Vegetable Classifier

Neural Networks and Genetic Algorithms

Table of Contents

1. Introduction

To complete the assignment for my Neural Networks and Genetic Algorithms course (NNGA), I chose to create a Neural Network that would classify images of vegetables. The model can discern between 15 different types of vegetables.

The dataset used is found online and a link to that website will be found in the “References” section at the end of this report. This dataset contains 1000 training photos for each one of the 15 categories mentioned, with 200 for validation and 200 for testing. In total, there are 21000 images with equal proportions and image resolution is 224×224 in *.jpg format.

2. Libraries

This project is written in Python and we use the following libraries:

- PyTorch – Building the model of our neural network and its training
    - torch – Core library
    - torch.nn – Loss functions and other modules
    - torch.optim – Optimization algorithms
    - torchvision.transforms – Preprocessing of the images before training
    - torch.utils.data – Utilities
- Scikit-learn – Evaluating the performance of the trained model
    - sklearn.metrics – Confusion matrix and accuracy score
- Matplotlib – Plotting the training and validation losses for each epoch
    - matplotlib.pyplot – Graph plotting
- Seaborn – Visualization of the confusion matrix
    - seaborn – Heat map for the confusion matrix

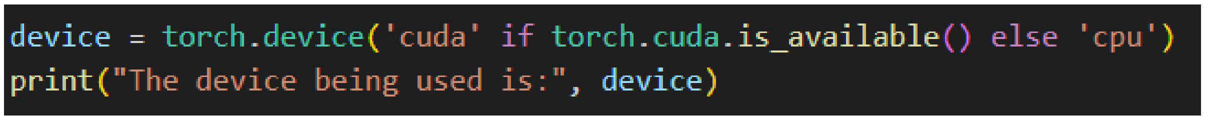
3. About the Model

To classify vegetable images, I utilized a Convolutional Neural Network (CNN). The architecture comprises four convolutional layers, each followed by a ReLU activation and max-pooling, along with four fully connected layers. Dropout layers were incorporated to mitigate overfitting.

- Convolutional Layers: These layers extract features from input images, essential for the classification task.
- Dropout Layers: During training, these layers randomly set a fraction of inputs to zero to prevent the model from overfitting on the training data.
- Fully Connected Layers: The final layers responsible for categorizing the input data into the correct vegetable class.

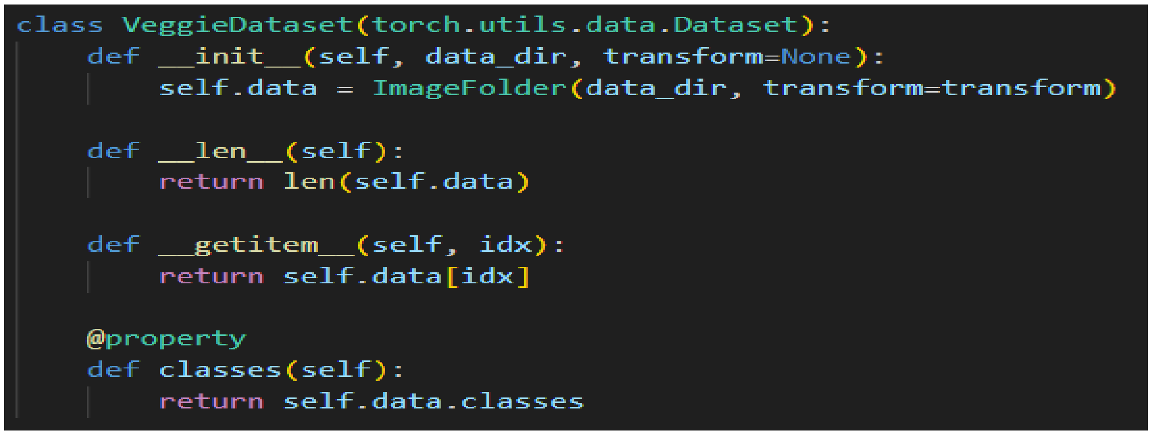
4. Code Description

The implementation begins with a check for CUDA cores for GPU acceleration. If a GPU is available, it is utilized for training; otherwise, the CPU is used as a fallback.



