# Tanawin-st123975-task1-slipt-data

October 27, 2023

[ ]:
```
!nvidia-smi
```

```
Sun Oct 22 18:56:57 2023
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.105.17   Driver Version: 525.105.17   CUDA Version: 12.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   71C    P0    29W /  70W |   2923MiB / 15360MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
+-----------------------------------------------------------------------------+
```

[ ]:
```
# from google.colab import files
# files.upload()
```

[ ]:
```
# from google.colab import drive

# drive.mount('/content/gdrive')
```

[ ]:
```
# !unzip /content/gdrive/MyDrive/AIT/ISE/computer-vision/lab/week-12/
 ↪vege_dataset.zip
```

[ ]:
```
import torch
import torchvision
from torchvision import datasets, models, transforms
import torch.nn as nn
import torch.optim as optim
import time
```

```
import os
from copy import copy
from copy import deepcopy
import torch.nn.functional as F

# Set device to GPU or CPU

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

[ ]:
```
import torch
import torchvision.transforms as transforms
from torchvision.datasets import ImageFolder
from torch.utils.data import DataLoader, random_split
```

[ ]:
```
class BasicBlock(nn.Module):
    '''
    BasicBlock: Simple residual block with two conv layers
    '''
    EXPANSION = 1
    def __init__(self, in_planes, out_planes, stride=1):
        super().__init__()
        self.conv1 = nn.Conv2d(in_planes, out_planes, kernel_size=3,␣
↪stride=stride, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(out_planes)
        self.conv2 = nn.Conv2d(out_planes, out_planes, kernel_size=3, stride=1,␣
↪padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(out_planes)
        self.shortcut = nn.Sequential()
        # If output size is not equal to input size, reshape it with 1x1␣
↪convolution
        if stride != 1 or in_planes != out_planes:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_planes, out_planes, kernel_size=1, stride=stride,␣
↪bias=False),
                nn.BatchNorm2d(out_planes)
            )

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        out += self.shortcut(x)
        out = F.relu(out)
        return out
```

[ ]:
```
class Bottleneck(nn.Module):
    '''
```

```
    BottleneckBlock: More powerful residual block with three convs, used for␣
↪Resnet50 and up
    '''
    EXPANSION = 4
    def __init__(self, in_planes, planes, stride=1):
        super().__init__()
        self.conv1 = nn.Conv2d(in_planes, planes, kernel_size=1, bias=False)
        self.bn1 = nn.BatchNorm2d(planes)
        self.conv2 = nn.Conv2d(planes, planes, kernel_size=3, stride=stride,␣
↪padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(planes)
        self.conv3 = nn.Conv2d(planes, self.EXPANSION * planes, kernel_size=1,␣
↪bias=False)
        self.bn3 = nn.BatchNorm2d(self.EXPANSION * planes)

        self.shortcut = nn.Sequential()
        # If the output size is not equal to input size, reshape it with 1x1␣
↪convolution
        if stride != 1 or in_planes != self.EXPANSION * planes:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_planes, self.EXPANSION * planes,
                          kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(self.EXPANSION * planes)
            )

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = F.relu(self.bn2(self.conv2(out)))
        out = self.bn3(self.conv3(out))
        out += self.shortcut(x)
        out = F.relu(out)
        return out
```

[ ]:
```
# class ResNet(nn.Module):
#     def __init__(self, block, num_blocks, num_classes=10):
#         super().__init__()
#         self.in_planes = 64
#         # Initial convolution
#         self.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1,␣
 ↪bias=False)
#         self.bn1 = nn.BatchNorm2d(64)
#         # Residual blocks
#         self.layer1 = self._make_layer(block, 64, num_blocks[0], stride=1)
#         self.layer2 = self._make_layer(block, 128, num_blocks[1], stride=2)
#         self.layer3 = self._make_layer(block, 256, num_blocks[2], stride=2)
#         self.layer4 = self._make_layer(block, 512, num_blocks[3], stride=2)
#         # FC layer = 1 layer
```

```
#         self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
#         self.linear = nn.Linear(512 * block.EXPANSION, num_classes)

#     def _make_layer(self, block, planes, num_blocks, stride):
#         strides = [stride] + [1] * (num_blocks-1)
#         layers = []
#         for stride in strides:
#             layers.append(block(self.in_planes, planes, stride))
#             self.in_planes = planes * block.EXPANSION
#         return nn.Sequential(*layers)

