resnet_explore

April 22, 2024

1 Imported Modules

[ ]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from torchvision.models import resnet50, ResNet50_Weights
import torchvision.transforms as T

import os
import math as m
from PIL import Image as I
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix as cm

from ray import train, tune
from ray.tune.schedulers import ASHAScheduler
from ray.train import Checkpoint
from ray.tune.search.optuna import OptunaSearch
```

2 Model Initialization, Layer Replacement, Device Location

[ ]:
```
x = resnet50(weights=ResNet50_Weights.DEFAULT)

for param in x.parameters():
    param.requires_grad = False

x.fc = nn.Linear(2048, 12)
nn.init.kaiming_uniform_(x.fc.weight, nonlinearity='relu')
x.fc.requires_grad = True

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

```
elif getattr(torch, 'has_mps', False):
    device = "mps"
else:
    device = "cpu"

x = x.to(device)
```

```
/var/folders/yj/n5srfxb95115x6hcn4jcx3000000gn/T/ipykernel_53838/4026304798.py:1
2: UserWarning: 'has_mps' is deprecated, please use
'torch.backends.mps.is_built()'
  elif getattr(torch, 'has_mps', False):
```

3 Unnormalization Class

[ ]:
```
class Unnormalize(object):
    def __init__(self, mean=None, std=None):
        self.mean = mean
        self.std = std

    def __call__(self, tensor):
        for t, m, s in zip(tensor, self.mean, self.std):
            t.mul_(s).add_(m)
        return tensor
```

4 Data Loading Function

[ ]:
```
data_labels = {}
unnorm = Unnormalize()

def load_images():
    #Bunch of arrays for storing and moving data
    aux_img = []
    aux_labels = []
    train_imgs = []
    train_labels = []
    test_imgs = []
    test_labels = []
    val_imgs = []
    val_labels = []
    idx = 0

    crop = T.CenterCrop((128, 128))

    #Cycle through various folders for images
    for folder in range(len(sorted(os.listdir('./Data')))):
        if sorted(os.listdir('./Data'))[folder] == '.DS_Store':
```

```
            continue
        else:
            aux_img = os.listdir('./Data/'+sorted(os.listdir('./Data'))[folder])

            for i in range(len(aux_img)):
                #Remove .DS_Store file from data
                if aux_img[i] == '.DS_Store':
                    aux_img.pop(i)
                    break

            #Resize Image and Convert Channels
            for i in range(len(aux_img)):
                try:
                    with I.open('./Data/'+sorted(os.listdir('./
↪Data'))[folder]+'/'+aux_img[i]) as img:
                        img = crop(img)

                        if img.mode != 'RGB':
                            img = img.convert('RGB')

                        img = np.array(img)
                        aux_img[i] = img
                except Exception as e:
                    print(f"{i}")
            aux_labels = []

            #Generate Labels
            for i in range(len(aux_img)):
                aux_labels.append(idx)
            idx += 1

            #Divide Data and Labels into Training, Testing, and Validation Sets
            train_imgs += aux_img[:int(m.ceil(len(aux_img)*0.6))]
            aux_img = aux_img[int(m.ceil(len(aux_img)*0.6)):]
            val_imgs += aux_img[:int(m.ceil(len(aux_img)*0.75))]
            aux_img = aux_img[int(m.ceil(len(aux_img)*0.75)):]
            test_imgs += aux_img

            train_labels += aux_labels[:int(m.ceil(len(aux_labels)*0.6))]
            aux_labels = aux_labels[int(m.ceil(len(aux_labels)*0.6)):]
            val_labels += aux_labels[:int(m.ceil(len(aux_labels)*0.75))]
            aux_labels = aux_labels[int(m.ceil(len(aux_labels)*0.75)):]
            test_labels += aux_labels
        data_labels[folder-1] = sorted(os.listdir('./Data'))[folder]

    #Data Normalization RGB -> [0, 1]
    aux_img = []
```

```
    aux_img = train_imgs + test_imgs + val_imgs
    mu = np.mean(aux_img, axis=(0, 1, 2))
    sigma = np.std(aux_img, axis=(0, 1, 2))
    unnorm.mean = mu
    unnorm.std = sigma

    normalize = T.Compose([
        T.Normalize(mu, sigma)
    ])

    #Conversion to tensors and recast data
    train_imgs, test_imgs, val_imgs = torch.tensor(train_imgs).to(torch.
↪float32), torch.tensor(test_imgs).to(torch.float32), torch.tensor(val_imgs).
↪to(torch.float32)
    train_labels, test_labels, val_labels = torch.tensor(train_labels), torch.
↪tensor(test_labels), torch.tensor(val_labels)

    #Shuffle Images Sets
    train_imgs, train_labels = shuffle_imgs(train_imgs, train_labels)
    test_imgs, test_labels = shuffle_imgs(test_imgs, test_labels)
    val_imgs, val_labels = shuffle_imgs(val_imgs, val_labels)

    #Normalize and Reorder Data for Processing
    train_imgs = normalize(train_imgs.permute(0, 3, 1, 2))
    test_imgs = normalize(test_imgs.permute(0, 3, 1, 2))
    val_imgs = normalize(val_imgs.permute(0, 3, 1, 2))

    train_set, test_set, val_set = TensorDataset(train_imgs, train_labels),␣
↪TensorDataset(test_imgs, test_labels), TensorDataset(val_imgs, val_labels)

    trainloader, testloader, valloader = DataLoader(dataset=train_set,␣
↪shuffle=True, batch_size=128), DataLoader(dataset=test_set, shuffle=True,␣
↪batch_size=64), DataLoader(dataset=val_set, shuffle=True, batch_size=64)

    return trainloader, testloader, valloader

#Shuffle Images according to Random Noise Generation
def shuffle_imgs(imgs, labels):
   B = imgs.size(0)
   shuffle = torch.randperm(B)
   return imgs[shuffle], labels[shuffle]
```

[ ]:
```
trainloader, testloader, valloader = load_images()
print(data_labels)
```

```
/var/folders/yj/n5srfxb951l5x6hcn4jcx3000000gn/T/ipykernel_53838/1063623792.py:7
8: UserWarning: Creating a tensor from a list of numpy.ndarrays is extremely
```

```
slow. Please consider converting the list to a single numpy.ndarray with
numpy.array() before converting to a tensor. (Triggered internally at
/Users/runner/work/pytorch/pytorch/pytorch/torch/csrc/utils/tensor_new.cpp:278.)
  train_imgs, test_imgs, val_imgs = torch.tensor(train_imgs).to(torch.float32),
torch.tensor(test_imgs).to(torch.float32),
torch.tensor(val_imgs).to(torch.float32)
```

```
{0: 'apples', 1: 'coconuts', 2: 'grapes', 3: 'guavas', 4: 'lemons', 5: 'limes',
6: 'mango', 7: 'oranges', 8: 'peaches', 9: 'starfruit', 10: 'strawberries', 11:
'watermelons'}
```

5 Train, Test, and Finetuning Functions

[ ]:
```
#Training Method with Adam Optimizer
def trainfunc(model, traindata, valdata, epochs):
    optimizer = optim.Adam(model.parameters(), lr=0.006349288161821683, eps=6.
 ↪5131448417080975e-09)
    #optimizer = optim.SGD(model.parameters(), lr=0.006349288161821683)
    lr_optim = optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=10)
    loss_list = []
    loss_list_aux = []
    validation_loss = []
    validation_loss_aux = []

    for i in range(epochs):
        tot = 0
        div = 0
        for imgs, labels in traindata:
            imgs = imgs.to(device)
            labels = labels.to(device)
            optimizer.zero_grad()

            logits = model(imgs)
            loss = F.cross_entropy(logits, labels)
            loss_list_aux.append(loss.item())

            tot += (torch.argmax(logits, axis=1) == labels).float().sum().item()
            div += imgs.shape[0]

            loss.backward()
            optimizer.step()
        print('Training Accuracy (epoch ', i, '): ', tot/div*100, '%')
        print('Mean Training Loss: ', torch.mean(torch.tensor(loss_list_aux)))
        loss_list.append(torch.mean(torch.tensor(loss_list_aux)))
        lr_optim.step()

        with torch.no_grad():
```

```
            div = 0
            tot = 0
            for imgs, labels in valdata:
                imgs = imgs.to(device)
                labels = labels.to(device)
                val_logits = model(imgs)
                validation_loss_aux.append(F.cross_entropy(val_logits, labels).
↪item())
                val_logits = nn.Softmax()(val_logits)
                pred = torch.argmax(val_logits, axis=1)
                tot += (pred == labels).float().sum().item()
                div += imgs.shape[0]
            validation_loss.append(torch.mean(torch.
↪tensor(validation_loss_aux)))
            print('Mean Validation Loss: ', torch.mean(torch.
↪tensor(validation_loss_aux)))
            print("Validation Accuracy: ", tot/div*100, "%")

    return model, loss_list, validation_loss

#Finetune Function
def finetune(model, data, valdata, epochs):
    model.fc.requires_grad = False
    model.avgpool.requires_grad = True
    model.layer4.requires_grad = True
    model.layer3.requires_grad = True
    model.layer2.requires_grad = True
    model.layer1.requires_grad = True
    model.maxpool.requires_grad = True
    model.conv1.requires_grad = True

    optimizer = optim.Adam(model.parameters(), lr=0.001)
    lr_sched = optim.lr_scheduler.StepLR(optimizer, step_size=2)
    loss_list = []
    loss_list_aux = []

    for i in range(epochs):
        for imgs, labels in data:
            imgs = imgs.to(device)
            labels = labels.to(device)
            optimizer.zero_grad()

            logits = model(imgs)
            loss = F.cross_entropy(logits, labels)
            loss_list_aux.append(loss.item())

            loss.backward()
```

```
            optimizer.step()

        loss_list.append(torch.mean(torch.tensor(loss_list_aux)))

        div = 0
        tot = 0
        for imgs, labels in valdata:
            imgs = imgs.to(device)
            labels = labels.to(device)
            val_logits = model(imgs)
            val_logits = nn.Softmax()(val_logits)
            pred = torch.argmax(val_logits, axis=1)
            tot += (pred == labels).float().sum().item()
            div += imgs.shape[0]
        print("Validation Accuracy: ", tot/div*100, "%")
        lr_sched.step()
    return model, loss_list

def test(model, data):
    div = 0
    total = 0

    for imgs, labels in data:
        imgs = imgs.to(device)
        labels = labels.to(device)
        logits = model(imgs)
        logits = nn.Softmax()(logits)
        pred = torch.argmax(logits, axis=1)
        div += imgs.shape[0]
        total += (pred == labels).float().sum().item()
    print("Accuracy: ", total/div*100, "%")
```

6 Confusion Matrix Function

[ ]:
```
def confusion_matrix(model, data):
    total_pred = []
    total_labels = []

    for imgs, labels in data:
        imgs = imgs.to(device)
        labels = labels.to(device)

        logits = model(imgs)
        logits = nn.Softmax()(logits)
        pred = torch.argmax(logits, axis=1)
```

```
        total_pred += list(pred.to('cpu'))
        total_labels += list(labels.to('cpu'))

    con_mat = cm(total_labels, total_pred)
    labels = [data_labels[i] for i in range(len(con_mat))]
    plt.figure(figsize=(10, 8))
    ax = sns.heatmap(con_mat, annot=True, cmap="inferno", xticklabels=labels,␣
 ↪yticklabels=labels)
    plt.title(f'ResNet50 Accuracy')
    plt.ylabel('True Labels')
    plt.xlabel('Predicted Labels')
    plt.show()
    plt.clf()
```

7 Hyperparameter Tuning Functions

[ ]:
```
def hfinetunetrain(model, data, optimizer, lr_sched):
    for imgs, labels in data:
        imgs, labels = imgs.to(device), labels.to(device)

        pred = model(imgs)
        optimizer.zero_grad()
        loss = F.cross_entropy(pred, labels)
        loss.backward()
        optimizer.step()
        lr_sched.step()
    if device == "cuda":
        torch.cuda.empty_cache()
    elif device == "mps":
        torch.mps.empty_cache()

def hfinetunetest(model, data):
    model.eval()
    correct = 0
    total = 0

    for imgs, labels in data:
        imgs, labels = imgs.to(device), labels.to(device)

        outputs = model(imgs)

        pred = torch.argmax(outputs, axis=1)

        total += imgs.shape[0]
        correct += (pred == labels).float().sum().item()
    if device == "cuda":
```

