NinjaCart

October 11, 2023

1 Ninjacart: CV Classification

1.1 Problem Statement

- Ninjacart is India's largest fresh produce supply chain company. They are pioneers in solving one of the toughest supply chain problems of the world by leveraging innovative technology. They source fresh produce from farmers and deliver them to businesses within 12 hours. An integral component of their automation process is the development of robust classifiers which can distinguish between images of different types of vegetables, while also correctly labeling images that do not contain any one type of vegetable as noise.
- As a starting point, ninjacart has provided us with a dataset scraped from the web which contains train and test folders, each having 4 sub-folders with images of onions, potatoes, tomatoes and some market scenes. We have been tasked with preparing a multiclass classifier for identifying these vegetables. The dataset provided has all the required images to achieve the task.

[29]:
```
import torch.nn as nn
import torch
from torchvision import transforms
from torch.utils.data import DataLoader,random_split
from torch.utils.tensorboard import SummaryWriter
from torchvision.datasets import ImageFolder
from torchvision import models
import torchmetrics
from torchsummary import summary

import pathlib
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import plotly.express as px
import seaborn as sns
import datetime
import random

from tqdm import tqdm_notebook as tqdm
```

[2]:

```python
sns.set_style("darkgrid")
device = "cuda" if torch.cuda.is_available() else "cpu"
```

[6]:

```python
train_path = pathlib.Path("/Scaler/NinjaCart_Project/ninjacart_data/train/")
test_path = pathlib.Path("/Scaler/NinjaCart_Project/ninjacart_data/test/")

train_count = {}
test_count = {}

print("*"*25,"Train Images", "*"*25)
print("\n")
count = 0
for i in os.listdir(train_path):
    for j in os.listdir(os.path.join(train_path,i)):
        if i not in train_count:
            train_count[i] = [j]
        train_count[i].append(j)
    count+=len(train_count[i])
    print(f"Total Images in {i} folder: {len(train_count[i])}")
print(f"Total train images: {count}")

print("*"*25,"Test Images", "*"*25)
print("\n")
count = 0
for i in os.listdir(test_path):
    for j in os.listdir(os.path.join(test_path,i)):
        if i not in test_count:
            test_count[i] = [j]
        test_count[i].append(j)
    count+=len(test_count[i])
    print(f"Total Images in {i} folder: {len(test_count[i])}")

print(f"Total test images: {count}")
```

```
************************* Train Images *************************


Total Images in indian market folder: 600
Total Images in onion folder: 850
Total Images in potato folder: 899
Total Images in tomato folder: 790
Total train images: 3139
************************* Test Images *************************


Total Images in indian market folder: 82
Total Images in onion folder: 84
Total Images in potato folder: 82
```

```
Total Images in tomato folder: 107
Total test images: 355
```

1.1.1 Exploratory Data Analysis.

Plotting class distribution & Visualizing Image dimensions with their plots

[4]:
```python
for i in os.listdir(train_path):
    for j in os.listdir(os.path.join(train_path,i)):
        img = plt.imread(os.path.join(train_path,i,j))
        plt.imshow(img)
        plt.title(f"{i} {img.shape}")
        plt.axis("off")
        plt.tight_layout()
        plt.show()
        break
```

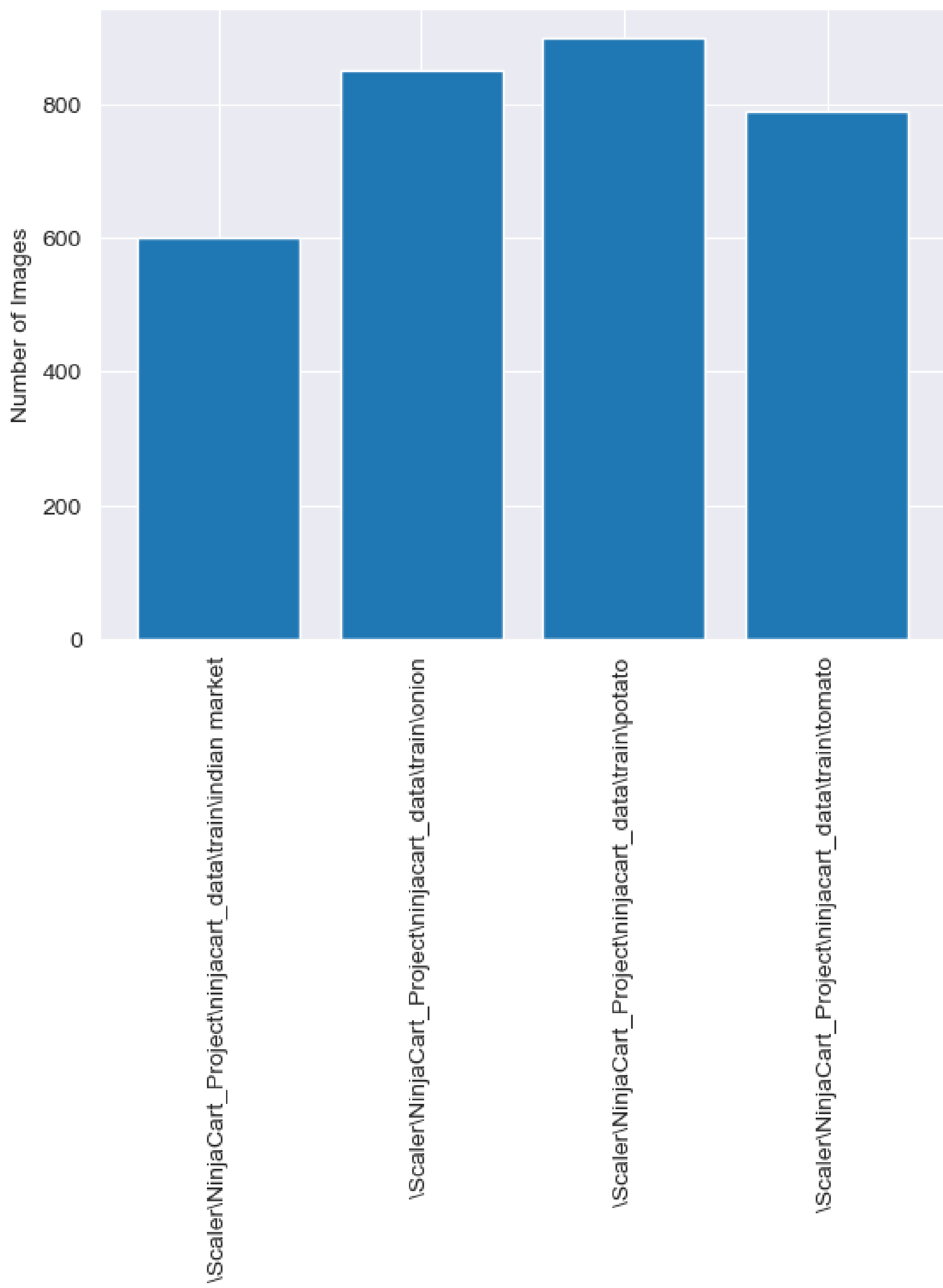
indian market (259, 194, 3)

onion (385, 640, 3)

potato (183, 275, 3)

tomato (500, 400, 3)

[5]:
```
subdirectories = [os.path.join(train_path, d) for d in os.listdir(train_path)␣
↪if os.path.isdir(os.path.join(train_path, d))]

# Count the number of images in each subdirectory
counts = [0] * len(subdirectories)
for i, directory in enumerate(subdirectories):
    counts[i] = len(os.listdir(directory))

# Create a bar chart to visualize the distribution
plt.bar(subdirectories, counts)
plt.xticks(rotation=90)
plt.ylabel("Number of Images")
plt.show()
```



Splitting the dataset into train, validation, and test set

[6]:
```
transform = transforms.Compose([
            transforms.RandomHorizontalFlip(),
```

```
            transforms.GaussianBlur(3),
            transforms.RandomRotation(10),
            transforms.Resize(500),
            transforms.CenterCrop(500),
            transforms.ToTensor(),
            transforms.Normalize([0.485, 0.456, 0.406],
                                 [0.229, 0.224, 0.225])
        ])
```

[7]:
```
Data_folder = ImageFolder(train_path,transform=transform)



train, valid = random_split(Data_folder,[int(0.8*len(Data_folder)),int(0.
 ↪2*len(Data_folder))],generator=torch.Generator().manual_seed(42))

train_data = DataLoader(train,batch_size=64,shuffle=True)
valid_data = DataLoader(valid,batch_size=64,shuffle=False)
```

[8]:
```
def plot_loss_accuracy(hist):
    fig, ax = plt.subplots(2,figsize=(8,8))
    fig.suptitle('Loss & Accuracy', fontsize=16)
    loss = []
    val_loss = []
    accu = []
    val_accu = []
    for i in hist:
        loss.append(i["loss"].item())
        val_loss.append(i["Val_Loss"].item())
        accu.append(i["train_accuracy"].item())
        val_accu.append(i["Val_Accu"].item())
    ax[0].plot(loss)
    ax[0].plot(val_loss)
    ax[0].set_xlabel("Epochs")
    ax[0].set_ylabel("Loss")
    ax[0].set_title("Loss Graph")


    ax[1].plot(accu)
    ax[1].plot(val_accu)
    ax[1].set_xlabel("Epochs")
    ax[1].set_ylabel("Accuracy")
    ax[1].set_title("Accuracy Graph")

    plt.show()
```

- Training_loop

[9]:

```python
def accuracy(pred,label):
    _, out = torch.max(pred,dim=1)
    return torch.tensor(torch.sum(out==label).item()/len(pred))

def validation_loss(model,validdata,loss):
    model.eval()
    with torch.no_grad():
        val_acc = []
        val_los = []
        for img,label in validdata:
            img = img.to(device)
            label = label.to(device)
            out = model(img)
            val_los.append(loss(out,label).item())
            val_acc.append(accuracy(out,label))
        return torch.tensor(val_los).mean(),torch.tensor(val_acc).mean()


def training_loop(model,train_data,valid_data,epochs,loss,optim):
    writer = SummaryWriter(log_dir=r"D:\Scaler\NinjaCart_Project\Logs")
    history = []
    for epoch in range(epochs+1):
        running_loss = []
        run_accuracy = []
        for img, label in train_data:
            img = img.to(device)
            label = label.to(device)
            out = model(img)
            train_loss=loss(out,label)
            train_loss.backward()
            optim.step()
            optim.zero_grad()
            running_loss.append(train_loss.item())
            run_accuracy.append(accuracy(out,label))
        val_loss, val_acc = validation_loss(model,valid_data,loss)
        history.append({"loss":torch.tensor(running_loss).mean(),
                        "train_accuracy":torch.tensor(run_accuracy).mean(),
                        "Val_Loss": val_loss,
                        "Val_Accu" : val_acc
                        })
        writer.add_scalar("Training loss x epoch", torch.tensor(running_loss).
 ↪mean(), epoch)
        writer.add_scalar("Validation loss x epoch", val_loss, epoch)
        writer.add_scalar("Train Accuracy x Epoch",torch.tensor(run_accuracy).
 ↪mean(),epoch)
        writer.add_scalar("Val Accuracy x Epoch",val_acc,epoch)
```

```
        print("{} Epoch {}, Training loss {},Train_accu {} Val_loss {}, Val␣
↪Accuracy {}".format(datetime.datetime.now(), epoch, torch.
↪tensor(running_loss).mean(),torch.tensor(run_accuracy).mean(),  val_loss,␣
↪val_acc))
    return history
```

