Name:

Darian Irani

Netid:

irani2

CS 441 - HW 5: Deep Learning and Applications

Complete the sections below. You do not need to fill out the checklist. **Be sure to select all relevant pages in Gradescope.**

Total Points Available **[ ] / 150**

1. Applications of AI
    1. Describe the applications [ ] / 15
    2. Positive impact [ ] / 7
    3. Negative impact [ ] / 8
2. Fine-tune Model for Pets Classification
    1. Qs about ResNet-34 structure [ ] / 10
    2. Epochs vs Loss Plots [ ] / 10
    3. Best Performance / Question [ ] / 10
3. CLIP: Contrastive Language-Image Pretraining
    1. Test CLIP zero-shot performance [ ] / 20
    2. Test CLIP linear probe performance [ ] / 10
    3. KNN on CLIP features [ ] / 10
4. Stretch Goals
    1. Compare word tokenizers [ ] / 20
    2. Implement/train custom network [ ] / 30

1. Answer “Applications of AI” Questions

1. **How is AI used in that application area?** What is the problem that AI is trying to solve, and what are the key AI/ML technologies involved? What are the technical challenges? (100+ words)

The integration of AI into home monitoring systems offers a blend of enhanced security, energy optimization, and improved elderly care. AI's ability to analyze real-time video feeds in security systems revolutionizes the way we approach home safety. It identifies potential threats with precision, minimizing the reliance on constant human monitoring and providing a proactive approach to security.

In the realm of elderly care, AI's role is equally transformative. By monitoring health indicators and daily activities, AI systems can swiftly notify caregivers in the event of an emergency, ensuring timely intervention and support. This technological advancement is a beacon of hope for enhancing the quality of life for the elderly, providing them with a sense of independence while ensuring their safety.

Energy management is another area where AI dominates, demonstrating its ability to learn and adapt to household patterns. It intelligently adjusts heating, lighting, and other systems, leading to significant energy savings and contributing to a more sustainable future.

Despite these benefits, integrating AI into home monitoring is not without its challenges. Ensuring the privacy and security of data, achieving compatibility among various smart devices, maintaining the accuracy and reliability of AI decision-making, and managing the extensive data generated are critical issues that need addressing. These challenges underscore the importance of establishing stringent security protocols, ensuring seamless device integration, and implementing precise and dependable AI functionalities.

2. **What is the actual or potential positive impact?** Who is impacted? (50+ words)

Firstly, it enhances security by intelligently analyzing behavior and detecting anomalies, reducing the risk of intrusions and enabling quick responses to threats. Secondly, AI-driven systems optimize energy consumption, leading to significant savings and environmental benefits through smart management of resources like heating, cooling, and lighting. Thirdly, in the realm of healthcare, particularly for elderly care, AI enables continuous monitoring, facilitating immediate medical assistance when needed, thereby ensuring safety and promoting independence. Overall, AI in home monitoring promises a future where homes are not only safer and more efficient but also adapt to the individual needs and well-being of their inhabitants, reflecting a profound positive impact on daily living and resource management.

3. **What is the actual or potential negative impact?** Who is impacted? (50+ words)

The actual or potential negative impact of AI in home monitoring includes privacy concerns, as constant surveillance and data collection might lead to unauthorized data access and misuse. Individuals could feel their personal space is invaded, undermining trust in technology. Additionally, over-reliance on AI may result in complacency, where residents ignore manual checks and maintenance, potentially leading to system failures. Job displacement is another concern, as AI could replace human roles in security and monitoring. These impacts primarily affect homeowners, tenants, and workers in the security and property management sectors, highlighting the need for ethical and secure AI implementation.

4. **What are your sources?** (include full citations and links if available) [required; -15 pts if not provided] Format is not as important as being clear about what source is used.

Ferguson, Ann. “How Ai-Powered Smart Home Security Revolutionizes Home Safety.” *DAVE*, 8 Sept. 2023, www.comeseedave.com/blog/ai-smart-home-security-revolution#

ThriveAdmin. “How AI Is Changing Home Security.” *Aeon Systems*, 4 May 2023, www.aeonsystems.net/how-ai-is-changing-home-security/

“Exploring the Future Impact of AI in Home Security.” *Vector Security*, www.vectorsecurity.com/exploring-the-future-impact-of-ai-in-home-security. Accessed 8 Apr. 2024.

2. CNN: Image Classification

1. **Answer these questions about the network structure of your model based on ResNet-34**

   1.1. How many parameters are there in total? (xx.x million)

21.3 million

   1.2. Which of these layers **do not** have trainable parameters? (choose more than 1)

   a) Convolutional
   b) BatchNorm
   c) ReLU
   d) Max pooling
   e) Fully connected

ReLU, Max pooling

1.3. True or false: In layers 1-4, whenever the feature map is downsampled by a factor of 2, the number of features is doubled.

True.
When transitioning from layer 1 to 2, the spatial dimension (stride) is downsampled by a factor of 2. The number of features is doubled (from 64 to 128). The same pattern is followed from layer 3 to 4.

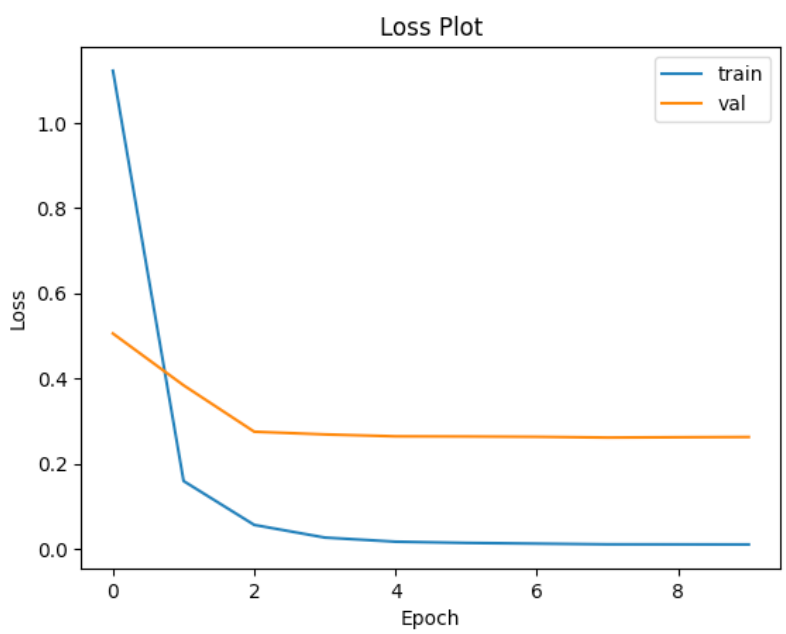
1.4. Which of these are applied immediately before the final fully connected layer? (choose one)

a) Convolutional
b) BatchNorm
c) ReLU
d) Max pooling
e) Average pooling
f) Fully connected