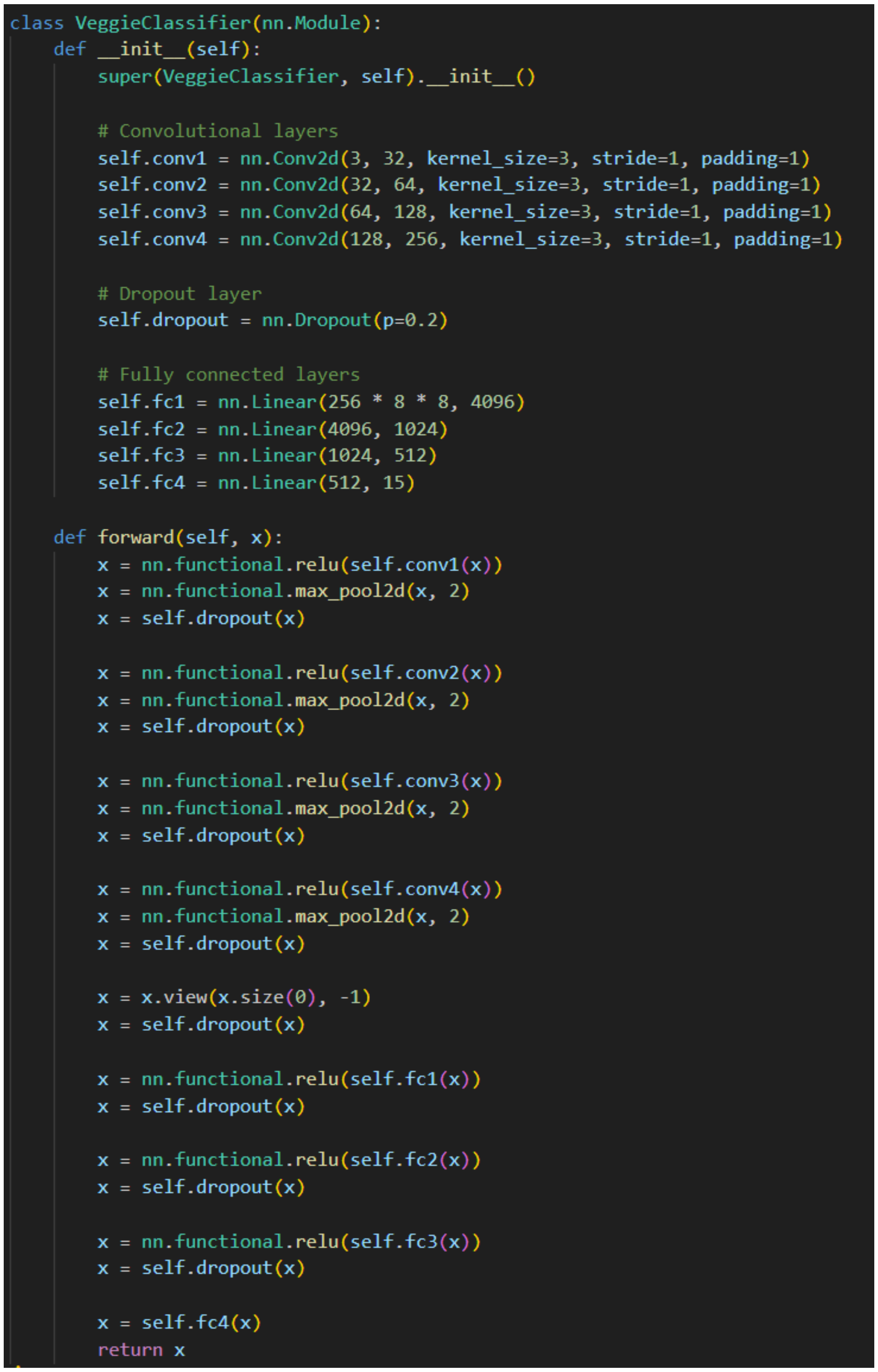
```python
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print("The device being used is:", device)
```

A custom class named VeggieDataset was created to load the images of vegetables from the directory. This class derived from PyTorch's 'torch.utils.data.Dataset' class, ensuring compatibility with PyTorch's DataLoader.