1.1.2 Creating model architecture and training

CNN Classifier model from scratch

[22]:
```
class BaseCnn(nn.Module):
    def __init__(self,num_channel) -> None:
        super().__init__()
        self.conv_layer1 = nn.Sequential(
            nn.
↪Conv2d(in_channels=num_channel,out_channels=num_channel*2,kernel_size=3,stride=1),
            nn.ReLU(),
            nn.MaxPool2d(2,2),
            nn.BatchNorm2d(num_channel*2)
        )
        self.conv_layer2 = nn.Sequential(
            nn.
↪Conv2d(in_channels=num_channel*2,out_channels=num_channel*4,kernel_size=3,stride=1),
            nn.ReLU(),
            nn.MaxPool2d(2,2),
            nn.BatchNorm2d(num_channel*4)
        )
        self.conv_layer3 = nn.Sequential(
            nn.
↪Conv2d(in_channels=num_channel*4,out_channels=num_channel*8,kernel_size=3,stride=1),
            nn.ReLU(),
            nn.MaxPool2d(2,2),
            nn.BatchNorm2d(num_channel*8)
        )
        self.fc_layer1 = nn.Sequential(
            nn.Linear(24*60*60,512),
            nn.ReLU()
        )
        self.fc_layer2 = nn.Sequential(
            nn.Linear(512,4),
            nn.LogSoftmax(dim=1)
        )

    def forward(self,x):
        out = self.conv_layer1(x)
        out = self.conv_layer2(out)
        out = self.conv_layer3(out)
```

```
        # print(out.shape)
        out = out.view(-1,24*60*60)

        out = self.fc_layer1(out)
        out = self.fc_layer2(out)
        return out
```

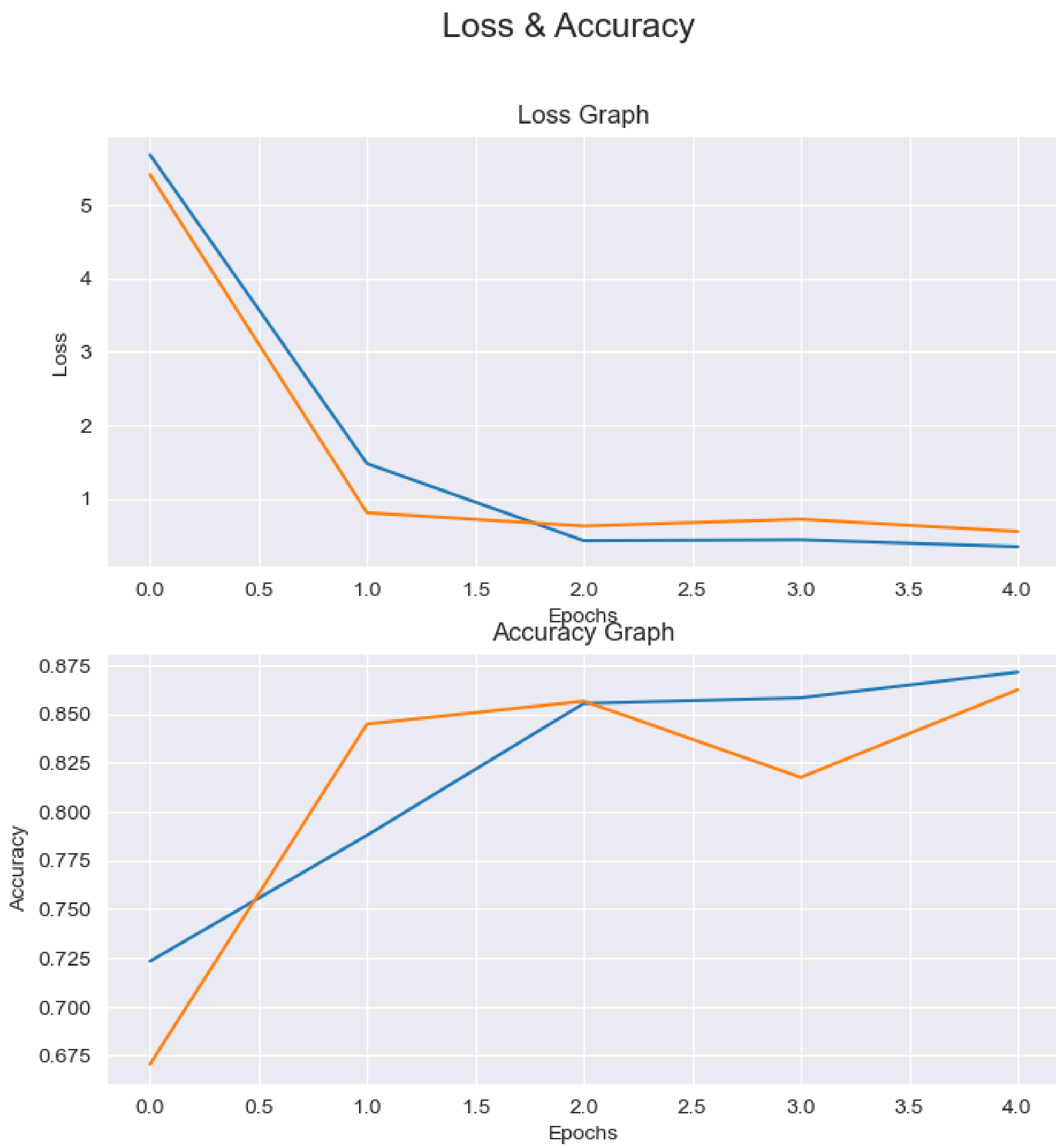
[50]:
```
model = BaseCnn(num_channel=3)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(params=model.parameters(),lr=0.001)

model.to(device)
```

[50]:
```
BaseCnn(
  (conv_layer1): Sequential(
    (0): Conv2d(3, 6, kernel_size=(3, 3), stride=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (3): BatchNorm2d(6, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  )
  (conv_layer2): Sequential(
    (0): Conv2d(6, 12, kernel_size=(3, 3), stride=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (3): BatchNorm2d(12, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  )
  (conv_layer3): Sequential(
    (0): Conv2d(12, 24, kernel_size=(3, 3), stride=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (3): BatchNorm2d(24, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  )
  (fc_layer1): Sequential(
    (0): Linear(in_features=86400, out_features=512, bias=True)
    (1): ReLU()
  )
  (fc_layer2): Sequential(
    (0): Linear(in_features=512, out_features=4, bias=True)
    (1): LogSoftmax(dim=1)
  )
)
```

[51]:
```
hist =␣
↪training_loop(epochs=5,model=model,loss=criterion,optim=optimizer,train_data=train_data,val
```

```
2023-10-11 22:27:27.338044 Epoch 0, Training loss 6.330111980438232,Train_accu
0.6819010376930237 Val_loss 3.962353229522705, Val_ Accuracy 0.6522058844566345
```

[57]:
```
plot_loss_accuracy(hist[:-1])
```



[64]:
```
weights_path = r"D:\Scaler\NinjaCart_Project\Weights\Basemodel.pth"
torch.save(model.state_dict(), weights_path)
```

ComplexCnn for smooth fitting

[23]:
```python
class ComplexCnn(nn.Module):
    def __init__(self,num_channel) -> None:
        super().__init__()
        self.conv_layer1 = nn.Sequential(
            nn.Conv2d(in_channels=num_channel,out_channels=num_channel*2,kernel_size=3),
            nn.ReLU(),
            nn.BatchNorm2d(num_channel*2),

            nn.Conv2d(in_channels=num_channel*2,out_channels=num_channel*4,stride=2,kernel_size=3),
            nn.ReLU(),
            nn.BatchNorm2d(num_channel*4),

            nn.Conv2d(in_channels=num_channel*4,out_channels=32,kernel_size=3),
            nn.ReLU(),
            nn.BatchNorm2d(32),
            nn.MaxPool2d(2,2),

            nn.Conv2d(in_channels=32,out_channels=64,kernel_size=3),
            nn.ReLU(),
            nn.BatchNorm2d(64),
            nn.MaxPool2d(2,2),

            nn.Conv2d(in_channels=64,out_channels=128,kernel_size=3),
            nn.ReLU(),
            nn.BatchNorm2d(128),
            nn.MaxPool2d(2,2)
        )

        self.fc_layer1 = nn.Sequential(
            nn.Linear(128*29*29,512),
            nn.ReLU(),

            nn.Linear(512,256),
            nn.ReLU(),

            nn.Linear(256,4),
            nn.LogSoftmax(dim=1)
        )

    def forward(self,x):
        out = self.conv_layer1(x)

        # print(out.shape)
        out = out.view(-1,128*29*29)
```

```
        out = self.fc_layer1(out)

        return out
```

[14]:
```
model = ComplexCnn(num_channel=3)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(params=model.parameters(),lr=0.001)

model.to(device)
```