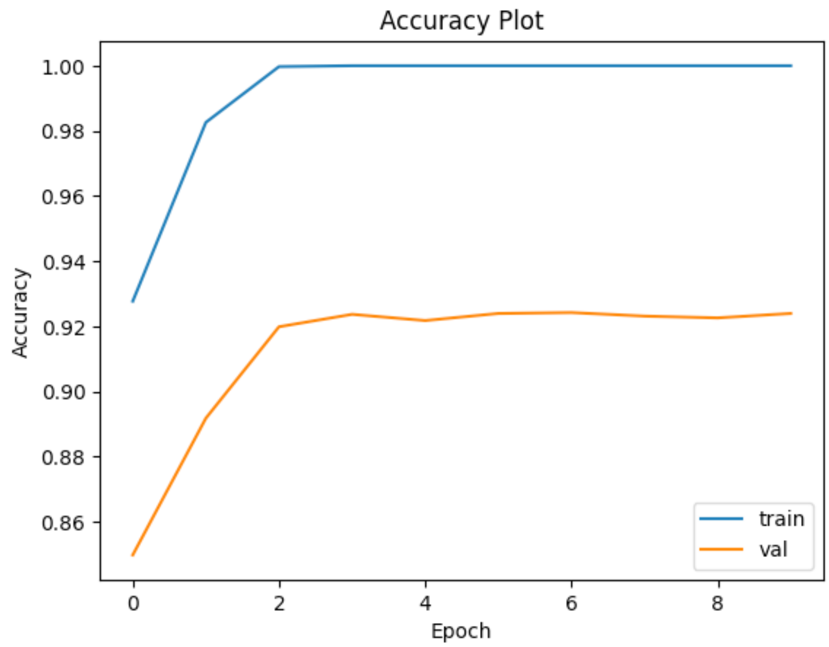
Average pooling

2. Plot accuracy and loss for at least 10 epochs

Loss



Accuracy



3. Best accuracy / question

Your best val (test_set) accuracy:

92.4%

True or False: Once the training accuracy reaches 100%, it's not possible to improve the model with further training.

False, having a training accuracy of 100% does not necessarily mean that no further improvements can be made. The model could be overfitting the data and including noise or outliers. Just because the training accuracy is 100% it may not perform well on the validation or test data and generalization may need to be improved through methods like hyperparameter adjusting or regularization.

3. CLIP: Contrastive Language-Image Pretraining

1. CLIP zero-shot performance

Your test accuracy (xx.x%)

67.6%

What is the key idea that provides zero-shot ability to CLIP? (choose one)

a. The visual model is trained to predict the most likely word based on a large dataset of labeled images.
b. A text model is trained to map words into a vector and a visual model to map patches into an equal sized vector, such that the vectors of image and its textual description are much more similar than those of non-corresponding images and descriptions.
c. CLIP learns to generate the most likely textual description, given the image.

b

2. **CLIP linear probe performance**

Your test accuracy (xx.x%)

93.4%

3. **KNN on CLIP feature**

Your test accuracy: (xx.x%)

85.8%

Best K:

9

4. Stretch Goals

a. Compare word tokenizers

Report 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." and one additional sentence of your choice. 20 points for reporting trained encodings of at least two models. 10 points for one model. You must train the models on WikiText-2 (should be included in notebook code).

BPE Encoding: ['I ', 'am ', 'lear', 'ning ', 'about ', 'word ', 'to', 'k', 'en', 'iz', 'er', 's', '. They ', 'are not ', 'very ', 'compl', 'ic', 'at', 'ed', ', and ', 'they are ', 'a ', 'good ', 'way to ', 'conver', 't ', 'natural ', 'text ', 'into ', 'to', 'k', 'ens', '.']

WordPiece Encoding: ['[UNK]']

Additional BPE Encoding: ['I ', 'have ', 'lear', 'n', 't ', 'a ', 'l', 'ot ', 'from ', 'Ap', 'pl', 'ied ', 'M', 'ach', 'ine ', 'L', 'ear', 'ning ', 'at ', 'U', 'I', 'U', 'C']

Additional WordPiece Encoding: ['I', '## ', '##have ', '##lear', '##n', '##t ', '##a ', '##lo', '##t ', '##from ', '##Ap', '##pl', '##ied ', '##Mac', '##hi', '##ne ', '##Le', '##ar', '##ning ', '##at ', '##U', '##I', '##U', '##C']

b. Custom network implementation and evaluation

i. Display the structure of the network you implemented

ii. Plot accuracy and loss

iii. Best test accuracy:

Acknowledgments / Attribution

List any outside sources for code or improvement ideas or “None”.

https://www.techtarget.com/searchcustomerexperience/definition/virtual-assistant-AI-assistant

https://mobidev.biz/blog/ai-virtual-assistant-technology-guide

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 [1]:
```
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 [2]:
```
# Mount and define data dir

datadir = "/Users/darian/Desktop/UIUC/Applied ML/HW5/Code/"
save_dir = "/Users/darian/Desktop/UIUC/Applied ML/HW5/Code/"
```

In [3]:
```
def load_pet_dataset(train_transform = None, test_transform = Non
e):
    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 [4]:
```python
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]
```

```
Label: Abyssinian
```

Out[4]:

2.2 Data Preprocess

In [5]:
```python
from torchvision import transforms
from torch.utils.data import DataLoader
```

In [6]:

```
# 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 th
at'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]),
        ])
```