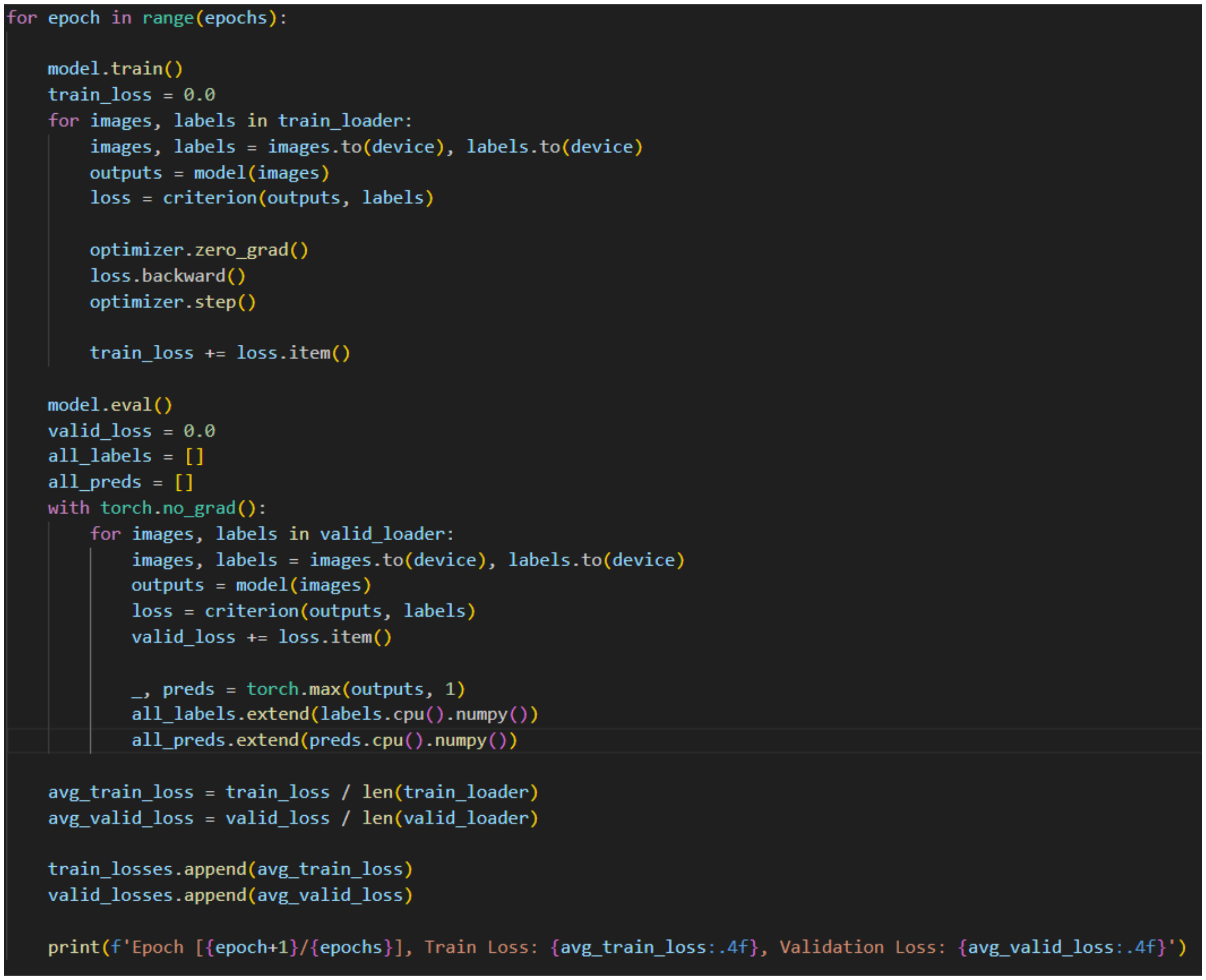
```python
class VeggieDataset(torch.utils.data.Dataset):
    def __init__(self, data_dir, transform=None):
        self.data = ImageFolder(data_dir, transform=transform)

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        return self.data[idx]

    @property
    def classes(self):
        return self.data.classes
```