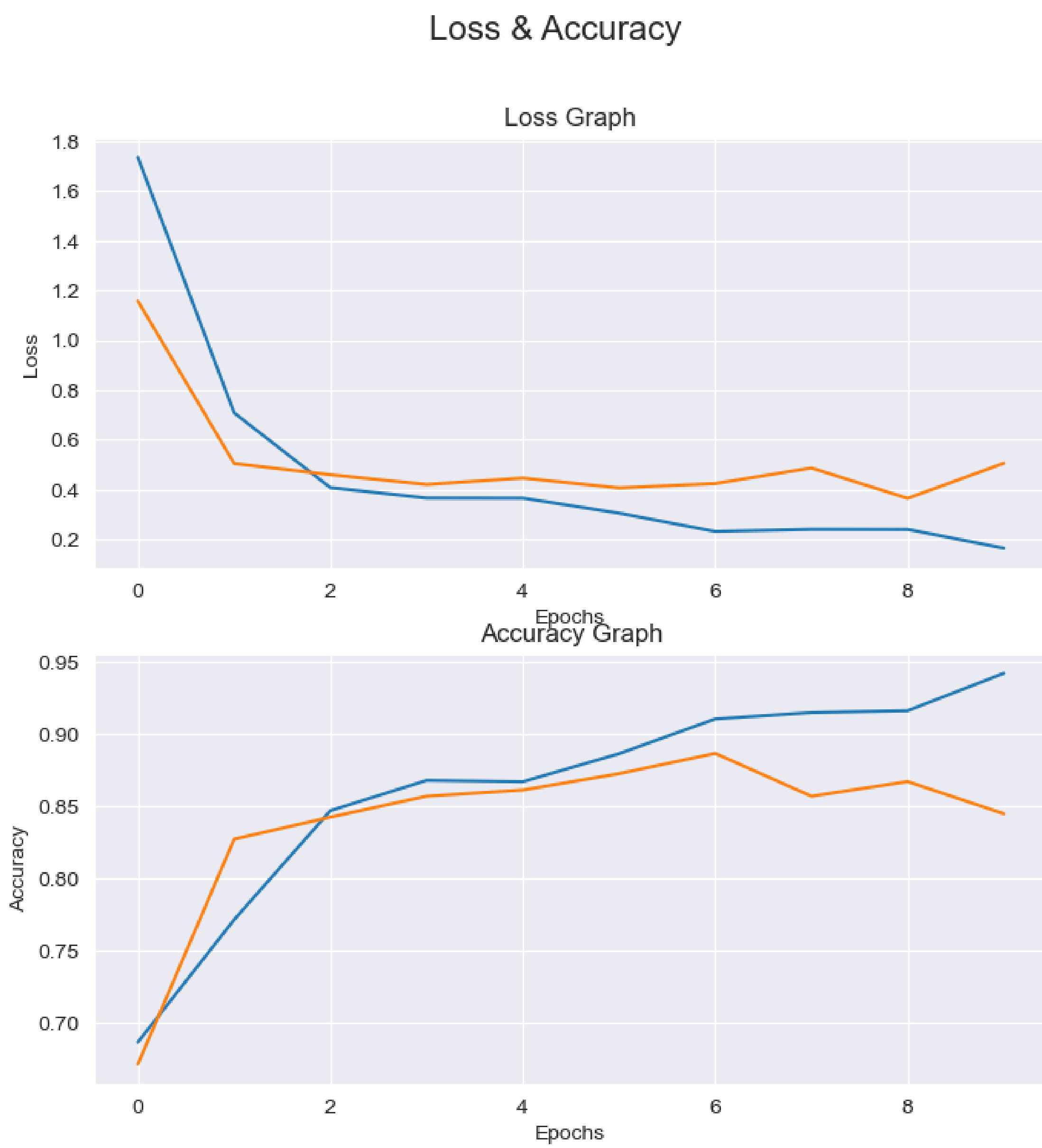
[14]:
```
ComplexCnn(
  (conv_layer1): Sequential(
    (0): Conv2d(3, 6, kernel_size=(3, 3), stride=(1, 1))
    (1): ReLU()
    (2): BatchNorm2d(6, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (3): Conv2d(6, 12, kernel_size=(3, 3), stride=(2, 2))
    (4): ReLU()
    (5): BatchNorm2d(12, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (6): Conv2d(12, 32, kernel_size=(3, 3), stride=(1, 1))
    (7): ReLU()
    (8): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (10): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
    (11): ReLU()
    (12): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (13): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (14): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1))
    (15): ReLU()
    (16): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (17): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (fc_layer1): Sequential(
    (0): Linear(in_features=107648, out_features=512, bias=True)
    (1): ReLU()
    (2): Linear(in_features=512, out_features=256, bias=True)
    (3): ReLU()
    (4): Linear(in_features=256, out_features=4, bias=True)
    (5): LogSoftmax(dim=1)
```

```
  )
)
```

[15]:
```
hist =␣
 ↪training_loop(epochs=10,model=model,loss=criterion,optim=optimizer,train_data=train_data,va
```

```
2023-10-11 15:14:26.086776 Epoch 0, Training loss 1.735496163368225,Train_accu
0.6869791746139526 Val_loss 1.1578682661056519, Val_ Accuracy 0.6717524528503418
2023-10-11 15:16:02.502520 Epoch 1, Training loss 0.7086995840072632,Train_accu
0.7718750238418579 Val_loss 0.5053683519363403, Val_ Accuracy 0.8276654481887817
2023-10-11 15:17:43.472027 Epoch 2, Training loss 0.4083729684352875,Train_accu
0.8473957777023315 Val_loss 0.46034055948257446, Val_ Accuracy
0.8429228067398071
2023-10-11 15:19:25.025675 Epoch 3, Training loss 0.3666824996471405,Train_accu
0.868359386920929 Val_loss 0.4212194085121155, Val_ Accuracy 0.8573835492134094
2023-10-11 15:21:07.972460 Epoch 4, Training loss 0.36614498496055603,Train_accu
0.8674479722976685 Val_loss 0.446850448846817, Val_ Accuracy 0.8616727590560913
2023-10-11 15:22:49.660638 Epoch 5, Training loss 0.3059066832065582,Train_accu
0.8868489265441895 Val_loss 0.4073648452758789, Val_ Accuracy 0.8730085492134094
2023-10-11 15:24:30.503931 Epoch 6, Training loss 0.23235023021697998,Train_accu
0.910937488079071 Val_loss 0.42485150694847107, Val_ Accuracy 0.8870710134506226
2023-10-11 15:26:04.386240 Epoch 7, Training loss 0.24108807742595673,Train_accu
0.9153645634651184 Val_loss 0.4873097836971283, Val_ Accuracy 0.8573835492134094
2023-10-11 15:27:33.373243 Epoch 8, Training loss 0.24055051803588867,Train_accu
0.9166666865348816 Val_loss 0.3655339479446411, Val_ Accuracy 0.8675245046615601
2023-10-11 15:29:00.891277 Epoch 9, Training loss 0.16508269309997559,Train_accu
0.942578136920929 Val_loss 0.5063896775245667, Val_ Accuracy 0.8452512621879578
2023-10-11 15:30:28.301085 Epoch 10, Training loss
6.742588961283176e+27,Train_accu 0.934374988079071 Val_loss 0.5418890118598938,
Val_ Accuracy 0.8651654124259949
```

[16]:
```
plot_loss_accuracy(hist[:-1])
```



[18]:
```
weights_path = r"D:\Scaler\NinjaCart_Project\Weights\Complexmodel.pth"
torch.save(model.state_dict(), weights_path)
```

ComplexCnn1 with l2 Regularization and Dropout parameters

[10]:
```
class ComplexCnn1(nn.Module):
    def __init__(self,num_channel) -> None:
        super().__init__()
        self.conv_layer1 = nn.Sequential(
            nn.
 ↪Conv2d(in_channels=num_channel,out_channels=num_channel*2,kernel_size=3),
```

```
            nn.ReLU(),
            nn.BatchNorm2d(num_channel*2),

            nn.
↪Conv2d(in_channels=num_channel*2,out_channels=num_channel*4,stride=2,kernel_size=3),
            nn.ReLU(),
            nn.BatchNorm2d(num_channel*4),

            nn.Conv2d(in_channels=num_channel*4,out_channels=32,kernel_size=3),
            nn.ReLU(),
            nn.BatchNorm2d(32),
            nn.MaxPool2d(2,2),

            nn.Conv2d(in_channels=32,out_channels=64,kernel_size=3),
            nn.ReLU(),
            nn.BatchNorm2d(64),
            nn.MaxPool2d(2,2),

            nn.Conv2d(in_channels=64,out_channels=128,kernel_size=3),
            nn.ReLU(),
            nn.BatchNorm2d(128),
            nn.MaxPool2d(2,2)
        )

        self.fc_layer1 = nn.Sequential(
            nn.Dropout(0.3),
            nn.Linear(128*29*29,512),
            nn.ReLU(),

            nn.Dropout(0.3),
            nn.Linear(512,256),
            nn.ReLU(),

            nn.Dropout(0.3),
            nn.Linear(256,4),
            nn.LogSoftmax(dim=1)
        )

    def forward(self,x):
        out = self.conv_layer1(x)

        # print(out.shape)
        out = out.view(-1,128*29*29)

        out = self.fc_layer1(out)

        return out
```

[11]:
```
model = ComplexCnn1(num_channel=3)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(params=model.parameters(),lr=0.
 ↪001,weight_decay=1e-5)

model.to(device)
```

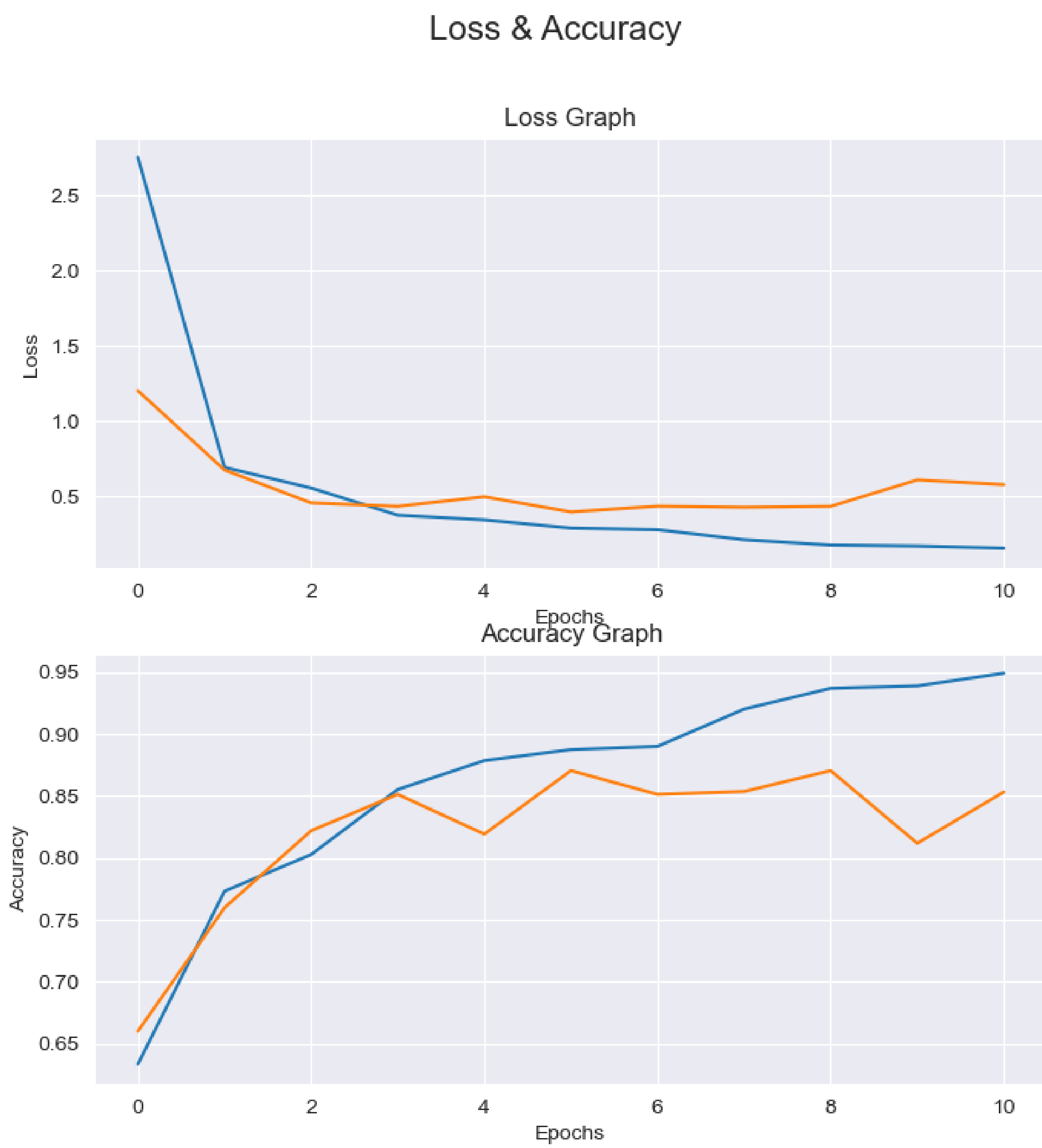
[11]:
```
ComplexCnn1(
  (conv_layer1): Sequential(
    (0): Conv2d(3, 6, kernel_size=(3, 3), stride=(1, 1))
    (1): ReLU()
    (2): BatchNorm2d(6, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (3): Conv2d(6, 12, kernel_size=(3, 3), stride=(2, 2))
    (4): ReLU()
    (5): BatchNorm2d(12, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (6): Conv2d(12, 32, kernel_size=(3, 3), stride=(1, 1))
    (7): ReLU()
    (8): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (10): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
    (11): ReLU()
    (12): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (13): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (14): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1))
    (15): ReLU()
    (16): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (17): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (fc_layer1): Sequential(
    (0): Dropout(p=0.3, inplace=False)
    (1): Linear(in_features=107648, out_features=512, bias=True)
    (2): ReLU()
    (3): Dropout(p=0.3, inplace=False)
    (4): Linear(in_features=512, out_features=256, bias=True)
    (5): ReLU()
    (6): Dropout(p=0.3, inplace=False)
    (7): Linear(in_features=256, out_features=4, bias=True)
    (8): LogSoftmax(dim=1)
  )
```

```
)
```