#     def forward(self, x):
#         out = F.relu(self.bn1(self.conv1(x)))
#         out = self.layer1(out)
#         out = self.layer2(out)
#         out = self.layer3(out)
#         out = self.layer4(out)
#         out = self.avgpool(out)
#         out = out.view(out.size(0), -1)
#         out = self.linear(out)
#         return out
```

[ ]:
```
class ResNet(nn.Module):
    def __init__(self, block, num_blocks, num_classes=10, dropout_prob=0.5):
        super(ResNet, self).__init__()
        self.in_planes = 64
        # Initial convolution
        self.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1,␣
 ↪bias=False)
        self.bn1 = nn.BatchNorm2d(64)
        # Residual blocks
        self.layer1 = self._make_layer(block, 64, num_blocks[0], stride=1)
        self.layer2 = self._make_layer(block, 128, num_blocks[1], stride=2)
        self.layer3 = self._make_layer(block, 256, num_blocks[2], stride=2)
        self.layer4 = self._make_layer(block, 512, num_blocks[3], stride=2)
        # FC layer = 1 layer
        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.linear = nn.Linear(512 * block.EXPANSION, num_classes)
        self.dropout = nn.Dropout(p=dropout_prob)  # Add dropout layer with␣
 ↪specified dropout probability

    def _make_layer(self, block, planes, num_blocks, stride):
        strides = [stride] + [1] * (num_blocks-1)
        layers = []
        for stride in strides:
            layers.append(block(self.in_planes, planes, stride))
            self.in_planes = planes * block.EXPANSION
```

```
        return nn.Sequential(*layers)

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.layer1(out)
        out = self.layer2(out)
        out = self.layer3(out)
        out = self.layer4(out)
        out = self.avgpool(out)
        out = out.view(out.size(0), -1)
        out = self.dropout(out)  # Apply dropout before the linear layer
        out = self.linear(out)
        return out
```

[ ]:
```
def ResNet18(num_classes = 10):
    '''
    First conv layer: 1
    4 residual blocks with two sets of two convolutions each: 2*2 + 2*2 + 2*2 +␣
 ↪2*2 = 16 conv layers
    last FC layer: 1
    Total layers: 1+16+1 = 18
    '''
    return ResNet(BasicBlock, [2, 2, 2, 2], num_classes)


def ResNet34(num_classes):
    '''
    First conv layer: 1
    4 residual blocks with [3, 4, 6, 3] sets of two convolutions each: 3*2 +␣
 ↪4*2 + 6*2 + 3*2 = 32
    last FC layer: 1
    Total layers: 1+32+1 = 34
    '''
    return ResNet(BasicBlock, [3, 4, 6, 3], num_classes)


def ResNet50(num_classes = 10):
    '''
    First conv layer: 1
    4 residual blocks with [3, 4, 6, 3] sets of three convolutions each: 3*3 +␣
 ↪4*3 + 6*3 + 3*3 = 48
    last FC layer: 1
    Total layers: 1+48+1 = 50
    '''
    return ResNet(Bottleneck, [3, 4, 6, 3], num_classes)
```

```
def ResNet101(num_classes = 10):
    '''
    First conv layer: 1
    4 residual blocks with [3, 4, 23, 3] sets of three convolutions each: 3*3 +␣
 ↪4*3 + 23*3 + 3*3 = 99
    last FC layer: 1
    Total layers: 1+99+1 = 101
    '''
    return ResNet(Bottleneck, [3, 4, 23, 3], num_classes)


def ResNet152(num_classes = 10):
    '''
    First conv layer: 1
    4 residual blocks with [3, 8, 36, 3] sets of three convolutions each: 3*3 +␣
 ↪8*3 + 36*3 + 3*3 = 150
    last FC layer: 1
    Total layers: 1+150+1 = 152
    '''
    return ResNet(Bottleneck, [3, 8, 36, 3], num_classes)
```

[ ]:
```
# import important library
from torch.utils.data import Dataset, DataLoader


class BasilDataset(Dataset):
    def __init__(self, root_path="/vege_dataset/", transform=None):
        return

    def __len__(self):
        return 0

    def __getitem__(self, i):
        return
```

[ ]:
```
from torch.utils.data import Dataset, DataLoader
from os import listdir
from PIL import Image