The VeggieClassifier class defines the CNN architecture. This includes the convolutional layers, dropout layers, and fully connected layers. The forward function directs the flow of data through the network.



```python
class VeggieClassifier(nn.Module):
    def __init__(self):
        super(VeggieClassifier, self).__init__()

        # Convolutional layers
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1)
        self.conv4 = nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1)

        # Dropout layer
        self.dropout = nn.Dropout(p=0.2)

        # Fully connected layers
        self.fc1 = nn.Linear(256 * 8 * 8, 4096)
        self.fc2 = nn.Linear(4096, 1024)
        self.fc3 = nn.Linear(1024, 512)
        self.fc4 = nn.Linear(512, 15)

    def forward(self, x):
        x = nn.functional.relu(self.conv1(x))
        x = nn.functional.max_pool2d(x, 2)
        x = self.dropout(x)

        x = nn.functional.relu(self.conv2(x))
        x = nn.functional.max_pool2d(x, 2)
        x = self.dropout(x)

        x = nn.functional.relu(self.conv3(x))
        x = nn.functional.max_pool2d(x, 2)
        x = self.dropout(x)

        x = nn.functional.relu(self.conv4(x))
        x = nn.functional.max_pool2d(x, 2)
        x = self.dropout(x)

        x = x.view(x.size(0), -1)
        x = self.dropout(x)

        x = nn.functional.relu(self.fc1(x))
        x = self.dropout(x)

        x = nn.functional.relu(self.fc2(x))
        x = self.dropout(x)

        x = nn.functional.relu(self.fc3(x))
        x = self.dropout(x)

        x = self.fc4(x)
        return x
```

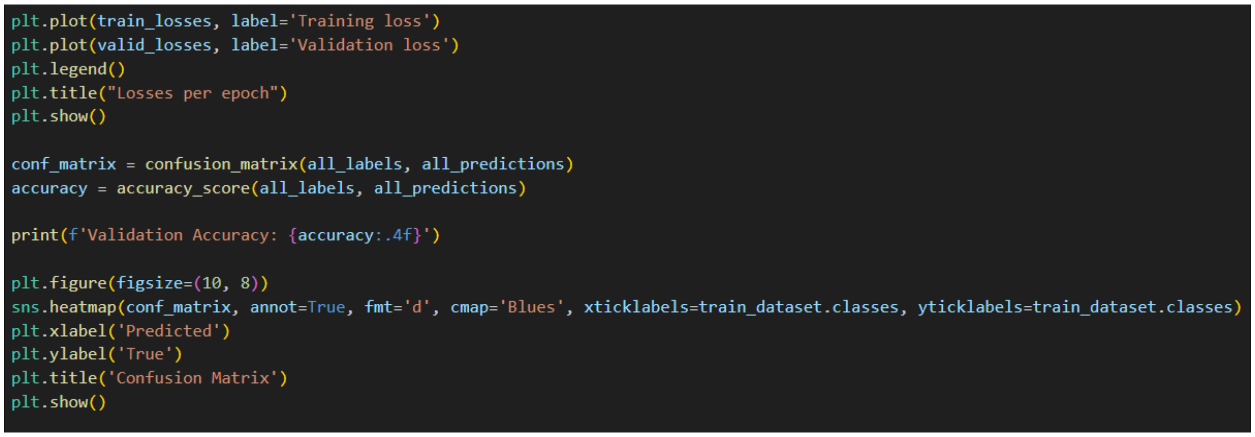
The model was trained for 30 epochs using the ADAM optimizer, known for its effectiveness in managing large datasets.



```python
for epoch in range(epochs):

    model.train()
    train_loss = 0.0
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        loss = criterion(outputs, labels)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        train_loss += loss.item()

    model.eval()
    valid_loss = 0.0
    all_labels = []
    all_preds = []
    with torch.no_grad():
        for images, labels in valid_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            loss = criterion(outputs, labels)
            valid_loss += loss.item()

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

    avg_train_loss = train_loss / len(train_loader)
    avg_valid_loss = valid_loss / len(valid_loader)

    train_losses.append(avg_train_loss)
    valid_losses.append(avg_valid_loss)

    print(f'Epoch [{epoch+1}/{epochs}], Train Loss: {avg_train_loss:.4f}, Validation Loss: {avg_valid_loss:.4f}')
```