[12]:
```
hist =␣
↪training_loop(model=model,train_data=train_data,valid_data=valid_data,epochs=10,loss=criter
```

```
2023-10-11 22:46:11.445330 Epoch 0, Training loss 2.755383253097534,Train_accu
0.6334635019302368 Val_loss 1.2032948732376099, Val_ Accuracy 0.6600490212440491
2023-10-11 22:47:53.123661 Epoch 1, Training loss 0.6931974291801453,Train_accu
0.7730468511581421 Val_loss 0.6757590174674988, Val_ Accuracy 0.7596813440322876
2023-10-11 22:49:34.036219 Epoch 2, Training loss 0.5559856295585632,Train_accu
0.8026041984558105 Val_loss 0.45714253187179565, Val_ Accuracy
0.8218137621879578
2023-10-11 22:51:09.774744 Epoch 3, Training loss 0.37543243169784546,Train_accu
0.8550781011581421 Val_loss 0.43442726135253906, Val_ Accuracy
0.8511335253715515
2023-10-11 22:52:43.598312 Epoch 4, Training loss 0.34399253129959106,Train_accu
0.8783854246139526 Val_loss 0.49789175391197205, Val_ Accuracy 0.819087028503418
2023-10-11 22:54:17.381893 Epoch 5, Training loss 0.2894912660121918,Train_accu
0.8872395753860474 Val_loss 0.39725929498672485, Val_ Accuracy
0.8702818751335144
2023-10-11 22:55:51.297775 Epoch 6, Training loss 0.2785736620426178,Train_accu
0.889843761920929 Val_loss 0.43500056862831116, Val_ Accuracy 0.8511335253715515
2023-10-11 22:57:24.984219 Epoch 7, Training loss 0.21191683411598206,Train_accu
0.919921875 Val_loss 0.4281575679779053, Val_ Accuracy 0.8534620404243469
2023-10-11 22:59:00.373154 Epoch 8, Training loss 0.17638280987739563,Train_accu
0.936718761920929 Val_loss 0.43429914116859436, Val_ Accuracy 0.8702512979507446
2023-10-11 23:00:33.915557 Epoch 9, Training loss 0.1697533130645752,Train_accu
0.938671886920929 Val_loss 0.6097890734672546, Val_ Accuracy 0.8116728067398071
2023-10-11 23:02:03.458717 Epoch 10, Training loss
0.15574601292610168,Train_accu 0.9488281011581421 Val_loss 0.5786613821983337,
Val_ Accuracy 0.8530637621879578
```

[13]:
```
plot_loss_accuracy(hist)
```



[14]:
```
weights_path = r"D:
↪\Scaler\NinjaCart_Project\Weights\Complexmodel_with_overfitting.pth"
torch.save(model.state_dict(), weights_path)
```

Pretrained Model – ResNet50

[11]:
```
model = models.resnet50(pretrained=True)
# since we are using the ResNet50 model as a feature extractor we set
# its parameters to non-trainable (by default they are trainable)
for param in model.parameters():
        param.requires_grad = False
```

```
# on to the current device
modelOutputFeats = model.fc.in_features
model.fc = nn.Linear(modelOutputFeats, 4)
model = model.to(device)
```

```
c:\Users\revan\anaconda3\lib\site-packages\torchvision\models\_utils.py:208:
UserWarning: The parameter 'pretrained' is deprecated since 0.13 and may be
removed in the future, please use 'weights' instead.
  warnings.warn(
c:\Users\revan\anaconda3\lib\site-packages\torchvision\models\_utils.py:223:
UserWarning: Arguments other than a weight enum or `None` for 'weights' are
deprecated since 0.13 and may be removed in the future. The current behavior is
equivalent to passing `weights=ResNet50_Weights.IMAGENET1K_V1`. You can also use
`weights=ResNet50_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet50-0676ba61.pth" to
C:\Users\revan/.cache\torch\hub\checkpoints\resnet50-0676ba61.pth

  0%|          | 0.00/97.8M [00:00<?, ?B/s]
```

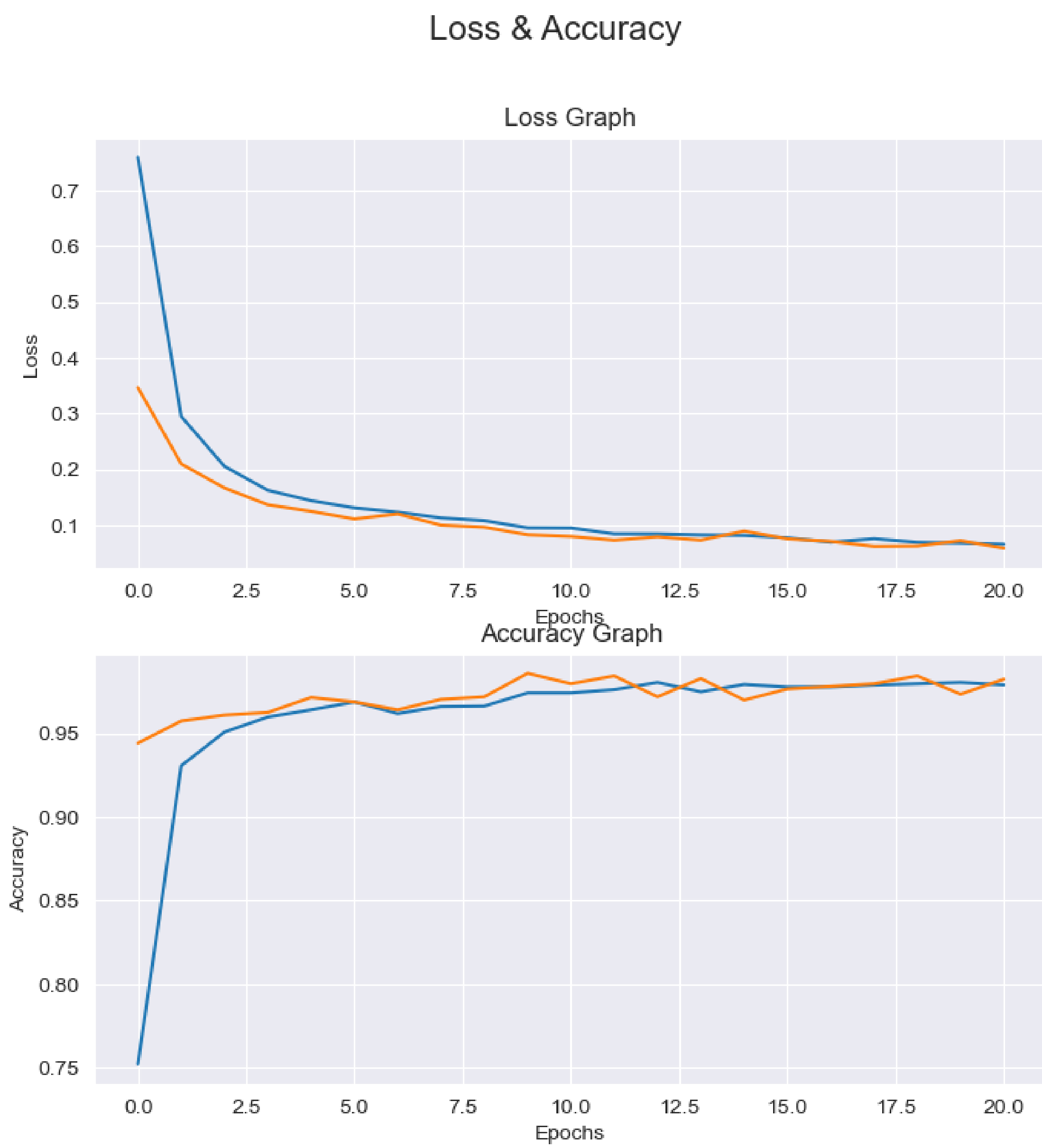
[12]:
```
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(params=model.parameters(),lr=0.001)
```