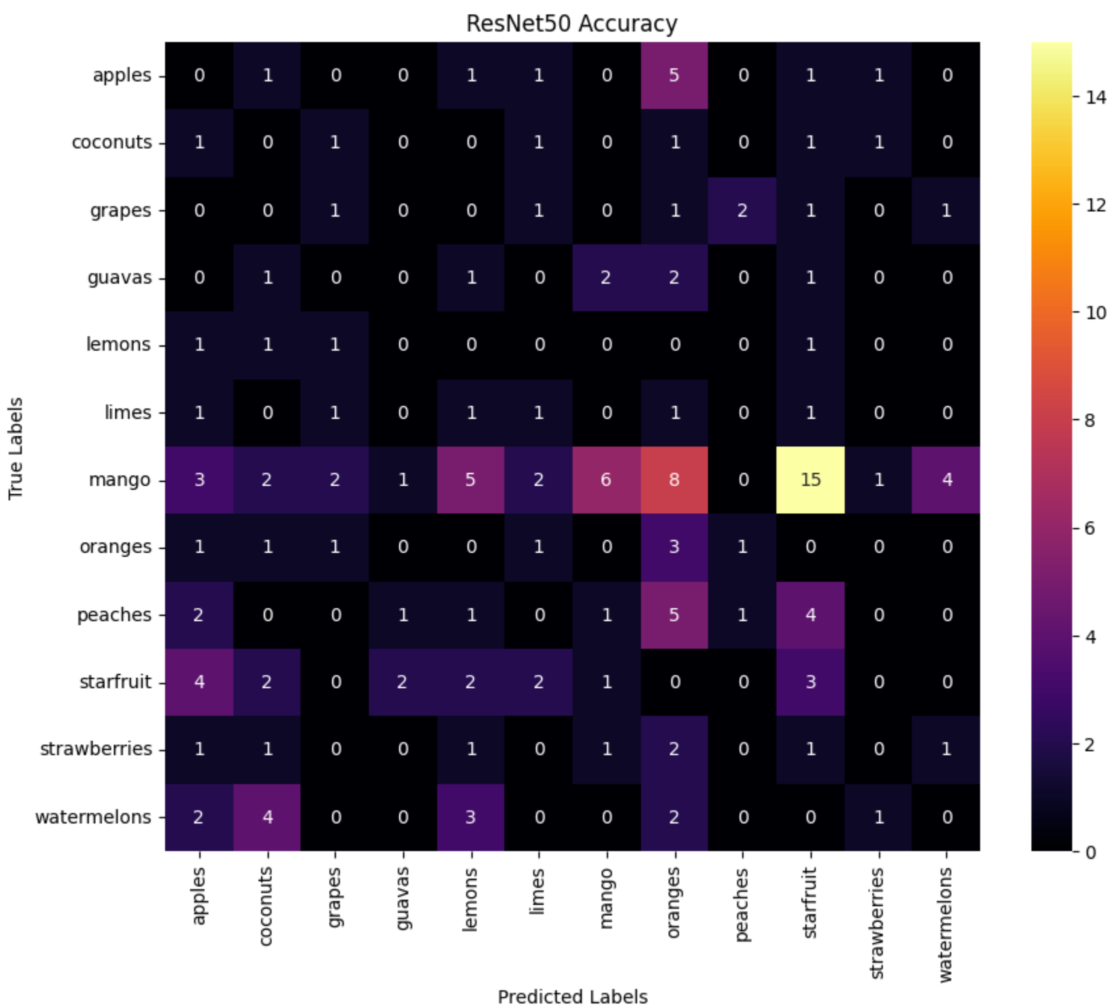
```
        torch.cuda.empty_cache()
    elif device == "mps":
        torch.mps.empty_cache()
    return correct/total

def objective(config):
    train_loader, test_loader, val_loader = load_images()

    model = resnet50(weights=ResNet50_Weights.DEFAULT).to(device)
    for param in x.parameters():
        param.requires_grad = False

    model.fc = nn.Sequential(nn.Linear(2048, config["l1"]),
                                    nn.ReLU(),
                                    nn.Linear(config["l1"], config["l2"]),
                                    nn.ReLU(),
                                    nn.Linear(config["l2"], 12)
                                    ).to(device)
    nn.init.kaiming_uniform_(x.fc.weight, nonlinearity='relu')
    x.fc.requires_grad = True

    optimizer = optim.Adam(model.parameters(), lr=config["lr"],␣
 ↪eps=config["eps"])
    lr_sched = optim.lr_scheduler.StepLR(optimizer,␣
 ↪step_size=config["lr_sched_step"])

    for epoch in range(10):
        hfinetunetrain(model, train_loader, optimizer, lr_sched)
        acc = hfinetunetest(model, val_loader)

        checkpoint = None
        if (epoch+1)%5 == 0:
            torch.save(model.state_dict(),
                       "Users/thomassigler/DeepLearningProject/
 ↪TuningCheckpoints/model.pth")
            checkpoint = Checkpoint.from_directory("Users/thomassigler/
 ↪DeepLearningProject/TuningCheckpoints")
        train.report({"mean_accuracy": acc}, checkpoint=checkpoint)

        if device == "cuda":
            torch.cuda.empty_cache()
        elif device == "mps":
            torch.mps.empty_cache()
```

8 Model Benchmark with Replaced Layer

[ ]:
```
test(x, testloader)

confusion_matrix(x, testloader)
```

```
/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-
packages/torch/nn/modules/module.py:1511: UserWarning: Implicit dimension choice
for softmax has been deprecated. Change the call to include dim=X as an
argument.
  return self._call_impl(*args, **kwargs)
```

```
Accuracy:  8.108108108108109 %
```




```
<Figure size 640x480 with 0 Axes>
```