5. Results Analysis

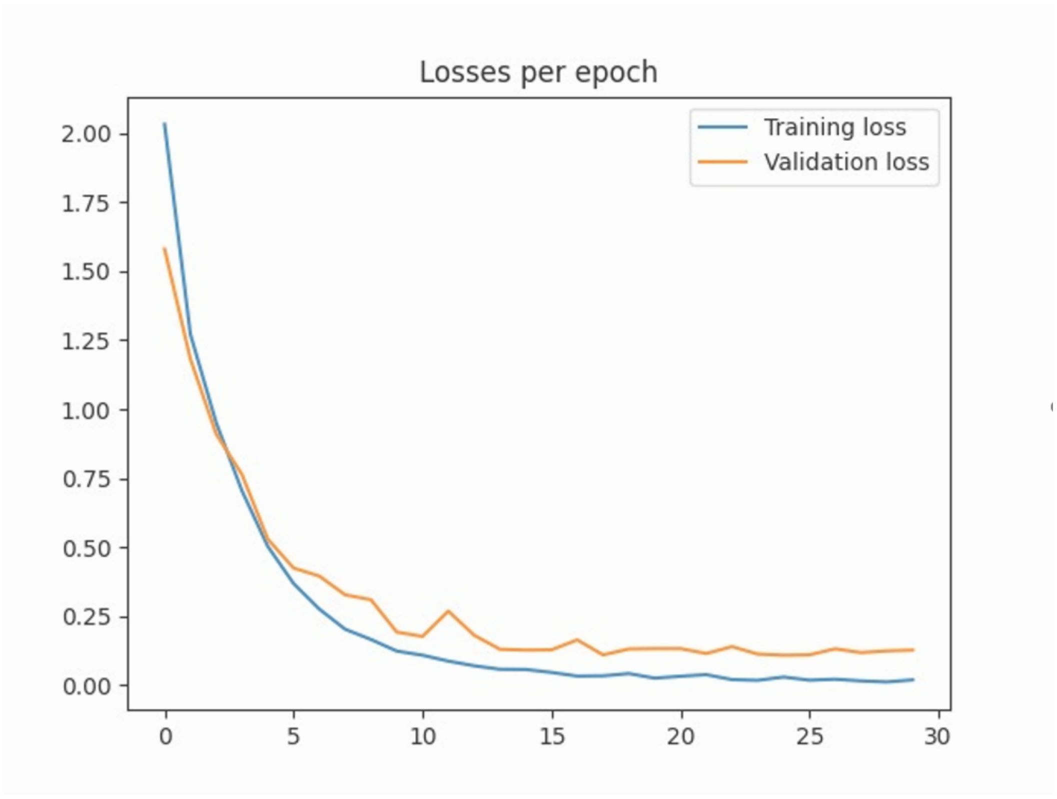
After 30 epochs, the model's performance was evaluated on the validation set. The loss values for both training and validation datasets were plotted to observe the model's learning behavior. Accuracy was calculated based on the confusion matrix generated from the predictions.