In [7]:

```
# Feel free to change
train_set, test_set = load_pet_dataset(train_transform, test_transf
orm)
train_loader = DataLoader(dataset=train_set,
                          batch_size=128,
                          shuffle=True,
                          num_workers=2)

test_loader = DataLoader(dataset=test_set,
                         batch_size=128,
                         shuffle=False,
                         num_workers=2)
```

2.3 Helper Functions

In [8]:
```python
# 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) + " pa
rams":^34s}' +'--']
  print('\n'.join(summary_str))

  # print model structure
  print(model)
```

In [9]:
```python
# 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_frequenc
y == 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_frequenc
y == 0 or i ==0 or i == 1)]
    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 [10]:
```python
device = torch.device("mps")
```

In [11]:
```python
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 C
rossEntropyLoss 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 be
low; https://pypi.org/project/tqdm/
    it_train = tqdm(enumerate(train_loader), total=len(train_loade
r), desc="Training ...", position = 0) # progress bar
    for i, (images, labels) in it_train:

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

        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 C
rossEntropyLoss 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 be
low; 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/gene
rated/torch.no_grad.html
         output = model(images) # do not compute gradient when perfor
ming prediction
        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 [12]:

```
import torch
import torchvision.models as models

# loads a pre-trained ResNet-34 model
model = torch.hub.load('pytorch/vision:v0.10.0', 'resnet34', pretra
ined=True)

# Number of target classes in the Pet dataset
num_target_classes = 37

# Replace the last layer (classification head) with a new linear la
yer for Pet classification
model.fc = torch.nn.Linear(in_features=model.fc.in_features, out_fe
atures=num_target_classes)

# Assuming 'device' is defined (e.g., device = torch.device("cuda"
if torch.cuda.is_available() else "cpu"))
model = model.to(device)

# Function to display the model structure; ensure it is correctly d
efined elsewhere in your code
# This function might simply be a print(model) or more detailed log
ging of model parameters
def display_model(model):
    print(model)

display_model(model)  # Displays the model structure and parameter
count
```

```
Using cache found in /Users/darian/.cache/torch/hub/pytorch_vision
_v0.10.0
/opt/homebrew/lib/python3.8/site-packages/torchvision/models/_util
s.py:208: UserWarning: The parameter 'pretrained' is deprecated si
nce 0.13 and may be removed in the future, please use 'weights' in
stead.
  warnings.warn(
/opt/homebrew/lib/python3.8/site-packages/torchvision/models/_util
s.py:223: UserWarning: Arguments other than a weight enum or `None
` for 'weights' are deprecated since 0.13 and may be removed in th
e future. The current behavior is equivalent to passing `weights=R
esNet34_Weights.IMAGENET1K_V1`. You can also use `weights=ResNet34
_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)

ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), paddin
g=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, tra
ck_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilatio
n=1, ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), p
adding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), p
adding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), p
adding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), p
adding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (2): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), p
adding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), p
adding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_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=Tru
e, track_running_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=Tru
e, track_running_stats=True)
        (downsample): Sequential(
          (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bi
as=False)
          (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=Tru
e, track_running_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=Tru
e, track_running_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=Tru
e, track_running_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=Tru
e, track_running_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=Tru
e, track_running_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=Tru
e, track_running_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=Tru
e, track_running_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=Tru
e, track_running_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=Tru
e, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), b
ias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=Tru
e, track_running_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=Tru
e, track_running_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=Tru
e, track_running_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=Tru
e, track_running_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=Tru
e, track_running_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=Tru
e, track_running_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=Tru
e, track_running_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=Tru
e, track_running_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=Tru
e, track_running_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=Tru
e, track_running_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=Tru
e, track_running_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=Tru
e, track_running_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=Tru
e, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), b
ias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=Tru
e, track_running_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=Tru
e, track_running_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=Tru
e, track_running_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=Tru
e, track_running_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=Tru
e, track_running_stats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=37, bias=True)
)
```

In [13]:
```
# Training Setting. Feel free to change.



num_epochs = 10
test_interval = 1

# TO DO: set initial learning rate
learn_rate = 0.1
optimizer = torch.optim.SGD(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

# Setting up CosineAnnealingLR
T_max = 10  # Duration of a single learning rate cycle
eta_min = 0.00001  # Minimum learning rate
lr_scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimize
r, T_max=T_max, eta_min=eta_min)

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="Traini
ng ...", position=1):

    # 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 epo
ch 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_l
oss)} \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 training accuracy
        train_accuracy_list.append(train_accuracy)

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

        test_accuracy, test_loss = test(test_loader, model, criteri
```

```
on) # Get testing accuracy and error
        test_losses.append(test_loss)
        test_accuracy_list.append(test_accuracy)
        print(f'Test (val) accuracy on epoch {str(epoch)} is {str(t
est_accuracy)} \n')

        # Checkpoints are used to save the model with best validati
on 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

        # if test_accuracy >= 0.9:
        #       print("Desired accuracy reached. Saving Model.")
        #       save_checkpoint(save_dir, model, save_name='best_mo
del.pth')
        #       success = True
        #       break  # Exit the loop early if the desired accurac
y is met
```

```
loss: 0.356: 100%|██████████| 29/29 [00:48<00:00,  1.66s/it]

Loss for Training on epoch 0 is 1.1219996213912964

Evaluating Network

Validating ...: 100%|██████████| 29/29 [00:21<00:00,  1.34it/s]

Mean Accuracy: 0.9277
Avg loss: 0.2854201726872346
Training accuracy on epoch 0 is 0.9277173913043478


Validating ...: 100%|██████████| 29/29 [00:20<00:00,  1.39it/s]

Mean Accuracy: 0.8498
Avg loss: 0.5052560028331033
Test (val) accuracy on epoch 0 is 0.8498228400109021

Saving Model

loss: 0.126: 100%|██████████| 29/29 [00:47<00:00,  1.63s/it]

Loss for Training on epoch 1 is 0.15913696587085724

Evaluating Network

Validating ...: 100%|██████████| 29/29 [00:20<00:00,  1.42it/s]

Mean Accuracy: 0.9826
Avg loss: 0.10214457943521697
Training accuracy on epoch 1 is 0.9826086956521739


Validating ...: 100%|██████████| 29/29 [00:20<00:00,  1.42it/s]
```