[13]:
```
hist =␣
  ↪training_loop(model=model,train_data=train_data,valid_data=valid_data,epochs=20,loss=criter
```

```
2023-10-11 17:34:43.100686 Epoch 0, Training loss 0.7595013380050659,Train_accu
0.751953125 Val_loss 0.3463861346244812, Val_ Accuracy 0.9441176652908325
2023-10-11 17:36:44.512801 Epoch 1, Training loss 0.29463303089141846,Train_accu
0.9305989146232605 Val_loss 0.20936469733715057, Val_ Accuracy
0.9574142694473267
2023-10-11 17:38:45.014544 Epoch 2, Training loss 0.20497548580169678,Train_accu
0.9507812261581421 Val_loss 0.1662873923778534, Val_ Accuracy 0.9609375
2023-10-11 17:40:43.290859 Epoch 3, Training loss 0.16220775246620178,Train_accu
0.959765613079071 Val_loss 0.13619402050971985, Val_ Accuracy 0.9624999761581421
2023-10-11 17:42:42.597787 Epoch 4, Training loss 0.1435335874557495,Train_accu
0.964062511920929 Val_loss 0.12438507378101349, Val_ Accuracy 0.9714767336845398
2023-10-11 17:44:42.348485 Epoch 5, Training loss 0.13055306673049927,Train_accu
0.96875 Val_loss 0.11077388375997543, Val_ Accuracy 0.96875
2023-10-11 17:46:43.640830 Epoch 6, Training loss 0.1229119673371315,Train_accu
0.9618489146232605 Val_loss 0.11982085555791855, Val_ Accuracy 0.964062511920929
2023-10-11 17:48:45.713106 Epoch 7, Training loss 0.1127258688211441,Train_accu
0.966015636920929 Val_loss 0.09960721433162689, Val_ Accuracy 0.9703124761581421
2023-10-11 17:50:49.903531 Epoch 8, Training loss 0.10769388824701309,Train_accu
0.9662760496139526 Val_loss 0.09590649604797363, Val_ Accuracy 0.971875011920929
2023-10-11 17:52:52.731025 Epoch 9, Training loss 0.09487268328666687,Train_accu
0.9742187261581421 Val_loss 0.08230968564748764, Val_ Accuracy
0.9859374761581421
2023-10-11 17:54:55.408599 Epoch 10, Training loss
```

```
0.09452280402183533,Train_accu 0.9742187261581421 Val_loss 0.0792941004037857,
Val_ Accuracy 0.979687511920929
2023-10-11 17:56:58.344620 Epoch 11, Training loss
0.08394793421030045,Train_accu 0.9761718511581421 Val_loss 0.07251139730215073,
Val_ Accuracy 0.984375
2023-10-11 17:59:01.391689 Epoch 12, Training loss
0.08382031321525574,Train_accu 0.98046875 Val_loss 0.07854972779750824, Val_
Accuracy 0.971875011920929
2023-10-11 18:01:03.315959 Epoch 13, Training loss
0.08196136355400085,Train_accu 0.9748698472976685 Val_loss 0.0725628137588501,
Val_ Accuracy 0.9828125238418579
2023-10-11 18:03:04.793246 Epoch 14, Training loss
0.08117194473743439,Train_accu 0.979296863079071 Val_loss 0.08899617940187454,
Val_ Accuracy 0.9699142575263977
2023-10-11 18:05:07.993827 Epoch 15, Training loss
0.07691022753715515,Train_accu 0.977734386920929 Val_loss 0.07497622072696686,
Val_ Accuracy 0.9765625
2023-10-11 18:07:12.850721 Epoch 16, Training loss
0.06896936148405075,Train_accu 0.977734386920929 Val_loss 0.0704292431473732,
Val_ Accuracy 0.9781249761581421
2023-10-11 18:09:17.325440 Epoch 17, Training loss
0.07525552064180374,Train_accu 0.9789062738418579 Val_loss 0.06152898818254471,
Val_ Accuracy 0.979687511920929
2023-10-11 18:11:22.115252 Epoch 18, Training loss
0.06869018077850342,Train_accu 0.979687511920929 Val_loss 0.06217086315155029,
Val_ Accuracy 0.984375
2023-10-11 18:13:24.963071 Epoch 19, Training loss 0.0675734356045723,Train_accu
0.98046875 Val_loss 0.07167433202266693, Val_ Accuracy 0.973437488079071
2023-10-11 18:15:26.757286 Epoch 20, Training loss
0.06533544510602951,Train_accu 0.9790364503860474 Val_loss 0.05827395245432854,
Val_ Accuracy 0.9824142456054688
```

[14]:
```
plot_loss_accuracy(hist)
```



[16]:
```
weights_path = r"D:\Scaler\NinjaCart_Project\Weights\ResNetmodel.pth"
torch.save(model.state_dict(), weights_path)
```

MobileNet

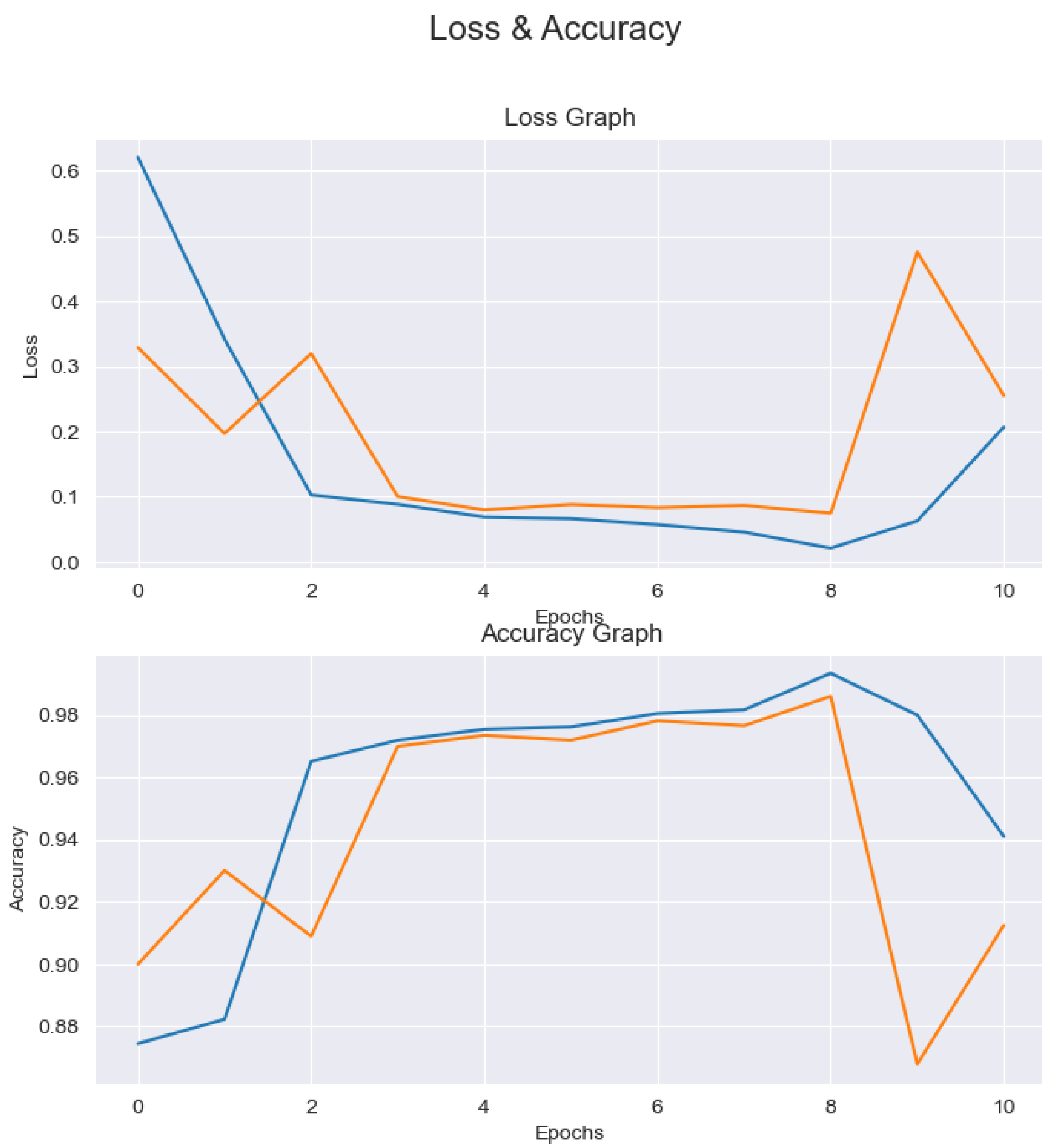
[33]:
```
model = models.mobilenet_v3_small(pretrained=True)
# since we are using the ResNet50 model as a feature extractor we set
# its parameters to non-trainable (by default they are trainable)
# on to the current device
model.classifier[3].out_features = 4
```

```
model = model.to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(params=model.parameters(),lr=0.001)
```

[34]:
```
hist =␣
  ↪training_loop(model=model,train_data=train_data,valid_data=valid_data,epochs=10,loss=criter
```

```
2023-10-11 20:56:25.749385 Epoch 0, Training loss 0.6206335425376892,Train_accu
0.8744791746139526 Val_loss 0.32880836725234985, Val_ Accuracy
0.8999693989753723
2023-10-11 20:58:18.099385 Epoch 1, Training loss 0.34168487787246704,Train_accu
0.8822916746139526 Val_loss 0.19669687747955322, Val_ Accuracy
0.9300551414489746
2023-10-11 20:59:48.573694 Epoch 2, Training loss 0.10252787917852402,Train_accu
0.9651042222976685 Val_loss 0.3195001184940338, Val_ Accuracy 0.9089460372924805
2023-10-11 21:01:15.518696 Epoch 3, Training loss 0.08810731023550034,Train_accu
0.971875011920929 Val_loss 0.10009181499481201, Val_ Accuracy 0.9699142575263977
2023-10-11 21:02:43.268059 Epoch 4, Training loss 0.06854832172393799,Train_accu
0.975390613079071 Val_loss 0.0793922170996666, Val_ Accuracy 0.973437488079071
2023-10-11 21:04:32.371338 Epoch 5, Training loss 0.06607703864574432,Train_accu
0.9761718511581421 Val_loss 0.08783284574747086, Val_ Accuracy 0.971875011920929
2023-10-11 21:06:15.774881 Epoch 6, Training loss 0.05698562413454056,Train_accu
0.98046875 Val_loss 0.0830567255616188, Val_ Accuracy 0.9781249761581421
2023-10-11 21:08:02.263644 Epoch 7, Training loss
0.045340877026319504,Train_accu 0.981640636920929 Val_loss 0.08632183074951172,
Val_ Accuracy 0.9765625
2023-10-11 21:10:14.668430 Epoch 8, Training loss 0.0207529254257679,Train_accu
0.993359386920929 Val_loss 0.07439843565225601, Val_ Accuracy 0.9859374761581421
2023-10-11 21:12:05.468076 Epoch 9, Training loss 0.06267683953046799,Train_accu
0.9799479246139526 Val_loss 0.4754170775413513, Val_ Accuracy 0.8678921461105347
2023-10-11 21:14:02.395725 Epoch 10, Training loss
0.20664146542549133,Train_accu 0.9410156011581421 Val_loss 0.25508445501327515,
Val_ Accuracy 0.9124693870544434
```

[36]:
```
plot_loss_accuracy(hist)
```



[37]:
```
summary(model, (3, 500, 500))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1        [-1, 16, 250, 250]             432
       BatchNorm2d-2        [-1, 16, 250, 250]              32
         Hardswish-3        [-1, 16, 250, 250]               0
            Conv2d-4        [-1, 16, 125, 125]             144
       BatchNorm2d-5        [-1, 16, 125, 125]              32
              ReLU-6        [-1, 16, 125, 125]               0
```