9 Training ResNet50 FC Layer

[ ]:
```
x, loss_list, val_loss = trainfunc(x, trainloader, valloader, 200)
test(x, testloader)

confusion_matrix(x, testloader)
```

```
Training Accuracy (epoch  0 ):  36.67745415318231 %
Mean Training Loss:  tensor(1.8886)
```

```
/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-
packages/torch/nn/modules/module.py:1511: UserWarning: Implicit dimension choice
for softmax has been deprecated. Change the call to include dim=X as an
argument.
  return self._call_impl(*args, **kwargs)
```

```
Mean Validation Loss:  tensor(1.2270)
Validation Accuracy:  64.64208242950107 %
Training Accuracy (epoch  1 ):  84.46601941747572 %
Mean Training Loss:  tensor(1.3071)
Mean Validation Loss:  tensor(1.0424)
Validation Accuracy:  74.62039045553145 %
Training Accuracy (epoch  2 ):  91.8015102481122 %
Mean Training Loss:  tensor(0.9962)
Mean Validation Loss:  tensor(0.9315)
Validation Accuracy:  77.87418655097615 %
Training Accuracy (epoch  3 ):  96.87162891046385 %
Mean Training Loss:  tensor(0.8044)
Mean Validation Loss:  tensor(0.8995)
Validation Accuracy:  79.39262472885032 %
Training Accuracy (epoch  4 ):  98.70550161812298 %
Mean Training Loss:  tensor(0.6768)
Mean Validation Loss:  tensor(0.8583)
Validation Accuracy:  80.4772234273319 %
Training Accuracy (epoch  5 ):  99.67637540453075 %
Mean Training Loss:  tensor(0.5845)
Mean Validation Loss:  tensor(0.8309)
Validation Accuracy:  78.52494577006507 %
Training Accuracy (epoch  6 ):  100.0 %
Mean Training Loss:  tensor(0.5173)
Mean Validation Loss:  tensor(0.8053)
Validation Accuracy:  79.82646420824295 %
Training Accuracy (epoch  7 ):  99.78425026968716 %
Mean Training Loss:  tensor(0.4655)
Mean Validation Loss:  tensor(0.7808)
Validation Accuracy:  81.77874186550976 %
Training Accuracy (epoch  8 ):  100.0 %
Mean Training Loss:  tensor(0.4249)
Mean Validation Loss:  tensor(0.7626)
```

```
Validation Accuracy:  81.34490238611714 %
Training Accuracy (epoch  9 ):  99.89212513484358 %
Mean Training Loss:  tensor(0.3923)
Mean Validation Loss:  tensor(0.7467)
Validation Accuracy:  80.26030368763557 %
Training Accuracy (epoch  10 ):  99.89212513484358 %
Mean Training Loss:  tensor(0.3654)
Mean Validation Loss:  tensor(0.7372)
Validation Accuracy:  79.39262472885032 %
Training Accuracy (epoch  11 ):  99.46062567421791 %
Mean Training Loss:  tensor(0.3435)
Mean Validation Loss:  tensor(0.7248)
Validation Accuracy:  81.12798264642083 %
Training Accuracy (epoch  12 ):  99.67637540453075 %
Mean Training Loss:  tensor(0.3246)
Mean Validation Loss:  tensor(0.7135)
Validation Accuracy:  80.26030368763557 %
Training Accuracy (epoch  13 ):  100.0 %
Mean Training Loss:  tensor(0.3082)
Mean Validation Loss:  tensor(0.7070)
Validation Accuracy:  82.86334056399133 %
Training Accuracy (epoch  14 ):  100.0 %
Mean Training Loss:  tensor(0.2936)
Mean Validation Loss:  tensor(0.7053)
Validation Accuracy:  81.56182212581345 %
Training Accuracy (epoch  15 ):  99.56850053937433 %
Mean Training Loss:  tensor(0.2820)
Mean Validation Loss:  tensor(0.7036)
Validation Accuracy:  81.12798264642083 %
Training Accuracy (epoch  16 ):  99.56850053937433 %
Mean Training Loss:  tensor(0.2707)
Mean Validation Loss:  tensor(0.6995)
Validation Accuracy:  80.26030368763557 %
Training Accuracy (epoch  17 ):  99.89212513484358 %
Mean Training Loss:  tensor(0.2595)
Mean Validation Loss:  tensor(0.6935)
Validation Accuracy:  82.86334056399133 %
Training Accuracy (epoch  18 ):  100.0 %
Mean Training Loss:  tensor(0.2485)
Mean Validation Loss:  tensor(0.6903)
Validation Accuracy:  82.21258134490239 %
Training Accuracy (epoch  19 ):  100.0 %
Mean Training Loss:  tensor(0.2384)
Mean Validation Loss:  tensor(0.6895)
Validation Accuracy:  81.12798264642083 %
Training Accuracy (epoch  20 ):  100.0 %
Mean Training Loss:  tensor(0.2289)
Mean Validation Loss:  tensor(0.6876)
```

```
Validation Accuracy:  82.21258134490239 %
Training Accuracy (epoch  21 ):  100.0 %
Mean Training Loss:  tensor(0.2202)
Mean Validation Loss:  tensor(0.6836)
Validation Accuracy:  83.29718004338395 %
Training Accuracy (epoch  22 ):  100.0 %
Mean Training Loss:  tensor(0.2121)
Mean Validation Loss:  tensor(0.6791)
Validation Accuracy:  82.86334056399133 %
Training Accuracy (epoch  23 ):  100.0 %
Mean Training Loss:  tensor(0.2045)
Mean Validation Loss:  tensor(0.6757)
Validation Accuracy:  82.646420824295 %
Training Accuracy (epoch  24 ):  100.0 %
Mean Training Loss:  tensor(0.1973)
Mean Validation Loss:  tensor(0.6728)
Validation Accuracy:  83.29718004338395 %
Training Accuracy (epoch  25 ):  100.0 %
Mean Training Loss:  tensor(0.1908)
Mean Validation Loss:  tensor(0.6683)
Validation Accuracy:  82.4295010845987 %
Training Accuracy (epoch  26 ):  99.89212513484358 %
Mean Training Loss:  tensor(0.1850)
Mean Validation Loss:  tensor(0.6650)
Validation Accuracy:  83.08026030368764 %
Training Accuracy (epoch  27 ):  100.0 %
Mean Training Loss:  tensor(0.1793)
Mean Validation Loss:  tensor(0.6602)
Validation Accuracy:  84.3817787418655 %
Training Accuracy (epoch  28 ):  100.0 %
Mean Training Loss:  tensor(0.1738)
Mean Validation Loss:  tensor(0.6602)
Validation Accuracy:  82.86334056399133 %
Training Accuracy (epoch  29 ):  100.0 %
Mean Training Loss:  tensor(0.1689)
Mean Validation Loss:  tensor(0.6595)
Validation Accuracy:  82.86334056399133 %
Training Accuracy (epoch  30 ):  100.0 %
Mean Training Loss:  tensor(0.1643)
Mean Validation Loss:  tensor(0.6604)
Validation Accuracy:  82.4295010845987 %
Training Accuracy (epoch  31 ):  100.0 %
Mean Training Loss:  tensor(0.1599)
Mean Validation Loss:  tensor(0.6596)
Validation Accuracy:  84.3817787418655 %
Training Accuracy (epoch  32 ):  100.0 %
Mean Training Loss:  tensor(0.1557)
Mean Validation Loss:  tensor(0.6573)
```

```
Validation Accuracy:  82.4295010845987 %
Training Accuracy (epoch  33 ):  100.0 %
Mean Training Loss:  tensor(0.1517)
Mean Validation Loss:  tensor(0.6552)
Validation Accuracy:  84.1648590021692 %
Training Accuracy (epoch  34 ):  100.0 %
Mean Training Loss:  tensor(0.1480)
Mean Validation Loss:  tensor(0.6536)
Validation Accuracy:  82.21258134490239 %
Training Accuracy (epoch  35 ):  100.0 %
Mean Training Loss:  tensor(0.1445)
Mean Validation Loss:  tensor(0.6506)
Validation Accuracy:  83.08026030368764 %
Training Accuracy (epoch  36 ):  100.0 %
Mean Training Loss:  tensor(0.1412)
Mean Validation Loss:  tensor(0.6496)
Validation Accuracy:  82.4295010845987 %
Training Accuracy (epoch  37 ):  99.78425026968716 %
Mean Training Loss:  tensor(0.1385)
Mean Validation Loss:  tensor(0.6486)
Validation Accuracy:  84.81561822125813 %
Training Accuracy (epoch  38 ):  100.0 %
Mean Training Loss:  tensor(0.1355)
Mean Validation Loss:  tensor(0.6483)
Validation Accuracy:  81.56182212581345 %
Training Accuracy (epoch  39 ):  100.0 %
Mean Training Loss:  tensor(0.1327)
Mean Validation Loss:  tensor(0.6478)
Validation Accuracy:  82.21258134490239 %
Training Accuracy (epoch  40 ):  99.89212513484358 %
Mean Training Loss:  tensor(0.1300)
Mean Validation Loss:  tensor(0.6466)
Validation Accuracy:  81.34490238611714 %
Training Accuracy (epoch  41 ):  100.0 %
Mean Training Loss:  tensor(0.1275)
Mean Validation Loss:  tensor(0.6443)
Validation Accuracy:  82.21258134490239 %
Training Accuracy (epoch  42 ):  100.0 %
Mean Training Loss:  tensor(0.1249)
Mean Validation Loss:  tensor(0.6418)
Validation Accuracy:  83.29718004338395 %
Training Accuracy (epoch  43 ):  100.0 %
Mean Training Loss:  tensor(0.1224)
Mean Validation Loss:  tensor(0.6395)
Validation Accuracy:  83.29718004338395 %
Training Accuracy (epoch  44 ):  100.0 %
Mean Training Loss:  tensor(0.1200)
Mean Validation Loss:  tensor(0.6374)
```

```
Validation Accuracy:  82.646420824295 %
Training Accuracy (epoch  45 ):  100.0 %
Mean Training Loss:  tensor(0.1177)
Mean Validation Loss:  tensor(0.6370)
Validation Accuracy:  83.51409978308027 %
Training Accuracy (epoch  46 ):  100.0 %
Mean Training Loss:  tensor(0.1155)
Mean Validation Loss:  tensor(0.6394)
Validation Accuracy:  82.86334056399133 %
Training Accuracy (epoch  47 ):  100.0 %
Mean Training Loss:  tensor(0.1134)
Mean Validation Loss:  tensor(0.6373)
Validation Accuracy:  83.73101952277658 %
Training Accuracy (epoch  48 ):  100.0 %
Mean Training Loss:  tensor(0.1112)
Mean Validation Loss:  tensor(0.6381)
Validation Accuracy:  81.99566160520607 %
Training Accuracy (epoch  49 ):  100.0 %
Mean Training Loss:  tensor(0.1092)
Mean Validation Loss:  tensor(0.6368)
Validation Accuracy:  82.86334056399133 %
Training Accuracy (epoch  50 ):  99.89212513484358 %
Mean Training Loss:  tensor(0.1075)
Mean Validation Loss:  tensor(0.6358)
Validation Accuracy:  82.646420824295 %
Training Accuracy (epoch  51 ):  100.0 %
Mean Training Loss:  tensor(0.1056)
Mean Validation Loss:  tensor(0.6356)
Validation Accuracy:  83.29718004338395 %
Training Accuracy (epoch  52 ):  100.0 %
Mean Training Loss:  tensor(0.1039)
Mean Validation Loss:  tensor(0.6348)
Validation Accuracy:  83.08026030368764 %
Training Accuracy (epoch  53 ):  100.0 %
Mean Training Loss:  tensor(0.1022)
Mean Validation Loss:  tensor(0.6385)
Validation Accuracy:  82.21258134490239 %
Training Accuracy (epoch  54 ):  100.0 %
Mean Training Loss:  tensor(0.1005)
Mean Validation Loss:  tensor(0.6371)
Validation Accuracy:  81.34490238611714 %
Training Accuracy (epoch  55 ):  100.0 %
Mean Training Loss:  tensor(0.0989)
Mean Validation Loss:  tensor(0.6389)
Validation Accuracy:  81.56182212581345 %
Training Accuracy (epoch  56 ):  100.0 %
Mean Training Loss:  tensor(0.0974)
Mean Validation Loss:  tensor(0.6390)
```

```
Validation Accuracy:  82.646420824295 %
Training Accuracy (epoch  57 ):  99.89212513484358 %
Mean Training Loss:  tensor(0.0960)
Mean Validation Loss:  tensor(0.6382)
Validation Accuracy:  80.91106290672451 %
Training Accuracy (epoch  58 ):  100.0 %
Mean Training Loss:  tensor(0.0946)
Mean Validation Loss:  tensor(0.6374)
Validation Accuracy:  83.73101952277658 %
Training Accuracy (epoch  59 ):  99.89212513484358 %
Mean Training Loss:  tensor(0.0932)
Mean Validation Loss:  tensor(0.6372)
Validation Accuracy:  82.21258134490239 %
Training Accuracy (epoch  60 ):  100.0 %
Mean Training Loss:  tensor(0.0919)
Mean Validation Loss:  tensor(0.6378)
Validation Accuracy:  82.86334056399133 %
Training Accuracy (epoch  61 ):  99.89212513484358 %
Mean Training Loss:  tensor(0.0906)
Mean Validation Loss:  tensor(0.6363)
Validation Accuracy:  84.81561822125813 %
Training Accuracy (epoch  62 ):  100.0 %
Mean Training Loss:  tensor(0.0894)
Mean Validation Loss:  tensor(0.6356)
Validation Accuracy:  82.646420824295 %
Training Accuracy (epoch  63 ):  100.0 %
Mean Training Loss:  tensor(0.0881)
Mean Validation Loss:  tensor(0.6351)
Validation Accuracy:  83.08026030368764 %
Training Accuracy (epoch  64 ):  100.0 %
Mean Training Loss:  tensor(0.0869)
Mean Validation Loss:  tensor(0.6356)
Validation Accuracy:  83.08026030368764 %
Training Accuracy (epoch  65 ):  100.0 %
Mean Training Loss:  tensor(0.0857)
Mean Validation Loss:  tensor(0.6346)
Validation Accuracy:  81.99566160520607 %
Training Accuracy (epoch  66 ):  100.0 %
Mean Training Loss:  tensor(0.0847)
Mean Validation Loss:  tensor(0.6347)
Validation Accuracy:  82.4295010845987 %
Training Accuracy (epoch  67 ):  100.0 %
Mean Training Loss:  tensor(0.0836)
Mean Validation Loss:  tensor(0.6341)
Validation Accuracy:  83.73101952277658 %
Training Accuracy (epoch  68 ):  100.0 %
Mean Training Loss:  tensor(0.0825)
