already satisfied: torchvision in /usr/local/lib/python3.9/dist-packages (from clip==1.0) (0.14.1+cu116)
Requirement already satisfied: wcwidth>=0.2.5 in /usr/local/lib/python3.9/dist-packages (from ftfy->clip==1.0) (0.2.6)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.9/dist-packages (from torch->clip==1.0) (4.5.0)
Requirement already satisfied: requests in /usr/local/lib/python3.9/dist-packages (from torchvision->clip==1.0) (2.27.1)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.9/dist-packages (from torchvision->clip==1.0) (8.4.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.9/dist-packages (from torchvision->clip==1.0) (1.22.4)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.9/dist-packages (from requests->torchvision->clip==1.0) (1.26.15)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.9/dist-packages (from requests->torchvision->clip==1.0) (3.4)
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.9/dist-packages (from requests->torchvision->clip==1.0) (2.0.12)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.9/dist-packages (from requests->torchvision->clip==1.0) (2022.12.7)
Building wheels for collected packages: clip
  Building wheel for clip (setup.py) ... done
  Created wheel for clip: filename=clip-1.0-py3-none-any.whl size=1369398 sha256=b91c43c2ffa7af7e98485d2396c2b6bcfd85bf87765f2462283d12b1b7bc0cce
  Stored in directory: /tmp/pip-ephem-wheel-cache-b8_j6m15/wheels/c8/e4/e1/11374c111387672fc2068dfbe0d4b424cb9cdd1b2e184a71b5
Successfully built clip
Installing collected packages: ftfy, clip
Successfully installed clip-1.0 ftfy-6.1.1
```

```
import clip
```

```
# 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
100%|████████████████████████████████████████| 338M/338M [00:05<00:00, 69.6MiB/s]
```

1.3 CLIP zero-shot prediction

```
"""The following is an example of using CLIP pre-trained model for zero-shot prediction task"""
# Prepare the inputs
n = 200
image, class_id = flower_train[n]
image_input = clip_preprocess(image).unsqueeze(0).to(device) # extract image and put in device memory
text_inputs = torch.cat([clip.tokenize(f"a photo of a {c}, a type of flower.") for c in flower_classes]).to(device) # put text to match to image

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

# Pick the top 5 most similar labels for the image
similarity = (100.0 * image_features @ text_features.T) # score is cosine similarity times 100
p_class_given_image= similarity.softmax(dim=-1)  # P(y|x) is score through softmax
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: giant white arum lily

Top predictions:

giant white arum lily: 60.01%
            lotus: 12.98%
       siam tulip: 8.25%
        anthurium: 3.55%
   morning glory: 1.87%
```

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

```
from  tqdm  import  tqdm
from  torch.utils.data  import  DataLoader


#  Load  flowers  dataset  again.  This  time,  with  clip_preprocess  as  transform
flower_train_trans,  flower_test_trans,  flower_classes  =  load_flower_data(img_transform=clip_preprocess)


def  clip_zero_shot(data_set,  classes):
        data_loader  =  DataLoader(data_set,  batch_size=64,  shuffle=False)    #  dataloader  lets  you  process  in  batch  which  is  way  faster
        num_correct  =  0
        num_iteration  =  len(data_set)
        for  n  in  range(num_iteration):
            image,  class_id  =  data_set[n]
            image_input  =  clip_preprocess(image).unsqueeze(0).to(device)
            text_inputs  =  torch.cat([clip.tokenize(f"a  photo  of  a  {c},  a  type  of  flower.")  for  c  in  flower_classes]).to(device)
            with  torch.no_grad():
                image_features  =  clip_model.encode_image(image_input)  #  compute  image  features  with  CLIP  model
                text_features  =  clip_model.encode_text(text_inputs)  #  compute  text  features  with  CLIP  model
            image_features  /=  image_features.norm(dim=-1,  keepdim=True)  #  unit-normalize  image  features
            text_features  /=  text_features.norm(dim=-1,  keepdim=True)
            similarity  =  (100.0  *  image_features  @  text_features.T)  #  score  is  cosine  similarity  times  100
            p_class_given_image=  similarity.softmax(dim=-1)    #  P(y|x)  is  score  through  softmax
            values,  indices  =  p_class_given_image[0].topk(5)  #  gets  the  top  5  labels
            if  class_id  ==  indices[0]:
                num_correct  +=  1
        accuracy  =  num_correct  /  num_iteration
        return  accuracy




accuracy  =  clip_zero_shot(data_set=flower_test,  classes=flower_classes)
print(f"\nAccuracy  =  {100*accuracy:.3f}%")
```

```
Accuracy = 67.843%
```

1.5 YOUR TASK: Test CLIP linear probe performance on Flowers 102

```
from sklearn.linear_model import LogisticRegression
from tqdm import tqdm
from torch.utils.data import DataLoader
import numpy as np


"""
In this part, train a linear classifier on CLIP features
return: image features, labels in numpy format.
"""
def get_features(data_set):
        # Needs code here
        all_feature = []
        all_label = []
        with torch.no_grad():
            for images, labels in tqdm(DataLoader(data_set, batch_size = 100)):
                features = model.encode_image(images.to(device))
                all_feature.append(features)
                all_label.append(labels)
        return torch.cat(all_feature).cpu().numpy(), torch.cat(all_label).cpu().numpy()


# Calculate the image features
device = "cuda" if torch.cuda.is_available() else "cpu"
model, preprocess = clip.load('ViT-B/32', device)
train_features, train_labels = get_features(flower_train_trans)
test_features, test_labels = get_features(flower_test_trans)
classifier = LogisticRegression(random_state = 0, C=0.316, max_iter = 1000, verbose = 1)
classifier.fit(train_features, train_labels)
# Perform logistic regression
# Needs code here