AdaptiveAvgPool2d-7	[-1, 16, 1, 1]	0
Conv2d-8	[-1, 8, 1, 1]	136
ReLU-9	[-1, 8, 1, 1]	0
Conv2d-10	[-1, 16, 1, 1]	144
Hardsigmoid-11	[-1, 16, 1, 1]	0
SqueezeExcitation-12	[-1, 16, 125, 125]	0
Conv2d-13	[-1, 16, 125, 125]	256
BatchNorm2d-14	[-1, 16, 125, 125]	32
InvertedResidual-15	[-1, 16, 125, 125]	0
Conv2d-16	[-1, 72, 125, 125]	1,152
BatchNorm2d-17	[-1, 72, 125, 125]	144
ReLU-18	[-1, 72, 125, 125]	0
Conv2d-19	[-1, 72, 63, 63]	648
BatchNorm2d-20	[-1, 72, 63, 63]	144
ReLU-21	[-1, 72, 63, 63]	0
Conv2d-22	[-1, 24, 63, 63]	1,728
BatchNorm2d-23	[-1, 24, 63, 63]	48
InvertedResidual-24	[-1, 24, 63, 63]	0
Conv2d-25	[-1, 88, 63, 63]	2,112
BatchNorm2d-26	[-1, 88, 63, 63]	176
ReLU-27	[-1, 88, 63, 63]	0
Conv2d-28	[-1, 88, 63, 63]	792
BatchNorm2d-29	[-1, 88, 63, 63]	176
ReLU-30	[-1, 88, 63, 63]	0
Conv2d-31	[-1, 24, 63, 63]	2,112
BatchNorm2d-32	[-1, 24, 63, 63]	48
InvertedResidual-33	[-1, 24, 63, 63]	0
Conv2d-34	[-1, 96, 63, 63]	2,304
BatchNorm2d-35	[-1, 96, 63, 63]	192
Hardswish-36	[-1, 96, 63, 63]	0
Conv2d-37	[-1, 96, 32, 32]	2,400
BatchNorm2d-38	[-1, 96, 32, 32]	192
Hardswish-39	[-1, 96, 32, 32]	0
AdaptiveAvgPool2d-40	[-1, 96, 1, 1]	0
Conv2d-41	[-1, 24, 1, 1]	2,328
ReLU-42	[-1, 24, 1, 1]	0
Conv2d-43	[-1, 96, 1, 1]	2,400
Hardsigmoid-44	[-1, 96, 1, 1]	0
SqueezeExcitation-45	[-1, 96, 32, 32]	0
Conv2d-46	[-1, 40, 32, 32]	3,840
BatchNorm2d-47	[-1, 40, 32, 32]	80
InvertedResidual-48	[-1, 40, 32, 32]	0
Conv2d-49	[-1, 240, 32, 32]	9,600
BatchNorm2d-50	[-1, 240, 32, 32]	480
Hardswish-51	[-1, 240, 32, 32]	0
Conv2d-52	[-1, 240, 32, 32]	6,000
BatchNorm2d-53	[-1, 240, 32, 32]	480
Hardswish-54	[-1, 240, 32, 32]	0

AdaptiveAvgPool2d-55	[-1, 240, 1, 1]	0
Conv2d-56	[-1, 64, 1, 1]	15,424
ReLU-57	[-1, 64, 1, 1]	0
Conv2d-58	[-1, 240, 1, 1]	15,600
Hardsigmoid-59	[-1, 240, 1, 1]	0
SqueezeExcitation-60	[-1, 240, 32, 32]	0
Conv2d-61	[-1, 40, 32, 32]	9,600
BatchNorm2d-62	[-1, 40, 32, 32]	80
InvertedResidual-63	[-1, 40, 32, 32]	0
Conv2d-64	[-1, 240, 32, 32]	9,600
BatchNorm2d-65	[-1, 240, 32, 32]	480
Hardswish-66	[-1, 240, 32, 32]	0
Conv2d-67	[-1, 240, 32, 32]	6,000
BatchNorm2d-68	[-1, 240, 32, 32]	480
Hardswish-69	[-1, 240, 32, 32]	0
AdaptiveAvgPool2d-70	[-1, 240, 1, 1]	0
Conv2d-71	[-1, 64, 1, 1]	15,424
ReLU-72	[-1, 64, 1, 1]	0
Conv2d-73	[-1, 240, 1, 1]	15,600
Hardsigmoid-74	[-1, 240, 1, 1]	0
SqueezeExcitation-75	[-1, 240, 32, 32]	0
Conv2d-76	[-1, 40, 32, 32]	9,600
BatchNorm2d-77	[-1, 40, 32, 32]	80
InvertedResidual-78	[-1, 40, 32, 32]	0
Conv2d-79	[-1, 120, 32, 32]	4,800
BatchNorm2d-80	[-1, 120, 32, 32]	240
Hardswish-81	[-1, 120, 32, 32]	0
Conv2d-82	[-1, 120, 32, 32]	3,000
BatchNorm2d-83	[-1, 120, 32, 32]	240
Hardswish-84	[-1, 120, 32, 32]	0
AdaptiveAvgPool2d-85	[-1, 120, 1, 1]	0
Conv2d-86	[-1, 32, 1, 1]	3,872
ReLU-87	[-1, 32, 1, 1]	0
Conv2d-88	[-1, 120, 1, 1]	3,960
Hardsigmoid-89	[-1, 120, 1, 1]	0
SqueezeExcitation-90	[-1, 120, 32, 32]	0
Conv2d-91	[-1, 48, 32, 32]	5,760
BatchNorm2d-92	[-1, 48, 32, 32]	96
InvertedResidual-93	[-1, 48, 32, 32]	0
Conv2d-94	[-1, 144, 32, 32]	6,912
BatchNorm2d-95	[-1, 144, 32, 32]	288
Hardswish-96	[-1, 144, 32, 32]	0
Conv2d-97	[-1, 144, 32, 32]	3,600
BatchNorm2d-98	[-1, 144, 32, 32]	288
Hardswish-99	[-1, 144, 32, 32]	0
AdaptiveAvgPool2d-100	[-1, 144, 1, 1]	0
Conv2d-101	[-1, 40, 1, 1]	5,800
ReLU-102	[-1, 40, 1, 1]	0

Conv2d-103	[-1, 144, 1, 1]	5,904
Hardsigmoid-104	[-1, 144, 1, 1]	0
SqueezeExcitation-105	[-1, 144, 32, 32]	0
Conv2d-106	[-1, 48, 32, 32]	6,912
BatchNorm2d-107	[-1, 48, 32, 32]	96
InvertedResidual-108	[-1, 48, 32, 32]	0
Conv2d-109	[-1, 288, 32, 32]	13,824
BatchNorm2d-110	[-1, 288, 32, 32]	576
Hardswish-111	[-1, 288, 32, 32]	0
Conv2d-112	[-1, 288, 16, 16]	7,200
BatchNorm2d-113	[-1, 288, 16, 16]	576
Hardswish-114	[-1, 288, 16, 16]	0
AdaptiveAvgPool2d-115	[-1, 288, 1, 1]	0
Conv2d-116	[-1, 72, 1, 1]	20,808
ReLU-117	[-1, 72, 1, 1]	0
Conv2d-118	[-1, 288, 1, 1]	21,024
Hardsigmoid-119	[-1, 288, 1, 1]	0
SqueezeExcitation-120	[-1, 288, 16, 16]	0
Conv2d-121	[-1, 96, 16, 16]	27,648
BatchNorm2d-122	[-1, 96, 16, 16]	192
InvertedResidual-123	[-1, 96, 16, 16]	0
Conv2d-124	[-1, 576, 16, 16]	55,296
BatchNorm2d-125	[-1, 576, 16, 16]	1,152
Hardswish-126	[-1, 576, 16, 16]	0
Conv2d-127	[-1, 576, 16, 16]	14,400
BatchNorm2d-128	[-1, 576, 16, 16]	1,152
Hardswish-129	[-1, 576, 16, 16]	0
AdaptiveAvgPool2d-130	[-1, 576, 1, 1]	0
Conv2d-131	[-1, 144, 1, 1]	83,088
ReLU-132	[-1, 144, 1, 1]	0
Conv2d-133	[-1, 576, 1, 1]	83,520
Hardsigmoid-134	[-1, 576, 1, 1]	0
SqueezeExcitation-135	[-1, 576, 16, 16]	0
Conv2d-136	[-1, 96, 16, 16]	55,296
BatchNorm2d-137	[-1, 96, 16, 16]	192
InvertedResidual-138	[-1, 96, 16, 16]	0
Conv2d-139	[-1, 576, 16, 16]	55,296
BatchNorm2d-140	[-1, 576, 16, 16]	1,152
Hardswish-141	[-1, 576, 16, 16]	0
Conv2d-142	[-1, 576, 16, 16]	14,400
BatchNorm2d-143	[-1, 576, 16, 16]	1,152
Hardswish-144	[-1, 576, 16, 16]	0
AdaptiveAvgPool2d-145	[-1, 576, 1, 1]	0
Conv2d-146	[-1, 144, 1, 1]	83,088
ReLU-147	[-1, 144, 1, 1]	0
Conv2d-148	[-1, 576, 1, 1]	83,520
Hardsigmoid-149	[-1, 576, 1, 1]	0
SqueezeExcitation-150	[-1, 576, 16, 16]	0

```
          Conv2d-151           [-1, 96, 16, 16]          55,296
     BatchNorm2d-152           [-1, 96, 16, 16]             192
InvertedResidual-153           [-1, 96, 16, 16]               0
          Conv2d-154          [-1, 576, 16, 16]          55,296
     BatchNorm2d-155          [-1, 576, 16, 16]           1,152
       Hardswish-156          [-1, 576, 16, 16]               0
AdaptiveAvgPool2d-157            [-1, 576, 1, 1]               0
          Linear-158                 [-1, 1024]         590,848
       Hardswish-159                 [-1, 1024]               0
         Dropout-160                 [-1, 1024]               0
          Linear-161                 [-1, 1000]       1,025,000
================================================================
Total params: 2,542,856
Trainable params: 2,542,856
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 2.86
Forward/backward pass size (MB): 176.26
Params size (MB): 9.70
Estimated Total Size (MB): 188.82
----------------------------------------------------------------
```

[38]:
```
weights_path = r"D:\Scaler\NinjaCart_Project\Weights\Mobilenet.pth"
torch.save(model.state_dict(), weights_path)
```

1.1.3 Testing the best model

Test_Data

[7]:
```
transform = transforms.Compose([
        transforms.Resize(500),
        transforms.CenterCrop(500),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])


Test_folder = ImageFolder(test_path,transform=transform)
test_data = DataLoader(Test_folder,batch_size=64,shuffle=False,num_workers=4)
```

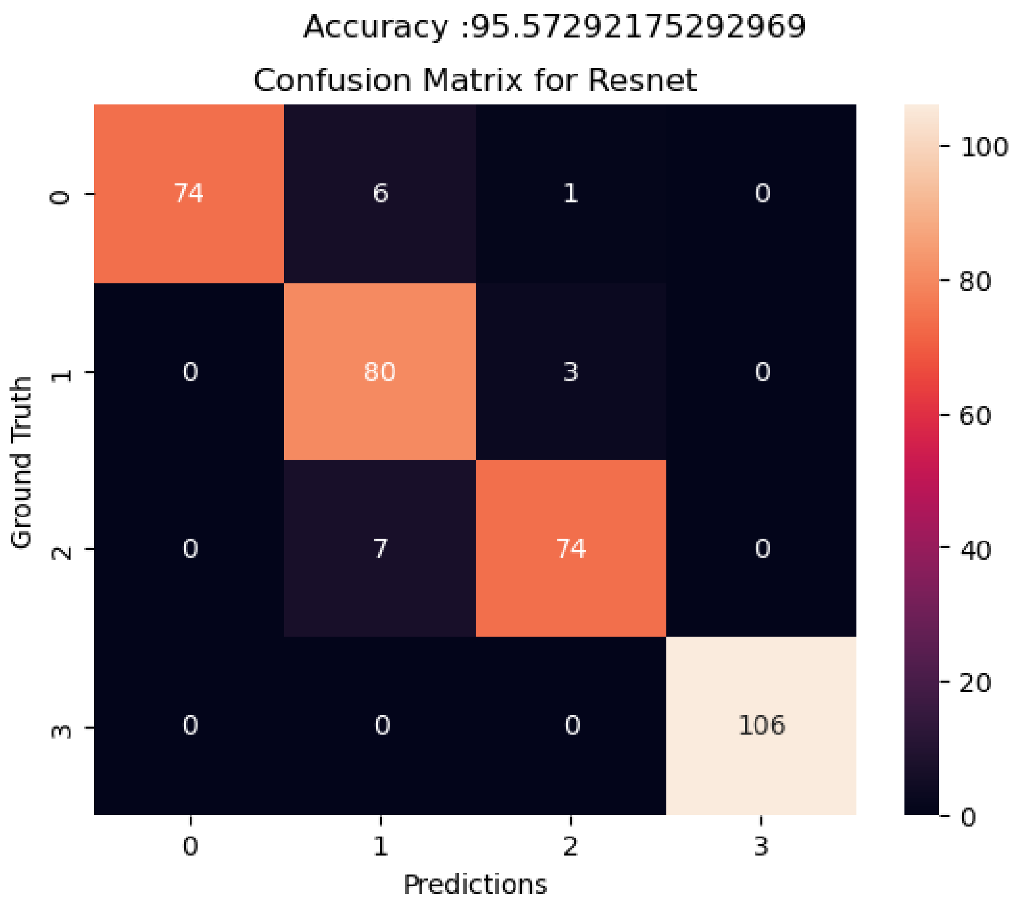
[8]:
```
acc = torchmetrics.Accuracy(task="multiclass",num_classes=4)
```