Mean Validation Loss:  tensor(0.6347)
```

```
Validation Accuracy:  83.73101952277658 %
Training Accuracy (epoch  69 ):  100.0 %
Mean Training Loss:  tensor(0.0815)
Mean Validation Loss:  tensor(0.6339)
Validation Accuracy:  82.86334056399133 %
Training Accuracy (epoch  70 ):  100.0 %
Mean Training Loss:  tensor(0.0804)
Mean Validation Loss:  tensor(0.6341)
Validation Accuracy:  83.08026030368764 %
Training Accuracy (epoch  71 ):  100.0 %
Mean Training Loss:  tensor(0.0794)
Mean Validation Loss:  tensor(0.6336)
Validation Accuracy:  83.08026030368764 %
Training Accuracy (epoch  72 ):  100.0 %
Mean Training Loss:  tensor(0.0785)
Mean Validation Loss:  tensor(0.6335)
Validation Accuracy:  81.12798264642083 %
Training Accuracy (epoch  73 ):  100.0 %
Mean Training Loss:  tensor(0.0775)
Mean Validation Loss:  tensor(0.6329)
Validation Accuracy:  81.99566160520607 %
Training Accuracy (epoch  74 ):  100.0 %
Mean Training Loss:  tensor(0.0766)
Mean Validation Loss:  tensor(0.6323)
Validation Accuracy:  82.86334056399133 %
Training Accuracy (epoch  75 ):  100.0 %
Mean Training Loss:  tensor(0.0756)
Mean Validation Loss:  tensor(0.6320)
Validation Accuracy:  83.29718004338395 %
Training Accuracy (epoch  76 ):  100.0 %
Mean Training Loss:  tensor(0.0748)
Mean Validation Loss:  tensor(0.6319)
Validation Accuracy:  82.4295010845987 %
Training Accuracy (epoch  77 ):  100.0 %
Mean Training Loss:  tensor(0.0739)
Mean Validation Loss:  tensor(0.6314)
Validation Accuracy:  82.86334056399133 %
Training Accuracy (epoch  78 ):  100.0 %
Mean Training Loss:  tensor(0.0731)
Mean Validation Loss:  tensor(0.6314)
Validation Accuracy:  83.29718004338395 %
Training Accuracy (epoch  79 ):  100.0 %
Mean Training Loss:  tensor(0.0723)
Mean Validation Loss:  tensor(0.6312)
Validation Accuracy:  83.51409978308027 %
Training Accuracy (epoch  80 ):  100.0 %
Mean Training Loss:  tensor(0.0715)
Mean Validation Loss:  tensor(0.6314)
```

```
Validation Accuracy:  83.08026030368764 %
Training Accuracy (epoch  81 ):  100.0 %
Mean Training Loss:  tensor(0.0707)
Mean Validation Loss:  tensor(0.6304)
Validation Accuracy:  83.29718004338395 %
Training Accuracy (epoch  82 ):  100.0 %
Mean Training Loss:  tensor(0.0699)
Mean Validation Loss:  tensor(0.6302)
Validation Accuracy:  82.4295010845987 %
Training Accuracy (epoch  83 ):  100.0 %
Mean Training Loss:  tensor(0.0692)
Mean Validation Loss:  tensor(0.6306)
Validation Accuracy:  83.08026030368764 %
Training Accuracy (epoch  84 ):  100.0 %
Mean Training Loss:  tensor(0.0685)
Mean Validation Loss:  tensor(0.6305)
Validation Accuracy:  84.1648590021692 %
Training Accuracy (epoch  85 ):  100.0 %
Mean Training Loss:  tensor(0.0678)
Mean Validation Loss:  tensor(0.6305)
Validation Accuracy:  82.646420824295 %
Training Accuracy (epoch  86 ):  100.0 %
Mean Training Loss:  tensor(0.0670)
Mean Validation Loss:  tensor(0.6309)
Validation Accuracy:  83.73101952277658 %
Training Accuracy (epoch  87 ):  100.0 %
Mean Training Loss:  tensor(0.0664)
Mean Validation Loss:  tensor(0.6304)
Validation Accuracy:  83.94793926247289 %
Training Accuracy (epoch  88 ):  100.0 %
Mean Training Loss:  tensor(0.0657)
Mean Validation Loss:  tensor(0.6298)
Validation Accuracy:  82.86334056399133 %
Training Accuracy (epoch  89 ):  100.0 %
Mean Training Loss:  tensor(0.0650)
Mean Validation Loss:  tensor(0.6288)
Validation Accuracy:  83.51409978308027 %
Training Accuracy (epoch  90 ):  100.0 %
Mean Training Loss:  tensor(0.0644)
Mean Validation Loss:  tensor(0.6290)
Validation Accuracy:  82.646420824295 %
Training Accuracy (epoch  91 ):  100.0 %
Mean Training Loss:  tensor(0.0637)
Mean Validation Loss:  tensor(0.6276)
Validation Accuracy:  82.646420824295 %
Training Accuracy (epoch  92 ):  100.0 %
Mean Training Loss:  tensor(0.0631)
Mean Validation Loss:  tensor(0.6278)
```

```
Validation Accuracy:  82.86334056399133 %
Training Accuracy (epoch  93 ):  100.0 %
Mean Training Loss:  tensor(0.0625)
Mean Validation Loss:  tensor(0.6275)
Validation Accuracy:  84.1648590021692 %
Training Accuracy (epoch  94 ):  100.0 %
Mean Training Loss:  tensor(0.0619)
Mean Validation Loss:  tensor(0.6272)
Validation Accuracy:  82.21258134490239 %
Training Accuracy (epoch  95 ):  100.0 %
Mean Training Loss:  tensor(0.0613)
Mean Validation Loss:  tensor(0.6260)
Validation Accuracy:  83.29718004338395 %
Training Accuracy (epoch  96 ):  99.78425026968716 %
Mean Training Loss:  tensor(0.0608)
Mean Validation Loss:  tensor(0.6254)
Validation Accuracy:  84.81561822125813 %
Training Accuracy (epoch  97 ):  100.0 %
Mean Training Loss:  tensor(0.0602)
Mean Validation Loss:  tensor(0.6249)
Validation Accuracy:  82.86334056399133 %
Training Accuracy (epoch  98 ):  100.0 %
Mean Training Loss:  tensor(0.0597)
Mean Validation Loss:  tensor(0.6260)
Validation Accuracy:  81.56182212581345 %
Training Accuracy (epoch  99 ):  100.0 %
Mean Training Loss:  tensor(0.0592)
Mean Validation Loss:  tensor(0.6261)
Validation Accuracy:  81.99566160520607 %
Training Accuracy (epoch  100 ):  100.0 %
Mean Training Loss:  tensor(0.0586)
Mean Validation Loss:  tensor(0.6264)
Validation Accuracy:  82.646420824295 %
Training Accuracy (epoch  101 ):  100.0 %
Mean Training Loss:  tensor(0.0581)
Mean Validation Loss:  tensor(0.6263)
Validation Accuracy:  82.86334056399133 %
Training Accuracy (epoch  102 ):  100.0 %
Mean Training Loss:  tensor(0.0576)
Mean Validation Loss:  tensor(0.6261)
Validation Accuracy:  81.99566160520607 %
Training Accuracy (epoch  103 ):  100.0 %
Mean Training Loss:  tensor(0.0570)
Mean Validation Loss:  tensor(0.6261)
Validation Accuracy:  81.99566160520607 %
Training Accuracy (epoch  104 ):  100.0 %
Mean Training Loss:  tensor(0.0565)
Mean Validation Loss:  tensor(0.6258)
```

```
Validation Accuracy:  81.99566160520607 %
Training Accuracy (epoch  105 ):  100.0 %
Mean Training Loss:  tensor(0.0560)
Mean Validation Loss:  tensor(0.6254)
Validation Accuracy:  82.86334056399133 %
Training Accuracy (epoch  106 ):  100.0 %
Mean Training Loss:  tensor(0.0556)
Mean Validation Loss:  tensor(0.6252)
Validation Accuracy:  83.08026030368764 %
Training Accuracy (epoch  107 ):  100.0 %
Mean Training Loss:  tensor(0.0551)
Mean Validation Loss:  tensor(0.6270)
Validation Accuracy:  82.86334056399133 %
Training Accuracy (epoch  108 ):  100.0 %
Mean Training Loss:  tensor(0.0546)
Mean Validation Loss:  tensor(0.6265)
Validation Accuracy:  83.51409978308027 %
Training Accuracy (epoch  109 ):  100.0 %
Mean Training Loss:  tensor(0.0541)
Mean Validation Loss:  tensor(0.6266)
Validation Accuracy:  83.51409978308027 %
Training Accuracy (epoch  110 ):  100.0 %
Mean Training Loss:  tensor(0.0537)
Mean Validation Loss:  tensor(0.6264)
Validation Accuracy:  84.1648590021692 %
Training Accuracy (epoch  111 ):  100.0 %
Mean Training Loss:  tensor(0.0533)
Mean Validation Loss:  tensor(0.6255)
Validation Accuracy:  84.81561822125813 %
Training Accuracy (epoch  112 ):  100.0 %
Mean Training Loss:  tensor(0.0528)
Mean Validation Loss:  tensor(0.6261)
Validation Accuracy:  83.51409978308027 %
Training Accuracy (epoch  113 ):  100.0 %
Mean Training Loss:  tensor(0.0524)
Mean Validation Loss:  tensor(0.6259)
Validation Accuracy:  82.21258134490239 %
Training Accuracy (epoch  114 ):  100.0 %
Mean Training Loss:  tensor(0.0520)
Mean Validation Loss:  tensor(0.6256)
Validation Accuracy:  83.51409978308027 %
Training Accuracy (epoch  115 ):  100.0 %
Mean Training Loss:  tensor(0.0516)
Mean Validation Loss:  tensor(0.6257)
Validation Accuracy:  82.4295010845987 %
Training Accuracy (epoch  116 ):  100.0 %
Mean Training Loss:  tensor(0.0512)
Mean Validation Loss:  tensor(0.6253)
```

```
Validation Accuracy:  83.51409978308027 %
Training Accuracy (epoch  117 ):  100.0 %
Mean Training Loss:  tensor(0.0508)
Mean Validation Loss:  tensor(0.6253)
Validation Accuracy:  82.21258134490239 %
Training Accuracy (epoch  118 ):  100.0 %
Mean Training Loss:  tensor(0.0504)
Mean Validation Loss:  tensor(0.6251)
Validation Accuracy:  81.99566160520607 %
Training Accuracy (epoch  119 ):  99.89212513484358 %
Mean Training Loss:  tensor(0.0501)
Mean Validation Loss:  tensor(0.6254)
Validation Accuracy:  83.08026030368764 %
Training Accuracy (epoch  120 ):  100.0 %
Mean Training Loss:  tensor(0.0497)
Mean Validation Loss:  tensor(0.6264)
Validation Accuracy:  80.26030368763557 %
Training Accuracy (epoch  121 ):  100.0 %
Mean Training Loss:  tensor(0.0493)
Mean Validation Loss:  tensor(0.6266)
Validation Accuracy:  82.4295010845987 %
Training Accuracy (epoch  122 ):  100.0 %
Mean Training Loss:  tensor(0.0490)
Mean Validation Loss:  tensor(0.6266)
Validation Accuracy:  82.646420824295 %
Training Accuracy (epoch  123 ):  100.0 %
Mean Training Loss:  tensor(0.0486)
Mean Validation Loss:  tensor(0.6266)
Validation Accuracy:  82.646420824295 %
Training Accuracy (epoch  124 ):  100.0 %
Mean Training Loss:  tensor(0.0482)
Mean Validation Loss:  tensor(0.6265)
Validation Accuracy:  82.646420824295 %
Training Accuracy (epoch  125 ):  100.0 %
Mean Training Loss:  tensor(0.0479)
Mean Validation Loss:  tensor(0.6265)
Validation Accuracy:  83.29718004338395 %
Training Accuracy (epoch  126 ):  100.0 %
Mean Training Loss:  tensor(0.0475)
Mean Validation Loss:  tensor(0.6261)
Validation Accuracy:  82.646420824295 %
Training Accuracy (epoch  127 ):  100.0 %
Mean Training Loss:  tensor(0.0472)
Mean Validation Loss:  tensor(0.6263)
Validation Accuracy:  82.86334056399133 %
Training Accuracy (epoch  128 ):  100.0 %
Mean Training Loss:  tensor(0.0469)
Mean Validation Loss:  tensor(0.6258)
```

```
Validation Accuracy:  84.3817787418655 %
Training Accuracy (epoch  129 ):  100.0 %
Mean Training Loss:  tensor(0.0465)
Mean Validation Loss:  tensor(0.6258)
Validation Accuracy:  82.646420824295 %
Training Accuracy (epoch  130 ):  100.0 %
Mean Training Loss:  tensor(0.0462)
Mean Validation Loss:  tensor(0.6255)
Validation Accuracy:  82.86334056399133 %
Training Accuracy (epoch  131 ):  100.0 %
Mean Training Loss:  tensor(0.0459)
Mean Validation Loss:  tensor(0.6257)
Validation Accuracy:  81.99566160520607 %
Training Accuracy (epoch  132 ):  100.0 %
Mean Training Loss:  tensor(0.0456)
Mean Validation Loss:  tensor(0.6254)
Validation Accuracy:  84.59869848156181 %
Training Accuracy (epoch  133 ):  100.0 %
Mean Training Loss:  tensor(0.0453)
Mean Validation Loss:  tensor(0.6254)
Validation Accuracy:  83.29718004338395 %
Training Accuracy (epoch  134 ):  100.0 %
Mean Training Loss:  tensor(0.0450)
Mean Validation Loss:  tensor(0.6252)
Validation Accuracy:  83.73101952277658 %
Training Accuracy (epoch  135 ):  100.0 %
Mean Training Loss:  tensor(0.0446)
Mean Validation Loss:  tensor(0.6258)
Validation Accuracy:  83.51409978308027 %
Training Accuracy (epoch  136 ):  100.0 %
Mean Training Loss:  tensor(0.0443)
Mean Validation Loss:  tensor(0.6257)
Validation Accuracy:  83.29718004338395 %
Training Accuracy (epoch  137 ):  100.0 %
Mean Training Loss:  tensor(0.0440)
Mean Validation Loss:  tensor(0.6255)
Validation Accuracy:  82.646420824295 %
Training Accuracy (epoch  138 ):  100.0 %
Mean Training Loss:  tensor(0.0437)
Mean Validation Loss:  tensor(0.6253)
Validation Accuracy:  84.1648590021692 %
Training Accuracy (epoch  139 ):  100.0 %
Mean Training Loss:  tensor(0.0435)
Mean Validation Loss:  tensor(0.6252)
Validation Accuracy:  83.29718004338395 %
Training Accuracy (epoch  140 ):  100.0 %
Mean Training Loss:  tensor(0.0432)
Mean Validation Loss:  tensor(0.6251)
```