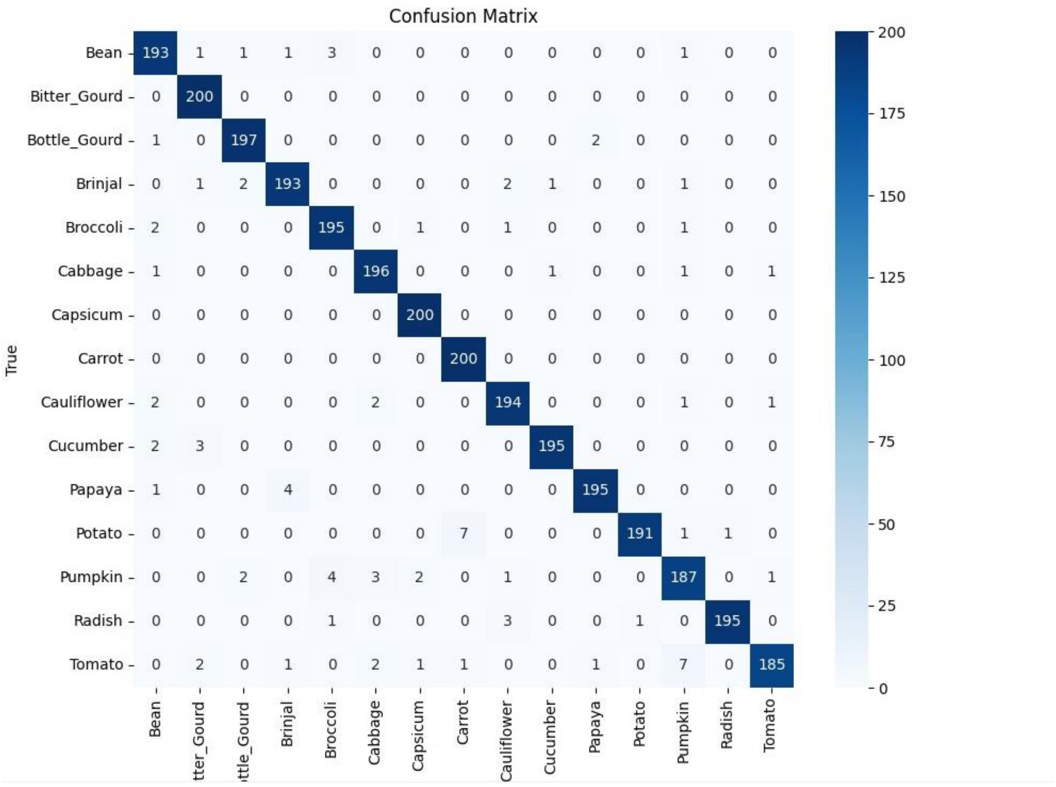


```
plt.plot(train_losses, label='Training loss')
plt.plot(valid_losses, label='Validation loss')
plt.legend()
plt.title("Losses per epoch")
plt.show()

conf_matrix = confusion_matrix(all_labels, all_predictions)
accuracy = accuracy_score(all_labels, all_predictions)

print(f'Validation Accuracy: {accuracy:.4f}')

plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=train_dataset.classes, yticklabels=train_dataset.classes)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

Observations:

- The model achieved a validation accuracy of 97%, which was the highest observed accuracy across multiple runs. Validation accuracies ranged between 93% and 97% in different trials.
- The training loss consistently decreased over the epochs, reaching near 0, while the validation loss plateaued and showed slight fluctuations after about 5-10 epochs.
- The confusion matrix shows that the model performed well across most categories.





	Bean	tter_Gourd	ttle_Gourd	Brinjal	Broccoli	Cabbage	Capsicum	Carrot	Cauliflower	Cucumber	Papaya	Potato	Pumpkin	Radish	Tomato
Bean	193	1	1	1	3	0	0	0	0	0	0	0	1	0	0
Bitter_Gourd	0	200	0	0	0	0	0	0	0	0	0	0	0	0	0
Bottle_Gourd	1	0	197	0	0	0	0	0	0	0	2	0	0	0	0
Brinjal	0	1	2	193	0	0	0	0	2	1	0	0	1	0	0
Broccoli	2	0	0	0	195	0	1	0	1	0	0	0	1	0	0
Cabbage	1	0	0	0	0	196	0	0	0	1	0	0	1	0	1
Capsicum	0	0	0	0	0	0	200	0	0	0	0	0	0	0	0
Carrot	0	0	0	0	0	0	0	200	0	0	0	0	0	0	0
Cauliflower	2	0	0	0	0	2	0	0	194	0	0	0	1	0	1
Cucumber	2	3	0	0	0	0	0	0	0	195	0	0	0	0	0
Papaya	1	0	0	4	0	0	0	0	0	0	195	0	0	0	0
Potato	0	0	0	0	0	0	0	7	0	0	0	191	1	1	0
Pumpkin	0	0	2	0	4	3	2	0	1	0	0	0	187	0	1
Radish	0	0	0	0	1	0	0	0	3	0	0	1	0	195	0
Tomato	0	2	0	1	0	2	1	1	0	0	1	0	7	0	185

6. Conclusions

The vegetable classification model performed well, achieving good results across multiple runs. These results show that the model is effective, even though we had slight overfitting when the validation loss fluctuated. Taking a good look at the confusion matrix, we can see that some of the vegetables have been misplaced, but that can corrected in a future implementation of this project.

7. References

Dataset: https://www.kaggle.com/code/chitwanmanchanda/vegetable-image-classification-using-cnn/notebook

8. Code