Loading all trained model

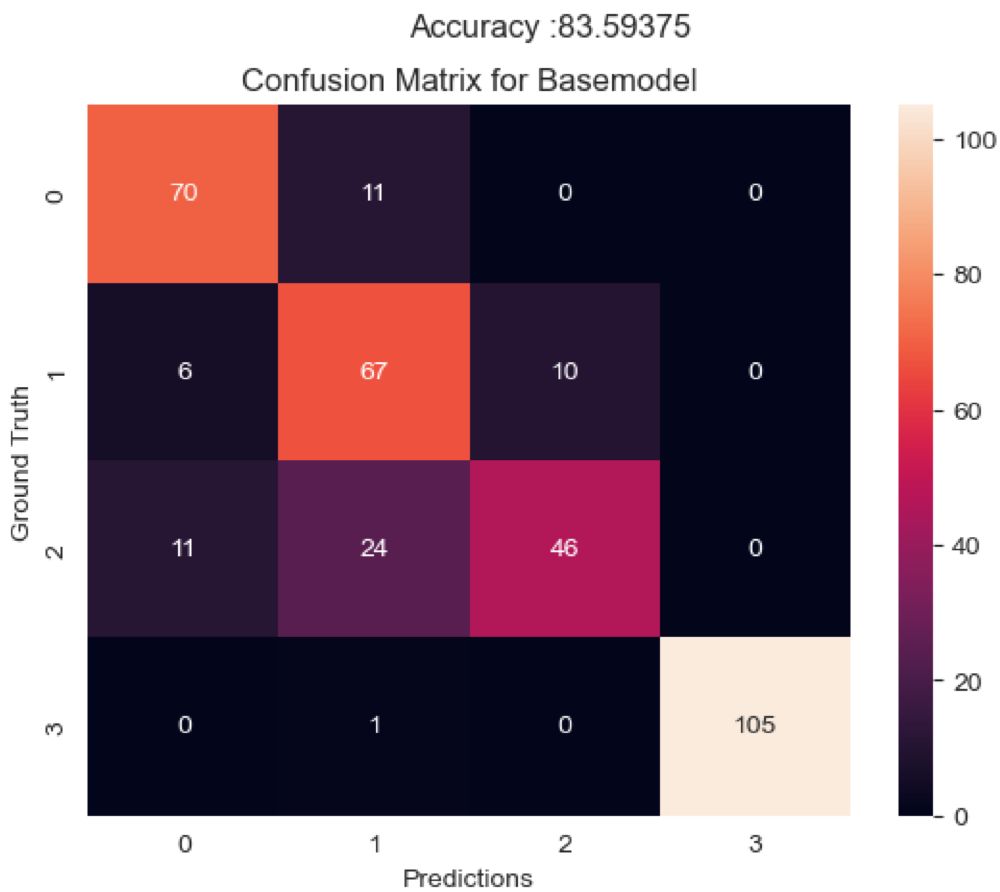
[12]:
```
def Testing(model,data,model_name):
    predictions = []
    ground_truth = []
    accuracy = []
    with torch.no_grad():
```

```
        for img, label in data:
            out = model(img)
            pred = [i.numpy().argmax() for i in out]
            accuracy.append(acc(out.argmax(dim=1),label))
            predictions = np.concatenate((predictions,pred))
            ground_truth = np.concatenate((ground_truth,label))
    confmat = torchmetrics.ConfusionMatrix(task="multiclass", num_classes=4)
    cm = confmat(torch.tensor(predictions),torch.tensor(ground_truth))
    plt.title(f"Confusion Matrix for {model_name}")
    plt.suptitle(f"Accuracy :{torch.tensor(accuracy).mean()*100}")
    sns.heatmap(cm,annot=True,fmt='d')
    plt.xlabel("Predictions")
    plt.ylabel("Ground Truth")
    plt.show()
```

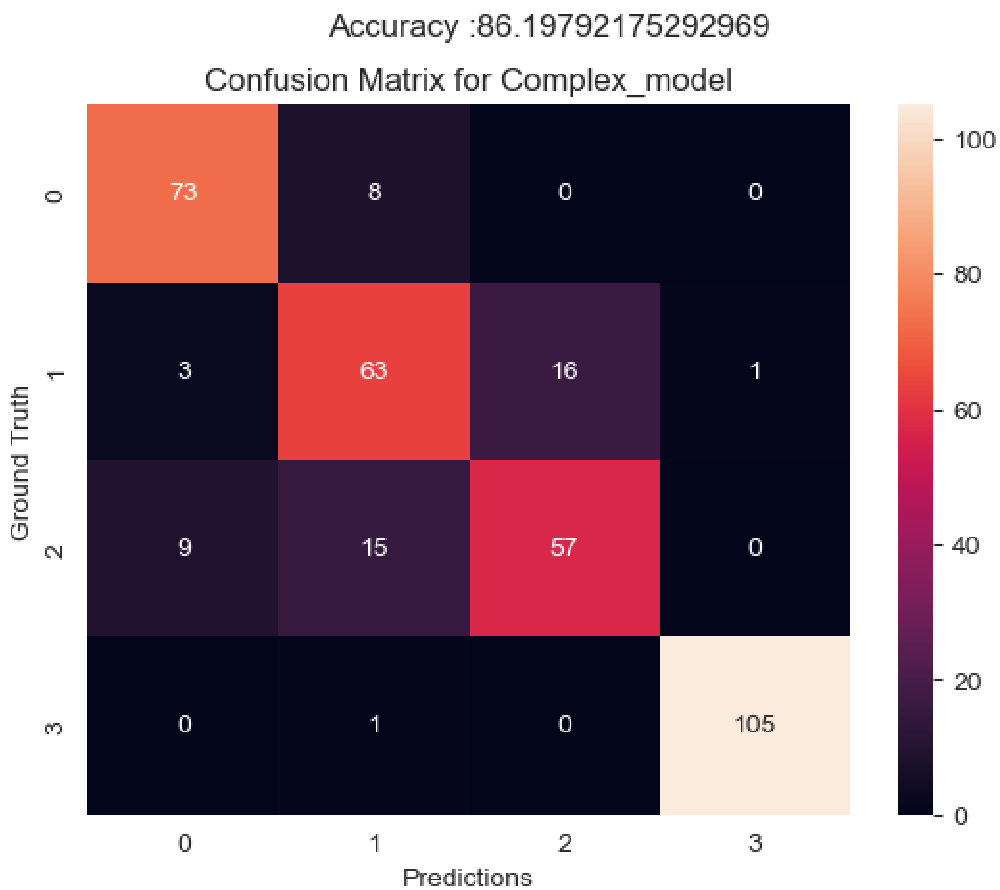
[13]:
```
Resnet_model = models.resnet50()
modelOutputFeats = Resnet_model.fc.in_features
Resnet_model.fc = nn.Linear(modelOutputFeats, 4)
Resnet_model.load_state_dict(torch.load(r"D:
 ↪\Scaler\NinjaCart_Project\Weights\ResNetmodel.pth"))
Resnet_model.eval()


Testing(model=Resnet_model,data=test_data,model_name="Resnet")
```



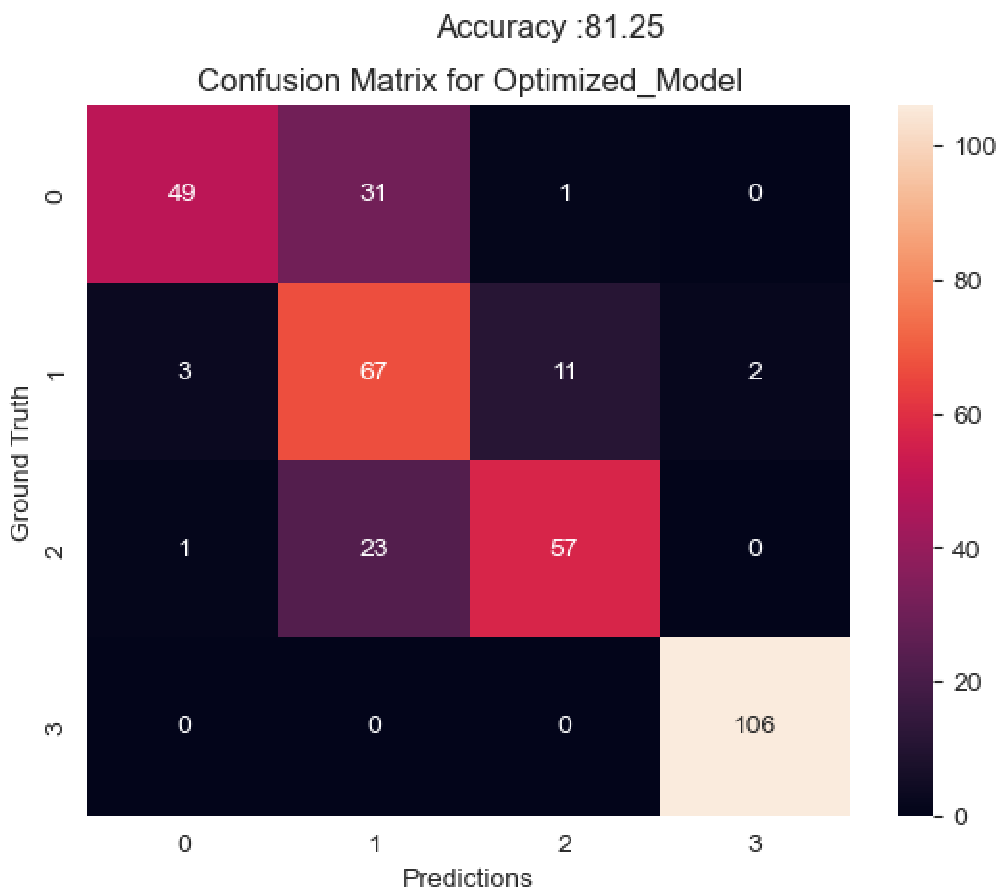
[24]:
```
Base_model = BaseCnn(num_channel=3)
Base_model.load_state_dict(torch.load(r"D:
↪\Scaler\NinjaCart_Project\Weights\Basemodel.pth"))
Base_model.eval()

Testing(model=Base_model,data=test_data,model_name="Basemodel")
```