# class BasilDataset(Dataset):
#     def __init__(self, root_path="vege_dataset/", transform=None):
#         # keep root directory
#         self.dir = root_path
#         # keep transform
#         self.transform = transform
```

```
#         # read all files in kapao and horapa folder
#         list_kaprao = listdir(root_path + 'kapao/')
#         list_horapa = listdir(root_path + 'horapa/')
#         # calculate all number for each class (just in case)
#         self.kaprao_len = len(list_kaprao)
#         self.horapa_len = len(list_horapa)

#         # put the data file path into ids
#         self.ids = [self.dir + 'kapao/' + file for file in list_kaprao if not␣
 ↪file.startswith('.')]
#         self.ids.extend([self.dir + 'horapa/' + file for file in list_horapa␣
 ↪if not file.startswith('.')])

#     def __len__(self):
#         return self.kaprao_len + self.horapa_len

#     def __getitem__(self, i):
#         idx = self.ids[i]
#         img_file = idx

#         # open photo
#         pil_img = Image.open(img_file)

#         # resize, normalize and convert to pytorch tensor
#         if self.transform:
#             img = self.transform(pil_img)
#         self.pil_img = pil_img

#         # get label from file list counter
#         if i < self.kaprao_len:
#             label = 0
#         else:
#             label = 1

#         return {
#             'image': img,
#             'label': label,
#             'file_name' : img_file,
#         }

from os import listdir
from PIL import Image
from torch.utils.data import Dataset

class BasilDataset(Dataset):
    def __init__(self, root_path="vege_dataset/", transform=None):
        self.dir = root_path
```

```
        self.transform = transform

        list_kaprao = [file for file in listdir(root_path + 'kapao/') if not␣
↪file.startswith('.')]
        list_horapa = [file for file in listdir(root_path + 'horapa/') if not␣
↪file.startswith('.')]

        self.kaprao_len = len(list_kaprao)
        self.horapa_len = len(list_horapa)

        self.ids = [self.dir + 'kapao/' + file for file in list_kaprao]
        self.ids.extend([self.dir + 'horapa/' + file for file in list_horapa])

    def __len__(self):
        return self.kaprao_len + self.horapa_len

    def __getitem__(self, i):
        if i >= len(self.ids):
            raise IndexError("list index out of range")

        idx = self.ids[i]
        img_file = idx

        pil_img = Image.open(img_file)

        if self.transform:
            img = self.transform(pil_img)
        self.pil_img = pil_img

        if i < self.kaprao_len:
            label = 0
        else:
            label = 1

        return {
            'image': img,
            'label': label,
            'file_name': img_file,
        }
```

[ ]:
```
root = "vege_dataset/"

transform = transforms.Compose([
    transforms.Resize(32),
    transforms.RandomCrop(28), # CenterCrop
    transforms.ToTensor(),
```

```
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.
↪225])])

dataset = BasilDataset(root, transform)
```

[ ]:
```
import matplotlib.pyplot as plt

output_label = ['kaprao', 'horapa']

batch = dataset[0]
image, label, filename = batch['image'], batch['label'], batch['file_name']
pil_img = Image.open(filename)

print(output_label[label])
print(filename)
# (3, 224, 224) pytorch
# pyplot -> (224,224,3)
plt.imshow(pil_img)
plt.show()
```

```
kaprao
vege_dataset/kapao/20220425_205015_resize.jpg
```

[ ]:
```
# train_loader = DataLoader(dataset, batch_size=64, shuffle=True,␣
 ↪pin_memory=True)
# Split the dataset into train and test sets
dataset_length = len(dataset)
train_size = int(0.9 * dataset_length)
test_size = dataset_length - train_size

train_dataset, test_dataset = random_split(dataset, [train_size, test_size])

batch_size = 32

# Create data loaders
train_loader = DataLoader(train_dataset, batch_size=batch_size,
                          shuffle=True, num_workers=32)
test_loader = DataLoader(test_dataset, batch_size=batch_size,
                         shuffle=False, num_workers=32)
```