```python
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, accuracy_score
from torch.utils.data import DataLoader
from torchvision.datasets import ImageFolder

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print("The device being used is:", device)

class VeggieDataset(torch.utils.data.Dataset):
    def __init__(self, data_dir, transform=None):
        self.data = ImageFolder(data_dir, transform=transform)

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        return self.data[idx]

    @property
    def classes(self):
        return self.data.classes

class VeggieClassifier(nn.Module):
    def __init__(self):
        super(VeggieClassifier, self).__init__()

        # Convolutional layers
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1)
        self.conv4 = nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1)

        # Dropout layer
        self.dropout = nn.Dropout(p=0.2)

        # Fully connected layers
        self.fc1 = nn.Linear(256 * 8 * 8, 4096)
        self.fc2 = nn.Linear(4096, 1024)
        self.fc3 = nn.Linear(1024, 512)
        self.fc4 = nn.Linear(512, 15)

    def forward(self, x):
        x = nn.functional.relu(self.conv1(x))
        x = nn.functional.max_pool2d(x, 2)
        x = self.dropout(x)

        x = nn.functional.relu(self.conv2(x))
        x = nn.functional.max_pool2d(x, 2)
        x = self.dropout(x)

        x = nn.functional.relu(self.conv3(x))
        x = nn.functional.max_pool2d(x, 2)
        x = self.dropout(x)

        x = nn.functional.relu(self.conv4(x))
        x = nn.functional.max_pool2d(x, 2)
        x = self.dropout(x)

        x = x.view(x.size(0), -1)
        x = self.dropout(x)

        x = nn.functional.relu(self.fc1(x))
        x = self.dropout(x)

        x = nn.functional.relu(self.fc2(x))
        x = self.dropout(x)

        x = nn.functional.relu(self.fc3(x))
        x = self.dropout(x)

        x = self.fc4(x)
        return x

transform = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.ToTensor(),
])

train_dataset_folder = './dataset/train'
train_dataset = VeggieDataset(train_dataset_folder, transform=transform)
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
```

```

valid_dataset_folder = './dataset/valid'
valid_dataset = VeggieDataset(valid_dataset_folder, transform=transform)
valid_loader = DataLoader(valid_dataset, batch_size=64, shuffle=False)

model = VeggieClassifier().to(device)

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.0001)

epochs = 30

train_losses = []
valid_losses = []

for epoch in range(epochs):

    model.train()
    train_loss = 0.0
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        loss = criterion(outputs, labels)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        train_loss += loss.item()

    model.eval()
    valid_loss = 0.0
    all_labels = []
    all_predictions = []
    with torch.no_grad():
        for images, labels in valid_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            loss = criterion(outputs, labels)
            valid_loss += loss.item()

            _, predictions = torch.max(outputs, 1)
            all_labels.extend(labels.cpu().numpy())
            all_predictions.extend(predictions.cpu().numpy())

```

```python
        avg_train_loss = train_loss / len(train_loader)
        avg_valid_loss = valid_loss / len(valid_loader)

        train_losses.append(avg_train_loss)
        valid_losses.append(avg_valid_loss)

        print(f'Epoch [{epoch+1}/{epochs}], Train Loss: {avg_train_loss:.4f}, Validation Loss: {avg_valid_loss:.4f}')

    plt.plot(train_losses, label='Training loss')
    plt.plot(valid_losses, label='Validation loss')
    plt.legend()
    plt.title("Losses per epoch")
    plt.show()

    conf_matrix = confusion_matrix(all_labels, all_predictions)
    accuracy = accuracy_score(all_labels, all_predictions)

    print(f'Validation Accuracy: {accuracy:.4f}')

    plt.figure(figsize=(10, 8))
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=train_dataset.classes, yticklabels=train_dataset.classes)
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.title('Confusion Matrix')
    plt.show()
```