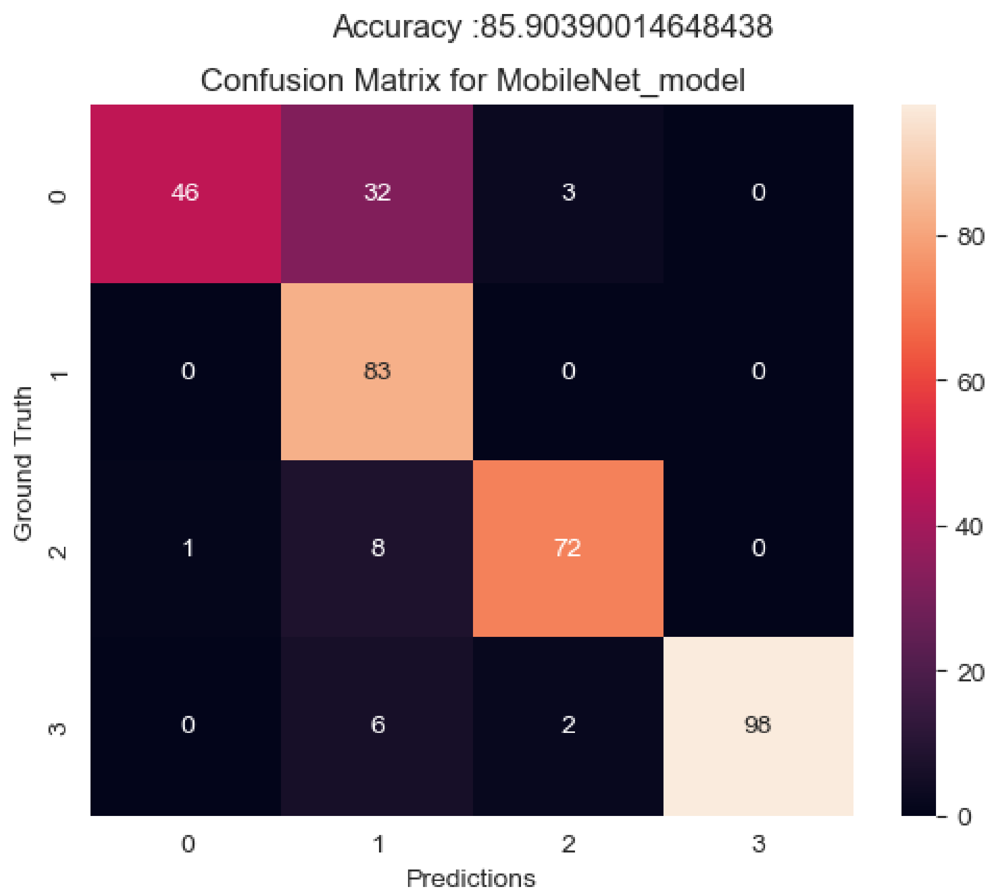
[25]:
```
Complex_model = ComplexCnn(num_channel=3)
Complex_model.load_state_dict(torch.load(r"D:
    ↪\Scaler\NinjaCart_Project\Weights\Complexmodel.pth"))
Complex_model.eval()

Testing(model=Complex_model,data=test_data,model_name="Complex_model")
```



[26]:
```
Optimized_Model = ComplexCnn1(num_channel=3)
Optimized_Model.load_state_dict(torch.load(r"D:
  ↪\Scaler\NinjaCart_Project\Weights\Complexmodel_with_overfitting.pth"))
Optimized_Model.eval()

Testing(model=Optimized_Model,data=test_data,model_name="Optimized_Model")
```



[28]:
```
MobileNet_model = models.mobilenet_v3_small()
MobileNet_model.classifier[3].out_features = 4
MobileNet_model.load_state_dict(torch.load(r"D:
  ↪\Scaler\NinjaCart_Project\Weights\Mobilenet.pth"))
MobileNet_model.eval()

Testing(model=MobileNet_model,data=test_data,model_name="MobileNet_model")
```



Random image samples prediction

[30]:
```
classes = ["indian_market","onion","potato","tomato"]
count = 0
with torch.no_grad():
    for i,(img, label) in enumerate(test_data):
        if i == count:
            image = img[random.randint(0,64)]
            out = Resnet_model(image.unsqueeze(0))
            pred = out.argmax()
            plt.imshow(image.numpy().transpose(1,2,0))
            plt.title(f"predicted:{classes[pred]} \n real:{classes[label[random.
 ↪randint(0,64)]]}")
            plt.axis("off")
            plt.tight_layout()
            plt.show()
            count+=1
        else:
            break
```

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

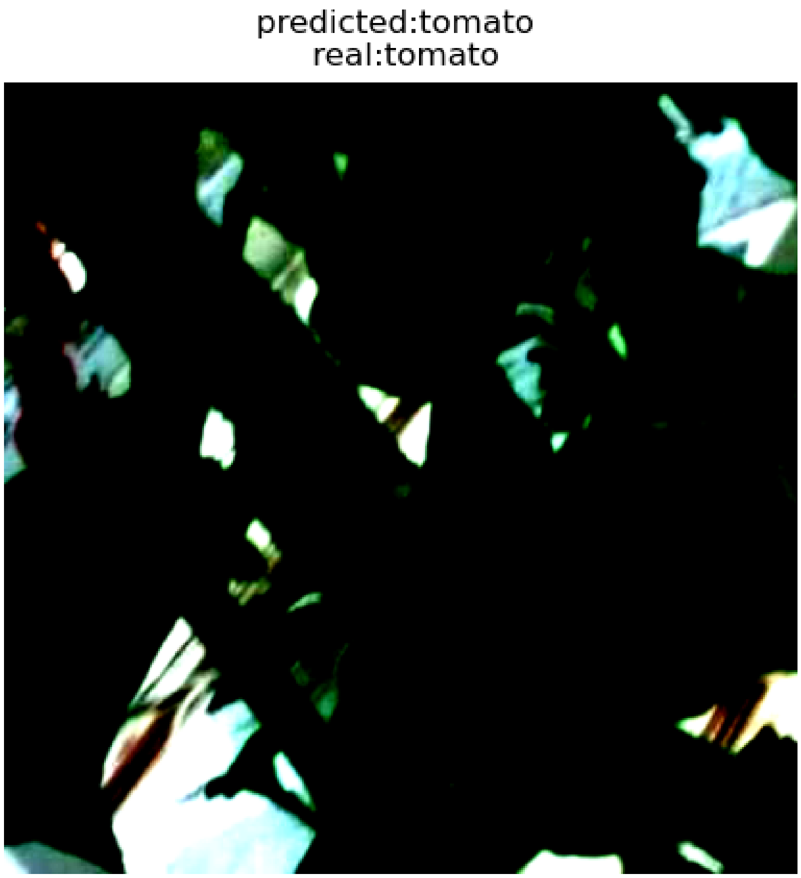

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```

predicted:potato
real:onion

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

predicted:potato
real:potato

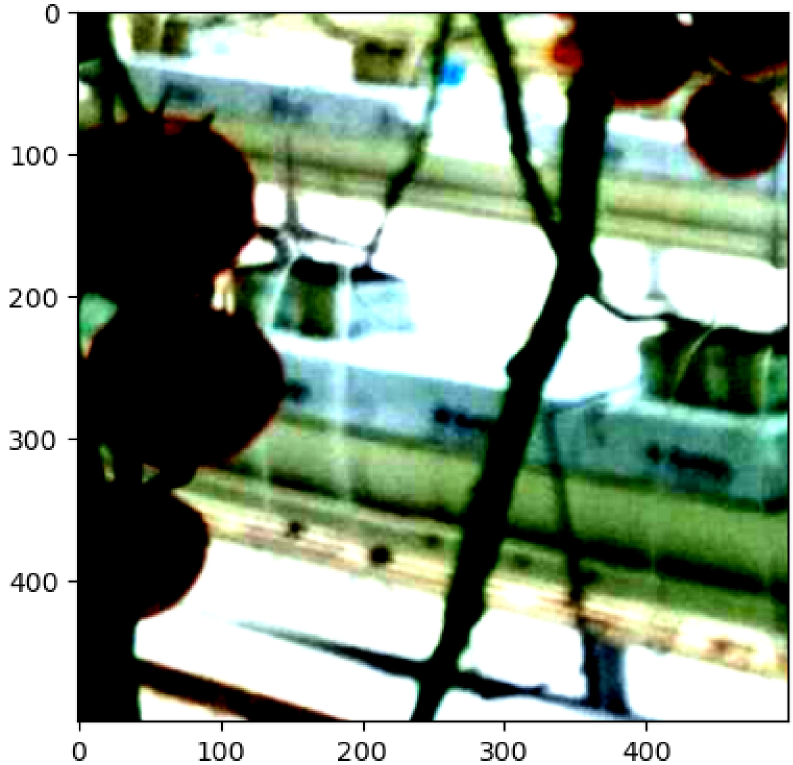
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```




```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
d:\Scaler\NinjaCart_Project\ninjacart_data\NinjaCart.ipynb Cell 61 line 1
      <a href='vscode-notebook-cell:/d%3A/Scaler/NinjaCart_Project/
 ↪ninjacart_data/NinjaCart.ipynb#Y114sZmlsZQ%3D%3D?line=7'>8</a> pred = out.
 ↪argmax()
      <a href='vscode-notebook-cell:/d%3A/Scaler/NinjaCart_Project/
 ↪ninjacart_data/NinjaCart.ipynb#Y114sZmlsZQ%3D%3D?line=8'>9</a> plt.
 ↪imshow(image.numpy().transpose(1,2,0))
---> <a href='vscode-notebook-cell:/d%3A/Scaler/NinjaCart_Project/ninjacart_data/
 ↪NinjaCart.ipynb#Y114sZmlsZQ%3D%3D?line=9'>10</a> plt.title(f"predicted:
 ↪{classes[pred]} \n real:{classes[label[random.randint(0,64)]]}")
     <a href='vscode-notebook-cell:/d%3A/Scaler/NinjaCart_Project/ninjacart_data/
 ↪NinjaCart.ipynb#Y114sZmlsZQ%3D%3D?line=10'>11</a> plt.axis("off")
     <a href='vscode-notebook-cell:/d%3A/Scaler/NinjaCart_Project/ninjacart_data/
 ↪NinjaCart.ipynb#Y114sZmlsZQ%3D%3D?line=11'>12</a> plt.tight_layout()
```

```
IndexError: index 37 is out of bounds for dimension 0 with size 31
```



1.1.4 Summary & Insights

- We have tried different types of models from scratch
- But when compared to basic models, the pretrained models are doing very well in the performance because of its core architecture
- our model can also tried to optimize the performance well but not thatv much

[ ]:

```

```