```
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:557:
UserWarning: This DataLoader will create 32 worker processes in total. Our
suggested max number of worker in current system is 8, which is smaller than
what this DataLoader is going to create. Please be aware that excessive worker
creation might get DataLoader running slow or even freeze, lower the worker
number to avoid potential slowness/freeze if necessary.
  warnings.warn(_create_warning_msg(
```

[ ]:
```
model = ResNet101(2).to(device)
```

[ ]:
```
import torch.optim as optim

criterion = nn.CrossEntropyLoss()

optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9,␣
 ↪weight_decay=0.0001)
# optimizer = optim.RMSprop(model.parameters(), lr=0.001, alpha=0.9,␣
 ↪weight_decay=0.0001)
# optimizer = optim.Adam(model.parameters(), lr=0.001, weight_decay=0.0001)
```

[ ]:
```
from torch.optim.lr_scheduler import ExponentialLR, MultiStepLR

scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.1)

scheduler1 = ExponentialLR(optimizer, gamma=0.9)
scheduler2 = MultiStepLR(optimizer, milestones=[30,80], gamma=0.1)
```

[ ]:
```
model_name = "resnet101_vege"

# Training loop
```

```
num_epochs = 50
train_loss_list = []
valid_loss_list = []
train_accuracy_list = []
valid_accuracy_list = []

for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0

    for batch in train_loader:  # Change here
        images, labels = batch['image'], batch['label']  # Change here
        images = images.to(device)
        labels = labels.to(device)

        optimizer.zero_grad()

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

        running_loss += loss.item()
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    train_loss = running_loss / len(train_loader)
    train_accuracy = correct / total

    # Validation loop
    model.eval()
    running_loss = 0.0
    correct = 0
    total = 0

    with torch.no_grad():
        for batch in test_loader:  # Change here
            images, labels = batch['image'], batch['label']  # Change here
            images = images.to(device)
            labels = labels.to(device)

            outputs = model(images)
            loss = criterion(outputs, labels)
```

```
            running_loss += loss.item()
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    valid_loss = running_loss / len(test_loader)
    valid_accuracy = correct / total

    # Store loss and accuracy values
    train_loss_list.append(train_loss)
    valid_loss_list.append(valid_loss)
    train_accuracy_list.append(train_accuracy)
    valid_accuracy_list.append(valid_accuracy)

    # Print the training/validation statistics
    print(f"Epoch: {epoch+1}/{num_epochs} | "
          f"Train Loss: {train_loss:.4f} | Train Acc: {train_accuracy:.4f} | "
          f"Valid Loss: {valid_loss:.4f} | Valid Acc: {valid_accuracy:.4f}")

    # Update the learning rate
    # scheduler.step()
    scheduler1.step()
    scheduler2.step()

# Save the trained model
torch.save(model.state_dict(), model_name+".pth")
```

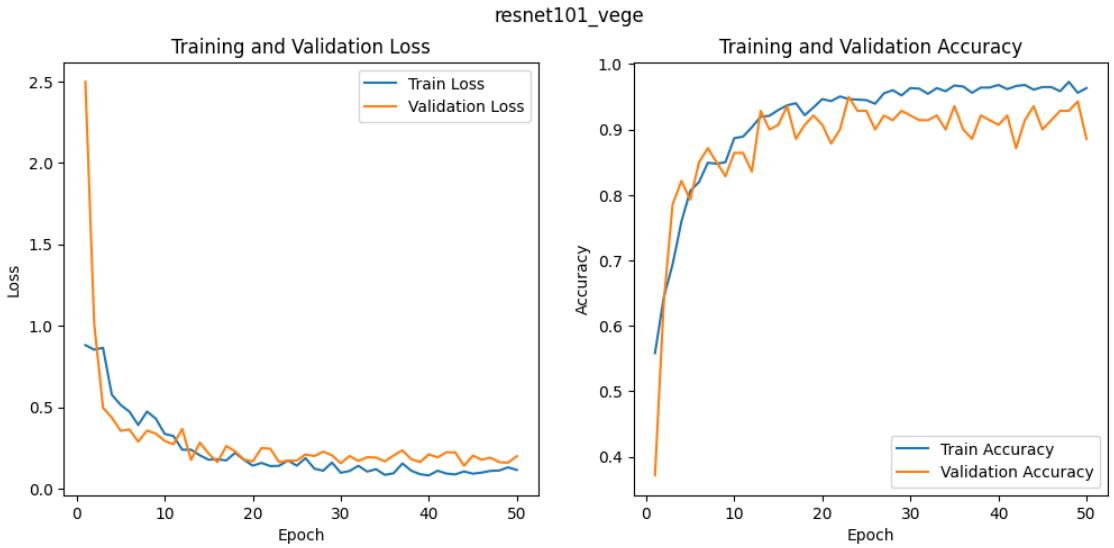
```
Epoch: 1/50 | Train Loss: 0.8811 | Train Acc: 0.5582 | Valid Loss: 2.4973 |
Valid Acc: 0.3714
Epoch: 2/50 | Train Loss: 0.8534 | Train Acc: 0.6435 | Valid Loss: 1.0124 |
Valid Acc: 0.6357
Epoch: 3/50 | Train Loss: 0.8646 | Train Acc: 0.6938 | Valid Loss: 0.4970 |
Valid Acc: 0.7857
Epoch: 4/50 | Train Loss: 0.5772 | Train Acc: 0.7592 | Valid Loss: 0.4367 |
Valid Acc: 0.8214
Epoch: 5/50 | Train Loss: 0.5143 | Train Acc: 0.8062 | Valid Loss: 0.3555 |
Valid Acc: 0.7929
Epoch: 6/50 | Train Loss: 0.4730 | Train Acc: 0.8198 | Valid Loss: 0.3634 |
Valid Acc: 0.8500
Epoch: 7/50 | Train Loss: 0.3911 | Train Acc: 0.8493 | Valid Loss: 0.2885 |
Valid Acc: 0.8714
Epoch: 8/50 | Train Loss: 0.4735 | Train Acc: 0.8477 | Valid Loss: 0.3561 |
Valid Acc: 0.8500
Epoch: 9/50 | Train Loss: 0.4297 | Train Acc: 0.8501 | Valid Loss: 0.3384 |
Valid Acc: 0.8286
Epoch: 10/50 | Train Loss: 0.3364 | Train Acc: 0.8868 | Valid Loss: 0.2930 |
Valid Acc: 0.8643
```