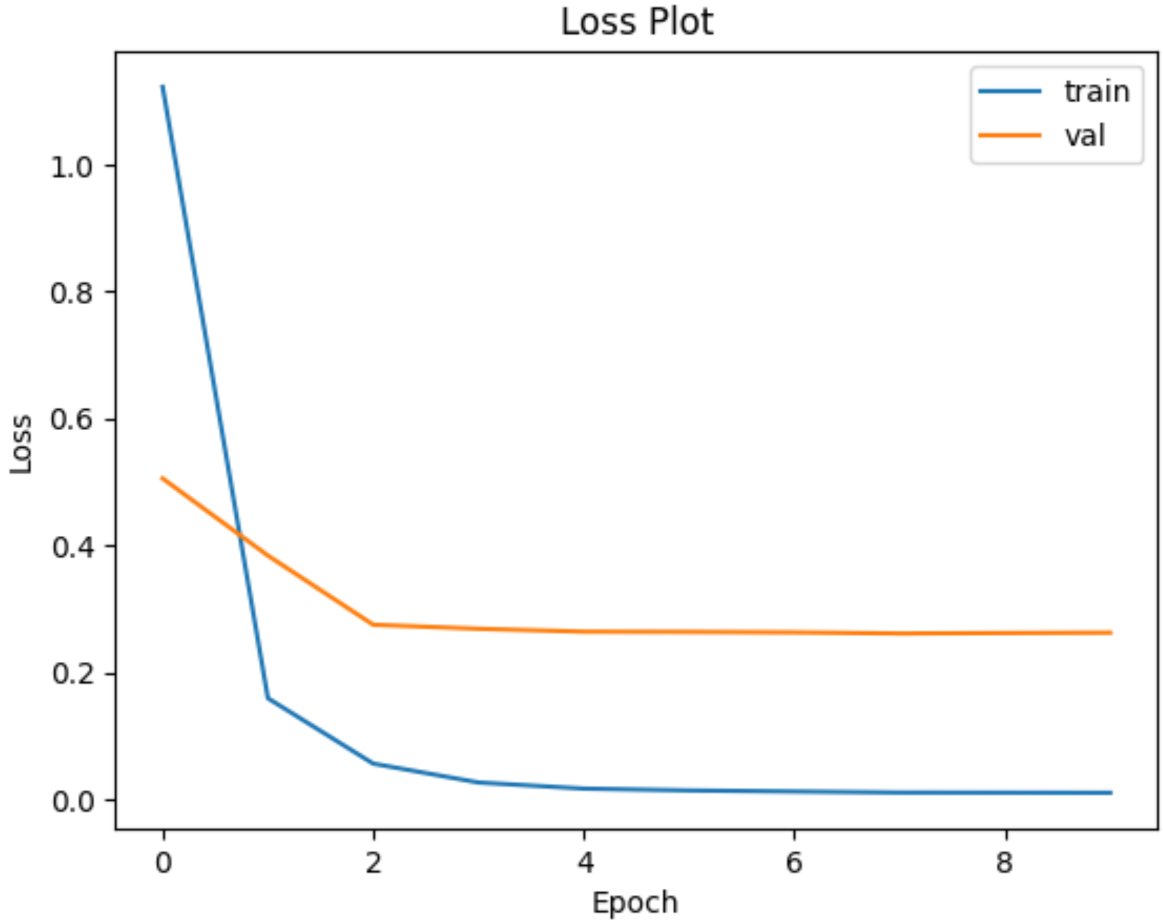
```
Mean Accuracy: 0.8918
Avg loss: 0.3837804488580802
Test (val) accuracy on epoch 1 is 0.8917961297356228

Saving Model

loss: 0.040: 100%|██████████| 29/29 [00:46<00:00,  1.61s/it]

Loss for Training on epoch 2 is 0.056164197623729706

Evaluating Network

Validating ...: 100%|██████████| 29/29 [00:20<00:00,  1.40it/s]

Mean Accuracy: 0.9997
Avg loss: 0.024761343631764937
Training accuracy on epoch 2 is 0.9997282608695652


Validating ...: 100%|██████████| 29/29 [00:20<00:00,  1.44it/s]

Mean Accuracy: 0.9199
Avg loss: 0.2748762646625782
Test (val) accuracy on epoch 2 is 0.919869174161897

Saving Model

loss: 0.017: 100%|██████████| 29/29 [00:46<00:00,  1.59s/it]

Loss for Training on epoch 3 is 0.026439199224114418

Evaluating Network

Validating ...: 100%|██████████| 29/29 [00:20<00:00,  1.43it/s]

Mean Accuracy: 1.0000
Avg loss: 0.013861465299951619
Training accuracy on epoch 3 is 1.0


Validating ...: 100%|██████████| 29/29 [00:20<00:00,  1.44it/s]

Mean Accuracy: 0.9237
Avg loss: 0.2685117555846428
Test (val) accuracy on epoch 3 is 0.9236849277732352

Saving Model

loss: 0.018: 100%|██████████| 29/29 [00:46<00:00,  1.60s/it]

Loss for Training on epoch 4 is 0.016797997057437897

Evaluating Network

Validating ...: 100%|██████████| 29/29 [00:20<00:00,  1.43it/s]

Mean Accuracy: 1.0000
Avg loss: 0.010497034398903107
Training accuracy on epoch 4 is 1.0
```