# Evaluate using the logistic regression classifier
# Needs code here
predictions = classifier.predict(test_features)
accuracy = np.mean((test_labels == predictions).astype(float))
print(f"\nAccuracy = {100*accuracy:.3f}%")
```

```
100%|████████████| 11/11 [00:13<00:00, 1.25s/it]
100%|████████████| 11/11 [00:13<00:00, 1.24s/it]
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

Accuracy = 93.627%
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    3.8s finished
```

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

```
from scipy import stats
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score


def knn(x_train, y_train, x_test, y_test, K=1):
        # Needs code here
        model = KNeighborsClassifier(n_neighbors = K)
        model.fit(x_train, y_train)
        y_pred = model.predict(x_test)
        accuracy = accuracy_score(y_test, y_pred)
        return accuracy


K = [1, 3, 5, 11, 21]
for n in K:
    accuracy = knn(train_features, train_labels, test_features, test_labels, K=n)
    print(f"\nAccuracy = {100*accuracy:.3f}%")
```

```
Accuracy = 84.118%

Accuracy = 83.137%

Accuracy = 84.608%

Accuracy = 85.000%

Accuracy = 79.804%
```

Part 2: Fine-Tune for Pets Image Classification

Include all the code for Part 2 in this section

2.1 Prepare Data

```
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


# Mount and define data dir
from google.colab import drive
drive.mount('/content/drive')
datadir = "/content/"
save_dir = "/content/drive/My Drive/CS441/hw3"


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



train_set, test_set = load_pet_dataset()

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

2.2 Data Preprocess

```
from torchvision import transforms
from torch.utils.data import DataLoader


# 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 size of ImageNet images
                        transforms.CenterCrop(224),
                        transforms.ToTensor(),
                        transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                                                std= [0.229, 0.224, 0.225]),
                ])


# 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

```
# 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":^34s}' +'--']
    print('\n'.join(summary_str))

    # print model structure
    print(model)


# 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 i ==0 or i == 1)]
        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 i ==0 or i == 1)]
        plt.plot(indices, train, label="train")
        plt.plot(indices, val, label="val")
        plt.title("Training 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.

```
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 CrossEntropyLoss from torch.nn)
        :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; https://pypi.org/project/tqdm/
        # it_train = tqdm(enumerate(train_loader), total=len(train_loader), desc="Training ...", position = 0) # progress bar
        for i, (images, labels) in it_train:

                # TO DO: read/understand and then uncomment these lines
                #images, labels = images.to(device), labels.to(device)
                #optimizer.zero_grad()
                #prediction = model(images)
                #loss = criterion(prediction, labels)
                #it_train.set_description(f'loss: {loss:.3f}')
                #loss.backward()
                #optimizer.step()
                #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 CrossEntropyLoss from torch.nn)
        :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; https://pypi.org/project/tqdm/
        #it_test = tqdm(enumerate(test_loader), total=len(test_loader), desc="Validating ...", position = 0)
        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/torch.no_grad.html
            #   output = model(images)
            #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


device = 'cuda'
# loads a pre-trained ResNet-34 model
model = torch.hub.load('pytorch/vision:v0.10.0', 'resnet34', pretrained=True)
target_class = 37
# TO DO: replace the last layer with a new linear layer for Pets classification

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



# Training Setting. Feel free to change.
num_epochs = 20
test_interval = 5

# TO DO: set initial learning rate
learn_rate = []
optimizer = torch.optim.Adam(model.parameters(), lr=learn_rate)

# TO DO: define your learning rate scheduler, e.g. StepLR
# https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.StepLR.html#torch.optim.lr_scheduler.StepLR
lr_scheduler = []

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 ...", position=1):
    train_loss = train(train_loader, model, criterion, optimizer) # Train the Network for one epoch
    # TO DO: uncomment the line below. It should be called each epoch to apply the lr_scheduler
    # lr_scheduler.step()
    train_losses.append(train_loss)
    print(f'Loss for Training on epoch {str(epoch)} is {str(train_loss)} \n')

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

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

        print(f'Training accuracy on epoch {str(epoch)} is {str(train_accuracy)} \n')

        test_accuracy, test_loss = test(test_loader, model, criterion) # Get testing accuracy and error
        test_losses.append(test_loss)
        test_accuracy_list.append(test_accuracy)

        print(f'Testing accuracy on epoch {str(epoch)} is {str(test_accuracy)} \n')

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

2.5 Plotting of losses and accuracy

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

2.6 Evaluating trained model

```
# TO DO: initialize your trained model as you did before so that you can load the parameters into it
model = torch.hub.load('pytorch/vision:v0.10.0', 'resnet34', pretrained=True).to(device)
# replace last layer

load_model(model, save_dir) # Load the trained weight

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

Part 3: No coding for this part

Part 4: Stretch Goals

Include any new code needed for Part 3 here

```
# example network definition that needs to be modified for custom network stretch goal

class Network(nn.Module):
    def __init__(self, num_classes=10, dropout = 0.5):
        super(Network, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=11, stride=4, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(64, 256, kernel_size=5, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
        )

        self.avgpool = nn.AdaptiveAvgPool2d((6, 6))
        self.classifier = nn.Sequential(
            nn.Dropout(p=dropout),
            nn.Linear(256 * 6 * 6, 512),
            nn.ReLU(inplace=True),
            nn.Dropout(p=dropout),
            nn.Linear(512, 512),
            nn.ReLU(inplace=True),
            nn.Linear(512, num_classes),
```

```
        )


    def forward(self, x):
        N, c, H, W = x.shape
        features = self.features(x)
        pooled_features = self.avgpool(features)
        output = self.classifier(torch.flatten(pooled_features, 1))
        return output
```

✓ 0s completed at 7:04 PM