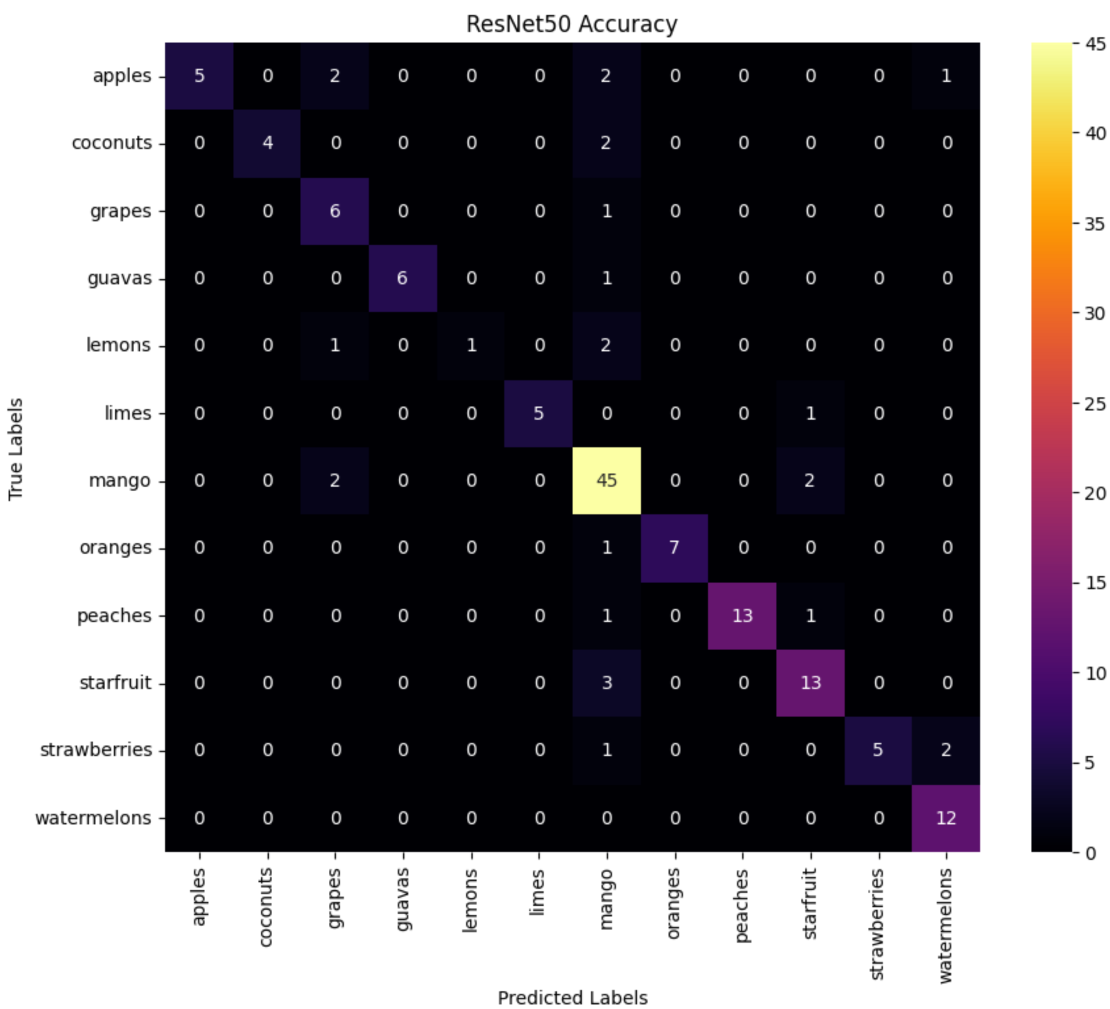
```
Validation Accuracy:  84.1648590021692 %
Training Accuracy (epoch  141 ):  100.0 %
Mean Training Loss:  tensor(0.0429)
Mean Validation Loss:  tensor(0.6247)
Validation Accuracy:  81.99566160520607 %
Training Accuracy (epoch  142 ):  100.0 %
Mean Training Loss:  tensor(0.0426)
Mean Validation Loss:  tensor(0.6250)
Validation Accuracy:  81.77874186550976 %
Training Accuracy (epoch  143 ):  100.0 %
Mean Training Loss:  tensor(0.0423)
Mean Validation Loss:  tensor(0.6251)
Validation Accuracy:  80.91106290672451 %
Training Accuracy (epoch  144 ):  100.0 %
Mean Training Loss:  tensor(0.0421)
Mean Validation Loss:  tensor(0.6257)
Validation Accuracy:  83.94793926247289 %
Training Accuracy (epoch  145 ):  100.0 %
Mean Training Loss:  tensor(0.0418)
Mean Validation Loss:  tensor(0.6255)
Validation Accuracy:  83.51409978308027 %
Training Accuracy (epoch  146 ):  100.0 %
Mean Training Loss:  tensor(0.0416)
Mean Validation Loss:  tensor(0.6266)
Validation Accuracy:  81.77874186550976 %
Training Accuracy (epoch  147 ):  100.0 %
Mean Training Loss:  tensor(0.0413)
Mean Validation Loss:  tensor(0.6264)
Validation Accuracy:  84.59869848156181 %
Training Accuracy (epoch  148 ):  100.0 %
Mean Training Loss:  tensor(0.0411)
Mean Validation Loss:  tensor(0.6270)
Validation Accuracy:  81.56182212581345 %
Training Accuracy (epoch  149 ):  100.0 %
Mean Training Loss:  tensor(0.0408)
Mean Validation Loss:  tensor(0.6267)
Validation Accuracy:  83.94793926247289 %
Training Accuracy (epoch  150 ):  100.0 %
Mean Training Loss:  tensor(0.0406)
Mean Validation Loss:  tensor(0.6266)
Validation Accuracy:  83.73101952277658 %
Training Accuracy (epoch  151 ):  100.0 %
Mean Training Loss:  tensor(0.0403)
Mean Validation Loss:  tensor(0.6273)
Validation Accuracy:  82.86334056399133 %
Training Accuracy (epoch  152 ):  100.0 %
Mean Training Loss:  tensor(0.0401)
Mean Validation Loss:  tensor(0.6277)
```

```
Validation Accuracy:  83.51409978308027 %
Training Accuracy (epoch  153 ):  100.0 %
Mean Training Loss:  tensor(0.0398)
Mean Validation Loss:  tensor(0.6281)
Validation Accuracy:  83.51409978308027 %
Training Accuracy (epoch  154 ):  100.0 %
Mean Training Loss:  tensor(0.0396)
Mean Validation Loss:  tensor(0.6282)
Validation Accuracy:  82.4295010845987 %
Training Accuracy (epoch  155 ):  100.0 %
Mean Training Loss:  tensor(0.0394)
Mean Validation Loss:  tensor(0.6280)
Validation Accuracy:  83.29718004338395 %
Training Accuracy (epoch  156 ):  100.0 %
Mean Training Loss:  tensor(0.0391)
Mean Validation Loss:  tensor(0.6275)
Validation Accuracy:  83.29718004338395 %
Training Accuracy (epoch  157 ):  100.0 %
Mean Training Loss:  tensor(0.0389)
Mean Validation Loss:  tensor(0.6273)
Validation Accuracy:  81.12798264642083 %
Training Accuracy (epoch  158 ):  100.0 %
Mean Training Loss:  tensor(0.0387)
Mean Validation Loss:  tensor(0.6273)
Validation Accuracy:  83.29718004338395 %
Training Accuracy (epoch  159 ):  100.0 %
Mean Training Loss:  tensor(0.0384)
Mean Validation Loss:  tensor(0.6272)
Validation Accuracy:  82.646420824295 %
Training Accuracy (epoch  160 ):  100.0 %
Mean Training Loss:  tensor(0.0382)
Mean Validation Loss:  tensor(0.6272)
Validation Accuracy:  83.29718004338395 %
Training Accuracy (epoch  161 ):  100.0 %
Mean Training Loss:  tensor(0.0380)
Mean Validation Loss:  tensor(0.6274)
Validation Accuracy:  82.86334056399133 %
Training Accuracy (epoch  162 ):  100.0 %
Mean Training Loss:  tensor(0.0378)
Mean Validation Loss:  tensor(0.6271)
Validation Accuracy:  82.646420824295 %
Training Accuracy (epoch  163 ):  100.0 %
Mean Training Loss:  tensor(0.0376)
Mean Validation Loss:  tensor(0.6271)
Validation Accuracy:  82.86334056399133 %
Training Accuracy (epoch  164 ):  100.0 %
Mean Training Loss:  tensor(0.0373)
Mean Validation Loss:  tensor(0.6268)
```

```
Validation Accuracy:  82.4295010845987 %
Training Accuracy (epoch  165 ):  100.0 %
Mean Training Loss:  tensor(0.0371)
Mean Validation Loss:  tensor(0.6268)
Validation Accuracy:  83.08026030368764 %
Training Accuracy (epoch  166 ):  100.0 %
Mean Training Loss:  tensor(0.0369)
Mean Validation Loss:  tensor(0.6267)
Validation Accuracy:  83.73101952277658 %
Training Accuracy (epoch  167 ):  100.0 %
Mean Training Loss:  tensor(0.0367)
Mean Validation Loss:  tensor(0.6264)
Validation Accuracy:  83.51409978308027 %
Training Accuracy (epoch  168 ):  100.0 %
Mean Training Loss:  tensor(0.0365)
Mean Validation Loss:  tensor(0.6269)
Validation Accuracy:  84.3817787418655 %
Training Accuracy (epoch  169 ):  100.0 %
Mean Training Loss:  tensor(0.0363)
Mean Validation Loss:  tensor(0.6270)
Validation Accuracy:  83.08026030368764 %
Training Accuracy (epoch  170 ):  100.0 %
Mean Training Loss:  tensor(0.0361)
Mean Validation Loss:  tensor(0.6269)
Validation Accuracy:  82.4295010845987 %
Training Accuracy (epoch  171 ):  100.0 %
Mean Training Loss:  tensor(0.0359)
Mean Validation Loss:  tensor(0.6271)
Validation Accuracy:  83.94793926247289 %
Training Accuracy (epoch  172 ):  100.0 %
Mean Training Loss:  tensor(0.0357)
Mean Validation Loss:  tensor(0.6272)
Validation Accuracy:  83.51409978308027 %
Training Accuracy (epoch  173 ):  100.0 %
Mean Training Loss:  tensor(0.0356)
Mean Validation Loss:  tensor(0.6275)
Validation Accuracy:  82.646420824295 %
Training Accuracy (epoch  174 ):  100.0 %
Mean Training Loss:  tensor(0.0354)
Mean Validation Loss:  tensor(0.6277)
Validation Accuracy:  83.51409978308027 %
Training Accuracy (epoch  175 ):  100.0 %
Mean Training Loss:  tensor(0.0352)
Mean Validation Loss:  tensor(0.6278)
Validation Accuracy:  83.94793926247289 %
Training Accuracy (epoch  176 ):  100.0 %
Mean Training Loss:  tensor(0.0350)
Mean Validation Loss:  tensor(0.6281)
```

```
Validation Accuracy:  82.646420824295 %
Training Accuracy (epoch  177 ):  100.0 %
Mean Training Loss:  tensor(0.0348)
Mean Validation Loss:  tensor(0.6278)
Validation Accuracy:  82.86334056399133 %
Training Accuracy (epoch  178 ):  100.0 %
Mean Training Loss:  tensor(0.0346)
Mean Validation Loss:  tensor(0.6282)
Validation Accuracy:  83.29718004338395 %
Training Accuracy (epoch  179 ):  100.0 %
Mean Training Loss:  tensor(0.0345)
Mean Validation Loss:  tensor(0.6282)
Validation Accuracy:  83.51409978308027 %
Training Accuracy (epoch  180 ):  100.0 %
Mean Training Loss:  tensor(0.0343)
Mean Validation Loss:  tensor(0.6277)
Validation Accuracy:  84.1648590021692 %
Training Accuracy (epoch  181 ):  100.0 %
Mean Training Loss:  tensor(0.0341)
Mean Validation Loss:  tensor(0.6279)
Validation Accuracy:  84.81561822125813 %
Training Accuracy (epoch  182 ):  100.0 %
Mean Training Loss:  tensor(0.0339)
Mean Validation Loss:  tensor(0.6280)
Validation Accuracy:  83.94793926247289 %
Training Accuracy (epoch  183 ):  100.0 %
Mean Training Loss:  tensor(0.0337)
Mean Validation Loss:  tensor(0.6288)
Validation Accuracy:  83.94793926247289 %
Training Accuracy (epoch  184 ):  100.0 %
Mean Training Loss:  tensor(0.0336)
Mean Validation Loss:  tensor(0.6288)
Validation Accuracy:  83.29718004338395 %
Training Accuracy (epoch  185 ):  100.0 %
Mean Training Loss:  tensor(0.0334)
Mean Validation Loss:  tensor(0.6289)
Validation Accuracy:  82.4295010845987 %
Training Accuracy (epoch  186 ):  100.0 %
Mean Training Loss:  tensor(0.0332)
Mean Validation Loss:  tensor(0.6285)
Validation Accuracy:  83.51409978308027 %
Training Accuracy (epoch  187 ):  100.0 %
Mean Training Loss:  tensor(0.0331)
Mean Validation Loss:  tensor(0.6286)
Validation Accuracy:  81.77874186550976 %
Training Accuracy (epoch  188 ):  100.0 %
Mean Training Loss:  tensor(0.0329)
Mean Validation Loss:  tensor(0.6283)
```

```
Validation Accuracy:  83.29718004338395 %
Training Accuracy (epoch  189 ):  100.0 %
Mean Training Loss:  tensor(0.0328)
Mean Validation Loss:  tensor(0.6280)
Validation Accuracy:  82.86334056399133 %
Training Accuracy (epoch  190 ):  100.0 %
Mean Training Loss:  tensor(0.0326)
Mean Validation Loss:  tensor(0.6282)
Validation Accuracy:  83.51409978308027 %
Training Accuracy (epoch  191 ):  100.0 %
Mean Training Loss:  tensor(0.0324)
Mean Validation Loss:  tensor(0.6283)
Validation Accuracy:  83.08026030368764 %
Training Accuracy (epoch  192 ):  100.0 %
Mean Training Loss:  tensor(0.0323)
Mean Validation Loss:  tensor(0.6286)
Validation Accuracy:  83.51409978308027 %
Training Accuracy (epoch  193 ):  100.0 %
Mean Training Loss:  tensor(0.0321)
Mean Validation Loss:  tensor(0.6283)
Validation Accuracy:  83.94793926247289 %
Training Accuracy (epoch  194 ):  100.0 %
Mean Training Loss:  tensor(0.0320)
Mean Validation Loss:  tensor(0.6285)
Validation Accuracy:  83.94793926247289 %
Training Accuracy (epoch  195 ):  100.0 %
Mean Training Loss:  tensor(0.0318)
Mean Validation Loss:  tensor(0.6284)
Validation Accuracy:  83.73101952277658 %
Training Accuracy (epoch  196 ):  100.0 %
Mean Training Loss:  tensor(0.0317)
Mean Validation Loss:  tensor(0.6282)
Validation Accuracy:  82.21258134490239 %
Training Accuracy (epoch  197 ):  99.89212513484358 %
Mean Training Loss:  tensor(0.0315)
Mean Validation Loss:  tensor(0.6291)
Validation Accuracy:  85.24945770065075 %
Training Accuracy (epoch  198 ):  100.0 %
Mean Training Loss:  tensor(0.0314)
Mean Validation Loss:  tensor(0.6287)
Validation Accuracy:  83.29718004338395 %
Training Accuracy (epoch  199 ):  100.0 %
Mean Training Loss:  tensor(0.0313)
Mean Validation Loss:  tensor(0.6299)
Validation Accuracy:  84.81561822125813 %
Accuracy:  81.08108108108108 %
```