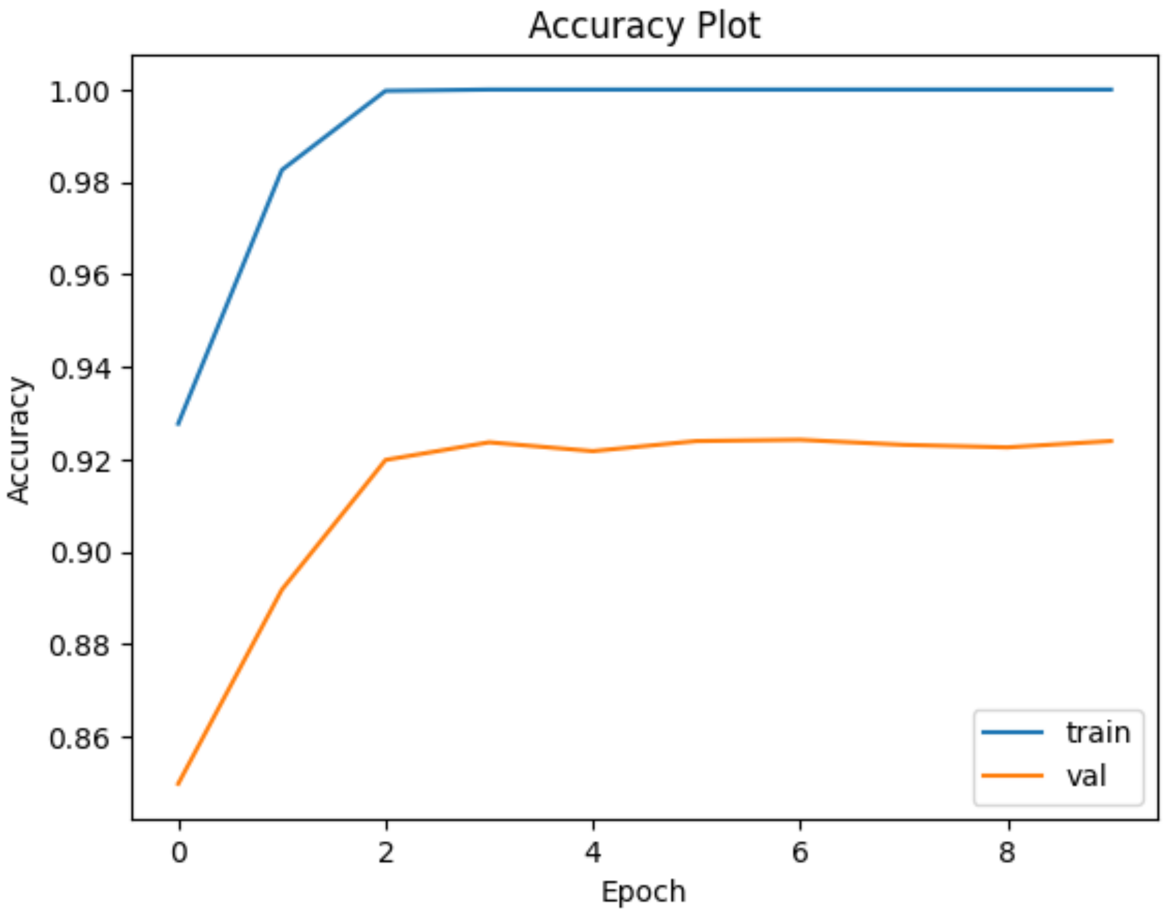
```
Validating ...: 100%|██████████| 29/29 [00:20<00:00,  1.45it/s]

Mean Accuracy: 0.9218
Avg loss: 0.2639962213820425
Test (val) accuracy on epoch 4 is 0.9217770509675661


loss: 0.011: 100%|██████████| 29/29 [00:46<00:00,  1.61s/it]

Loss for Training on epoch 5 is 0.013916683383286

Evaluating Network

Validating ...: 100%|██████████| 29/29 [00:21<00:00,  1.34it/s]

Mean Accuracy: 1.0000
Avg loss: 0.008532094271403962
Training accuracy on epoch 5 is 1.0


Validating ...: 100%|██████████| 29/29 [00:20<00:00,  1.43it/s]

Mean Accuracy: 0.9240
Avg loss: 0.26364428614234103
Test (val) accuracy on epoch 5 is 0.9239574816026165

Saving Model

loss: 0.013: 100%|██████████| 29/29 [00:47<00:00,  1.64s/it]

Loss for Training on epoch 6 is 0.012299755588173866

Evaluating Network

Validating ...: 100%|██████████| 29/29 [00:20<00:00,  1.43it/s]

Mean Accuracy: 1.0000
Avg loss: 0.007661389395723055
Training accuracy on epoch 6 is 1.0


Validating ...: 100%|██████████| 29/29 [00:20<00:00,  1.43it/s]

Mean Accuracy: 0.9242
Avg loss: 0.26288495277022494
Test (val) accuracy on epoch 6 is 0.9242300354319978

Saving Model

loss: 0.010: 100%|██████████| 29/29 [00:46<00:00,  1.59s/it]

Loss for Training on epoch 7 is 0.010628700256347656

Evaluating Network

Validating ...: 100%|██████████| 29/29 [00:20<00:00,  1.43it/s]
```

```
Mean Accuracy: 1.0000
Avg loss: 0.007380804384191488
Training accuracy on epoch 7 is 1.0


Validating ...: 100%|██████████| 29/29 [00:20<00:00,  1.44it/s]

Mean Accuracy: 0.9231
Avg loss: 0.2611089857487843
Test (val) accuracy on epoch 7 is 0.9231398201144726


loss: 0.014: 100%|██████████| 29/29 [00:46<00:00,  1.59s/it]

Loss for Training on epoch 8 is 0.01050286553800106

Evaluating Network

Validating ...: 100%|██████████| 29/29 [00:20<00:00,  1.43it/s]

Mean Accuracy: 1.0000
Avg loss: 0.007227688128578252
Training accuracy on epoch 8 is 1.0


Validating ...: 100%|██████████| 29/29 [00:20<00:00,  1.44it/s]

Mean Accuracy: 0.9226
Avg loss: 0.26182977829513876
Test (val) accuracy on epoch 8 is 0.92259471245571


loss: 0.009: 100%|██████████| 29/29 [00:46<00:00,  1.59s/it]

Loss for Training on epoch 9 is 0.010310512967407703

Evaluating Network

Validating ...: 100%|██████████| 29/29 [00:20<00:00,  1.43it/s]

Mean Accuracy: 1.0000
Avg loss: 0.007085385106118588
Training accuracy on epoch 9 is 1.0


Validating ...: 100%|██████████| 29/29 [00:20<00:00,  1.45it/s]
Training ...: 100%|██████████| 10/10 [15:44<00:00, 94.49s/it]

Mean Accuracy: 0.9240
Avg loss: 0.2624028641088256
Test (val) accuracy on epoch 9 is 0.9239574816026165
```

2.5 Plotting of losses and accuracy

In [14]:
```
plot_losses(train_losses, test_losses, test_interval, num_epochs)
plot_accuracy(train_accuracy_list, test_accuracy_list, test_interva
l, num_epochs)
```