```
Epoch: 11/50 | Train Loss: 0.3232 | Train Acc: 0.8892 | Valid Loss: 0.2724 |
Valid Acc: 0.8643
Epoch: 12/50 | Train Loss: 0.2388 | Train Acc: 0.9035 | Valid Loss: 0.3668 |
Valid Acc: 0.8357
Epoch: 13/50 | Train Loss: 0.2391 | Train Acc: 0.9195 | Valid Loss: 0.1763 |
Valid Acc: 0.9286
Epoch: 14/50 | Train Loss: 0.2045 | Train Acc: 0.9211 | Valid Loss: 0.2828 |
Valid Acc: 0.9000
Epoch: 15/50 | Train Loss: 0.1777 | Train Acc: 0.9298 | Valid Loss: 0.2150 |
Valid Acc: 0.9071
Epoch: 16/50 | Train Loss: 0.1805 | Train Acc: 0.9370 | Valid Loss: 0.1626 |
Valid Acc: 0.9357
Epoch: 17/50 | Train Loss: 0.1733 | Train Acc: 0.9402 | Valid Loss: 0.2615 |
Valid Acc: 0.8857
Epoch: 18/50 | Train Loss: 0.2198 | Train Acc: 0.9219 | Valid Loss: 0.2296 |
Valid Acc: 0.9071
Epoch: 19/50 | Train Loss: 0.1775 | Train Acc: 0.9338 | Valid Loss: 0.1790 |
Valid Acc: 0.9214
Epoch: 20/50 | Train Loss: 0.1411 | Train Acc: 0.9466 | Valid Loss: 0.1671 |
Valid Acc: 0.9071
Epoch: 21/50 | Train Loss: 0.1581 | Train Acc: 0.9434 | Valid Loss: 0.2499 |
Valid Acc: 0.8786
Epoch: 22/50 | Train Loss: 0.1386 | Train Acc: 0.9506 | Valid Loss: 0.2452 |
Valid Acc: 0.9000
Epoch: 23/50 | Train Loss: 0.1402 | Train Acc: 0.9466 | Valid Loss: 0.1629 |
Valid Acc: 0.9500
Epoch: 24/50 | Train Loss: 0.1733 | Train Acc: 0.9458 | Valid Loss: 0.1717 |
Valid Acc: 0.9286
Epoch: 25/50 | Train Loss: 0.1412 | Train Acc: 0.9450 | Valid Loss: 0.1726 |
Valid Acc: 0.9286
Epoch: 26/50 | Train Loss: 0.1873 | Train Acc: 0.9394 | Valid Loss: 0.2099 |
Valid Acc: 0.9000
Epoch: 27/50 | Train Loss: 0.1230 | Train Acc: 0.9553 | Valid Loss: 0.2003 |
Valid Acc: 0.9214
Epoch: 28/50 | Train Loss: 0.1101 | Train Acc: 0.9601 | Valid Loss: 0.2268 |
Valid Acc: 0.9143
Epoch: 29/50 | Train Loss: 0.1604 | Train Acc: 0.9522 | Valid Loss: 0.2051 |
Valid Acc: 0.9286
Epoch: 30/50 | Train Loss: 0.0975 | Train Acc: 0.9633 | Valid Loss: 0.1557 |
Valid Acc: 0.9214
Epoch: 31/50 | Train Loss: 0.1085 | Train Acc: 0.9625 | Valid Loss: 0.2006 |
Valid Acc: 0.9143
Epoch: 32/50 | Train Loss: 0.1405 | Train Acc: 0.9545 | Valid Loss: 0.1701 |
Valid Acc: 0.9143
Epoch: 33/50 | Train Loss: 0.1049 | Train Acc: 0.9633 | Valid Loss: 0.1931 |
Valid Acc: 0.9214
Epoch: 34/50 | Train Loss: 0.1197 | Train Acc: 0.9585 | Valid Loss: 0.1910 |
Valid Acc: 0.9000
```