<Figure size 640x480 with 0 Axes>

10 Finetuning Pre-trained Layers

[ ]:
```
x, loss_list_val = finetune(x, valloader, valloader, 200)
test(x, testloader)
```

```
/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-
packages/torch/nn/modules/module.py:1511: UserWarning: Implicit dimension choice
for softmax has been deprecated. Change the call to include dim=X as an
argument.
  return self._call_impl(*args, **kwargs)
```

```
Validation Accuracy:  85.46637744034707 %
Validation Accuracy:  85.24945770065075 %
Validation Accuracy:  86.55097613882863 %
```

```
Validation Accuracy:  84.59869848156181 %
Validation Accuracy:  85.24945770065075 %
Validation Accuracy:  86.11713665943601 %
Validation Accuracy:  87.20173535791757 %
Validation Accuracy:  88.06941431670282 %
Validation Accuracy:  85.68329718004338 %
Validation Accuracy:  85.46637744034707 %
Validation Accuracy:  85.46637744034707 %
Validation Accuracy:  88.06941431670282 %
Validation Accuracy:  87.41865509761388 %
Validation Accuracy:  86.55097613882863 %
Validation Accuracy:  86.55097613882863 %
Validation Accuracy:  86.98481561822126 %
Validation Accuracy:  83.94793926247289 %
Validation Accuracy:  85.68329718004338 %
Validation Accuracy:  86.33405639913232 %
Validation Accuracy:  86.98481561822126 %
Validation Accuracy:  86.55097613882863 %
Validation Accuracy:  87.20173535791757 %
Validation Accuracy:  85.90021691973969 %
Validation Accuracy:  85.68329718004338 %
Validation Accuracy:  85.46637744034707 %
Validation Accuracy:  86.55097613882863 %
Validation Accuracy:  85.03253796095444 %
Validation Accuracy:  86.76789587852495 %
Validation Accuracy:  86.33405639913232 %
Validation Accuracy:  86.55097613882863 %
Validation Accuracy:  85.90021691973969 %
Validation Accuracy:  86.55097613882863 %
Validation Accuracy:  85.68329718004338 %
Validation Accuracy:  86.76789587852495 %
Validation Accuracy:  86.98481561822126 %
Validation Accuracy:  85.90021691973969 %
Validation Accuracy:  87.20173535791757 %
Validation Accuracy:  85.68329718004338 %
Validation Accuracy:  88.06941431670282 %
Validation Accuracy:  85.24945770065075 %
Validation Accuracy:  87.20173535791757 %
Validation Accuracy:  86.98481561822126 %
Validation Accuracy:  86.33405639913232 %
Validation Accuracy:  86.98481561822126 %
Validation Accuracy:  86.55097613882863 %
Validation Accuracy:  86.76789587852495 %
Validation Accuracy:  86.55097613882863 %
Validation Accuracy:  86.33405639913232 %
Validation Accuracy:  87.85249457700651 %
Validation Accuracy:  86.98481561822126 %
Validation Accuracy:  85.46637744034707 %
```

```
Validation Accuracy:  85.68329718004338 %
Validation Accuracy:  85.90021691973969 %
Validation Accuracy:  85.68329718004338 %
Validation Accuracy:  85.90021691973969 %
Validation Accuracy:  87.20173535791757 %
Validation Accuracy:  85.68329718004338 %
Validation Accuracy:  85.24945770065075 %
Validation Accuracy:  84.1648590021692 %
Validation Accuracy:  87.41865509761388 %
Validation Accuracy:  85.90021691973969 %
Validation Accuracy:  85.90021691973969 %
Validation Accuracy:  84.81561822125813 %
Validation Accuracy:  86.55097613882863 %
Validation Accuracy:  85.90021691973969 %
Validation Accuracy:  86.33405639913232 %
Validation Accuracy:  86.76789587852495 %
Validation Accuracy:  86.33405639913232 %
Validation Accuracy:  86.76789587852495 %
Validation Accuracy:  86.76789587852495 %
Validation Accuracy:  85.90021691973969 %
Validation Accuracy:  87.20173535791757 %
Validation Accuracy:  86.55097613882863 %
Validation Accuracy:  88.28633405639913 %
Validation Accuracy:  86.11713665943601 %
Validation Accuracy:  86.76789587852495 %
Validation Accuracy:  85.46637744034707 %
Validation Accuracy:  85.46637744034707 %
Validation Accuracy:  85.46637744034707 %
Validation Accuracy:  86.55097613882863 %
Validation Accuracy:  86.33405639913232 %
Validation Accuracy:  86.33405639913232 %
Validation Accuracy:  85.90021691973969 %
Validation Accuracy:  85.68329718004338 %
Validation Accuracy:  85.68329718004338 %
Validation Accuracy:  87.20173535791757 %
Validation Accuracy:  87.20173535791757 %
Validation Accuracy:  85.90021691973969 %
Validation Accuracy:  87.20173535791757 %
Validation Accuracy:  86.11713665943601 %
Validation Accuracy:  85.24945770065075 %
Validation Accuracy:  86.98481561822126 %
Validation Accuracy:  86.33405639913232 %
Validation Accuracy:  86.11713665943601 %
Validation Accuracy:  86.11713665943601 %
Validation Accuracy:  85.68329718004338 %
Validation Accuracy:  86.76789587852495 %
Validation Accuracy:  84.59869848156181 %
Validation Accuracy:  87.63557483731019 %
```

```
Validation Accuracy:  85.46637744034707 %
Validation Accuracy:  87.20173535791757 %
Validation Accuracy:  85.24945770065075 %
Validation Accuracy:  87.41865509761388 %
Validation Accuracy:  86.33405639913232 %
Validation Accuracy:  86.98481561822126 %
Validation Accuracy:  85.68329718004338 %
Validation Accuracy:  87.85249457700651 %
Validation Accuracy:  87.20173535791757 %
Validation Accuracy:  87.20173535791757 %
Validation Accuracy:  86.76789587852495 %
Validation Accuracy:  87.20173535791757 %
Validation Accuracy:  85.90021691973969 %
Validation Accuracy:  87.63557483731019 %
Validation Accuracy:  84.59869848156181 %
Validation Accuracy:  85.03253796095444 %
Validation Accuracy:  85.03253796095444 %
Validation Accuracy:  86.76789587852495 %
Validation Accuracy:  86.98481561822126 %
Validation Accuracy:  85.68329718004338 %
Validation Accuracy:  84.1648590021692 %
Validation Accuracy:  86.55097613882863 %
Validation Accuracy:  87.20173535791757 %
Validation Accuracy:  85.46637744034707 %
Validation Accuracy:  85.24945770065075 %
Validation Accuracy:  86.98481561822126 %
Validation Accuracy:  86.33405639913232 %
Validation Accuracy:  87.85249457700651 %
Validation Accuracy:  87.85249457700651 %
Validation Accuracy:  85.46637744034707 %
Validation Accuracy:  87.20173535791757 %
Validation Accuracy:  87.41865509761388 %
Validation Accuracy:  86.98481561822126 %
Validation Accuracy:  87.63557483731019 %
Validation Accuracy:  87.63557483731019 %
Validation Accuracy:  85.46637744034707 %
Validation Accuracy:  87.41865509761388 %
Validation Accuracy:  87.20173535791757 %
Validation Accuracy:  86.33405639913232 %
Validation Accuracy:  86.11713665943601 %
Validation Accuracy:  86.33405639913232 %
Validation Accuracy:  86.55097613882863 %
Validation Accuracy:  85.46637744034707 %
Validation Accuracy:  86.98481561822126 %
Validation Accuracy:  85.03253796095444 %
Validation Accuracy:  85.68329718004338 %
Validation Accuracy:  86.76789587852495 %
Validation Accuracy:  86.11713665943601 %
```

```
Validation Accuracy:  86.98481561822126 %
Validation Accuracy:  85.68329718004338 %
Validation Accuracy:  87.41865509761388 %
Validation Accuracy:  86.76789587852495 %
Validation Accuracy:  88.28633405639913 %
Validation Accuracy:  86.76789587852495 %
Validation Accuracy:  87.85249457700651 %
Validation Accuracy:  86.98481561822126 %
Validation Accuracy:  87.85249457700651 %
Validation Accuracy:  86.33405639913232 %
Validation Accuracy:  85.68329718004338 %
Validation Accuracy:  86.11713665943601 %
Validation Accuracy:  85.68329718004338 %
Validation Accuracy:  86.55097613882863 %
Validation Accuracy:  86.76789587852495 %
Validation Accuracy:  85.68329718004338 %
Validation Accuracy:  85.90021691973969 %
Validation Accuracy:  85.68329718004338 %
Validation Accuracy:  85.68329718004338 %
Validation Accuracy:  86.76789587852495 %
Validation Accuracy:  87.41865509761388 %
Validation Accuracy:  86.55097613882863 %
Validation Accuracy:  85.46637744034707 %
Validation Accuracy:  86.76789587852495 %
Validation Accuracy:  85.03253796095444 %
Validation Accuracy:  86.33405639913232 %
Validation Accuracy:  84.3817787418655 %
Validation Accuracy:  87.20173535791757 %
Validation Accuracy:  87.41865509761388 %
Validation Accuracy:  86.76789587852495 %
Validation Accuracy:  86.11713665943601 %
Validation Accuracy:  87.41865509761388 %
Validation Accuracy:  85.46637744034707 %
Validation Accuracy:  86.11713665943601 %
Validation Accuracy:  85.90021691973969 %
Validation Accuracy:  88.93709327548807 %
Validation Accuracy:  85.03253796095444 %
Validation Accuracy:  87.20173535791757 %
Validation Accuracy:  86.98481561822126 %
Validation Accuracy:  87.63557483731019 %
Validation Accuracy:  86.33405639913232 %
Validation Accuracy:  84.81561822125813 %
Validation Accuracy:  86.76789587852495 %
Validation Accuracy:  85.46637744034707 %
Validation Accuracy:  86.55097613882863 %
Validation Accuracy:  87.20173535791757 %
Validation Accuracy:  85.24945770065075 %
Validation Accuracy:  87.85249457700651 %
```

```
Validation Accuracy:  85.68329718004338 %
Validation Accuracy:  86.55097613882863 %
Validation Accuracy:  86.11713665943601 %
Validation Accuracy:  86.33405639913232 %
Validation Accuracy:  86.76789587852495 %
Accuracy:  85.8108108108108 %
```

11 Hyperparameter Tuning

Hyper-parameter tuning requires that all file path names in this script be their absolute path names. Otherwise, this portion of the program will crash. This is due to how Raytune runs its testing environment, which is in a different file from this script.