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


Accuracy Plot
1.00
0.98
0.96
0.94
0.92
0.90
0.88
0.86
Accuracy
train
val
0
2
4
6
8
Epoch

Part 3: CLIP: Contrastive Language-Image Pretraining

Include all the code for Part 3 in this section

3.1 Prepare data

[Here (https://drive.google.com/file/d/1zJ1KfymSfsbmD6QS-F0eUC8T1PkqW0_j/view?usp=sharing)](https://drive.google.com/file/d/1zJ1KfymSfsbmD6QS-F0eUC8T1PkqW0_j/view?usp=sharing) 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
from google.colab import drive
import torch
from torchvision.datasets import Flowers102
%matplotlib inline
from matplotlib import pyplot as plt
```

In [ ]:

```
drive.mount('/content/drive', force_remount = True)
datadir = "/content/drive/My Drive/CS441"  # if you copy the json t
o Google Drive, you'll just have to load the images once
# datadir = "."
```

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

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=i
mg_transform, download=do_download)
    test_set = Flowers102(root=datadir, split='val', transform=img_
transform, download=do_download)
    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 usi
ng Colabs and want to use GPU for speed,
#            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()
```

In [ ]:
```
len(flower_train), len(flower_test)  # output should be (1020, 102
0)
```

Out[ ]:
```
(1020, 1020)
```

In [ ]:
```
# 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
```

Out[ ]:

3.2 Prepare CLIP model

In [ ]:
```
# !conda install --yes -c pytorch pytorch=1.7.1 torchvision cudatoolkit=11.0
# !pip install ftfy regex tqdm
!pip install pytorch==1.7.1 torchvision cudatoolkit==11.0
```

```
ERROR: Could not find a version that satisfies the requirement pytorch==1.7.1 (from versions: 0.1.2, 1.0.2)
ERROR: No matching distribution found for pytorch==1.7.1
```

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-ucqbvr9l
  Running command git clone --filter=blob:none --quiet https://github.com/openai/CLIP.git /tmp/pip-req-build-ucqbvr9l
  Resolved https://github.com/openai/CLIP.git to commit a1d071733d7111c9c014f024669f959182114e33
  Preparing metadata (setup.py) ... done
Collecting ftfy (from clip==1.0)
  Downloading ftfy-6.2.0-py3-none-any.whl (54 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 54.4/54.4 kB 413.7 kB/s eta 0:00:00
Requirement already satisfied: regex in /usr/local/lib/python3.10/dist-packages (from clip==1.0) (2023.12.25)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from clip==1.0) (4.66.2)
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (from clip==1.0) (2.2.1+cu121)
Requirement already satisfied: torchvision in /usr/local/lib/python3.10/dist-packages (from clip==1.0) (0.17.1+cu121)
Requirement already satisfied: wcwidth<0.3.0,>=0.2.12 in /usr/local/lib/python3.10/dist-packages (from ftfy->clip==1.0) (0.2.13)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch->clip==1.0) (3.13.4)
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/python3.10/dist-packages (from torch->clip==1.0) (4.11.0)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch->clip==1.0) (1.12)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch->clip==1.0) (3.3)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch->clip==1.0) (3.1.3)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch->clip==1.0) (2023.6.0)
Collecting nvidia-cuda-nvrtc-cu12==12.1.105 (from torch->clip==1.0)
  Using cached nvidia_cuda_nvrtc_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (23.7 MB)
Collecting nvidia-cuda-runtime-cu12==12.1.105 (from torch->clip==1.0)
  Using cached nvidia_cuda_runtime_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (823 kB)
Collecting nvidia-cuda-cupti-cu12==12.1.105 (from torch->clip==1.0)
  Using cached nvidia_cuda_cupti_cu12-12.1.105-py3-none-manylinux1
```

```
_x86_64.whl (14.1 MB)
Collecting nvidia-cudnn-cu12==8.9.2.26 (from torch->clip==1.0)
  Using cached nvidia_cudnn_cu12-8.9.2.26-py3-none-manylinux1_x86_
64.whl (731.7 MB)
Collecting nvidia-cublas-cu12==12.1.3.1 (from torch->clip==1.0)
  Using cached nvidia_cublas_cu12-12.1.3.1-py3-none-manylinux1_x86
_64.whl (410.6 MB)
Collecting nvidia-cufft-cu12==11.0.2.54 (from torch->clip==1.0)
  Using cached nvidia_cufft_cu12-11.0.2.54-py3-none-manylinux1_x86
_64.whl (121.6 MB)
Collecting nvidia-curand-cu12==10.3.2.106 (from torch->clip==1.0)
  Using cached nvidia_curand_cu12-10.3.2.106-py3-none-manylinux1_x
86_64.whl (56.5 MB)
Collecting nvidia-cusolver-cu12==11.4.5.107 (from torch->clip==1.
0)
  Using cached nvidia_cusolver_cu12-11.4.5.107-py3-none-manylinux1
_x86_64.whl (124.2 MB)
Collecting nvidia-cusparse-cu12==12.1.0.106 (from torch->clip==1.
0)
  Using cached nvidia_cusparse_cu12-12.1.0.106-py3-none-manylinux1
_x86_64.whl (196.0 MB)
Collecting nvidia-nccl-cu12==2.19.3 (from torch->clip==1.0)
  Using cached nvidia_nccl_cu12-2.19.3-py3-none-manylinux1_x86_64.
whl (166.0 MB)
Collecting nvidia-nvtx-cu12==12.1.105 (from torch->clip==1.0)
  Using cached nvidia_nvtx_cu12-12.1.105-py3-none-manylinux1_x86_6
4.whl (99 kB)
Requirement already satisfied: triton==2.2.0 in /usr/local/lib/pyt
hon3.10/dist-packages (from torch->clip==1.0) (2.2.0)
Collecting nvidia-nvjitlink-cu12 (from nvidia-cusolver-cu12==11.4.
5.107->torch->clip==1.0)
  Using cached nvidia_nvjitlink_cu12-12.4.127-py3-none-manylinux20
14_x86_64.whl (21.1 MB)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/
dist-packages (from torchvision->clip==1.0) (1.25.2)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/loca
l/lib/python3.10/dist-packages (from torchvision->clip==1.0) (9.4.
0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/p
ython3.10/dist-packages (from jinja2->torch->clip==1.0) (2.1.5)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/pyth
on3.10/dist-packages (from sympy->torch->clip==1.0) (1.3.0)
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=
1369499 sha256=2fb43b46e13a41a2ea3f90c30f17034f27ed626dd9fce379274
3fbc4fb07c984
  Stored in directory: /tmp/pip-ephem-wheel-cache-h1n9iwtz/wheels/
da/2b/4c/d6691fa9597aac8bb85d2ac13b112deb897d5b50f5ad9a37e4
Successfully built clip
Installing collected packages: nvidia-nvtx-cu12, nvidia-nvjitlink-
cu12, nvidia-nccl-cu12, nvidia-curand-cu12, nvidia-cufft-cu12, nvi
dia-cuda-runtime-cu12, nvidia-cuda-nvrtc-cu12, nvidia-cuda-cupti-c
u12, nvidia-cublas-cu12, ftfy, nvidia-cusparse-cu12, nvidia-cudnn-
cu12, nvidia-cusolver-cu12, clip
Successfully installed clip-1.0 ftfy-6.2.0 nvidia-cublas-cu12-12.
1.3.1 nvidia-cuda-cupti-cu12-12.1.105 nvidia-cuda-nvrtc-cu12-12.1.
```