```
Epoch: 35/50 | Train Loss: 0.0847 | Train Acc: 0.9673 | Valid Loss: 0.1675 |
Valid Acc: 0.9357
Epoch: 36/50 | Train Loss: 0.0945 | Train Acc: 0.9657 | Valid Loss: 0.2036 |
Valid Acc: 0.9000
Epoch: 37/50 | Train Loss: 0.1548 | Train Acc: 0.9561 | Valid Loss: 0.2355 |
Valid Acc: 0.8857
Epoch: 38/50 | Train Loss: 0.1104 | Train Acc: 0.9641 | Valid Loss: 0.1811 |
Valid Acc: 0.9214
Epoch: 39/50 | Train Loss: 0.0889 | Train Acc: 0.9641 | Valid Loss: 0.1633 |
Valid Acc: 0.9143
Epoch: 40/50 | Train Loss: 0.0814 | Train Acc: 0.9681 | Valid Loss: 0.2104 |
Valid Acc: 0.9071
Epoch: 41/50 | Train Loss: 0.1102 | Train Acc: 0.9617 | Valid Loss: 0.1918 |
Valid Acc: 0.9214
Epoch: 42/50 | Train Loss: 0.0929 | Train Acc: 0.9665 | Valid Loss: 0.2235 |
Valid Acc: 0.8714
Epoch: 43/50 | Train Loss: 0.0875 | Train Acc: 0.9681 | Valid Loss: 0.2222 |
Valid Acc: 0.9143
Epoch: 44/50 | Train Loss: 0.1056 | Train Acc: 0.9609 | Valid Loss: 0.1403 |
Valid Acc: 0.9357
Epoch: 45/50 | Train Loss: 0.0921 | Train Acc: 0.9649 | Valid Loss: 0.2027 |
Valid Acc: 0.9000
Epoch: 46/50 | Train Loss: 0.0987 | Train Acc: 0.9649 | Valid Loss: 0.1781 |
Valid Acc: 0.9143
Epoch: 47/50 | Train Loss: 0.1091 | Train Acc: 0.9585 | Valid Loss: 0.1904 |
Valid Acc: 0.9286
Epoch: 48/50 | Train Loss: 0.1115 | Train Acc: 0.9729 | Valid Loss: 0.1624 |
Valid Acc: 0.9286
Epoch: 49/50 | Train Loss: 0.1308 | Train Acc: 0.9561 | Valid Loss: 0.1599 |
Valid Acc: 0.9429
Epoch: 50/50 | Train Loss: 0.1148 | Train Acc: 0.9633 | Valid Loss: 0.1996 |
Valid Acc: 0.8857
```

[ ]:
```python
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
# Plot training and validation loss
plt.plot(range(1, num_epochs+1), train_loss_list, label='Train Loss')
plt.plot(range(1, num_epochs+1), valid_loss_list, label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()

plt.subplot(1, 2, 2)
# Plot training and validation accuracy
plt.plot(range(1, num_epochs+1), train_accuracy_list, label='Train Accuracy')
```

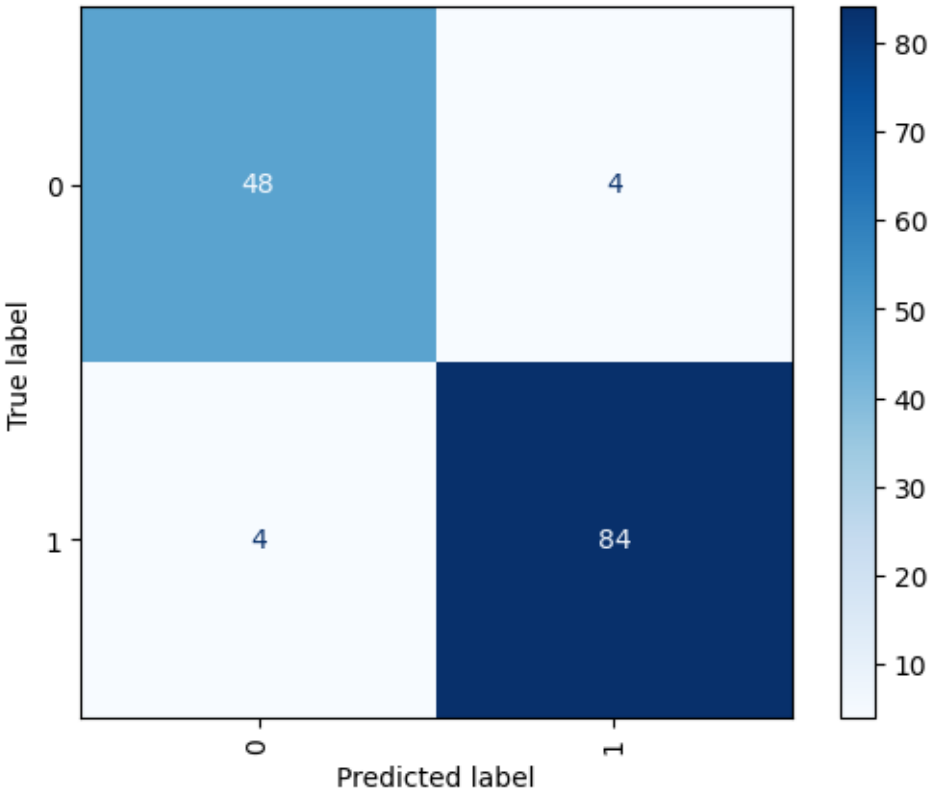
```
plt.plot(range(1, num_epochs+1), valid_accuracy_list, label='Validation␣
 ↪Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()

plt.suptitle(model_name)
plt.show()
```




[ ]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import numpy as np

loaded_model = ResNet101(2)
loaded_model.load_state_dict(torch.load('/content/resnet101_vege.pth'))
loaded_model.to(device)  # Move the loaded model to the same device as the data

# Test the loaded model
def test_loaded_model(model, test_dataloader, device):
    model.eval()  # Set the model to evaluation mode
    all_preds = []
    all_labels = []
    with torch.no_grad():
        for batch in test_dataloader:
            inputs, labels = batch['image'], batch['label']
            inputs = inputs.to(device)
            labels = labels.to(device)
            outputs = model(inputs)
```

```
            _, preds = torch.max(outputs, 1)
            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())
    return all_preds, all_labels

# Test the loaded model
all_preds, all_labels = test_loaded_model(loaded_model, test_loader, device)

# Generate confusion matrix
cm = confusion_matrix(all_labels, all_preds)

# Display the confusion matrix using ConfusionMatrixDisplay
cmd = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=np.
 ↪unique(all_labels))

# Plot the confusion matrix
cmd.plot(cmap='Blues', xticks_rotation='vertical')

# Show the plot
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:557:
UserWarning: This DataLoader will create 32 worker processes in total. Our
suggested max number of worker in current system is 8, which is smaller than
what this DataLoader is going to create. Please be aware that excessive worker
creation might get DataLoader running slow or even freeze, lower the worker
number to avoid potential slowness/freeze if necessary.
  warnings.warn(_create_warning_msg(
```


48
4
4
84
0
1
True label
0
1
Predicted label
80
70
60
50
40
30
20
10