This protion of the script can be slow if CUDA is not used. There is 0 support for Apple Metal by Ray, so the cpu will be used for testing hyperparameters even if items are loaded to MPS.

Ray was built for Linux and Linux-based systems. Running this portion of the script on a Windows device can be difficult and has been a known issue since late 2023. This is due to a fix the library implemented for a Pyarrow 7+ bug.

[ ]:
```
search_space = {"lr": tune.loguniform(1e-4, 1e-1), "eps": tune.
 ↪loguniform(1e-10, 1e-6), "l1": tune.randint(50, 2049), "l2": tune.
 ↪randint(12, 101), "lr_sched_step": tune.randint(1, 21)}
tuner = tune.Tuner(objective, tune_config=tune.
 ↪TuneConfig(metric="mean_accuracy", mode="max", search_alg=OptunaSearch(),␣
 ↪num_samples=2), param_space=search_space)

results = tuner.fit()
print("Best Config is: ", results.get_best_result().config)
```

```
<IPython.core.display.HTML object>
```

```
(raylet) Warning: The actor ImplicitFunc is very large (90 MiB). Check
that its definition is not implicitly capturing a large array or other object in
scope. Tip: use ray.put() to put large objects in the Ray object store.
```

```
(objective pid=50259) /var/folders/yj/n5srfxb951l5x6hcn4jcx3000000gn/T/
ipykernel_51918/3592750841.py:78: UserWarning: Creating a tensor from a list of
numpy.ndarrays is extremely slow. Please consider converting the list to a
single numpy.ndarray with numpy.array() before converting to a tensor.
(Triggered internally at
/Users/runner/work/pytorch/pytorch/pytorch/torch/csrc/utils/tensor_new.cpp:278.)
(objective pid=50277) /var/folders/yj/n5srfxb951l5x6hcn4jcx3000000gn/T/
ipykernel_51918/3592750841.py:78: UserWarning: Creating a tensor from a list of
numpy.ndarrays is extremely slow. Please consider converting the list to a
single numpy.ndarray with numpy.array() before converting to a tensor.
(Triggered internally at
/Users/runner/work/pytorch/pytorch/pytorch/torch/csrc/utils/tensor_new.cpp:278.)
2024-04-22 18:10:08,284 WARNING util.py:202 -- The `on_step_begin` operation
took 0.550 s, which may be a performance bottleneck.
```

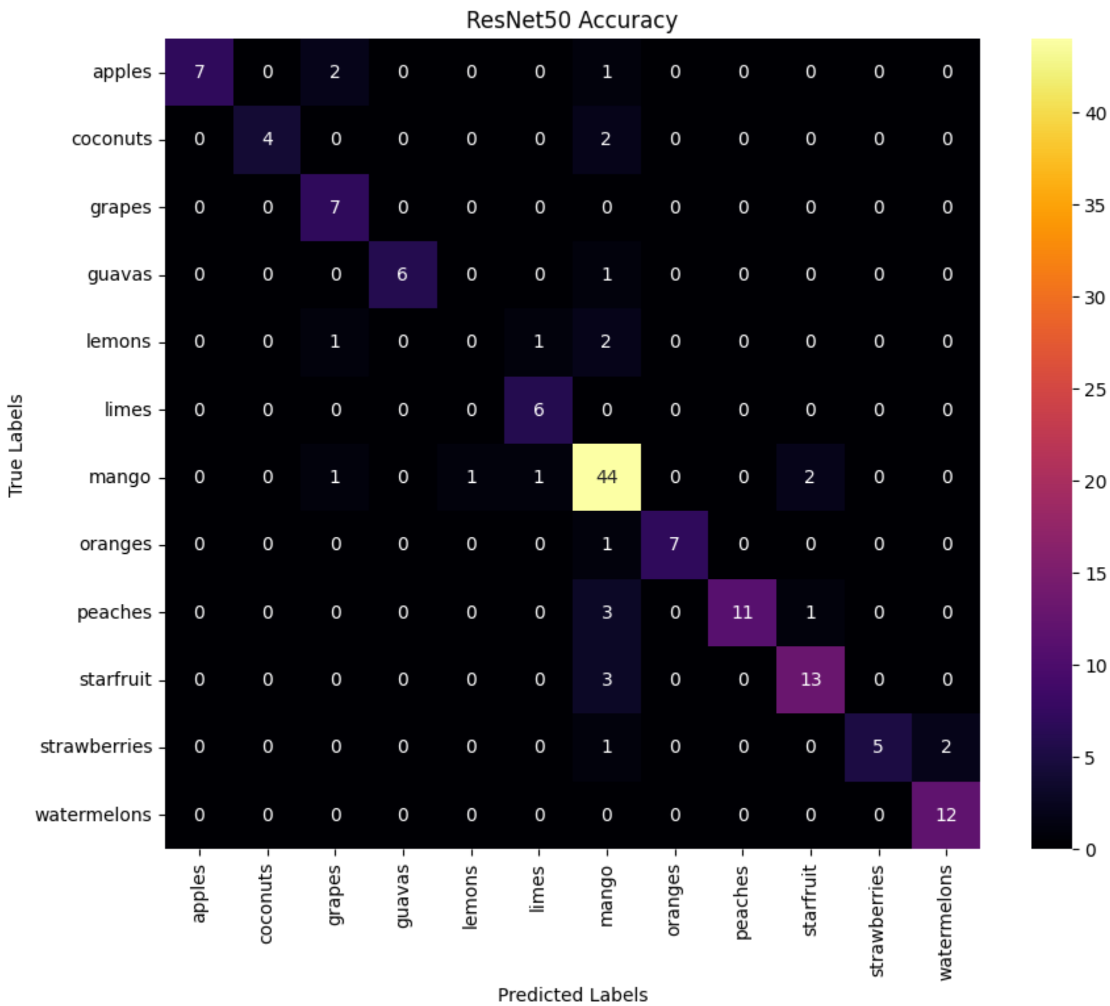
```
(objective pid=50259) Checkpoint successfully created at:
Checkpoint(filesystem=local, path=/Users/thomassigler/ray_results/objective_2024
-04-22_17-59-
50/objective_b9f0a4dd_1_eps=0.0000,l1=1881,l2=42,lr=0.0002,lr_sched_step=6_2024-
04-22_17-59-55/checkpoint_000000)
(objective pid=50277) Checkpoint successfully created at:
Checkpoint(filesystem=local, path=/Users/thomassigler/ray_results/objective_2024
-04-22_17-59-
50/objective_f4523408_2_eps=0.0000,l1=719,l2=72,lr=0.0125,lr_sched_step=6_2024-
04-22_18-00-11/checkpoint_000000)
(objective pid=50259) Checkpoint successfully created at:
Checkpoint(filesystem=local, path=/Users/thomassigler/ray_results/objective_2024
-04-22_17-59-
50/objective_b9f0a4dd_1_eps=0.0000,l1=1881,l2=42,lr=0.0002,lr_sched_step=6_2024-
04-22_17-59-55/checkpoint_000001)
(objective pid=50277) Checkpoint successfully created at:
Checkpoint(filesystem=local, path=/Users/thomassigler/ray_results/objective_2024
-04-22_17-59-
50/objective_f4523408_2_eps=0.0000,l1=719,l2=72,lr=0.0125,lr_sched_step=6_2024-
04-22_18-00-11/checkpoint_000001)
2024-04-22 18:27:02,663 INFO tune.py:1016 -- Wrote the latest version of all
result files and experiment state to
'/Users/thomassigler/ray_results/objective_2024-04-22_17-59-50' in 0.0138s.
2024-04-22 18:27:02,697 INFO tune.py:1048 -- Total run time: 1632.33 seconds
(1630.50 seconds for the tuning loop).

Best Config is:  {'lr': 0.00023301901230003848, 'eps': 2.887185712485118e-09,
'l1': 1881, 'l2': 42, 'lr_sched_step': 6}
```

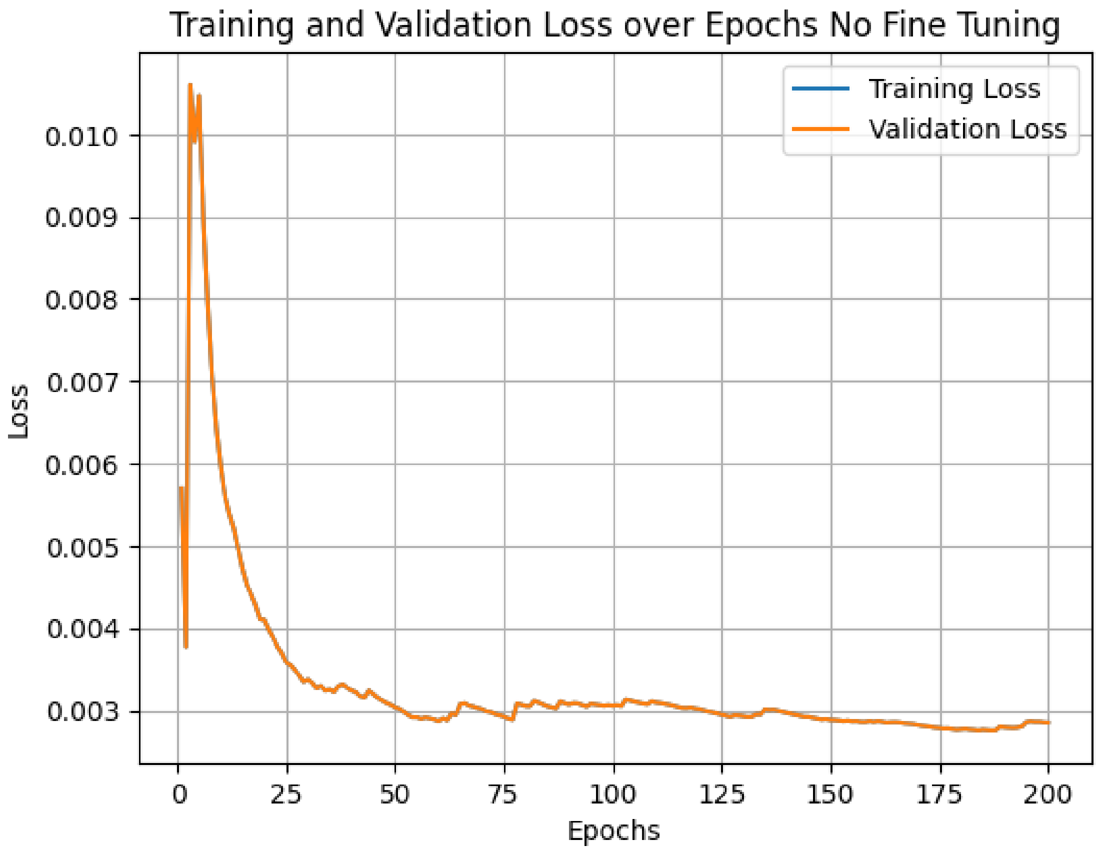
12 Plotting Graphs

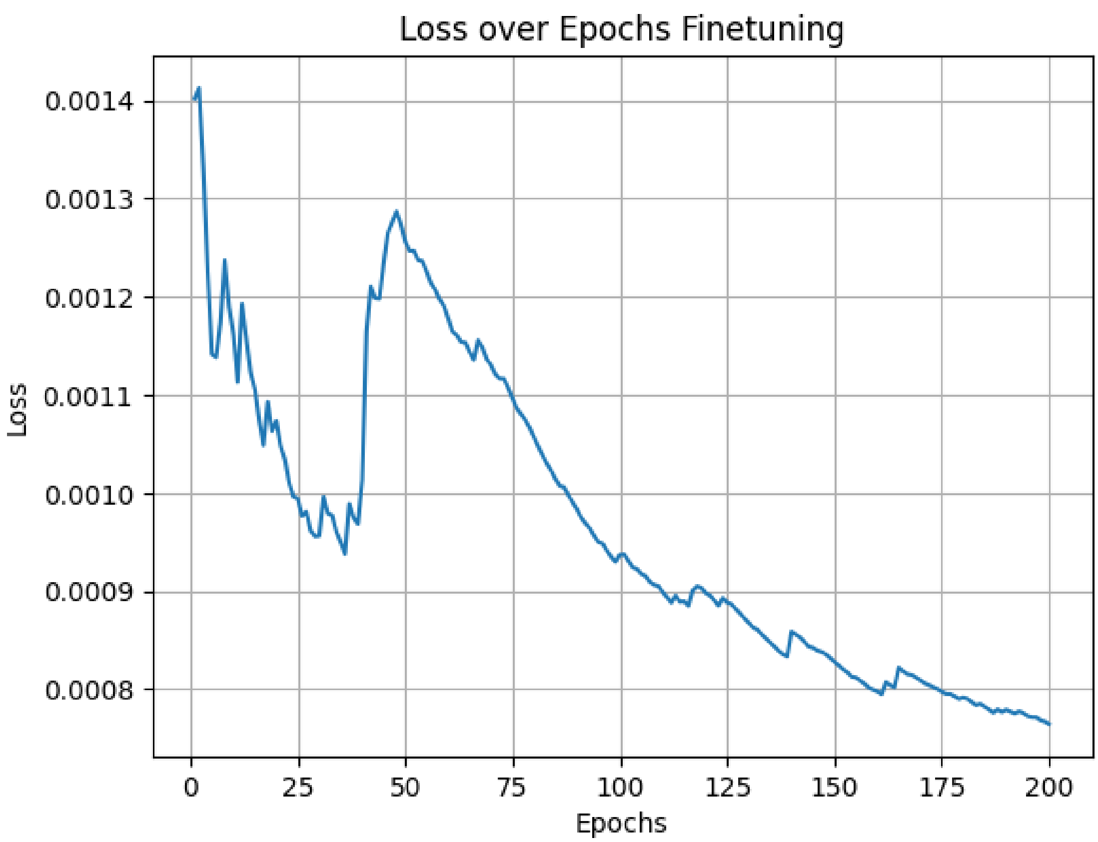
[ ]:
```
confusion_matrix(x, testloader)

plt.plot(np.linspace(1, len(loss_list), len(loss_list)), loss_list,␣
 ↪label='Training Loss')
plt.plot(np.linspace(1, len(val_loss), len(loss_list)), loss_list,␣
 ↪label='Validation Loss')
plt.title('Training and Validation Loss over Epochs No Fine Tuning')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.grid(True)
plt.legend()
plt.show()
plt.clf()

plt.plot(np.linspace(1, len(loss_list_val), len(loss_list_val)), loss_list_val)
plt.title('Loss over Epochs Finetuning')
```

```python
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.grid(True)
plt.show()
plt.clf()
```

```
/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-
packages/torch/nn/modules/module.py:1511: UserWarning: Implicit dimension choice
for softmax has been deprecated. Change the call to include dim=X as an
argument.
  return self._call_impl(*args, **kwargs)
```




Training and Validation Loss over Epochs No Fine Tuning
Training Loss
Validation Loss
Loss
Epochs
0.010
0.009
0.008
0.007
0.006
0.005
0.004
0.003
0
25
50
75
100
125
150
175
200



13 Optimizer Testing and Visualization

[ ]:
```python
def optimizer_testing(data, model, optimizer, scheduler, val_data, epochs=100):
    model = model.to("mps")
    acc_list = []
    for epoch in range(epochs):
        count = 0
        total = 0
        for imgs, labels in data:
            imgs, labels = imgs.to(device), labels.to(device)

            logits = model(imgs)
            loss = F.cross_entropy(logits, labels)

            loss.backward()
            optimizer.step()
        scheduler.step()

        for imgs, labels in val_data:
```

```
            imgs, labels = imgs.to(device), labels.to(device)

            logits = model(imgs)
            loss = F.cross_entropy(logits, labels)

            loss.backward()
            optimizer.step()

            pred = torch.argmax(logits, axis=1)
            count += (pred == labels).float().sum().item()
            total += imgs.shape[0]

        acc_list.append(count/total*100)
    return acc_list
```

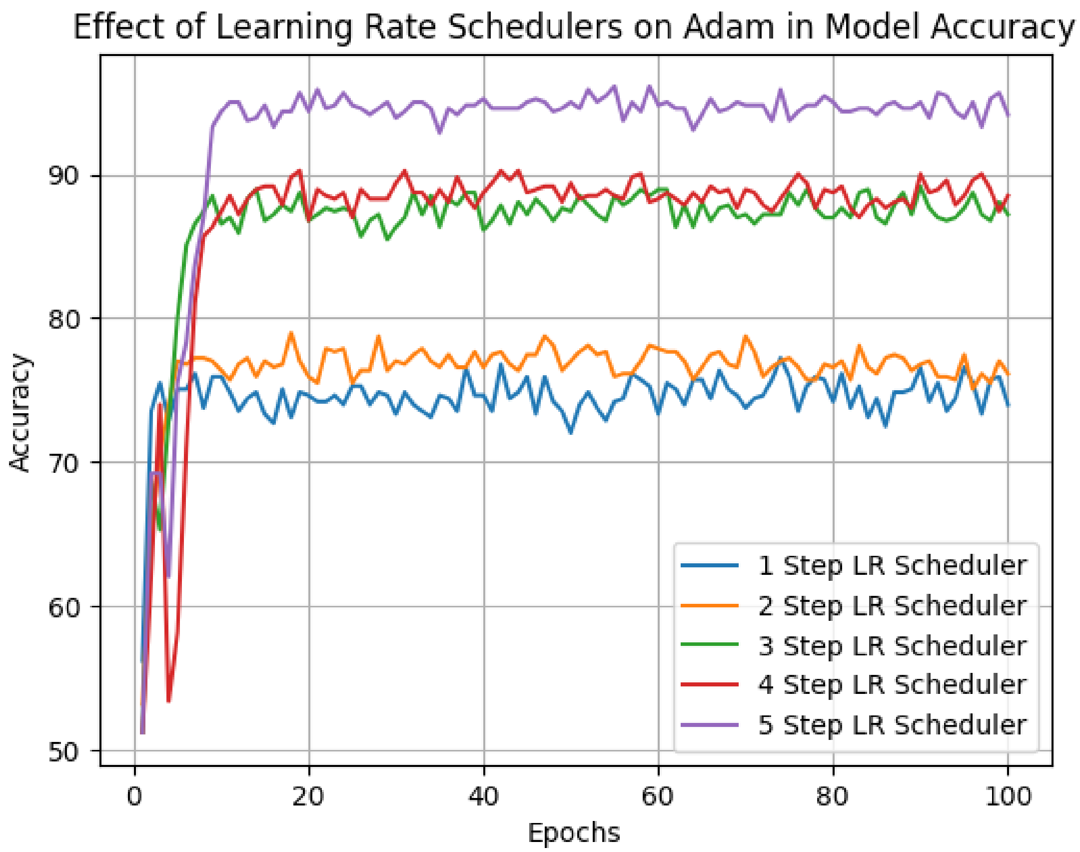
[ ]:
```
aux_model = resnet50(weights=ResNet50_Weights.DEFAULT)

for param in aux_model.parameters():
    param.requires_grad = False

aux_model.fc = nn.Linear(2048, 12)
nn.init.kaiming_uniform_(aux_model.fc.weight, nonlinearity='relu')
aux_model.fc.requires_grad = True
optimizer = torch.optim.Adam(aux_model.parameters(), lr=0.01)
Adam_list_1 = optimizer_testing(trainloader, aux_model, optimizer, optim.
 ↪lr_scheduler.StepLR(optimizer, step_size=1), valloader)
print("Done - 1")
aux_model = resnet50(weights=ResNet50_Weights.DEFAULT)

for param in aux_model.parameters():
    param.requires_grad = False

aux_model.fc = nn.Linear(2048, 12)
nn.init.kaiming_uniform_(aux_model.fc.weight, nonlinearity='relu')
aux_model.fc.requires_grad = True
optimizer = torch.optim.Adam(aux_model.parameters(), lr=0.01)
Adam_list_2 = optimizer_testing(trainloader, aux_model, optimizer, optim.
 ↪lr_scheduler.StepLR(optimizer, step_size=2), valloader)
print("Done - 2")
aux_model = resnet50(weights=ResNet50_Weights.DEFAULT)

for param in aux_model.parameters():
    param.requires_grad = False

aux_model.fc = nn.Linear(2048, 12)
nn.init.kaiming_uniform_(aux_model.fc.weight, nonlinearity='relu')
aux_model.fc.requires_grad = True
```

```
optimizer = torch.optim.Adam(aux_model.parameters(), lr=0.01)
Adam_list_3 = optimizer_testing(trainloader, aux_model, optimizer, optim.
 ↪lr_scheduler.StepLR(optimizer, step_size=3), valloader)
print("Done - 3")
aux_model = resnet50(weights=ResNet50_Weights.DEFAULT)

for param in aux_model.parameters():
    param.requires_grad = False

aux_model.fc = nn.Linear(2048, 12)
nn.init.kaiming_uniform_(aux_model.fc.weight, nonlinearity='relu')
aux_model.fc.requires_grad = True
optimizer = torch.optim.Adam(aux_model.parameters(), lr=0.01)
Adam_list_4 = optimizer_testing(trainloader, aux_model, optimizer, optim.
 ↪lr_scheduler.StepLR(optimizer, step_size=4), valloader)
print("Done - 4")
aux_model = resnet50(weights=ResNet50_Weights.DEFAULT)

for param in aux_model.parameters():
    param.requires_grad = False

aux_model.fc = nn.Linear(2048, 12)
nn.init.kaiming_uniform_(aux_model.fc.weight, nonlinearity='relu')
aux_model.fc.requires_grad = True
optimizer = torch.optim.Adam(aux_model.parameters(), lr=0.01)
Adam_list_5 = optimizer_testing(trainloader, aux_model, optimizer, optim.
 ↪lr_scheduler.StepLR(optimizer, step_size=5), valloader)
print("Done - 5")
```

```
Done - 1
Done - 2
Done - 3
Done - 4
Done - 5
```

[ ]:
```
plt.clf()
plt.plot(np.linspace(1, len(Adam_list_1), len(Adam_list_1)), Adam_list_1,␣
 ↪label="1 Step LR Scheduler")
plt.plot(np.linspace(1, len(Adam_list_2), len(Adam_list_2)), Adam_list_2,␣
 ↪label="2 Step LR Scheduler")
plt.plot(np.linspace(1, len(Adam_list_1), len(Adam_list_1)), Adam_list_3,␣
 ↪label="3 Step LR Scheduler")
plt.plot(np.linspace(1, len(Adam_list_1), len(Adam_list_1)), Adam_list_4,␣
 ↪label="4 Step LR Scheduler")
plt.plot(np.linspace(1, len(Adam_list_1), len(Adam_list_1)), Adam_list_5,␣
 ↪label="5 Step LR Scheduler")
plt.title("Effect of Learning Rate Schedulers on Adam in Model Accuracy")
```

```
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.grid(True)
plt.legend()
plt.show()
plt.clf()
```




```
<Figure size 640x480 with 0 Axes>
```