```
105 nvidia-cuda-runtime-cu12-12.1.105 nvidia-cudnn-cu12-8.9.2.26 n
vidia-cufft-cu12-11.0.2.54 nvidia-curand-cu12-10.3.2.106 nvidia-cu
solver-cu12-11.4.5.107 nvidia-cusparse-cu12-12.1.0.106 nvidia-nccl
-cu12-2.19.3 nvidia-nvjitlink-cu12-12.4.127 nvidia-nvtx-cu12-12.1.
105
```

In [ ]:
```
import clip
import torch
```

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 downl
oad
clip_model, clip_preprocess = clip.load("ViT-B/32", device=device)
```

```
cpu

100%|██████████████████████████████████████████| 338M/338M [00:03<0
0:00, 104MiB/s]
```

3.3 CLIP zero-shot prediction

In [ ]:

```
"""The following is an example of using CLIP pre-trained model for
zero-shot prediction task"""
# Prepare the inputs
n = 100  # image index to use
image, class_id = flower_train[n]
image_input = clip_preprocess(image).unsqueeze(0).to(device) # extr
act image and put in device memory
text_inputs = torch.cat([clip.tokenize(f"a photo of a {c}, a type o
f flower.") for c in flower_classes]).to(device) # put text to matc
h to image in device memory

# 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 t
ext 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-no
rmalize 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 l
abels

# 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():.2
f}%")
image
```

```
Ground truth: snapdragon

Top predictions:

        sweet pea: 30.35%
     garden phlox: 26.37%
       snapdragon: 25.95%
      wallflower: 4.24%
    bougainvillea: 1.91%
```

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 tr
ansform (you don't have to call clip_preprocess again)
flower_train_trans, flower_test_trans, flower_classes = load_flower
_data(img_transform=clip_preprocess)
```

In [ ]:

```python
def clip_zero_shot(data_set, classes):
    data_loader = DataLoader(data_set, batch_size=32, shuffle=False)  # dataloader lets you process in batch which is way faster (when using GPU)

    # TO DO: Needs code here

    total_correct = 0
    total_images = 0

    class_descriptions = [f"a photo of a {c}, a type of flower" for c in classes]
    text_tokens = clip.tokenize(class_descriptions).to(device)

    with torch.no_grad():
        text_features = clip_model.encode_text(text_tokens)
        text_features /= text_features.norm(dim=-1, keepdim=True)

        # Iterate over all batches
        for images, labels in tqdm(data_loader, desc="Evaluating"):
            images = images.to(device)
            labels = labels.to(device)

            # Calculate image features
            image_features = clip_model.encode_image(images)
            image_features /= image_features.norm(dim=-1, keepdim=True)

            # Calculate similarity and predictions
            similarity = image_features @ text_features.T
            predictions = similarity.argmax(dim=-1)

            # Update correct predictions counter
            total_correct += (predictions == labels).sum().item()
            total_images += labels.size(0)

    # Calculate accuracy
    accuracy = total_correct / total_images
    return accuracy
```

In [ ]:

```python
torch.cuda.empty_cache()
```

In [ ]:

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

```
Evaluating: 100%|██████████| 32/32 [00:14<00:00,  2.23it/s]

Accuracy = 67.647%
```

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 vectors.
"""
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().numpy()
```

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, verbose=1)
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))
print(f"\nAccuracy = {100*accuracy:.3f}%")
```

```
100%|██████████| 11/11 [06:48<00:00, 37.18s/it]
100%|██████████| 11/11 [06:47<00:00, 37.03s/it]

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.ne
ighbors.NeighborsClassifier or FAISS or own implementation)

from sklearn.neighbors import KNeighborsClassifier

model = KNeighborsClassifier(n_neighbors=9)

model.fit(train_features, train_labels)

# Evaluate accuracy on test_features, test_labels
predictions = model.predict(test_features)
accuracy = np.mean((test_labels == predictions).astype(float))

print(f"\nAccuracy = {100*accuracy:.3f}%")
```

```
Accuracy = 85.784%
```

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 (https://github.com/huggingface/tokenizers)](https://github.com/huggingface/tokenizers)

In [ ]:
```
!pip install tokenizers
```

```
Requirement already satisfied: tokenizers in /usr/local/lib/python
3.10/dist-packages (0.15.2)
Requirement already satisfied: huggingface_hub<1.0,>=0.16.4 in /us
r/local/lib/python3.10/dist-packages (from tokenizers) (0.20.3)
Requirement already satisfied: filelock in /usr/local/lib/python3.
10/dist-packages (from huggingface_hub<1.0,>=0.16.4->tokenizers) (
3.13.4)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/
python3.10/dist-packages (from huggingface_hub<1.0,>=0.16.4->token
izers) (2023.6.0)
Requirement already satisfied: requests in /usr/local/lib/python3.
10/dist-packages (from huggingface_hub<1.0,>=0.16.4->tokenizers) (
2.31.0)
Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/pyth
on3.10/dist-packages (from huggingface_hub<1.0,>=0.16.4->tokenizer
s) (4.66.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/pytho
n3.10/dist-packages (from huggingface_hub<1.0,>=0.16.4->tokenizer
s) (6.0.1)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/
local/lib/python3.10/dist-packages (from huggingface_hub<1.0,>=0.1
6.4->tokenizers) (4.11.0)
Requirement already satisfied: packaging>=20.9 in /usr/local/lib/p
ython3.10/dist-packages (from huggingface_hub<1.0,>=0.16.4->tokeni
zers) (24.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/lo
cal/lib/python3.10/dist-packages (from requests->huggingface_hub<
1.0,>=0.16.4->tokenizers) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/pyth
on3.10/dist-packages (from requests->huggingface_hub<1.0,>=0.16.4-
>tokenizers) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/li
b/python3.10/dist-packages (from requests->huggingface_hub<1.0,>=
0.16.4->tokenizers) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/li
b/python3.10/dist-packages (from requests->huggingface_hub<1.0,>=
0.16.4->tokenizers) (2024.2.2)
```

In [4]:
```
import requests
from tokenizers import Tokenizer
from tokenizers.models import BPE, WordPiece
from tokenizers.trainers import BpeTrainer, WordPieceTrainer

def download_file(url):
    local_filename = url.split('/')[-1]
    with requests.get(url, stream=True) as r:
        r.raise_for_status()
        with open(local_filename, 'wb') as f:
            for chunk in r.iter_content(chunk_size=8192):
                f.write(chunk)
    return local_filename

# Download WikiText-2 training data
url = "https://raw.githubusercontent.com/pytorch/examples/master/wo
```

```
rd_language_model/data/wikitext-2/train.txt"
train_file = download_file(url)



# Initialize tokenizers
bpe_tokenizer = Tokenizer(BPE(unk_token="[UNK]"))
wordpiece_tokenizer = Tokenizer(WordPiece(unk_token="[UNK]"))

# Initialize trainers
bpe_trainer = BpeTrainer(vocab_size=8000, special_tokens=["[UNK]",
"[CLS]", "[SEP]", "[PAD]", "[MASK]"])
wordpiece_trainer = WordPieceTrainer(vocab_size=8000, special_token
s=["[UNK]", "[CLS]", "[SEP]", "[PAD]", "[MASK]"])

# Train BPE
bpe_tokenizer.train([train_file], bpe_trainer)

# Train WordPiece
wordpiece_tokenizer.train([train_file], wordpiece_trainer)

# Save tokenizers
bpe_tokenizer.save("bpe_tokenizer.json")
wordpiece_tokenizer.save("wordpiece_tokenizer.json")


sample_text = "I am learning about word tokenizers. They are not ve
ry complicated, and they are a good way to convert natural text int
o tokens."
additional_text = "I have learnt a lot from Applied Machine Learnin
g at UIUC"

# Encode with BPE
bpe_encoded = bpe_tokenizer.encode(sample_text)
print("BPE Encoding:", bpe_encoded.tokens)

# Encode with WordPiece
wordpiece_encoded = wordpiece_tokenizer.encode(sample_text)
print("WordPiece Encoding:", wordpiece_encoded.tokens)

# Encode additional sentence
bpe_encoded_additional = bpe_tokenizer.encode(additional_text)
wordpiece_encoded_additional = wordpiece_tokenizer.encode(additiona
l_text)
print("Additional BPE Encoding:", bpe_encoded_additional.tokens)
print("Additional WordPiece Encoding:", wordpiece_encoded_additiona
l.tokens)
```

```
BPE Encoding: ['I ', 'am ', 'lear', 'ning ', 'about ', 'word ', 't
o', 'k', 'en', 'iz', 'er', 's', '. They ', 'are not ', 'very ', 'c
ompl', 'ic', 'at', 'ed', ', and ', 'they are ', 'a ', 'good ', 'wa
y to ', 'conver', 't ', 'natural ', 'text ', 'into ', 'to', 'k', '
ens', '.']
WordPiece Encoding: ['[UNK]']
Additional BPE Encoding: ['I ', 'have ', 'lear', 'n', 't ', 'a ',
'l', 'ot ', 'from ', 'Ap', 'pl', 'ied ', 'M', 'ach', 'ine ', 'L',
'ear', 'ning ', 'at ', 'U', 'I', 'U', 'C']
Additional WordPiece Encoding: ['I', '## ', '##have ', '##lear',
'##n', '##t ', '##a ', '##lo', '##t ', '##from ', '##Ap', '##pl',
'##ied ', '##Mac', '##hi', '##ne ', '##Le', '##ar', '##ning ', '##
at ', '##U', '##I', '##U', '##C']
```

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 n
etwork 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
```

In [ ]:

```
# from https://gist.github.com/jonathanagustin/b67b97ef12c53a8dec27
b343dca4abba
# install can take a minute

import os
# @title Convert Notebook to PDF. Save Notebook to given directory
NOTEBOOKS_DIR = "/content/drive/My Drive/CS441/24SP/hw2" # @param {
type:"string"}
NOTEBOOK_NAME = "CS441_SP24_HW2_Solution.ipynb" # @param {type:"str
ing"}
#---------------------------------------------------------------------
------------#
from google.colab import drive
drive.mount("/content/drive/", force_remount=True)
NOTEBOOK_PATH = f"{NOTEBOOKS_DIR}/{NOTEBOOK_NAME}"
assert os.path.exists(NOTEBOOK_PATH), f"NOTEBOOK NOT FOUND: {NOTEBO
OK_PATH}"
!apt install -y texlive-xetex texlive-fonts-recommended texlive-pla
in-generic > /dev/null 2>&1
!jupyter nbconvert "$NOTEBOOK_PATH" --to pdf > /dev/null 2>&1
NOTEBOOK_PDF = NOTEBOOK_PATH.rsplit('.', 1)[0] + '.pdf'
assert os.path.exists(NOTEBOOK_PDF), f"ERROR MAKING PDF: {NOTEBOOK_
PDF}"
print(f"PDF CREATED: {NOTEBOOK_PDF}")
```

In [ ]: