dl_project

May 19, 2024

1 The aim of this project is to build a deep neural network capable of analyzing images captured by smartphones to quickly and accurately identify crop diseases.

We are analyzing 54,306 images of plant leaves, which are categorized into 38 class labels. Each class label corresponds to a crop-disease pair, and our goal is to predict this pair from the image of the plant leaf. These images are sourced from the dataset available at the following repository: https://github.com/digitalepidemiologylab/plantvillage_deeplearning_paper_dataset.

To train our AI-based image recognition system, we will utilize this dataset. In all our experiments, we utilize three different versions of the PlantVillage dataset. We begin with the original dataset in color, then we explore a grayscale version, and finally, we conduct our experiments on a version where the leaves are segmented. This approach allows us to assess the performance and robustness of our image recognition system in various contexts. We analyze how variations such as color, grayscale, and leaf segmentation can impact the model's results. By understanding how our system behaves under these different conditions, we can better evaluate its ability to generalize and operate effectively in real-world environments. These three versions of the data are already available via the above-mentioned link.

The different of crop disease types used in this project :

0: Grape___healthy

1: Peach___Bacterial_spot

2: Apple___healthy

3: Orange___Haunglongbing_(Citrus_greening)

4: Corn_(maize)___healthy

5: Tomato___Septoria_leaf_spot

6: Tomato___healthy

7: Corn_(maize)__Common_rust$

8: Tomato___Early_blight

9: Potato___Late_blight

10: Peach___healthy

11: Corn_(maize)___Northern_Leaf_Blight

12: Blueberry___healthy

13: Grape___Leaf_blight_(Isariopsis_Leaf_Spot)

14: Tomato___Leaf_Mold

15: Soybean___healthy

16: Cherry_(including_sour)___healthy

17: Tomato___Spider_mites Two-spotted_spider_mite

18: Potato___healthy

19: Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot

20: Cherry_(including_sour)___Powdery_mildew

21: Apple___Cedar_apple_rust

22: Squash___Powdery_mildew

23: Tomato___Late_blight

24: Grape___Black_rot

25: Pepper,_bell___healthy

26: Tomato___Target_Spot

27: Apple___Black_rot

28: Tomato___Bacterial_spot

29: Strawberry___healthy

30: Pepper,_bell___Bacterial_spot

31: Raspberry___healthy

32: Tomato___Tomato_Yellow_Leaf_Curl_Virus

33: Apple___Apple_scab

34: Potato___Early_blight

35: Tomato___Tomato_mosaic_virus

36: Strawberry___Leaf_scorch

37: Grape___Esca_(Black_Measles)

[1]:
```
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import torch.utils.data as data
import matplotlib.pyplot as plt
```

```
from collections import defaultdict
import seaborn as sns

from PIL import Image, ImageFile
from torchvision import datasets, transforms, utils
from torch.utils.data import ConcatDataset
from torch.utils.data import DataLoader

from sklearn import manifold
from sklearn import decomposition
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

import subprocess
import shutil

import os
import random
import time
```

#Understanding the data

[2]:
```
SEED = 1

random.seed(SEED)
np.random.seed(SEED)
torch.manual_seed(SEED)
torch.cuda.manual_seed(SEED)
torch.backends.cudnn.deterministic = True
```

[3]:
```
ImageFile.LOAD_TRUNCATED_IMAGES = True

def extract_folder(repo_url, clone_dir, folder_name):
    if os.path.exists(folder_name):
        os.chdir("..")
        shutil.move(os.path.join(clone_dir, folder_name), folder_name)
        print(f"Folder '{folder_name}' extracted successfully.")

        total_files = 0
        classes = []
        # Count number of files in each directory
        for root, dirs, files in os.walk(folder_name):
            for directory in dirs:
                dir_path = os.path.join(root, directory)
                classes.append(directory)
                num_files = len([name for name in os.listdir(dir_path) if os.
 ↪path.isfile(os.path.join(dir_path, name))])
                total_files += num_files
```

```
                print(f"Directory '{directory}' contains {num_files} files.")

        print(f"Total number of files: {total_files}")

    else:
        print(f"Folder '{folder_name}' not found in the repository.")
        classes = []

    print()
    print(f"total number of classes : '{len(classes)}'.")
    print(classes)

    return classes
```

[ ]:
```
# Clone the repository
repo_url = "https://github.com/digitalepidemiologylab/
 ↪plantvillage_deeplearning_paper_dataset.git"
clone_dir = "plantvillage_deeplearning_paper_dataset"
subprocess.run(["git", "clone", repo_url, clone_dir])
os.chdir(clone_dir)

# Raw color images
classes = extract_folder(repo_url, clone_dir, "raw/color")
```

```
Folder 'raw/color' extracted successfully.
Directory 'Tomato___Early_blight' contains 1000 files.
Directory 'Apple___Black_rot' contains 621 files.
Directory 'Tomato___healthy' contains 1591 files.
Directory 'Peach___healthy' contains 360 files.
Directory 'Strawberry___healthy' contains 456 files.
Directory 'Apple___healthy' contains 1645 files.
Directory 'Tomato___Late_blight' contains 1909 files.
Directory 'Corn_(maize)___healthy' contains 1162 files.
Directory 'Corn_(maize)___Common_rust_' contains 1192 files.
Directory 'Pepper,_bell___Bacterial_spot' contains 997 files.
Directory 'Soybean___healthy' contains 5090 files.
Directory 'Grape___Leaf_blight_(Isariopsis_Leaf_Spot)' contains 1076 files.
Directory 'Strawberry___Leaf_scorch' contains 1109 files.
Directory 'Tomato___Tomato_Yellow_Leaf_Curl_Virus' contains 5357 files.
Directory 'Potato___healthy' contains 152 files.
Directory 'Tomato___Tomato_mosaic_virus' contains 373 files.
Directory 'Orange___Haunglongbing_(Citrus_greening)' contains 5507 files.
Directory 'Corn_(maize)___Northern_Leaf_Blight' contains 985 files.
Directory 'Pepper,_bell___healthy' contains 1478 files.
Directory 'Tomato___Septoria_leaf_spot' contains 1771 files.
Directory 'Grape___Esca_(Black_Measles)' contains 1383 files.
Directory 'Tomato___Target_Spot' contains 1404 files.
Directory 'Tomato___Leaf_Mold' contains 952 files.
```

```
Directory 'Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot' contains 513
files.
Directory 'Cherry_(including_sour)___Powdery_mildew' contains 1052 files.
Directory 'Apple___Cedar_apple_rust' contains 275 files.
Directory 'Raspberry___healthy' contains 371 files.
Directory 'Grape___healthy' contains 423 files.
Directory 'Apple___Apple_scab' contains 630 files.
Directory 'Peach___Bacterial_spot' contains 2297 files.
Directory 'Cherry_(including_sour)___healthy' contains 854 files.
Directory 'Tomato___Bacterial_spot' contains 2127 files.
Directory 'Grape___Black_rot' contains 1180 files.
Directory 'Squash___Powdery_mildew' contains 1835 files.
Directory 'Potato___Early_blight' contains 1000 files.
Directory 'Blueberry___healthy' contains 1502 files.
Directory 'Potato___Late_blight' contains 1000 files.
Directory 'Tomato___Spider_mites Two-spotted_spider_mite' contains 1676 files.
Total number of files: 54305

total number of classes : '38'.
['Tomato___Early_blight', 'Apple___Black_rot', 'Tomato___healthy',
'Peach___healthy', 'Strawberry___healthy', 'Apple___healthy',
'Tomato___Late_blight', 'Corn_(maize)___healthy', 'Corn_(maize)___Common_rust_',
'Pepper,_bell___Bacterial_spot', 'Soybean___healthy',
'Grape___Leaf_blight_(Isariopsis_Leaf_Spot)', 'Strawberry___Leaf_scorch',
'Tomato___Tomato_Yellow_Leaf_Curl_Virus', 'Potato___healthy',
'Tomato___Tomato_mosaic_virus', 'Orange___Haunglongbing_(Citrus_greening)',
'Corn_(maize)___Northern_Leaf_Blight', 'Pepper,_bell___healthy',
'Tomato___Septoria_leaf_spot', 'Grape___Esca_(Black_Measles)',
'Tomato___Target_Spot', 'Tomato___Leaf_Mold',
'Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot',
'Cherry_(including_sour)___Powdery_mildew', 'Apple___Cedar_apple_rust',
'Raspberry___healthy', 'Grape___healthy', 'Apple___Apple_scab',
'Peach___Bacterial_spot', 'Cherry_(including_sour)___healthy',
'Tomato___Bacterial_spot', 'Grape___Black_rot', 'Squash___Powdery_mildew',
'Potato___Early_blight', 'Blueberry___healthy', 'Potato___Late_blight',
'Tomato___Spider_mites Two-spotted_spider_mite']
```

Unbalanced Distribution: The distribution appears to be highly unbalanced. This conclusion is drawn from the significant disparity in the heights of the bars representing different crop disease classes. Some classes have a much larger number of files compared to others.

Imbalance Severity: Classes such as "Tomato_**Tomato_Yellow_Leaf_Curl_Virus", "Orange**_Haunglongbing_(Citrus_greening)", and "Soybean_**healthy" have notably higher numbers of files compared to the rest. Conversely, classes like "Potato**_healthy", "Apple_**Cedar_apple_rust", and "Corn_(maize)**_Cercospora_leaf_spot Gray_leaf_spot" have relatively fewer files.

Impact on Model Performance: The presence of highly unbalanced classes can potentially impact the performance of machine learning models trained on this dataset. Models may have a tendency

to be biased towards the majority classes, leading to poorer performance on minority classes.

Data Augmentation or Sampling Strategies: Addressing class imbalance may require techniques such as data augmentation for minority classes.

That leads to improved model performance and reduced overfitting. However, it's crucial to select augmentation techniques carefully and evaluate models appropriately to avoid introducing bias

#A. Data Preprocessing.

[6]:
```
class CropDiseaseDataset(data.Dataset):
    def __init__(self, root_dir, train=True, validation=False, gray_scale=False,␣
↪segmented=False):
        """Initializes a dataset containing images and labels."""
        super().__init__()
        self.gray_scale = gray_scale
        self.root_dir = root_dir
        self.train = train
        self.validation = validation
        self.segmented = segmented

        # Get all jpg images
        self.files = []
        for class_name in os.listdir(self.root_dir):
            class_dir = os.path.join(self.root_dir, class_name)
            # Filter files based on the segmented condition
            if self.segmented:
                files = [os.path.join(class_dir, file) for file in os.
↪listdir(class_dir) if file.endswith(".jpg")]
            else:
                files = [os.path.join(class_dir, file) for file in os.
↪listdir(class_dir) if file.endswith(".JPG")]
            self.files.extend(files)

        random.shuffle(self.files)

        # Define the split ratios
        train_ratio = 0.65
        validation_ratio = 0.15

        self.classes = []
        for root, dirs, _ in os.walk(root_dir):
            for directory in dirs:
                self.classes.append(directory)

        # Split the dataset into train, validation, and test sets
        if self.train:
            num_train_files = int(len(self.files) * train_ratio)
            self.files = self.files[:num_train_files]
```

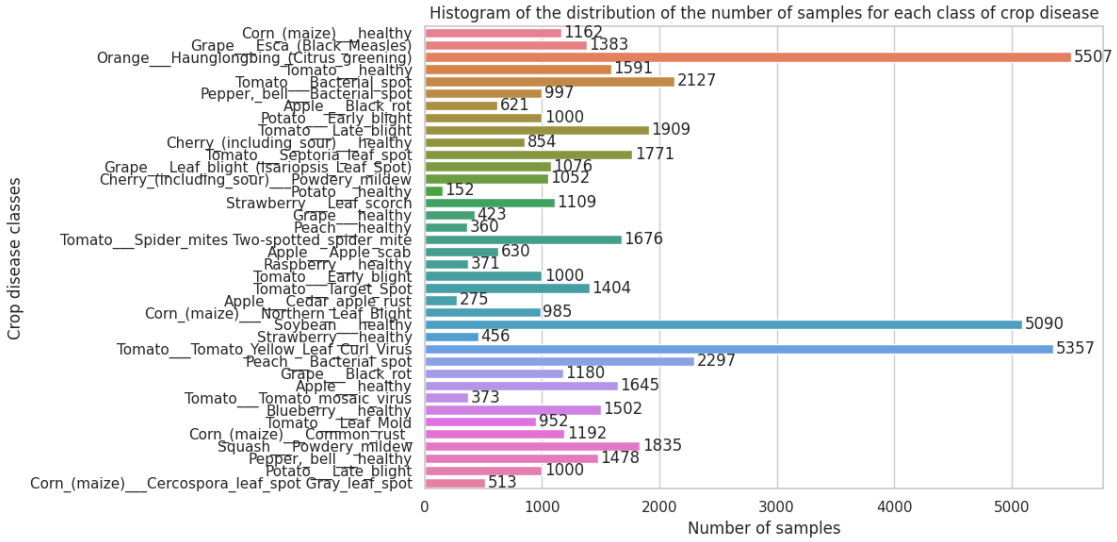
```
        elif self.validation:
            num_train_files = int(len(self.files) * train_ratio)
            num_validation_files = int(len(self.files) * validation_ratio)
            self.files = self.files[num_train_files:num_train_files +␣
↪num_validation_files]
        else:
            num_train_files = int(len(self.files) * train_ratio)
            num_validation_files = int(len(self.files) * validation_ratio)
            self.files = self.files[num_train_files + num_validation_files:]

        # Remove all invalid files
        def is_valid(filename):
            """Check that a file is not corrupted, is in RGB, and has non-zero␣
↪size"""
            if os.path.getsize(filename) == 0:
                return False
            valid_extensions = ['.jpg']
            _, ext = os.path.splitext(filename)
            if ext.lower() not in valid_extensions:
                return False
            try:
                with Image.open(filename, 'r') as img:
                    img.verify()
                    mode = img.mode
                    return mode == 'RGB'
            except:
                return False

        wrong = []
        for i in range(len(self.files)):
            if not is_valid(self.files[i]):
                wrong.append(i)
        self.files = np.delete(self.files, wrong)
        random.shuffle(self.files)

        if self.gray_scale:
            self.means, self.stds = self.calculate_mean_std_gray()
            self.means = np.array(self.means)
            self.stds = np.array(self.stds)
        else:
            self.means, self.stds = self.calculate_mean_std()

        self.transform = self.get_transform()

    def calculate_mean_std(self):
        sum_channel = np.zeros(3)
        squared_sum_channel = np.zeros(3)
```

```
        total_pixels = 0
        total_images = len(self.files)

        # Iterate through all images to accumulate sums
        for file in self.files:
            img = Image.open(file)

            img_array = np.array(img)
            total_pixels += img_array.size / 3
            sum_channel += np.sum(img_array, axis=(0, 1)) / 255.0
            squared_sum_channel += np.sum((img_array / 255.0) ** 2, axis=(0, 1))

        means = sum_channel / total_pixels
        stds = np.sqrt((squared_sum_channel / total_pixels) - (means ** 2))

        return means.tolist(), stds.tolist()

    def calculate_mean_std_gray(self):
        sum_channel = np.zeros(1)  # Grayscale has only one channel
        squared_sum_channel = np.zeros(1)
        total_pixels = 0
        total_images = len(self.files)

        # Iterate through all images to accumulate sums
        for file in self.files:
            img = Image.open(file).convert("L")  # Convert image to grayscale

            img_array = np.array(img)
            total_pixels += img_array.size
            sum_channel += np.sum(img_array) / 255.0
            squared_sum_channel += np.sum((img_array / 255.0) ** 2)

        means = sum_channel / total_pixels
        stds = np.sqrt((squared_sum_channel / total_pixels) - (means ** 2))

        return [means], [stds]

    def get_transform(self):
        if self.gray_scale:
            return transforms.Compose([
                transforms.RandomVerticalFlip(),
                transforms.RandomHorizontalFlip(),
                transforms.RandomApply([
                    transforms.ColorJitter(
                        brightness=torch.rand(1).item(),
                        contrast=torch.rand(1).item(),
                        saturation=torch.rand(1).item(),
```

```
                        hue=torch.rand(1).item() * 0.5,
                    )
                ]),
                transforms.Resize((120, 120)),
                transforms.Grayscale(num_output_channels=1),  # Convert to␣
↪grayscale
                transforms.ToTensor(),
                transforms.Normalize(self.means, self.stds)
            ])
        else:
            return transforms.Compose([
                transforms.RandomVerticalFlip(),
                transforms.RandomHorizontalFlip(),
                transforms.RandomApply([
                    transforms.ColorJitter(
                        brightness=torch.rand(1).item(),
                        contrast=torch.rand(1).item(),
                        saturation=torch.rand(1).item(),
                        hue=torch.rand(1).item() * 0.5,
                    )
                ]),
                transforms.Resize((120, 120)),
                transforms.ToTensor(),
                transforms.Normalize(self.means, self.stds)
            ])

    def __len__(self):
        """Returns the size of the dataset."""
        return len(self.files)

    def __getitem__(self, index):
        """Returns the index-th data item of the dataset."""
        if index < 0 or index >= self.__len__():
            raise IndexError(
                'Wrong index (must be in [{}, {}], given: {})'
                .format(0, self.__len__(), index)
            )

        # Load image and transform
        img = Image.open(self.files[index])
        img = self.transform(img)

        # Get label
        class_name = os.path.basename(os.path.dirname(self.files[index]))
        label = self.classes.index(class_name)

        return img, torch.tensor(label)
```

[ ]:
```
# Provided data: directory names and corresponding file counts
data_ = {
    'Corn_(maize)___healthy': 1162,
    'Grape___Esca_(Black_Measles)': 1383,
    'Orange___Haunglongbing_(Citrus_greening)': 5507,
    'Tomato___healthy': 1591,
    'Tomato___Bacterial_spot': 2127,
    'Pepper,_bell___Bacterial_spot': 997,
    'Apple___Black_rot': 621,
    'Potato___Early_blight': 1000,
    'Tomato___Late_blight': 1909,
    'Cherry_(including_sour)___healthy': 854,
    'Tomato___Septoria_leaf_spot': 1771,
    'Grape___Leaf_blight_(Isariopsis_Leaf_Spot)': 1076,
    'Cherry_(including_sour)___Powdery_mildew': 1052,
    'Potato___healthy': 152,
    'Strawberry___Leaf_scorch': 1109,
    'Grape___healthy': 423,
    'Peach___healthy': 360,
    'Tomato___Spider_mites Two-spotted_spider_mite': 1676,
    'Apple___Apple_scab': 630,
    'Raspberry___healthy': 371,
    'Tomato___Early_blight': 1000,
    'Tomato___Target_Spot': 1404,
    'Apple___Cedar_apple_rust': 275,
    'Corn_(maize)___Northern_Leaf_Blight': 985,
    'Soybean___healthy': 5090,
    'Strawberry___healthy': 456,
    'Tomato___Tomato_Yellow_Leaf_Curl_Virus': 5357,
    'Peach___Bacterial_spot': 2297,
    'Grape___Black_rot': 1180,
    'Apple___healthy': 1645,
    'Tomato___Tomato_mosaic_virus': 373,
    'Blueberry___healthy': 1502,
    'Tomato___Leaf_Mold': 952,
    'Corn_(maize)___Common_rust_': 1192,
    'Squash___Powdery_mildew': 1835,
    'Pepper,_bell___healthy': 1478,
    'Potato___Late_blight': 1000,
    'Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot': 513
}

directories = list(data_.keys())
file_counts = list(data_.values())
colors = sns.color_palette("husl", len(directories))

sns.set(style="whitegrid")
```

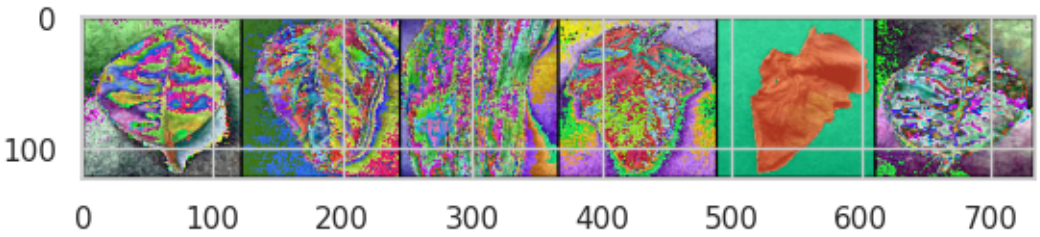
```
plt.figure(figsize=(12, 6))
sns.barplot(x=file_counts, y=directories, hue=directories, dodge=False,␣
 ↪palette=colors, legend=False)
for i, count in enumerate(file_counts):
    plt.text(count + 20, i, str(count), va='center')

plt.xlabel('Number of samples')
plt.ylabel('Crop disease classes')
plt.title('Histogram of the distribution of the number of samples for each class␣
 ↪of crop disease')
plt.tight_layout()
plt.show()
```



[ ]:
```
def show_images(img):
    plt.imshow(transforms.functional.to_pil_image(img))
    plt.show()
```

Checking out.

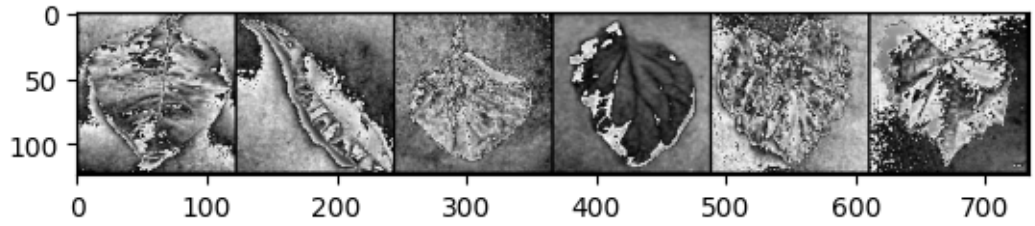
[ ]:
```
extracted_folder = "raw/color"
my_dataset = CropDiseaseDataset(root_dir=extracted_folder, train=True,␣
 ↪gray_scale=False, segmented = False)
train_loader = DataLoader(my_dataset, batch_size=6, shuffle=True, num_workers=2)

images, labels = next(iter(train_loader))
show_images(utils.make_grid(images))
print(*[classes[l] for l in labels])
```




```
Orange___Haunglongbing_(Citrus_greening) Tomato___Bacterial_spot
Corn_(maize)___Northern_Leaf_Blight Tomato___Bacterial_spot Tomato___Late_blight
Cherry_(including_sour)___Powdery_mildew
```

[ ]:
```
my_dataset = CropDiseaseDataset(root_dir=extracted_folder, train=True,␣
 ↪gray_scale=True, segmented= False)
train_loader = DataLoader(my_dataset, batch_size=6, shuffle=True, num_workers=2)

images, labels = next(iter(train_loader))
show_images(utils.make_grid(images))
print(*[my_dataset.classes[l] for l in labels])
```

```
Tomato___Tomato_Yellow_Leaf_Curl_Virus Soybean___healthy
Grape___Esca_(Black_Measles) Orange___Haunglongbing_(Citrus_greening)
Apple___healthy Pepper,_bell___healthy
```




```
Pepper,_bell___healthy Peach___Bacterial_spot Soybean___healthy
Tomato___Tomato_Yellow_Leaf_Curl_Virus Grape___Black_rot Grape___Black_rot
```

#B. Defining the convolutional network architecture.

Initialize our parameters, determine the number of model parameters and a function to tell us how long an epoch takes.

[7]:
```
def initialize_parameters(m):
    if isinstance(m, nn.Conv2d):
        nn.init.kaiming_normal_(m.weight.data, nonlinearity='relu')
        nn.init.constant_(m.bias.data, 0)
```

```
    elif isinstance(m, nn.Linear):
        nn.init.xavier_normal_(m.weight.data, gain=nn.init.
↪calculate_gain('relu'))
        nn.init.constant_(m.bias.data, 0)

def count_parameters(model):
    return sum(p.numel() for p in model.parameters() if p.requires_grad)

def epoch_time(start_time, end_time):
    elapsed_time = end_time - start_time
    elapsed_mins = int(elapsed_time / 60)
    elapsed_secs = int(elapsed_time - (elapsed_mins * 60))
    return elapsed_mins, elapsed_secs
```

[8]:
```
class CNN(nn.Module):
    def __init__(self, grayscale=False):
        super(CNN, self).__init__()
        if grayscale:
            in_channels = 1  # Set input channels to 1 for grayscale images
        else:
            in_channels = 3  # Keep input channels as 3 for RGB images
        self.conv1 = nn.Conv2d(in_channels, 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)
        self.pool = nn.MaxPool2d(2)
        self.fc1 = nn.Linear(256 * 7 * 7, 512)
        self.fc2 = nn.Linear(512, 64)
        self.fc3 = nn.Linear(64, 38)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))
        x = self.pool(F.relu(self.conv4(x)))
        x = x.view(-1, 256 * 7 * 7)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

[ ]:
```
model = CNN()
print(model)
```

```
CNN(
  (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
```

```
  (conv3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv4): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  (fc1): Linear(in_features=12544, out_features=512, bias=True)
  (fc2): Linear(in_features=512, out_features=64, bias=True)
  (fc3): Linear(in_features=64, out_features=38, bias=True)
)
```

2 C. Optimization stage (hyperparameter selections)

[ ]:
```
# Spliiting the dataset
extracted_folder = "raw/color"
train_set = CropDiseaseDataset(root_dir=extracted_folder, train=True,␣
↪validation=False, gray_scale=False, segmented=False)
validation_set = CropDiseaseDataset(root_dir=extracted_folder, train=False,␣
↪validation=True, gray_scale=False, segmented=False)
test_set = CropDiseaseDataset(root_dir=extracted_folder, train=False,␣
↪validation=False, gray_scale=False, segmented=False)

trainloader = torch.utils.data.DataLoader(train_set, batch_size=64,␣
↪shuffle=True, num_workers=2)
validationloader = torch.utils.data.DataLoader(validation_set, batch_size=64,␣
↪shuffle=False, num_workers=2)
testloader = torch.utils.data.DataLoader(test_set, batch_size=64, shuffle=False,␣
↪num_workers=2)
```

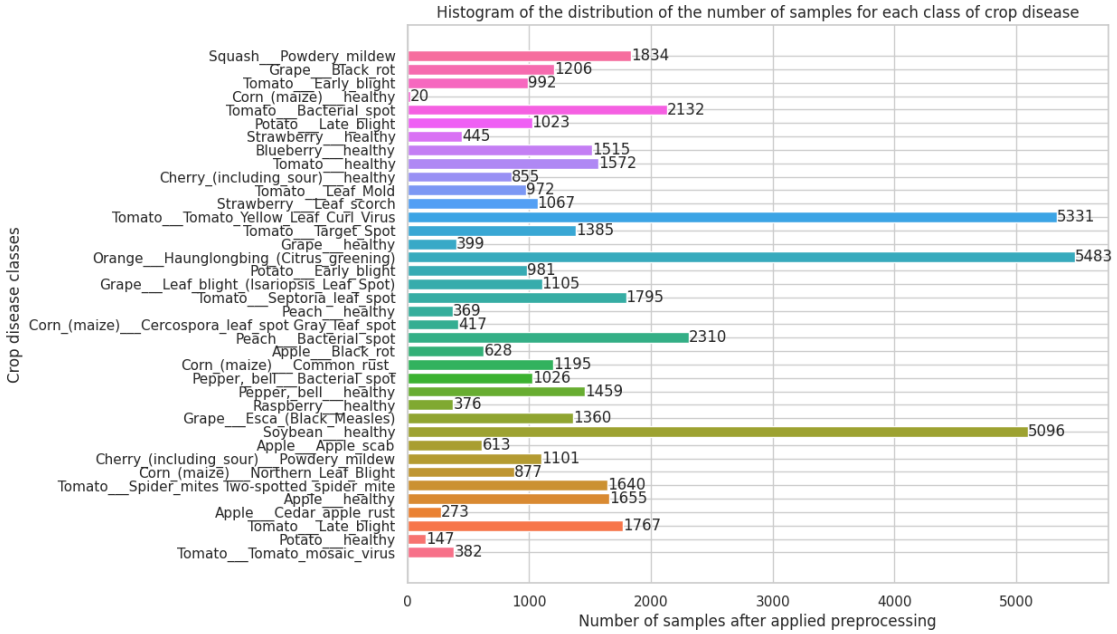
[ ]:
```
print(f"Train set: '{len(train_set)}' images,", f"Validation set:␣
↪'{len(validation_set)}' images,", f"Test set: '{len(test_set)}' images")
```

```
Train set: '34321' images, Validation set: '7920' images, Test set: '10562'
images
```

[ ]:
```
merged_dataset = ConcatDataset([train_set, validation_set, test_set])

# Create a single loader for the combined dataset
mergedloader = DataLoader(merged_dataset, batch_size=64, shuffle=True,␣
↪num_workers=2)
```

[ ]:
```
class_counts = defaultdict(int)
for images, labels in mergedloader:
    for label in labels:
        class_counts[label.item()] += 1

class_labels = sorted(class_counts.keys())
counts = [class_counts[label] for label in class_labels]
colors = sns.color_palette("husl", len(class_labels))
```

```
class_names = [train_set.classes[label] for label in class_labels]

sns.set(style="whitegrid")
plt.figure(figsize=(10, 8))
plt.barh(class_names, counts, color=colors)
plt.xlabel('Number of samples after applied preprocessing')
plt.ylabel('Crop disease classes')
plt.title('Histogram of the distribution of the number of samples for each class␣
 ↪of crop disease')

for i, (count, name) in enumerate(zip(counts, class_names)):
    plt.text(count, i, str(count), ha='left', va='center')

plt.show()
```



[9]:
```
device = 'cpu'
if torch.cuda.is_available():
    device = 'cuda'

# Define the training method
def train(model, num_epochs, trainloader, testloader, criterion, optimizer,␣
 ↪validation_phase=False):
    train_avg_loss = []
    test_avg_loss = []
    test_accuracy = []
```

```
train_accuracy = []

for i in range(num_epochs):
    train_losses = []
    test_losses = []
    correct_train = 0
    total_train = 0

    for x, y in trainloader:
        start_time = time.monotonic()
        x, y = x.to(device), y.to(device)

        pred = model(x)
        loss = criterion(pred, y)
        train_losses.append(loss.detach())

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

        # Compute training accuracy
        y_pred_train = pred.argmax(dim=-1)
        correct_train += (y_pred_train == y).sum().item()
        total_train += y.size(0)

    with torch.no_grad():
        correct_test = 0
        total_test = 0

        # Choose loader based on phase
        if validation_phase:
            loader = validationloader
        else:
            loader = testloader

        for x, y in loader:
            x, y = x.to(device), y.to(device)

            pred = model(x)
            loss = criterion(pred, y)
            test_losses.append(loss)

            y_pred_test = pred.argmax(dim=-1)
            correct_test += (y_pred_test == y).sum().item()
            total_test += y.size(0)

        # Compute accuracy
```

```
            accuracy_train = correct_train / total_train
            accuracy_test = correct_test / total_test
            test_accuracy.append(accuracy_test)

        train_avg_loss.append(torch.mean(torch.tensor(train_losses)).detach())
        test_avg_loss.append(torch.mean(torch.tensor(test_losses)).detach())

        if validation_phase:
            print("Epoch [{}/{}], Validation Loss: {:.4f}, Validation Accuracy:␣
↪{:.2f}%, Train Loss: {:.4f}, Train Accuracy: {:.2f}%".format(
                i+1, num_epochs, test_avg_loss[-1], accuracy_test*100,␣
↪train_avg_loss[-1], accuracy_train*100))
        else:
            print("Epoch [{}/{}], Test Loss: {:.4f}, Test Accuracy: {:.2f}%,␣
↪Train Loss: {:.4f}, Train Accuracy: {:.2f}%".format(
                i+1, num_epochs, test_avg_loss[-1], accuracy_test*100,␣
↪train_avg_loss[-1], accuracy_train*100))

        end_time = time.monotonic()
        epoch_mins, epoch_secs = epoch_time(start_time, end_time)

        print(f'| Epoch Time: {epoch_mins}m {epoch_secs}s')

        train_accuracy.append(accuracy_train)

    # Save the model at the end of the training
    torch.save(model.state_dict(), 'model.pt')

    return train_avg_loss, test_avg_loss, test_accuracy, train_accuracy
```

[10]:
```
# define a method for plotting the results
def plot_training_results(train_avg_loss, validation_avg_loss,␣
↪validation_accuracy, is_validation=True):
    sns.set(style="whitegrid")
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))

    if is_validation:
        loss_label = 'Validation Loss'
        accuracy_label = 'Validation Accuracy'
    else:
        loss_label = 'Test Loss'
        accuracy_label = 'Test Accuracy'

    # Plot train and validation/test loss
    ax1.plot(train_avg_loss, label='Train Loss')
    ax1.plot(validation_avg_loss, label=loss_label)
    ax1.set_xlabel('Epoch')
```

```
    ax1.set_ylabel('Loss')
    ax1.legend(loc='upper right')

    # Plot validation/test accuracy
    ax2.plot(validation_accuracy, label=accuracy_label)
    ax2.set_xlabel('Epoch')
    ax2.set_ylabel('Accuracy')
    ax2.legend(loc='upper left')

    plt.show()
```

Learning rate selection

[ ]:
```
num_epochs = 20
learning_rates = [1e-3,1e-2,1e-4]

# Lists to store results for each learning rate
train_losses_list = []
validation_losses_list = []
train_accuracy_list = []
validation_accuracy_list = []

for idx,lr in enumerate(learning_rates):
    network = CNN().to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = torch.optim.AdamW(network.parameters(), lr=lr)

    print(f"Training model for configuration {idx+ 1}...")
    # Train the model
    train_avg_loss, validation_avg_loss, validation_accuracy, train_accuracy =␣
 ↪train(network, num_epochs, trainloader, validationloader, criterion,␣
 ↪optimizer,validation_phase=True)

    # Store results for plotting
    train_losses_list.append(train_avg_loss)
    validation_losses_list.append(validation_avg_loss)
    train_accuracy_list.append(train_accuracy)
    validation_accuracy_list.append(validation_accuracy)
```

```
Training model for configuration 1...
Epoch [1/20], Validation Loss: 1.1557, Validation Accuracy: 65.19%, Train Loss:
1.9920, Train Accuracy: 45.28%
| Epoch Time: 0m 36s
Epoch [2/20], Validation Loss: 0.6847, Validation Accuracy: 77.85%, Train Loss:
0.9250, Train Accuracy: 71.22%
| Epoch Time: 0m 36s
Epoch [3/20], Validation Loss: 0.4793, Validation Accuracy: 84.39%, Train Loss:
0.6446, Train Accuracy: 79.49%
```

```
| Epoch Time: 0m 35s
Epoch [4/20], Validation Loss: 0.4588, Validation Accuracy: 85.06%, Train Loss:
0.5095, Train Accuracy: 83.55%
| Epoch Time: 0m 35s
Epoch [5/20], Validation Loss: 0.3131, Validation Accuracy: 89.44%, Train Loss:
0.4359, Train Accuracy: 85.84%
| Epoch Time: 0m 36s
Epoch [6/20], Validation Loss: 0.3292, Validation Accuracy: 89.00%, Train Loss:
0.3662, Train Accuracy: 87.85%
| Epoch Time: 0m 37s
Epoch [7/20], Validation Loss: 0.3100, Validation Accuracy: 89.90%, Train Loss:
0.3319, Train Accuracy: 89.07%
| Epoch Time: 0m 36s
Epoch [8/20], Validation Loss: 0.3023, Validation Accuracy: 90.03%, Train Loss:
0.3054, Train Accuracy: 89.90%
| Epoch Time: 0m 36s
Epoch [9/20], Validation Loss: 0.2389, Validation Accuracy: 92.15%, Train Loss:
0.2752, Train Accuracy: 90.79%
| Epoch Time: 0m 37s
Epoch [10/20], Validation Loss: 0.2192, Validation Accuracy: 92.73%, Train Loss:
0.2495, Train Accuracy: 91.76%
| Epoch Time: 0m 35s
Epoch [11/20], Validation Loss: 0.2609, Validation Accuracy: 91.67%, Train Loss:
0.2377, Train Accuracy: 92.22%
| Epoch Time: 0m 35s
Epoch [12/20], Validation Loss: 0.1786, Validation Accuracy: 94.57%, Train Loss:
0.2244, Train Accuracy: 92.57%
| Epoch Time: 0m 36s
Epoch [13/20], Validation Loss: 0.2449, Validation Accuracy: 92.25%, Train Loss:
0.1971, Train Accuracy: 93.48%
| Epoch Time: 0m 35s
Epoch [14/20], Validation Loss: 0.1949, Validation Accuracy: 93.76%, Train Loss:
0.1991, Train Accuracy: 93.49%
| Epoch Time: 0m 36s
Epoch [15/20], Validation Loss: 0.1860, Validation Accuracy: 94.15%, Train Loss:
0.1792, Train Accuracy: 93.99%
| Epoch Time: 0m 36s
Epoch [16/20], Validation Loss: 0.2143, Validation Accuracy: 93.04%, Train Loss:
0.1726, Train Accuracy: 94.32%
| Epoch Time: 0m 35s
Epoch [17/20], Validation Loss: 0.1831, Validation Accuracy: 93.80%, Train Loss:
0.1748, Train Accuracy: 94.32%
| Epoch Time: 0m 35s
Epoch [18/20], Validation Loss: 0.1868, Validation Accuracy: 94.46%, Train Loss:
0.1538, Train Accuracy: 94.87%
| Epoch Time: 0m 35s
Epoch [19/20], Validation Loss: 0.1580, Validation Accuracy: 94.89%, Train Loss:
0.1540, Train Accuracy: 94.85%
```

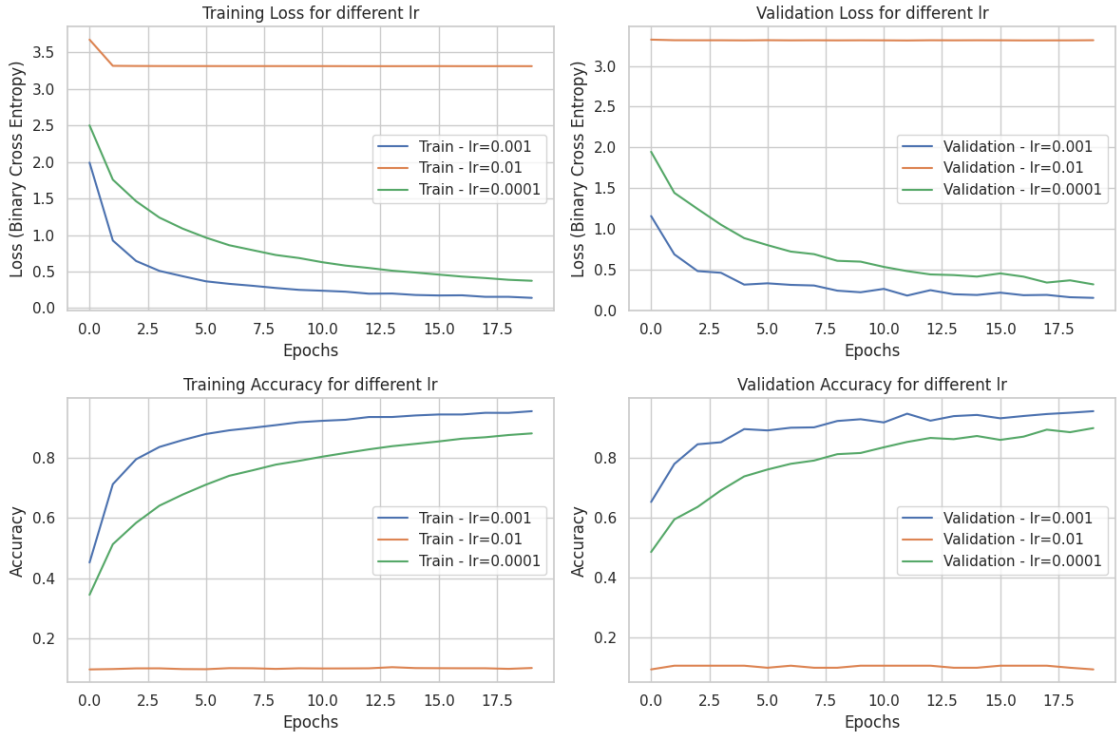
```
| Epoch Time: 0m 36s
Epoch [20/20], Validation Loss: 0.1504, Validation Accuracy: 95.40%, Train Loss:
0.1398, Train Accuracy: 95.41%
| Epoch Time: 0m 36s
Training model for configuration 2...
Epoch [1/20], Validation Loss: 3.3230, Validation Accuracy: 9.47%, Train Loss:
3.6765, Train Accuracy: 9.86%
| Epoch Time: 0m 36s
Epoch [2/20], Validation Loss: 3.3161, Validation Accuracy: 10.71%, Train Loss:
3.3176, Train Accuracy: 9.98%
| Epoch Time: 0m 36s
Epoch [3/20], Validation Loss: 3.3154, Validation Accuracy: 10.71%, Train Loss:
3.3157, Train Accuracy: 10.20%
| Epoch Time: 0m 36s
Epoch [4/20], Validation Loss: 3.3154, Validation Accuracy: 10.71%, Train Loss:
3.3149, Train Accuracy: 10.21%
| Epoch Time: 0m 36s
Epoch [5/20], Validation Loss: 3.3140, Validation Accuracy: 10.71%, Train Loss:
3.3144, Train Accuracy: 9.97%
| Epoch Time: 0m 36s
Epoch [6/20], Validation Loss: 3.3163, Validation Accuracy: 10.03%, Train Loss:
3.3144, Train Accuracy: 9.92%
| Epoch Time: 0m 36s
Epoch [7/20], Validation Loss: 3.3144, Validation Accuracy: 10.71%, Train Loss:
3.3142, Train Accuracy: 10.28%
| Epoch Time: 0m 36s
Epoch [8/20], Validation Loss: 3.3154, Validation Accuracy: 10.03%, Train Loss:
3.3142, Train Accuracy: 10.24%
| Epoch Time: 0m 36s
Epoch [9/20], Validation Loss: 3.3142, Validation Accuracy: 10.03%, Train Loss:
3.3142, Train Accuracy: 10.03%
| Epoch Time: 0m 36s
Epoch [10/20], Validation Loss: 3.3151, Validation Accuracy: 10.71%, Train Loss:
3.3140, Train Accuracy: 10.23%
| Epoch Time: 0m 36s
Epoch [11/20], Validation Loss: 3.3145, Validation Accuracy: 10.71%, Train Loss:
3.3140, Train Accuracy: 10.17%
| Epoch Time: 0m 36s
Epoch [12/20], Validation Loss: 3.3129, Validation Accuracy: 10.71%, Train Loss:
3.3136, Train Accuracy: 10.18%
| Epoch Time: 0m 35s
Epoch [13/20], Validation Loss: 3.3157, Validation Accuracy: 10.71%, Train Loss:
3.3131, Train Accuracy: 10.22%
| Epoch Time: 0m 35s
Epoch [14/20], Validation Loss: 3.3147, Validation Accuracy: 10.03%, Train Loss:
3.3131, Train Accuracy: 10.60%
| Epoch Time: 0m 35s
Epoch [15/20], Validation Loss: 3.3155, Validation Accuracy: 10.03%, Train Loss:
```

```
3.3134, Train Accuracy: 10.30%
| Epoch Time: 0m 36s
Epoch [16/20], Validation Loss: 3.3151, Validation Accuracy: 10.71%, Train Loss:
3.3138, Train Accuracy: 10.26%
| Epoch Time: 0m 36s
Epoch [17/20], Validation Loss: 3.3136, Validation Accuracy: 10.71%, Train Loss:
3.3134, Train Accuracy: 10.24%
| Epoch Time: 0m 36s
Epoch [18/20], Validation Loss: 3.3141, Validation Accuracy: 10.71%, Train Loss:
3.3132, Train Accuracy: 10.24%
| Epoch Time: 0m 36s
Epoch [19/20], Validation Loss: 3.3144, Validation Accuracy: 10.03%, Train Loss:
3.3137, Train Accuracy: 10.04%
| Epoch Time: 0m 36s
Epoch [20/20], Validation Loss: 3.3160, Validation Accuracy: 9.47%, Train Loss:
3.3131, Train Accuracy: 10.33%
| Epoch Time: 0m 36s
Training model for configuration 3...
Epoch [1/20], Validation Loss: 1.9452, Validation Accuracy: 48.54%, Train Loss:
2.5022, Train Accuracy: 34.58%
| Epoch Time: 0m 36s
Epoch [2/20], Validation Loss: 1.4404, Validation Accuracy: 59.39%, Train Loss:
1.7583, Train Accuracy: 51.33%
| Epoch Time: 0m 36s
Epoch [3/20], Validation Loss: 1.2426, Validation Accuracy: 63.55%, Train Loss:
1.4628, Train Accuracy: 58.43%
| Epoch Time: 0m 35s
Epoch [4/20], Validation Loss: 1.0496, Validation Accuracy: 69.05%, Train Loss:
1.2395, Train Accuracy: 64.10%
| Epoch Time: 0m 37s
Epoch [5/20], Validation Loss: 0.8843, Validation Accuracy: 73.70%, Train Loss:
1.0881, Train Accuracy: 67.79%
| Epoch Time: 0m 35s
Epoch [6/20], Validation Loss: 0.7984, Validation Accuracy: 75.98%, Train Loss:
0.9645, Train Accuracy: 71.07%
| Epoch Time: 0m 36s
Epoch [7/20], Validation Loss: 0.7193, Validation Accuracy: 77.89%, Train Loss:
0.8608, Train Accuracy: 74.01%
| Epoch Time: 0m 35s
Epoch [8/20], Validation Loss: 0.6875, Validation Accuracy: 78.98%, Train Loss:
0.7938, Train Accuracy: 75.81%
| Epoch Time: 0m 35s
Epoch [9/20], Validation Loss: 0.6055, Validation Accuracy: 81.10%, Train Loss:
0.7270, Train Accuracy: 77.71%
| Epoch Time: 0m 35s
Epoch [10/20], Validation Loss: 0.5954, Validation Accuracy: 81.49%, Train Loss:
0.6852, Train Accuracy: 78.95%
| Epoch Time: 0m 36s
```

```
Epoch [11/20], Validation Loss: 0.5309, Validation Accuracy: 83.38%, Train Loss:
0.6278, Train Accuracy: 80.34%
| Epoch Time: 0m 36s
Epoch [12/20], Validation Loss: 0.4790, Validation Accuracy: 85.16%, Train Loss:
0.5818, Train Accuracy: 81.58%
| Epoch Time: 0m 36s
Epoch [13/20], Validation Loss: 0.4383, Validation Accuracy: 86.50%, Train Loss:
0.5493, Train Accuracy: 82.76%
| Epoch Time: 0m 36s
Epoch [14/20], Validation Loss: 0.4303, Validation Accuracy: 86.10%, Train Loss:
0.5120, Train Accuracy: 83.81%
| Epoch Time: 0m 36s
Epoch [15/20], Validation Loss: 0.4119, Validation Accuracy: 87.15%, Train Loss:
0.4866, Train Accuracy: 84.59%
| Epoch Time: 0m 36s
Epoch [16/20], Validation Loss: 0.4508, Validation Accuracy: 85.86%, Train Loss:
0.4585, Train Accuracy: 85.39%
| Epoch Time: 0m 36s
Epoch [17/20], Validation Loss: 0.4111, Validation Accuracy: 86.93%, Train Loss:
0.4313, Train Accuracy: 86.30%
| Epoch Time: 0m 36s
Epoch [18/20], Validation Loss: 0.3382, Validation Accuracy: 89.26%, Train Loss:
0.4110, Train Accuracy: 86.78%
| Epoch Time: 0m 35s
Epoch [19/20], Validation Loss: 0.3657, Validation Accuracy: 88.41%, Train Loss:
0.3873, Train Accuracy: 87.51%
| Epoch Time: 0m 35s
Epoch [20/20], Validation Loss: 0.3162, Validation Accuracy: 89.77%, Train Loss:
0.3739, Train Accuracy: 88.05%
| Epoch Time: 0m 35s
```

[ ]:
```python
# Set Seaborn style
sns.set(style="whitegrid")
fig, axs = plt.subplots(2, 2, figsize=(12, 8))

for i, lr in enumerate(learning_rates):
    axs[0, 0].plot(train_losses_list[i], label=f'Train - lr={lr}')
    axs[0, 1].plot(validation_losses_list[i], label=f'Validation - lr={lr}')
    axs[1, 0].plot(train_accuracy_list[i], label=f'Train - lr={lr}')
    axs[1, 1].plot(validation_accuracy_list[i], label=f'Validation - lr={lr}')

axs[0, 0].set_title('Training Loss for different lr')
axs[0, 0].legend()
axs[0, 0].set_xlabel('Epochs')
axs[0, 0].set_ylabel('Loss (Binary Cross Entropy)')

axs[0, 1].set_title('Validation Loss for different lr')
```

```
axs[0, 1].legend()
axs[0, 1].set_xlabel('Epochs')
axs[0, 1].set_ylabel('Loss (Binary Cross Entropy)')

axs[1, 0].set_title('Training Accuracy for different lr')
axs[1, 0].legend()
axs[1, 0].set_xlabel('Epochs')
axs[1, 0].set_ylabel('Accuracy')

axs[1, 1].set_title('Validation Accuracy for different lr')
axs[1, 1].legend()
axs[1, 1].set_xlabel('Epochs')
axs[1, 1].set_ylabel('Accuracy')

plt.tight_layout()
plt.show()
```



The best learning among those which are tested is 10^{-3}.

Optimizer selection

[ ]:
```
num_epochs = 18
learning_rate = 1e-3
optimizers = ['RMSprop','Adam','SGD']
```

```
# Lists to store results for each learning rate
train_losses_list = []
validation_losses_list = []
train_accuracy_list = []
validation_accuracy_list = []

for idx, optimizer_name in enumerate(optimizers):
    network = CNN().to(device)
    criterion = nn.CrossEntropyLoss()
    print(f"Training model for configuration {idx+ 1}...")
    if optimizer_name == 'Adam':
        optimizer = optim.Adam(network.parameters(), lr=learning_rate)
    elif optimizer_name == 'SGD':
        optimizer = optim.SGD(network.parameters(), lr=learning_rate)
    elif optimizer_name == 'RMSprop':
        optimizer = optim.RMSprop(network.parameters(), lr=learning_rate)

    # Train the model
    train_avg_loss, validation_avg_loss, validation_accuracy, train_accuracy =␣
↪train(network, num_epochs, trainloader, validationloader, criterion,␣
↪optimizer,validation_phase=True)

    # Store results for plotting
    train_losses_list.append(train_avg_loss)
    validation_losses_list.append(validation_avg_loss)
    train_accuracy_list.append(train_accuracy)
    validation_accuracy_list.append(validation_accuracy)
```

```
Training model for configuration 1...
Epoch [1/18], Validation Loss: 2.3102, Validation Accuracy: 37.93%, Train Loss:
2.6718, Train Accuracy: 29.80%
| Epoch Time: 0m 35s
Epoch [2/18], Validation Loss: 1.6797, Validation Accuracy: 54.92%, Train Loss:
1.4774, Train Accuracy: 56.45%
| Epoch Time: 0m 35s
Epoch [3/18], Validation Loss: 1.5975, Validation Accuracy: 61.59%, Train Loss:
0.9859, Train Accuracy: 69.38%
| Epoch Time: 0m 35s
Epoch [4/18], Validation Loss: 0.9087, Validation Accuracy: 73.72%, Train Loss:
0.7607, Train Accuracy: 75.72%
| Epoch Time: 0m 34s
Epoch [5/18], Validation Loss: 0.9755, Validation Accuracy: 73.48%, Train Loss:
0.6256, Train Accuracy: 79.81%
| Epoch Time: 0m 35s
Epoch [6/18], Validation Loss: 1.4048, Validation Accuracy: 65.20%, Train Loss:
0.5347, Train Accuracy: 82.70%
| Epoch Time: 0m 34s
```

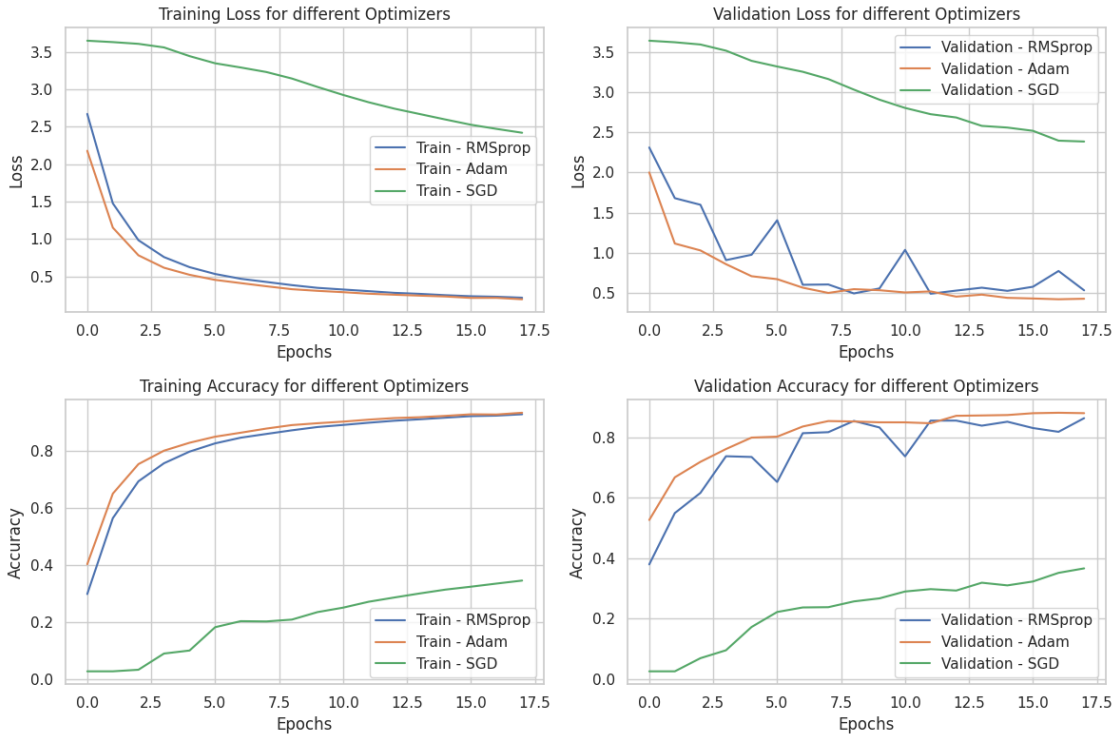
```
Epoch [7/18], Validation Loss: 0.6019, Validation Accuracy: 81.35%, Train Loss:
0.4708, Train Accuracy: 84.70%
| Epoch Time: 0m 37s
Epoch [8/18], Validation Loss: 0.6055, Validation Accuracy: 81.69%, Train Loss:
0.4285, Train Accuracy: 86.01%
| Epoch Time: 0m 35s
Epoch [9/18], Validation Loss: 0.4942, Validation Accuracy: 85.42%, Train Loss:
0.3859, Train Accuracy: 87.28%
| Epoch Time: 0m 34s
Epoch [10/18], Validation Loss: 0.5562, Validation Accuracy: 83.24%, Train Loss:
0.3496, Train Accuracy: 88.44%
| Epoch Time: 0m 35s
Epoch [11/18], Validation Loss: 1.0359, Validation Accuracy: 73.69%, Train Loss:
0.3274, Train Accuracy: 89.16%
| Epoch Time: 0m 34s
Epoch [12/18], Validation Loss: 0.4902, Validation Accuracy: 85.51%, Train Loss:
0.3056, Train Accuracy: 89.94%
| Epoch Time: 0m 35s
Epoch [13/18], Validation Loss: 0.5294, Validation Accuracy: 85.52%, Train Loss:
0.2830, Train Accuracy: 90.64%
| Epoch Time: 0m 35s
Epoch [14/18], Validation Loss: 0.5655, Validation Accuracy: 83.84%, Train Loss:
0.2696, Train Accuracy: 91.16%
| Epoch Time: 0m 36s
Epoch [15/18], Validation Loss: 0.5261, Validation Accuracy: 85.15%, Train Loss:
0.2520, Train Accuracy: 91.72%
| Epoch Time: 0m 34s
Epoch [16/18], Validation Loss: 0.5772, Validation Accuracy: 83.06%, Train Loss:
0.2376, Train Accuracy: 92.26%
| Epoch Time: 0m 37s
Epoch [17/18], Validation Loss: 0.7734, Validation Accuracy: 81.81%, Train Loss:
0.2309, Train Accuracy: 92.44%
| Epoch Time: 0m 36s
Epoch [18/18], Validation Loss: 0.5326, Validation Accuracy: 86.33%, Train Loss:
0.2181, Train Accuracy: 92.92%
| Epoch Time: 0m 36s
Training model for configuration 2...
Epoch [1/18], Validation Loss: 1.9999, Validation Accuracy: 52.63%, Train Loss:
2.1795, Train Accuracy: 40.28%
| Epoch Time: 0m 39s
Epoch [2/18], Validation Loss: 1.1150, Validation Accuracy: 66.76%, Train Loss:
1.1533, Train Accuracy: 65.06%
| Epoch Time: 0m 35s
Epoch [3/18], Validation Loss: 1.0292, Validation Accuracy: 71.87%, Train Loss:
0.7842, Train Accuracy: 75.41%
| Epoch Time: 0m 36s
Epoch [4/18], Validation Loss: 0.8582, Validation Accuracy: 76.11%, Train Loss:
0.6181, Train Accuracy: 80.14%
```

```
| Epoch Time: 0m 36s
Epoch [5/18], Validation Loss: 0.7083, Validation Accuracy: 79.90%, Train Loss:
0.5239, Train Accuracy: 82.91%
| Epoch Time: 0m 35s
Epoch [6/18], Validation Loss: 0.6718, Validation Accuracy: 80.20%, Train Loss:
0.4561, Train Accuracy: 85.04%
| Epoch Time: 0m 36s
Epoch [7/18], Validation Loss: 0.5657, Validation Accuracy: 83.56%, Train Loss:
0.4130, Train Accuracy: 86.45%
| Epoch Time: 0m 36s
Epoch [8/18], Validation Loss: 0.5004, Validation Accuracy: 85.37%, Train Loss:
0.3708, Train Accuracy: 87.89%
| Epoch Time: 0m 37s
Epoch [9/18], Validation Loss: 0.5473, Validation Accuracy: 85.23%, Train Loss:
0.3317, Train Accuracy: 89.12%
| Epoch Time: 0m 34s
Epoch [10/18], Validation Loss: 0.5339, Validation Accuracy: 84.95%, Train Loss:
0.3098, Train Accuracy: 89.77%
| Epoch Time: 0m 36s
Epoch [11/18], Validation Loss: 0.5063, Validation Accuracy: 84.94%, Train Loss:
0.2920, Train Accuracy: 90.33%
| Epoch Time: 0m 36s
Epoch [12/18], Validation Loss: 0.5192, Validation Accuracy: 84.63%, Train Loss:
0.2716, Train Accuracy: 91.02%
| Epoch Time: 0m 35s
Epoch [13/18], Validation Loss: 0.4544, Validation Accuracy: 87.11%, Train Loss:
0.2581, Train Accuracy: 91.59%
| Epoch Time: 0m 36s
Epoch [14/18], Validation Loss: 0.4790, Validation Accuracy: 87.22%, Train Loss:
0.2458, Train Accuracy: 91.87%
| Epoch Time: 0m 37s
Epoch [15/18], Validation Loss: 0.4393, Validation Accuracy: 87.36%, Train Loss:
0.2346, Train Accuracy: 92.35%
| Epoch Time: 0m 35s
Epoch [16/18], Validation Loss: 0.4313, Validation Accuracy: 87.97%, Train Loss:
0.2154, Train Accuracy: 92.90%
| Epoch Time: 0m 39s
Epoch [17/18], Validation Loss: 0.4215, Validation Accuracy: 88.12%, Train Loss:
0.2161, Train Accuracy: 92.83%
| Epoch Time: 0m 38s
Epoch [18/18], Validation Loss: 0.4281, Validation Accuracy: 87.98%, Train Loss:
0.1973, Train Accuracy: 93.45%
| Epoch Time: 0m 38s
Training model for configuration 3...
Epoch [1/18], Validation Loss: 3.6395, Validation Accuracy: 2.55%, Train Loss:
3.6493, Train Accuracy: 2.68%
| Epoch Time: 0m 36s
Epoch [2/18], Validation Loss: 3.6201, Validation Accuracy: 2.55%, Train Loss:
```

```
3.6305, Train Accuracy: 2.68%
| Epoch Time: 0m 36s
Epoch [3/18], Validation Loss: 3.5921, Validation Accuracy: 6.89%, Train Loss:
3.6074, Train Accuracy: 3.23%
| Epoch Time: 0m 37s
Epoch [4/18], Validation Loss: 3.5153, Validation Accuracy: 9.52%, Train Loss:
3.5595, Train Accuracy: 8.92%
| Epoch Time: 0m 35s
Epoch [5/18], Validation Loss: 3.3889, Validation Accuracy: 17.22%, Train Loss:
3.4433, Train Accuracy: 9.98%
| Epoch Time: 0m 35s
Epoch [6/18], Validation Loss: 3.3182, Validation Accuracy: 22.18%, Train Loss:
3.3477, Train Accuracy: 18.17%
| Epoch Time: 0m 36s
Epoch [7/18], Validation Loss: 3.2527, Validation Accuracy: 23.69%, Train Loss:
3.2914, Train Accuracy: 20.28%
| Epoch Time: 0m 35s
Epoch [8/18], Validation Loss: 3.1621, Validation Accuracy: 23.78%, Train Loss:
3.2311, Train Accuracy: 20.19%
| Epoch Time: 0m 37s
Epoch [9/18], Validation Loss: 3.0303, Validation Accuracy: 25.68%, Train Loss:
3.1442, Train Accuracy: 20.84%
| Epoch Time: 0m 37s
Epoch [10/18], Validation Loss: 2.9054, Validation Accuracy: 26.69%, Train Loss:
3.0319, Train Accuracy: 23.47%
| Epoch Time: 0m 36s
Epoch [11/18], Validation Loss: 2.8026, Validation Accuracy: 28.96%, Train Loss:
2.9255, Train Accuracy: 25.03%
| Epoch Time: 0m 36s
Epoch [12/18], Validation Loss: 2.7247, Validation Accuracy: 29.75%, Train Loss:
2.8275, Train Accuracy: 27.10%
| Epoch Time: 0m 36s
Epoch [13/18], Validation Loss: 2.6839, Validation Accuracy: 29.28%, Train Loss:
2.7424, Train Accuracy: 28.58%
| Epoch Time: 0m 36s
Epoch [14/18], Validation Loss: 2.5792, Validation Accuracy: 31.87%, Train Loss:
2.6698, Train Accuracy: 30.02%
| Epoch Time: 0m 35s
Epoch [15/18], Validation Loss: 2.5584, Validation Accuracy: 30.98%, Train Loss:
2.5974, Train Accuracy: 31.36%
| Epoch Time: 0m 36s
Epoch [16/18], Validation Loss: 2.5178, Validation Accuracy: 32.27%, Train Loss:
2.5271, Train Accuracy: 32.39%
| Epoch Time: 0m 36s
Epoch [17/18], Validation Loss: 2.3955, Validation Accuracy: 35.14%, Train Loss:
2.4717, Train Accuracy: 33.48%
| Epoch Time: 0m 35s
Epoch [18/18], Validation Loss: 2.3846, Validation Accuracy: 36.62%, Train Loss:
```

```
2.4202, Train Accuracy: 34.53%
| Epoch Time: 0m 35s
```

[ ]:
```python
sns.set(style="whitegrid")

fig, axs = plt.subplots(2, 2, figsize=(12, 8))

for i, optimizer_name in enumerate(optimizers):
    axs[0, 0].plot(train_losses_list[i], label=f'Train - {optimizer_name}')
    axs[0, 1].plot(validation_losses_list[i], label=f'Validation -␣
 ↪{optimizer_name}')
    axs[1, 0].plot(train_accuracy_list[i], label=f'Train - {optimizer_name}')
    axs[1, 1].plot(validation_accuracy_list[i], label=f'Validation -␣
 ↪{optimizer_name}')

axs[0, 0].set_title('Training Loss for different Optimizers')
axs[0, 0].legend()
axs[0, 0].set_xlabel('Epochs')
axs[0, 0].set_ylabel('Loss')

axs[0, 1].set_title('Validation Loss for different Optimizers')
axs[0, 1].legend()
axs[0, 1].set_xlabel('Epochs')
axs[0, 1].set_ylabel('Loss')

axs[1, 0].set_title('Training Accuracy for different Optimizers')
axs[1, 0].legend()
axs[1, 0].set_xlabel('Epochs')
axs[1, 0].set_ylabel('Accuracy')

axs[1, 1].set_title('Validation Accuracy for different Optimizers')
axs[1, 1].legend()
axs[1, 1].set_xlabel('Epochs')
axs[1, 1].set_ylabel('Accuracy')

plt.tight_layout()
plt.show()
```



3 D. 1) Train the model over the color images

[ ]:
```
# Define hyperparameters
learning_rate = 1e-3
num_epochs = 25

network = CNN(grayscale=False).to(device)
network.apply(initialize_parameters)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(network.parameters(), lr=learning_rate)

print(f'The model has {count_parameters(network):,} trainable parameters')
```

```
The model has 6,846,758 trainable parameters
```

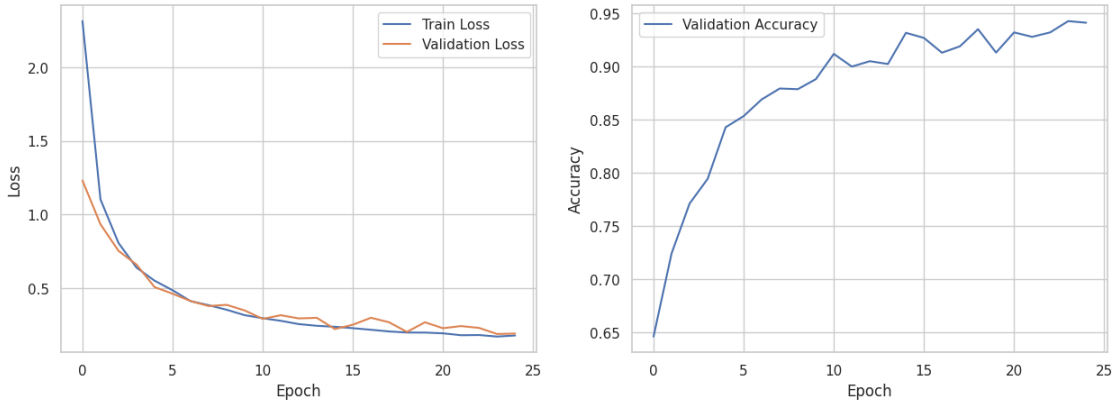
[ ]:
```
# Train the model
train_avg_loss, validation_avg_loss, validation_accuracy, train_accuracy =␣
↪train(network, num_epochs, trainloader, validationloader, criterion,␣
↪optimizer,validation_phase=True)
```

```
Epoch [1/25], Validation Loss: 1.2302, Validation Accuracy: 64.63%, Train Loss:
2.3124, Train Accuracy: 44.40%
| Epoch Time: 0m 36s
```

```
Epoch [2/25], Validation Loss: 0.9330, Validation Accuracy: 72.44%, Train Loss:
1.1015, Train Accuracy: 67.76%
| Epoch Time: 0m 37s
Epoch [3/25], Validation Loss: 0.7526, Validation Accuracy: 77.12%, Train Loss:
0.8058, Train Accuracy: 75.54%
| Epoch Time: 0m 37s
Epoch [4/25], Validation Loss: 0.6592, Validation Accuracy: 79.43%, Train Loss:
0.6387, Train Accuracy: 80.11%
| Epoch Time: 0m 36s
Epoch [5/25], Validation Loss: 0.5066, Validation Accuracy: 84.28%, Train Loss:
0.5500, Train Accuracy: 82.81%
| Epoch Time: 0m 36s
Epoch [6/25], Validation Loss: 0.4616, Validation Accuracy: 85.33%, Train Loss:
0.4851, Train Accuracy: 84.62%
| Epoch Time: 0m 35s
Epoch [7/25], Validation Loss: 0.4115, Validation Accuracy: 86.89%, Train Loss:
0.4111, Train Accuracy: 86.85%
| Epoch Time: 0m 35s
Epoch [8/25], Validation Loss: 0.3779, Validation Accuracy: 87.92%, Train Loss:
0.3836, Train Accuracy: 87.96%
| Epoch Time: 0m 36s
Epoch [9/25], Validation Loss: 0.3860, Validation Accuracy: 87.85%, Train Loss:
0.3526, Train Accuracy: 88.75%
| Epoch Time: 0m 36s
Epoch [10/25], Validation Loss: 0.3478, Validation Accuracy: 88.79%, Train Loss:
0.3158, Train Accuracy: 90.00%
| Epoch Time: 0m 37s
Epoch [11/25], Validation Loss: 0.2908, Validation Accuracy: 91.16%, Train Loss:
0.2952, Train Accuracy: 90.59%
| Epoch Time: 0m 36s
Epoch [12/25], Validation Loss: 0.3157, Validation Accuracy: 89.97%, Train Loss:
0.2776, Train Accuracy: 91.14%
| Epoch Time: 0m 37s
Epoch [13/25], Validation Loss: 0.2940, Validation Accuracy: 90.48%, Train Loss:
0.2554, Train Accuracy: 91.79%
| Epoch Time: 0m 37s
Epoch [14/25], Validation Loss: 0.2978, Validation Accuracy: 90.21%, Train Loss:
0.2438, Train Accuracy: 92.27%
| Epoch Time: 0m 36s
Epoch [15/25], Validation Loss: 0.2218, Validation Accuracy: 93.14%, Train Loss:
0.2371, Train Accuracy: 92.28%
| Epoch Time: 0m 37s
Epoch [16/25], Validation Loss: 0.2509, Validation Accuracy: 92.66%, Train Loss:
0.2276, Train Accuracy: 92.88%
| Epoch Time: 0m 39s
Epoch [17/25], Validation Loss: 0.2982, Validation Accuracy: 91.28%, Train Loss:
0.2169, Train Accuracy: 93.09%
| Epoch Time: 0m 37s
```

```
Epoch [18/25], Validation Loss: 0.2682, Validation Accuracy: 91.88%, Train Loss:
0.2055, Train Accuracy: 93.53%
| Epoch Time: 0m 38s
Epoch [19/25], Validation Loss: 0.2024, Validation Accuracy: 93.48%, Train Loss:
0.1981, Train Accuracy: 93.70%
| Epoch Time: 0m 38s
Epoch [20/25], Validation Loss: 0.2676, Validation Accuracy: 91.29%, Train Loss:
0.1977, Train Accuracy: 93.80%
| Epoch Time: 0m 37s
Epoch [21/25], Validation Loss: 0.2273, Validation Accuracy: 93.18%, Train Loss:
0.1927, Train Accuracy: 94.00%
| Epoch Time: 0m 38s
Epoch [22/25], Validation Loss: 0.2419, Validation Accuracy: 92.77%, Train Loss:
0.1798, Train Accuracy: 94.32%
| Epoch Time: 0m 37s
Epoch [23/25], Validation Loss: 0.2293, Validation Accuracy: 93.18%, Train Loss:
0.1811, Train Accuracy: 94.18%
| Epoch Time: 0m 38s
Epoch [24/25], Validation Loss: 0.1881, Validation Accuracy: 94.24%, Train Loss:
0.1706, Train Accuracy: 94.63%
| Epoch Time: 0m 38s
Epoch [25/25], Validation Loss: 0.1911, Validation Accuracy: 94.09%, Train Loss:
0.1780, Train Accuracy: 94.52%
| Epoch Time: 0m 37s
```

[ ]:
```
plot_training_results(train_avg_loss, validation_avg_loss, validation_accuracy,␣
↪is_validation=True)
```



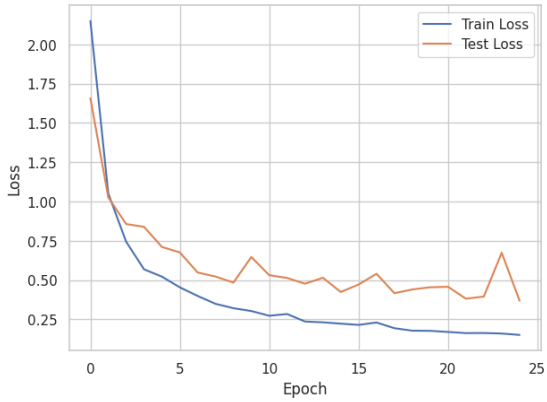
[ ]:
```
# Combining train and validation datasets for testing the model
combined_train_set = ConcatDataset([train_set, validation_set])
combined_train_loader = torch.utils.data.DataLoader(combined_train_set,␣
↪batch_size=64, shuffle=True, num_workers=2)
```

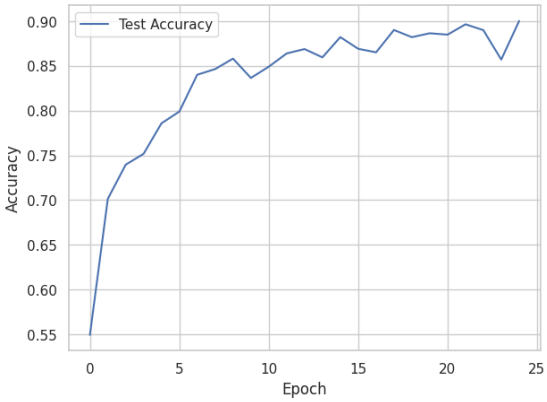
[ ]:
```
# Test the model
network = CNN(grayscale=False).to(device)
network.apply(initialize_parameters)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(network.parameters(), lr=learning_rate)

train_avg_loss, test_avg_loss, test_accuracy, train_accuracy = train(network,␣
 ↪num_epochs, combined_train_loader, testloader, criterion,␣
 ↪optimizer,validation_phase=False)
```

```
Epoch [1/25], Test Loss: 1.6582, Test Accuracy: 54.94%, Train Loss: 2.1497,
Train Accuracy: 46.78%
| Epoch Time: 0m 50s
Epoch [2/25], Test Loss: 1.0267, Test Accuracy: 70.14%, Train Loss: 1.0515,
Train Accuracy: 68.69%
| Epoch Time: 0m 49s
Epoch [3/25], Test Loss: 0.8566, Test Accuracy: 73.95%, Train Loss: 0.7440,
Train Accuracy: 76.77%
| Epoch Time: 0m 48s
Epoch [4/25], Test Loss: 0.8391, Test Accuracy: 75.17%, Train Loss: 0.5684,
Train Accuracy: 82.01%
| Epoch Time: 0m 47s
Epoch [5/25], Test Loss: 0.7107, Test Accuracy: 78.57%, Train Loss: 0.5218,
Train Accuracy: 83.45%
| Epoch Time: 0m 49s
Epoch [6/25], Test Loss: 0.6762, Test Accuracy: 79.89%, Train Loss: 0.4537,
Train Accuracy: 85.73%
| Epoch Time: 0m 47s
Epoch [7/25], Test Loss: 0.5481, Test Accuracy: 84.00%, Train Loss: 0.3987,
Train Accuracy: 86.85%
| Epoch Time: 0m 49s
Epoch [8/25], Test Loss: 0.5222, Test Accuracy: 84.63%, Train Loss: 0.3486,
Train Accuracy: 88.63%
| Epoch Time: 0m 49s
Epoch [9/25], Test Loss: 0.4841, Test Accuracy: 85.80%, Train Loss: 0.3208,
Train Accuracy: 89.67%
| Epoch Time: 0m 48s
Epoch [10/25], Test Loss: 0.6469, Test Accuracy: 83.64%, Train Loss: 0.3027,
Train Accuracy: 90.18%
| Epoch Time: 0m 49s
Epoch [11/25], Test Loss: 0.5310, Test Accuracy: 84.90%, Train Loss: 0.2727,
Train Accuracy: 91.07%
| Epoch Time: 0m 48s
Epoch [12/25], Test Loss: 0.5133, Test Accuracy: 86.38%, Train Loss: 0.2838,
Train Accuracy: 90.83%
| Epoch Time: 0m 48s
Epoch [13/25], Test Loss: 0.4769, Test Accuracy: 86.87%, Train Loss: 0.2360,
Train Accuracy: 92.14%
```

```
| Epoch Time: 0m 48s
Epoch [14/25], Test Loss: 0.5146, Test Accuracy: 85.94%, Train Loss: 0.2309,
Train Accuracy: 92.55%
| Epoch Time: 0m 48s
Epoch [15/25], Test Loss: 0.4240, Test Accuracy: 88.20%, Train Loss: 0.2224,
Train Accuracy: 92.79%
| Epoch Time: 0m 49s
Epoch [16/25], Test Loss: 0.4716, Test Accuracy: 86.90%, Train Loss: 0.2145,
Train Accuracy: 93.08%
| Epoch Time: 0m 48s
Epoch [17/25], Test Loss: 0.5401, Test Accuracy: 86.50%, Train Loss: 0.2297,
Train Accuracy: 92.73%
| Epoch Time: 0m 49s
Epoch [18/25], Test Loss: 0.4163, Test Accuracy: 89.00%, Train Loss: 0.1935,
Train Accuracy: 93.84%
| Epoch Time: 0m 48s
Epoch [19/25], Test Loss: 0.4402, Test Accuracy: 88.19%, Train Loss: 0.1776,
Train Accuracy: 94.36%
| Epoch Time: 0m 48s
Epoch [20/25], Test Loss: 0.4542, Test Accuracy: 88.63%, Train Loss: 0.1766,
Train Accuracy: 94.35%
| Epoch Time: 0m 49s
Epoch [21/25], Test Loss: 0.4572, Test Accuracy: 88.48%, Train Loss: 0.1696,
Train Accuracy: 94.56%
| Epoch Time: 0m 49s
Epoch [22/25], Test Loss: 0.3810, Test Accuracy: 89.63%, Train Loss: 0.1622,
Train Accuracy: 94.92%
| Epoch Time: 0m 49s
Epoch [23/25], Test Loss: 0.3951, Test Accuracy: 88.98%, Train Loss: 0.1627,
Train Accuracy: 94.83%
| Epoch Time: 0m 48s
Epoch [24/25], Test Loss: 0.6739, Test Accuracy: 85.69%, Train Loss: 0.1595,
Train Accuracy: 95.06%
| Epoch Time: 0m 48s
Epoch [25/25], Test Loss: 0.3686, Test Accuracy: 90.00%, Train Loss: 0.1507,
Train Accuracy: 95.26%
| Epoch Time: 0m 49s
```

[ ]:
```
plot_training_results(train_avg_loss, test_avg_loss, test_accuracy,␣
↪is_validation=False)
```





Examining the model

[ ]:
```
network.load_state_dict(torch.load('model.pt'))
```

[ ]: <All keys matched successfully>

[26]:
```
def get_predictions(model, iterator, device):
    model.eval()
    model.to(device)

    images = []
    labels = []
    probs = []
    corrects = []

    with torch.no_grad():
        for (x, y) in iterator:
            x = x.to(device)
            y = y.to(device)

            outputs = model(x)
            if isinstance(outputs, tuple):
                y_pred = outputs[0]
            else:
                y_pred = outputs

            y_prob = F.softmax(y_pred, dim=-1)
            _, predicted = torch.max(y_prob, 1)

            images.append(x.cpu())
            labels.append(y.cpu())
            probs.append(y_prob.cpu())
```

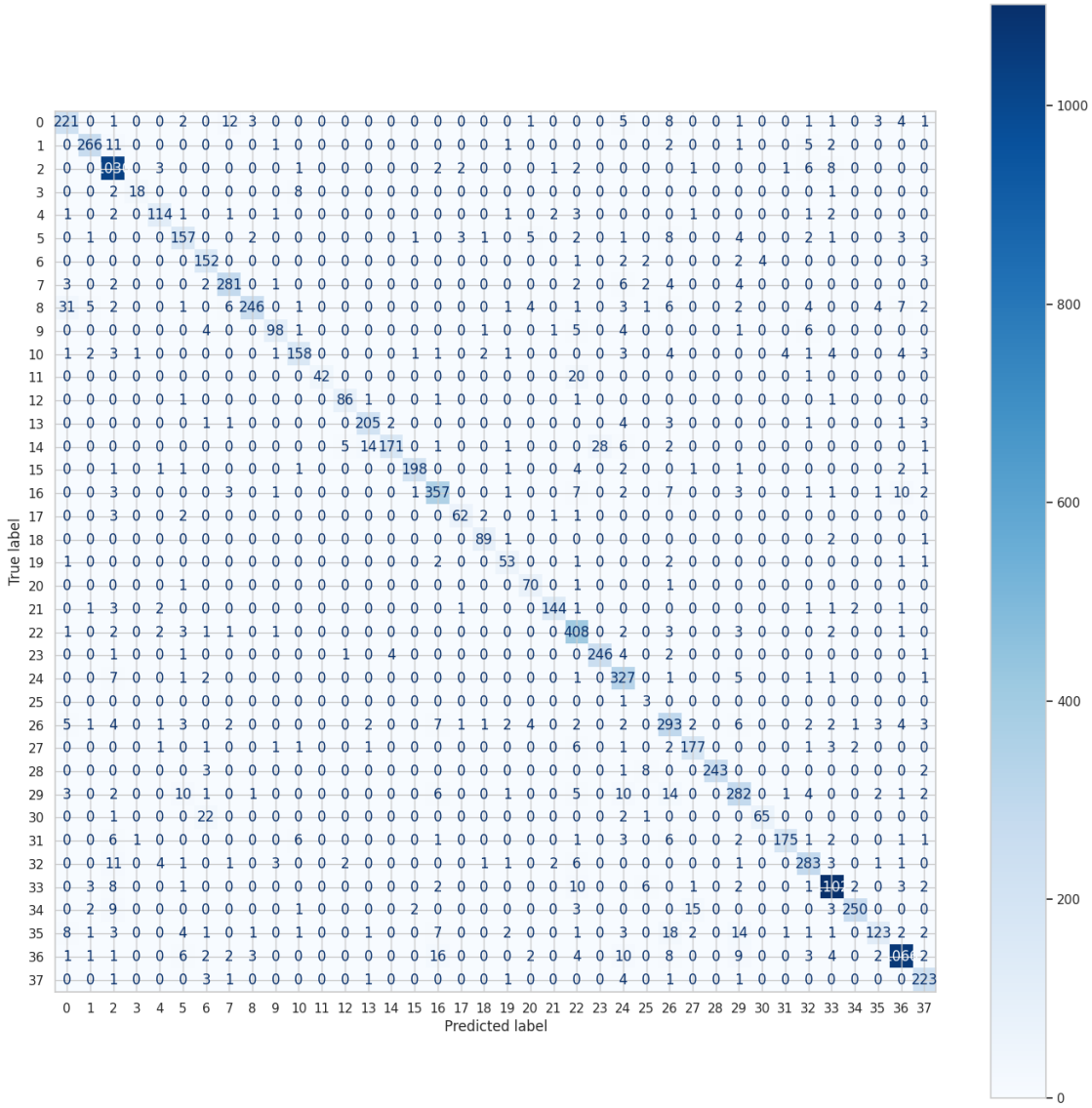
```
            # Determine correctness of predictions
            correct = predicted.eq(y)
            corrects.extend(correct.cpu().tolist())

    images = torch.cat(images, dim=0)
    labels = torch.cat(labels, dim=0)
    probs = torch.cat(probs, dim=0)

    return images, labels, probs, corrects
```

[ ]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
images, labels, probs, corrects = get_predictions(network, testloader, device)
pred_labels = torch.argmax(probs, 1)

print(f"There are {len(corrects)} correct predictions.")
```

```
There are 10562 correct predictions.
```

[27]:
```
# Define the plot_confusion_matrix function
def plot_confusion_matrix(labels, pred_labels):
    fig = plt.figure(figsize=(17, 17))
    ax = fig.add_subplot(1, 1, 1)
    cm = confusion_matrix(labels, pred_labels)
    cm = ConfusionMatrixDisplay(cm, display_labels=range(38))
    cm.plot(values_format='d', cmap='Blues', ax=ax)
```

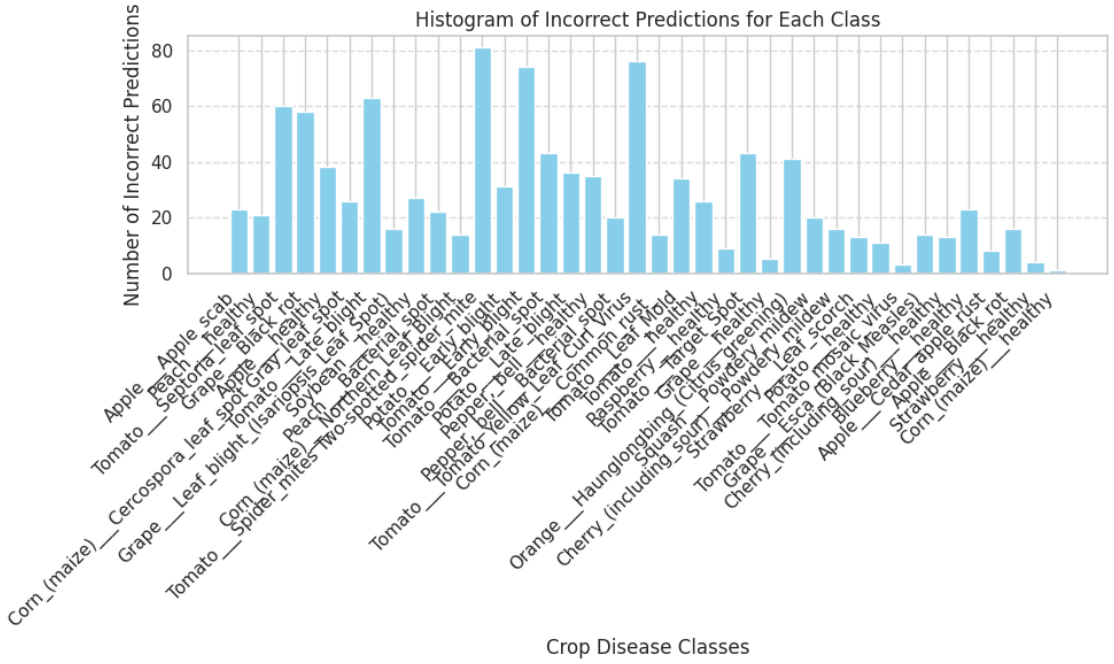
[ ]:
```
plot_confusion_matrix(labels, pred_labels)
```



[ ]:
```
# Now, extract and sort incorrect examples
incorrect_examples = []
for image, label, prob, correct in zip(images, labels, probs, corrects):
    if not correct:
        incorrect_examples.append((image, label, prob))

incorrect_examples.sort(reverse=True, key=lambda x: torch.max(x[2], dim=0).
 ↪values)
print(f"There are {len(incorrect_examples)} incorrect predictions.")
```

```
There are 1078 incorrect predictions.
```

[ ]:
```
incorrect_counts = {}

# Count the occurrences of each class label in the incorrect examples
for _, label, _ in incorrect_examples:
    if label.item() not in incorrect_counts:
        incorrect_counts[label.item()] = 1
    else:
        incorrect_counts[label.item()] += 1

class_names = [train_set.classes[label] for label in incorrect_counts.keys()]

# Plotting the histogram
plt.figure(figsize=(10, 6))
plt.bar(class_names, incorrect_counts.values(), color='skyblue')
plt.xlabel('Crop Disease Classes')
plt.ylabel('Number of Incorrect Predictions')
plt.title('Histogram of Incorrect Predictions for Each Class')
plt.xticks(rotation=45, ha='right')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```



[ ]:
```
data = {'Class Name': class_names, 'Incorrect Counts': list(incorrect_counts.
 ↪values())}
df1 = pd.DataFrame(data)
```

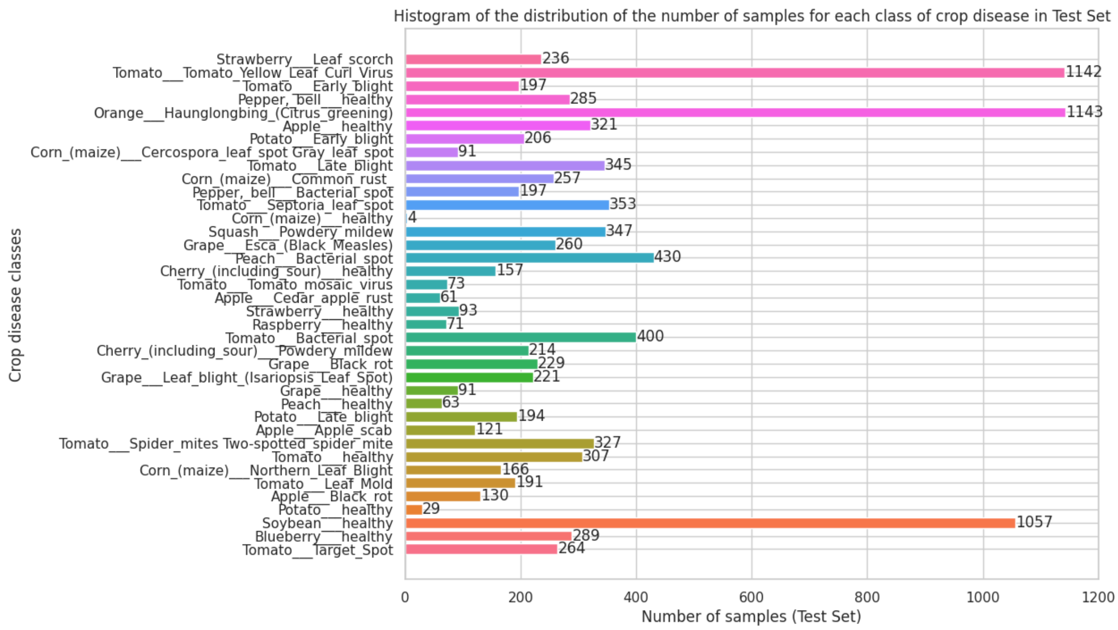
```
df1.head(5)
```

[ ]:

	Class Name	Incorrect Counts
0	Apple___Apple_scab	23
1	Peach___healthy	21
2	Tomato___Septoria_leaf_spot	60
3	Grape___Black_rot	58
4	Apple___healthy	38

[ ]:
```
class_counts = defaultdict(int)
for images, labels in testloader:
    for label in labels:
        class_counts[label.item()] += 1

class_labels = sorted(class_counts.keys())
counts = [class_counts[label] for label in class_labels]
colors = sns.color_palette("husl", len(class_labels))


class_names = [train_set.classes[label] for label in class_labels]

sns.set(style="whitegrid")
plt.figure(figsize=(10, 8))
plt.barh(class_names, counts, color=colors)
plt.xlabel('Number of samples (Test Set)')
plt.ylabel('Crop disease classes')
plt.title('Histogram of the distribution of the number of samples for each class␣
 ↪of crop disease in Test Set')

for i, (count, name) in enumerate(zip(counts, class_names)):
    plt.text(count, i, str(count), ha='left', va='center')

plt.show()
```

```
/usr/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning: os.fork()
was called. os.fork() is incompatible with multithreaded code, and JAX is
multithreaded, so this will likely lead to a deadlock.
  self.pid = os.fork()
/usr/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning: os.fork()
was called. os.fork() is incompatible with multithreaded code, and JAX is
multithreaded, so this will likely lead to a deadlock.
  self.pid = os.fork()
```



[ ]:
```python
data = {'Class Name': class_names, 'Counts': counts}
df2 = pd.DataFrame(data)
df2.head(5)
```

[ ]:
```
              Class Name  Counts
0  Tomato___Target_Spot     264
1   Blueberry___healthy     289
2     Soybean___healthy    1057
3      Potato___healthy      29
4     Apple___Black_rot     130
```

[ ]:
```python
merged_df = pd.merge(df1, df2, on='Class Name', suffixes=('_incorrect', '_test'))
merged_df.head(3)
```

[ ]:
```
                    Class Name  Incorrect Counts  Counts
0           Apple___Apple_scab                23     121
1              Peach___healthy                21      63
2  Tomato___Septoria_leaf_spot                60     353
```

[ ]:
```python
# Calculate the rate of success for each class in percentage
merged_df['Success Rate (%)'] = ((merged_df['Counts'] - merged_df['Incorrect␣
↪Counts']) / merged_df['Counts']) * 100
merged_df
```

[ ]:
```
                                      Class Name  Incorrect Counts  \
0                             Apple___Apple_scab                23
```

1	Peach___healthy	21
2	Tomato___Septoria_leaf_spot	60
3	Grape___Black_rot	58
4	Apple___healthy	38
5	Corn_(maize)___Cercospora_leaf_spot Gray_leaf_...	26
6	Tomato___Late_blight	63
7	Grape___Leaf_blight_(Isariopsis_Leaf_Spot)	16
8	Soybean___healthy	27
9	Peach___Bacterial_spot	22
10	Corn_(maize)___Northern_Leaf_Blight	14
11	Tomato___Spider_mites Two-spotted_spider_mite	81
12	Potato___Early_blight	31
13	Tomato___Early_blight	74
14	Tomato___Bacterial_spot	43
15	Potato___Late_blight	36
16	Pepper,_bell___healthy	35
17	Pepper,_bell___Bacterial_spot	20
18	Tomato___Tomato_Yellow_Leaf_Curl_Virus	76
19	Corn_(maize)___Common_rust_	14
20	Tomato___Leaf_Mold	34
21	Tomato___healthy	26
22	Raspberry___healthy	9
23	Tomato___Target_Spot	43
24	Grape___healthy	5
25	Orange___Haunglongbing_(Citrus_greening)	41
26	Squash___Powdery_mildew	20
27	Cherry_(including_sour)___Powdery_mildew	16
28	Strawberry___Leaf_scorch	13
29	Potato___healthy	11
30	Tomato___Tomato_mosaic_virus	3
31	Grape___Esca_(Black_Measles)	14
32	Cherry_(including_sour)___healthy	13
33	Blueberry___healthy	23
34	Apple___Cedar_apple_rust	8
35	Apple___Black_rot	16
36	Strawberry___healthy	4
37	Corn_(maize)___healthy	1

	Counts	Success Rate (%)
0	121	80.991736
1	63	66.666667
2	353	83.002833
3	229	74.672489
4	321	88.161994
5	91	71.428571
6	345	81.739130
7	221	92.760181

```
8     1057          97.445601
9      430          94.883721
10     166          91.566265
11     327          75.229358
12     206          84.951456
13     197          62.436548
14     400          89.250000
15     194          81.443299
16     285          87.719298
17     197          89.847716
18    1142          93.345009
19     257          94.552529
20     191          82.198953
21     307          91.530945
22      71          87.323944
23     264          83.712121
24      91          94.505495
25    1143          96.412948
26     347          94.236311
27     214          92.523364
28     236          94.491525
29      29          62.068966
30      73          95.890411
31     260          94.615385
32     157          91.719745
33     289          92.041522
34      61          86.885246
35     130          87.692308
36      93          95.698925
37       4          75.000000
```

[38]:
```python
def plot_most_incorrect(incorrect, n_images):
    rows = int(np.sqrt(n_images))
    cols = int(np.sqrt(n_images))

    fig = plt.figure(figsize=(20, 10))
    for i in range(rows*cols):
        ax = fig.add_subplot(rows, cols, i+1)
        image, true_label, probs = incorrect[i]
        true_prob = probs[true_label]
        incorrect_prob, incorrect_label = torch.max(probs, dim=0)

        # Normalize image tensor to [0, 1]
        image = image.permute(1, 2, 0).cpu().numpy()
        image_min, image_max = np.min(image), np.max(image)
        image = (image - image_min) / (image_max - image_min)
```

```
        ax.imshow(image, cmap='bone')
        ax.set_title(f'true label: {true_label} ({true_prob:.3f})\n'
                     f'pred label: {incorrect_label.item()} ({incorrect_prob:.
↪3f})')
        ax.axis('off')
    fig.subplots_adjust(hspace=0.5)
```

[ ]:
```
N_IMAGES = 25

plot_most_incorrect(incorrect_examples, N_IMAGES)
```

[ ]:
```
!pip install tqdm
```

```
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages
(4.66.2)
```

[24]:
```
from tqdm import tqdm

def get_representations(model, iterator, device):

    model.eval()

    outputs = []
    intermediates = []
    labels = []

    with torch.no_grad():
```

```
        for (x, y) in tqdm(iterator):

            x = x.to(device)

            # Ensure only one value is unpacked
            output = model(x)
            if isinstance(output, tuple):
                y_pred, h = output
            else:
                y_pred = output
                h = None

            outputs.append(y_pred.cpu())
            intermediates.append(h.cpu() if h is not None else None)
            labels.append(y)

    outputs = torch.cat(outputs, dim=0)
    intermediates = torch.cat(intermediates, dim=0) if intermediates[0] is not␣
↪None else None
    labels = torch.cat(labels, dim=0)

    return outputs, intermediates, labels
```

[ ]:
```
outputs, intermediates, labels = get_representations(network,
                                                     trainloader,
                                                     device)
```

```
  0%|          | 0/537 [00:00<?,
?it/s]/usr/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning:
os.fork() was called. os.fork() is incompatible with multithreaded code, and JAX
is multithreaded, so this will likely lead to a deadlock.
  self.pid = os.fork()
100%|| 537/537 [03:06<00:00,  2.88it/s]
```

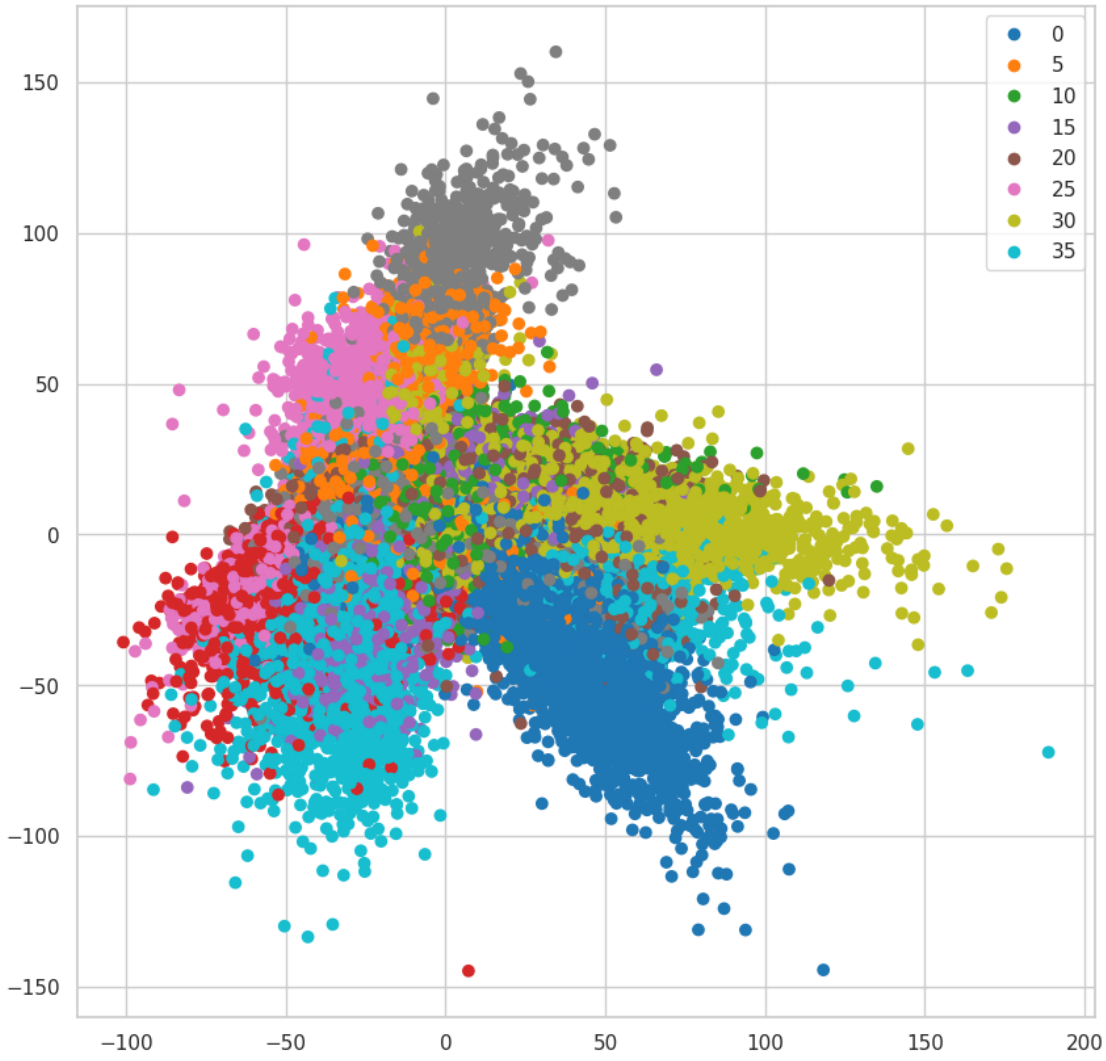
[ ]:
```
def get_pca(data, n_components=2):
    pca = decomposition.PCA()
    pca.n_components = n_components
    pca_data = pca.fit_transform(data)
    return pca_data
```

[ ]:
```
def plot_representations(data, labels, n_images=None):
    if n_images is not None:
        data = data[:n_images]
        labels = labels[:n_images]
    fig = plt.figure(figsize=(10, 10))
    ax = fig.add_subplot(111)
```

```
    scatter = ax.scatter(data[:, 0], data[:, 1], c=labels, cmap='tab10')
    handles, labels = scatter.legend_elements()
    ax.legend(handles=handles, labels=labels)
```

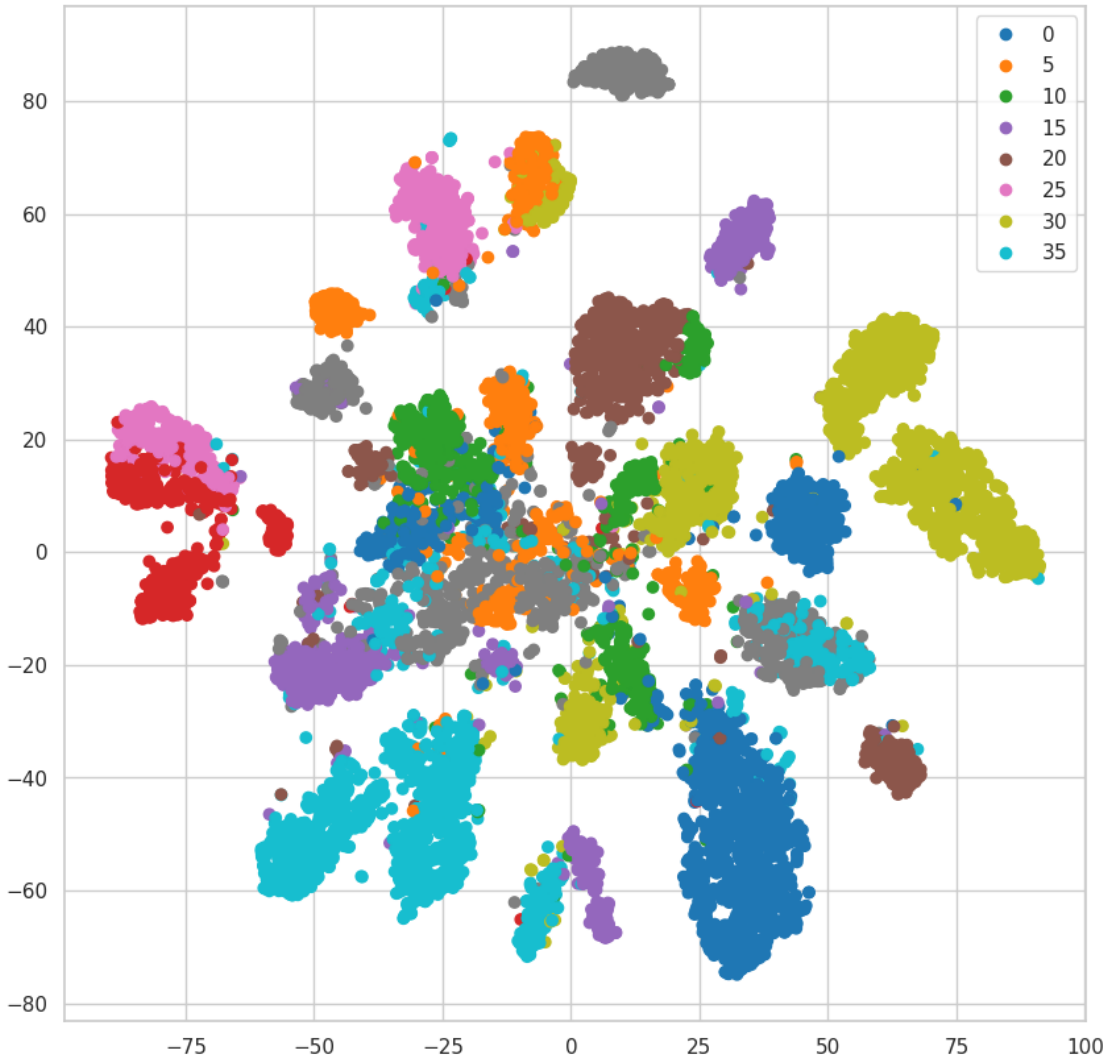
[ ]:
```
output_pca_data = get_pca(outputs)
```

[ ]:
```
plot_representations(output_pca_data, labels)
```



[ ]:
```
def get_tsne(data, n_components=2, n_images=None):
    if n_images is not None:
        data = data[:n_images]
    tsne = manifold.TSNE(n_components=n_components, random_state=0)
    tsne_data = tsne.fit_transform(data)
    return tsne_data
```

[ ]:
```
N_IMAGES = len(test_set)

output_tsne_data = get_tsne(outputs, n_images=N_IMAGES)
plot_representations(output_tsne_data, labels, n_images=N_IMAGES)
```



4 D. 2) Train the model over the grey images

[ ]:
```
# Clone the repository
repo_url = "https://github.com/digitalepidemiologylab/
 ↪plantvillage_deeplearning_paper_dataset.git"
clone_dir = "plantvillage_deeplearning_paper_dataset"
subprocess.run(["git", "clone", repo_url, clone_dir])
os.chdir(clone_dir)
```

```
# Raw color images
classes = extract_folder(repo_url, clone_dir, "raw/grayscale")
```

```
Folder 'raw/grayscale' extracted successfully.
Directory 'Tomato___Late_blight' contains 1909 files.
Directory 'Apple___Cedar_apple_rust' contains 275 files.
Directory 'Corn_(maize)___Common_rust_' contains 1192 files.
Directory 'Peach___Bacterial_spot' contains 2297 files.
Directory 'Grape___Black_rot' contains 1180 files.
Directory 'Blueberry___healthy' contains 1502 files.
Directory 'Pepper,_bell___Bacterial_spot' contains 997 files.
Directory 'Grape___Leaf_blight_(Isariopsis_Leaf_Spot)' contains 1076 files.
Directory 'Peach___healthy' contains 360 files.
Directory 'Pepper,_bell___healthy' contains 1478 files.
Directory 'Corn_(maize)___healthy' contains 1162 files.
Directory 'Tomato___Spider_mites Two-spotted_spider_mite' contains 1676 files.
Directory 'Corn_(maize)___Northern_Leaf_Blight' contains 985 files.
Directory 'Orange___Haunglongbing_(Citrus_greening)' contains 5507 files.
Directory 'Squash___Powdery_mildew' contains 1835 files.
Directory 'Potato___healthy' contains 152 files.
Directory 'Apple___healthy' contains 1645 files.
Directory 'Strawberry___healthy' contains 456 files.
Directory 'Tomato___Septoria_leaf_spot' contains 1771 files.
Directory 'Tomato___Leaf_Mold' contains 952 files.
Directory 'Cherry_(including_sour)___healthy' contains 854 files.
Directory 'Tomato___Tomato_mosaic_virus' contains 373 files.
Directory 'Potato___Late_blight' contains 1000 files.
Directory 'Grape___healthy' contains 423 files.
Directory 'Potato___Early_blight' contains 1000 files.
Directory 'Apple___Black_rot' contains 621 files.
Directory 'Cherry_(including_sour)___Powdery_mildew' contains 1052 files.
Directory 'Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot' contains 513
files.
Directory 'Tomato___healthy' contains 1591 files.
Directory 'Tomato___Target_Spot' contains 1404 files.
Directory 'Tomato___Tomato_Yellow_Leaf_Curl_Virus' contains 5357 files.
Directory 'Tomato___Bacterial_spot' contains 2127 files.
Directory 'Soybean___healthy' contains 5090 files.
Directory 'Apple___Apple_scab' contains 630 files.
Directory 'Raspberry___healthy' contains 371 files.
Directory 'Grape___Esca_(Black_Measles)' contains 1383 files.
Directory 'Strawberry___Leaf_scorch' contains 1109 files.
Directory 'Tomato___Early_blight' contains 1000 files.
Total number of files: 54305

total number of classes : '38'.
['Tomato___Late_blight', 'Apple___Cedar_apple_rust',
```

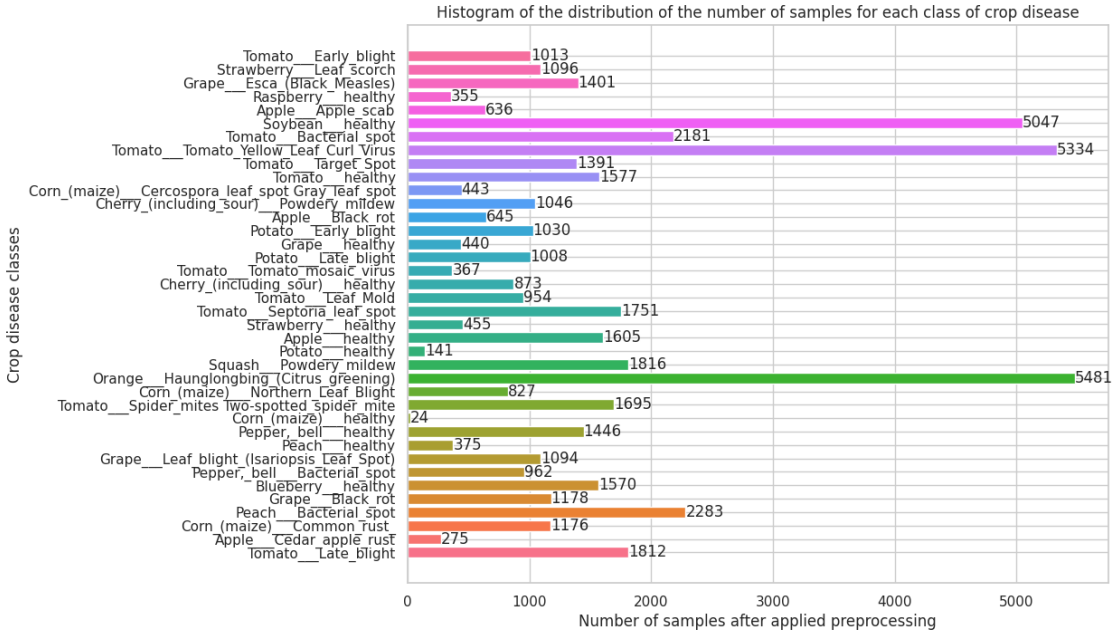
```
'Corn_(maize)___Common_rust_', 'Peach___Bacterial_spot', 'Grape___Black_rot',
'Blueberry___healthy', 'Pepper,_bell___Bacterial_spot',
'Grape___Leaf_blight_(Isariopsis_Leaf_Spot)', 'Peach___healthy',
'Pepper,_bell___healthy', 'Corn_(maize)___healthy', 'Tomato___Spider_mites Two-
spotted_spider_mite', 'Corn_(maize)___Northern_Leaf_Blight',
'Orange___Haunglongbing_(Citrus_greening)', 'Squash___Powdery_mildew',
'Potato___healthy', 'Apple___healthy', 'Strawberry___healthy',
'Tomato___Septoria_leaf_spot', 'Tomato___Leaf_Mold',
'Cherry_(including_sour)___healthy', 'Tomato___Tomato_mosaic_virus',
'Potato___Late_blight', 'Grape___healthy', 'Potato___Early_blight',
'Apple___Black_rot', 'Cherry_(including_sour)___Powdery_mildew',
'Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot', 'Tomato___healthy',
'Tomato___Target_Spot', 'Tomato___Tomato_Yellow_Leaf_Curl_Virus',
'Tomato___Bacterial_spot', 'Soybean___healthy', 'Apple___Apple_scab',
'Raspberry___healthy', 'Grape___Esca_(Black_Measles)',
'Strawberry___Leaf_scorch', 'Tomato___Early_blight']
```

[ ]:
```
# Spliiting the dataset
extracted_folder = "raw/grayscale"
train_set = CropDiseaseDataset(root_dir=extracted_folder, train=True,␣
↪validation=False, gray_scale=True)
validation_set = CropDiseaseDataset(root_dir=extracted_folder, train=False,␣
↪validation=True, gray_scale=True)
test_set = CropDiseaseDataset(root_dir=extracted_folder, train=False,␣
↪validation=False, gray_scale=True)

trainloader = torch.utils.data.DataLoader(train_set, batch_size=64,␣
↪shuffle=True, num_workers=2)
validationloader = torch.utils.data.DataLoader(validation_set, batch_size=64,␣
↪shuffle=False, num_workers=2)
testloader = torch.utils.data.DataLoader(test_set, batch_size=64, shuffle=False,␣
↪num_workers=2)
```

[ ]:
```
merged_dataset = ConcatDataset([train_set, validation_set, test_set])

# Create a single loader for the combined dataset
mergedloader = DataLoader(merged_dataset, batch_size=64, shuffle=True,␣
↪num_workers=2)
```

[ ]:
```
class_counts = defaultdict(int)
for images, labels in mergedloader:
    for label in labels:
        class_counts[label.item()] += 1

class_labels = sorted(class_counts.keys())
counts = [class_counts[label] for label in class_labels]
colors = sns.color_palette("husl", len(class_labels))
```

```
class_names = [train_set.classes[label] for label in class_labels]

sns.set(style="whitegrid")
plt.figure(figsize=(10, 8))
plt.barh(class_names, counts, color=colors)
plt.xlabel('Number of samples after applied preprocessing')
plt.ylabel('Crop disease classes')
plt.title('Histogram of the distribution of the number of samples for each class␣
 ↪of crop disease')

for i, (count, name) in enumerate(zip(counts, class_names)):
    plt.text(count, i, str(count), ha='left', va='center')

plt.show()
```



[ ]:
```
# Combining train and validation datasets for testing the model
combined_train_set = ConcatDataset([train_set, validation_set])
combined_train_loader = torch.utils.data.DataLoader(combined_train_set,␣
 ↪batch_size=64, shuffle=True, num_workers=2)
```

[ ]:
```
# Define hyperparameters
learning_rate = 1e-3
num_epochs = 25

network = CNN(grayscale=True).to(device)
```

```
network.apply(initialize_parameters)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(network.parameters(), lr=learning_rate)

print(f'The model has {count_parameters(network):,} trainable parameters')
```

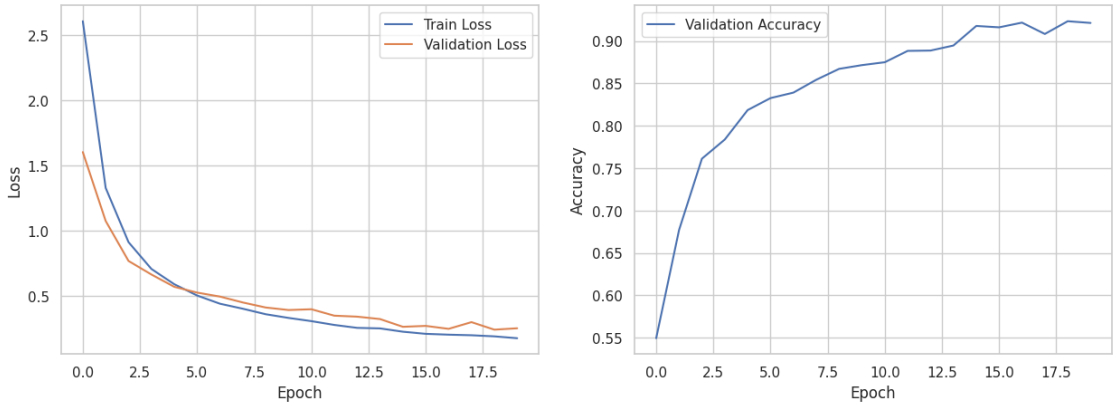
```
The model has 6,846,182 trainable parameters
```

[ ]:
```
# Train the model
train_avg_loss, validation_avg_loss, validation_accuracy, train_accuracy =␣
↪train(network, num_epochs, trainloader, validationloader, criterion,␣
↪optimizer,validation_phase=True)
```

```
Epoch [1/20], Validation Loss: 1.6054, Validation Accuracy: 54.95%, Train Loss:
2.6091, Train Accuracy: 36.80%
| Epoch Time: 0m 42s
Epoch [2/20], Validation Loss: 1.0772, Validation Accuracy: 67.71%, Train Loss:
1.3309, Train Accuracy: 61.05%
| Epoch Time: 0m 41s
Epoch [3/20], Validation Loss: 0.7704, Validation Accuracy: 76.12%, Train Loss:
0.9146, Train Accuracy: 71.57%
| Epoch Time: 0m 35s
Epoch [4/20], Validation Loss: 0.6666, Validation Accuracy: 78.38%, Train Loss:
0.7092, Train Accuracy: 77.35%
| Epoch Time: 0m 33s
Epoch [5/20], Validation Loss: 0.5722, Validation Accuracy: 81.86%, Train Loss:
0.5921, Train Accuracy: 81.03%
| Epoch Time: 0m 34s
Epoch [6/20], Validation Loss: 0.5278, Validation Accuracy: 83.27%, Train Loss:
0.5053, Train Accuracy: 83.48%
| Epoch Time: 0m 35s
Epoch [7/20], Validation Loss: 0.4972, Validation Accuracy: 83.91%, Train Loss:
0.4425, Train Accuracy: 85.19%
| Epoch Time: 0m 35s
Epoch [8/20], Validation Loss: 0.4520, Validation Accuracy: 85.43%, Train Loss:
0.4040, Train Accuracy: 86.79%
| Epoch Time: 0m 34s
Epoch [9/20], Validation Loss: 0.4134, Validation Accuracy: 86.72%, Train Loss:
0.3619, Train Accuracy: 88.12%
| Epoch Time: 0m 35s
Epoch [10/20], Validation Loss: 0.3941, Validation Accuracy: 87.16%, Train Loss:
0.3335, Train Accuracy: 89.00%
| Epoch Time: 0m 35s
Epoch [11/20], Validation Loss: 0.4001, Validation Accuracy: 87.50%, Train Loss:
0.3088, Train Accuracy: 89.83%
| Epoch Time: 0m 35s
Epoch [12/20], Validation Loss: 0.3507, Validation Accuracy: 88.84%, Train Loss:
0.2798, Train Accuracy: 90.71%
```

```
| Epoch Time: 0m 35s
Epoch [13/20], Validation Loss: 0.3432, Validation Accuracy: 88.88%, Train Loss:
0.2573, Train Accuracy: 91.49%
| Epoch Time: 0m 34s
Epoch [14/20], Validation Loss: 0.3248, Validation Accuracy: 89.47%, Train Loss:
0.2532, Train Accuracy: 91.45%
| Epoch Time: 0m 34s
Epoch [15/20], Validation Loss: 0.2655, Validation Accuracy: 91.78%, Train Loss:
0.2278, Train Accuracy: 92.47%
| Epoch Time: 0m 36s
Epoch [16/20], Validation Loss: 0.2724, Validation Accuracy: 91.62%, Train Loss:
0.2115, Train Accuracy: 93.07%
| Epoch Time: 0m 35s
Epoch [17/20], Validation Loss: 0.2497, Validation Accuracy: 92.17%, Train Loss:
0.2051, Train Accuracy: 93.28%
| Epoch Time: 0m 34s
Epoch [18/20], Validation Loss: 0.3011, Validation Accuracy: 90.83%, Train Loss:
0.2006, Train Accuracy: 93.28%
| Epoch Time: 0m 35s
Epoch [19/20], Validation Loss: 0.2435, Validation Accuracy: 92.34%, Train Loss:
0.1922, Train Accuracy: 93.76%
| Epoch Time: 0m 34s
Epoch [20/20], Validation Loss: 0.2541, Validation Accuracy: 92.13%, Train Loss:
0.1778, Train Accuracy: 94.07%
| Epoch Time: 0m 35s
```

[ ]:
```
plot_training_results(train_avg_loss, validation_avg_loss, validation_accuracy,␣
↪is_validation=True)
```



[ ]:
```
# Test the model
network = CNN(grayscale=True).to(device)
network.apply(initialize_parameters)
criterion = nn.CrossEntropyLoss()
```

```
optimizer = torch.optim.Adam(network.parameters(), lr=learning_rate)

train_avg_loss, test_avg_loss, test_accuracy, train_accuracy = train(network,␣
 ↪num_epochs, combined_train_loader, testloader, criterion,␣
 ↪optimizer,validation_phase=False)
```

```
Epoch [1/25], Test Loss: 1.7172, Test Accuracy: 50.78%, Train Loss: 2.3190,
Train Accuracy: 42.54%
| Epoch Time: 0m 33s
Epoch [2/25], Test Loss: 0.8323, Test Accuracy: 73.48%, Train Loss: 1.0529,
Train Accuracy: 68.46%
| Epoch Time: 0m 31s
Epoch [3/25], Test Loss: 0.7229, Test Accuracy: 77.05%, Train Loss: 0.7042,
Train Accuracy: 78.11%
| Epoch Time: 0m 33s
Epoch [4/25], Test Loss: 0.4907, Test Accuracy: 84.00%, Train Loss: 0.5606,
Train Accuracy: 81.93%
| Epoch Time: 0m 31s
Epoch [5/25], Test Loss: 0.7209, Test Accuracy: 77.61%, Train Loss: 0.4717,
Train Accuracy: 84.80%
| Epoch Time: 0m 33s
Epoch [6/25], Test Loss: 0.4400, Test Accuracy: 85.35%, Train Loss: 0.4541,
Train Accuracy: 85.46%
| Epoch Time: 0m 31s
Epoch [7/25], Test Loss: 0.3997, Test Accuracy: 87.07%, Train Loss: 0.3484,
Train Accuracy: 88.79%
| Epoch Time: 0m 33s
Epoch [8/25], Test Loss: 0.3278, Test Accuracy: 89.38%, Train Loss: 0.3116,
Train Accuracy: 89.84%
| Epoch Time: 0m 31s
Epoch [9/25], Test Loss: 0.3531, Test Accuracy: 88.52%, Train Loss: 0.2761,
Train Accuracy: 90.86%
| Epoch Time: 0m 33s
Epoch [10/25], Test Loss: 0.2792, Test Accuracy: 91.25%, Train Loss: 0.2498,
Train Accuracy: 91.78%
| Epoch Time: 0m 31s
Epoch [11/25], Test Loss: 0.2628, Test Accuracy: 91.56%, Train Loss: 0.2423,
Train Accuracy: 91.97%
| Epoch Time: 0m 33s
Epoch [12/25], Test Loss: 0.2685, Test Accuracy: 91.57%, Train Loss: 0.2226,
Train Accuracy: 92.61%
| Epoch Time: 0m 31s
Epoch [13/25], Test Loss: 0.2460, Test Accuracy: 92.43%, Train Loss: 0.2113,
Train Accuracy: 93.10%
| Epoch Time: 0m 32s
Epoch [14/25], Test Loss: 0.2656, Test Accuracy: 91.86%, Train Loss: 0.1896,
Train Accuracy: 93.83%
| Epoch Time: 0m 32s
```

```
Epoch [15/25], Test Loss: 0.2523, Test Accuracy: 92.02%, Train Loss: 0.1852,
Train Accuracy: 93.91%
| Epoch Time: 0m 32s
Epoch [16/25], Test Loss: 0.3001, Test Accuracy: 91.14%, Train Loss: 0.1656,
Train Accuracy: 94.51%
| Epoch Time: 0m 33s
Epoch [17/25], Test Loss: 0.2508, Test Accuracy: 92.46%, Train Loss: 0.1604,
Train Accuracy: 94.67%
| Epoch Time: 0m 31s
Epoch [18/25], Test Loss: 0.2286, Test Accuracy: 93.09%, Train Loss: 0.1603,
Train Accuracy: 94.84%
| Epoch Time: 0m 33s
Epoch [19/25], Test Loss: 0.2289, Test Accuracy: 93.31%, Train Loss: 0.1426,
Train Accuracy: 95.45%
| Epoch Time: 0m 32s
Epoch [20/25], Test Loss: 0.2485, Test Accuracy: 93.19%, Train Loss: 0.1473,
Train Accuracy: 95.37%
| Epoch Time: 0m 33s
Epoch [21/25], Test Loss: 0.2828, Test Accuracy: 92.18%, Train Loss: 0.1380,
Train Accuracy: 95.54%
| Epoch Time: 0m 31s
Epoch [22/25], Test Loss: 0.2161, Test Accuracy: 93.82%, Train Loss: 0.1390,
Train Accuracy: 95.63%
| Epoch Time: 0m 32s
Epoch [23/25], Test Loss: 0.2313, Test Accuracy: 93.73%, Train Loss: 0.1278,
Train Accuracy: 95.90%
| Epoch Time: 0m 32s
Epoch [24/25], Test Loss: 0.2608, Test Accuracy: 93.14%, Train Loss: 0.1208,
Train Accuracy: 96.08%
| Epoch Time: 0m 32s
Epoch [25/25], Test Loss: 0.2242, Test Accuracy: 93.79%, Train Loss: 0.1158,
Train Accuracy: 96.30%
| Epoch Time: 0m 31s
```

[ ]:
```
plot_training_results(train_avg_loss, test_avg_loss, test_accuracy,␣
↪is_validation=False)
```

Examining the model.

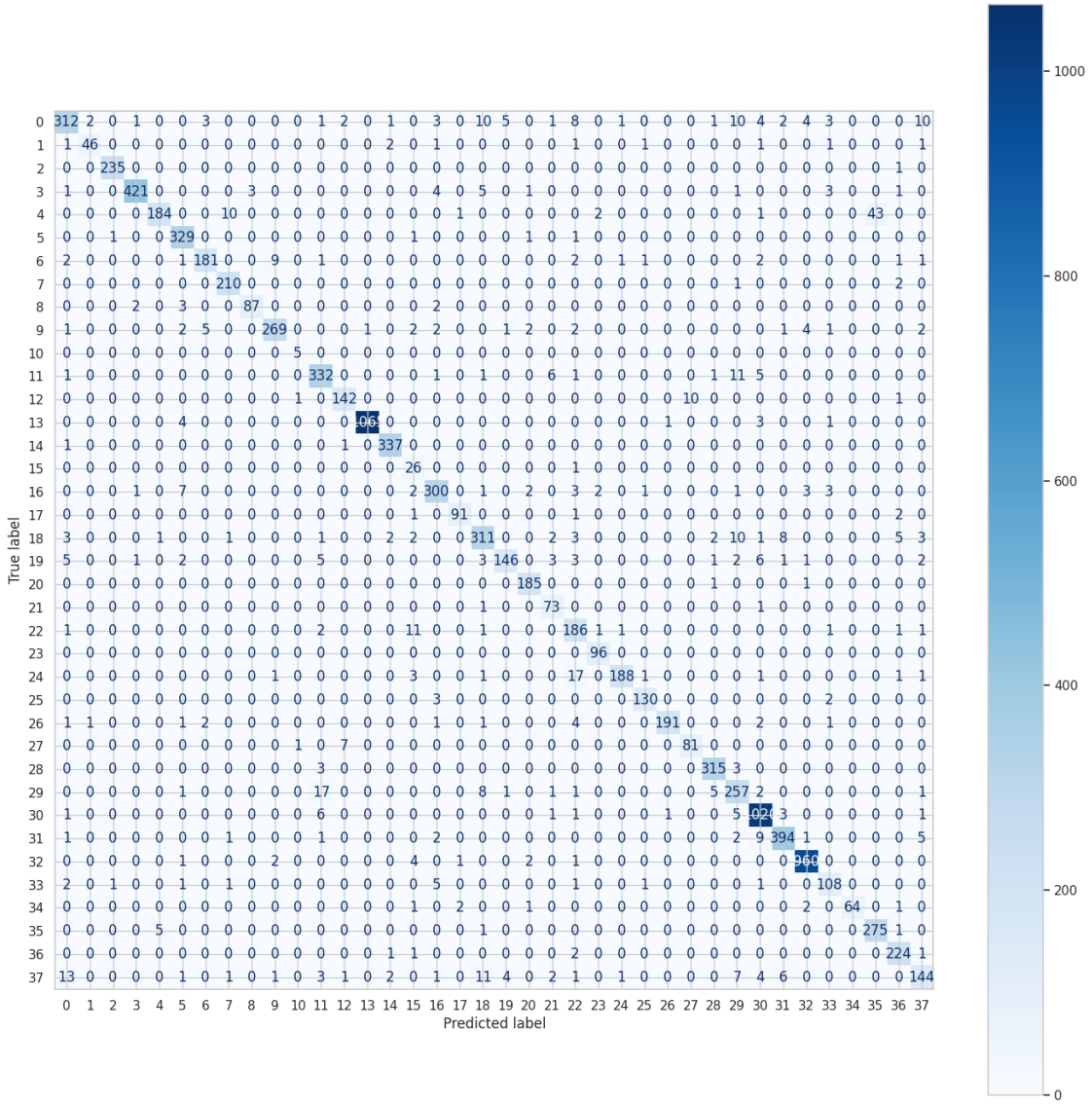
[ ]:
```
network.load_state_dict(torch.load('model.pt'))
```

[ ]: <All keys matched successfully>

[ ]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
images, labels, probs, corrects = get_predictions(network, testloader, device)
pred_labels = torch.argmax(probs, 1)

print(f"There are {len(corrects)} correct predictions.")
```

There are 10562 correct predictions.

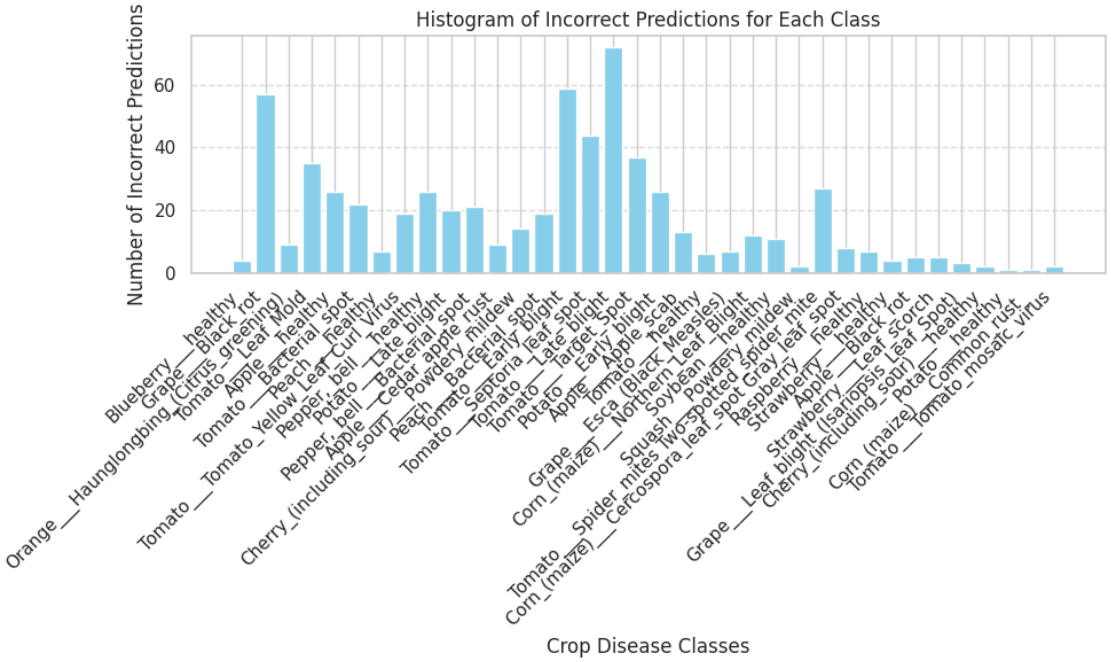
[ ]:
```
plot_confusion_matrix(labels, pred_labels)
```



[ ]:
```
# Now, extract and sort incorrect examples
incorrect_examples = []
for image, label, prob, correct in zip(images, labels, probs, corrects):
    if not correct:
        incorrect_examples.append((image, label, prob))

incorrect_examples.sort(reverse=True, key=lambda x: torch.max(x[2], dim=0).
 ↪values)
print(f"There are {len(incorrect_examples)} incorrect predictions.")
```

```
There are 642 incorrect predictions.
```

[ ]:
```
incorrect_counts = {}

# Count the occurrences of each class label in the incorrect examples
for _, label, _ in incorrect_examples:
    if label.item() not in incorrect_counts:
        incorrect_counts[label.item()] = 1
    else:
        incorrect_counts[label.item()] += 1

class_names = [train_set.classes[label] for label in incorrect_counts.keys()]

# Plotting the histogram
plt.figure(figsize=(10, 6))
plt.bar(class_names, incorrect_counts.values(), color='skyblue')
plt.xlabel('Crop Disease Classes')
plt.ylabel('Number of Incorrect Predictions')
plt.title('Histogram of Incorrect Predictions for Each Class')
plt.xticks(rotation=45, ha='right')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```



[ ]:
```
data = {'Class Name': class_names, 'Incorrect Counts': list(incorrect_counts.
↪values())}
df1 = pd.DataFrame(data)
```

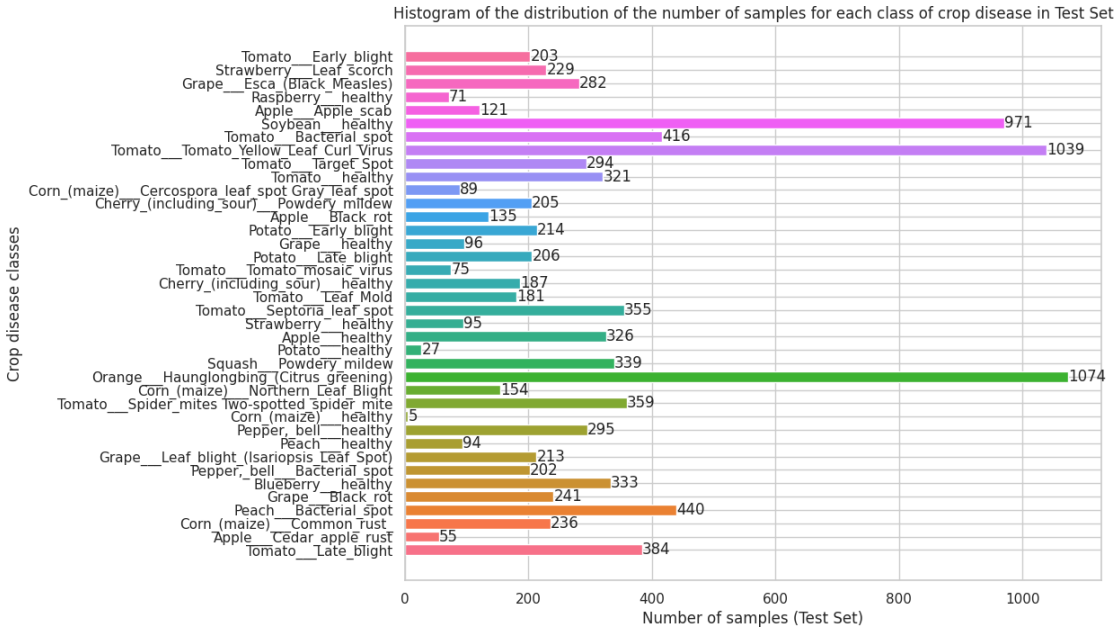
```
df1.head(5)
```

[ ]:

	Class Name	Incorrect Counts
0	Blueberry___healthy	4
1	Grape___Black_rot	57
2	Orange___Haunglongbing_(Citrus_greening)	9
3	Tomato___Leaf_Mold	35
4	Apple___healthy	26

[ ]:
```
class_counts = defaultdict(int)
for images, labels in testloader:
    for label in labels:
        class_counts[label.item()] += 1

class_labels = sorted(class_counts.keys())
counts = [class_counts[label] for label in class_labels]
colors = sns.color_palette("husl", len(class_labels))


class_names = [train_set.classes[label] for label in class_labels]

sns.set(style="whitegrid")
plt.figure(figsize=(10, 8))
plt.barh(class_names, counts, color=colors)
plt.xlabel('Number of samples (Test Set)')
plt.ylabel('Crop disease classes')
plt.title('Histogram of the distribution of the number of samples for each class␣
 ↪of crop disease in Test Set')

for i, (count, name) in enumerate(zip(counts, class_names)):
    plt.text(count, i, str(count), ha='left', va='center')

plt.show()
```

```
/usr/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning: os.fork()
was called. os.fork() is incompatible with multithreaded code, and JAX is
multithreaded, so this will likely lead to a deadlock.
  self.pid = os.fork()
/usr/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning: os.fork()
was called. os.fork() is incompatible with multithreaded code, and JAX is
multithreaded, so this will likely lead to a deadlock.
  self.pid = os.fork()
```



[ ]:
```
data = {'Class Name': class_names, 'Counts': counts}
df2 = pd.DataFrame(data)
df2.head(5)
```

[ ]:

	Class Name	Counts
0	Tomato___Late_blight	384
1	Apple___Cedar_apple_rust	55
2	Corn_(maize)___Common_rust_	236
3	Peach___Bacterial_spot	440
4	Grape___Black_rot	241

[ ]:
```
merged_df = pd.merge(df1, df2, on='Class Name', suffixes=('_incorrect', '_test'))
merged_df.head(3)
```

[ ]:

	Class Name	Incorrect Counts	Counts
0	Blueberry___healthy	4	333
1	Grape___Black_rot	57	241
2	Orange___Haunglongbing_(Citrus_greening)	9	1074

[ ]:
```
# Calculate the rate of success for each class in percentage
merged_df['Success Rate (%)'] = ((merged_df['Counts'] - merged_df['Incorrect␣
↪Counts']) / merged_df['Counts']) * 100
merged_df
```

[ ]:

	Class Name	Incorrect Counts	\
0	Blueberry___healthy	4	

1	Grape___Black_rot	57
2	Orange___Haunglongbing_(Citrus_greening)	9
3	Tomato___Leaf_Mold	35
4	Apple___healthy	26
5	Tomato___Bacterial_spot	22
6	Peach___healthy	7
7	Tomato___Tomato_Yellow_Leaf_Curl_Virus	19
8	Pepper,_bell___healthy	26
9	Potato___Late_blight	20
10	Pepper,_bell___Bacterial_spot	21
11	Apple___Cedar_apple_rust	9
12	Cherry_(including_sour)___Powdery_mildew	14
13	Peach___Bacterial_spot	19
14	Tomato___Early_blight	59
15	Tomato___Septoria_leaf_spot	44
16	Tomato___Late_blight	72
17	Tomato___Target_Spot	37
18	Potato___Early_blight	26
19	Apple___Apple_scab	13
20	Tomato___healthy	6
21	Grape___Esca_(Black_Measles)	7
22	Corn_(maize)___Northern_Leaf_Blight	12
23	Soybean___healthy	11
24	Squash___Powdery_mildew	2
25	Tomato___Spider_mites Two-spotted_spider_mite	27
26	Corn_(maize)___Cercospora_leaf_spot Gray_leaf_...	8
27	Raspberry___healthy	7
28	Strawberry___healthy	4
29	Apple___Black_rot	5
30	Strawberry___Leaf_scorch	5
31	Grape___Leaf_blight_(Isariopsis_Leaf_Spot)	3
32	Cherry_(including_sour)___healthy	2
33	Potato___healthy	1
34	Corn_(maize)___Common_rust_	1
35	Tomato___Tomato_mosaic_virus	2

	Counts	Success Rate (%)
0	333	98.798799
1	241	76.348548
2	1074	99.162011
3	181	80.662983
4	326	92.024540
5	416	94.711538
6	94	92.553191
7	1039	98.171319
8	295	91.186441
9	206	90.291262

```
10     202         89.603960
11      55         83.636364
12     205         93.170732
13     440         95.681818
14     203         70.935961
15     355         87.605634
16     384         81.250000
17     294         87.414966
18     214         87.850467
19     121         89.256198
20     321         98.130841
21     282         97.517730
22     154         92.207792
23     971         98.867147
24     339         99.410029
25     359         92.479109
26      89         91.011236
27      71         90.140845
28      95         95.789474
29     135         96.296296
30     229         97.816594
31     213         98.591549
32     187         98.930481
33      27         96.296296
34     236         99.576271
35      75         97.333333
```

[ ]:
```
N_IMAGES = 25

plot_most_incorrect(incorrect_examples, N_IMAGES)
```

[ ]:

[ ]:

[ ]:

5 D. 3) Train the model over the segmented images

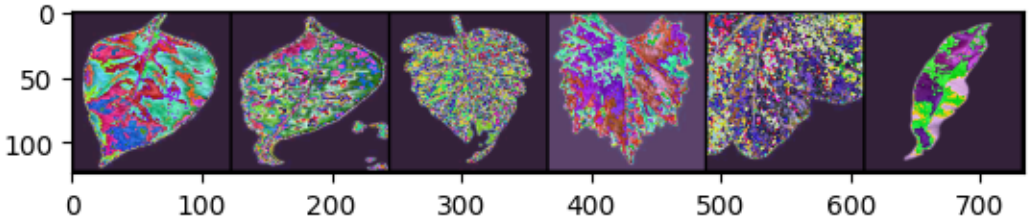
[4]:
```
repo_url = "https://github.com/digitalepidemiologylab/
 ↪plantvillage_deeplearning_paper_dataset.git"
clone_dir = "plantvillage_deeplearning_paper_dataset"
subprocess.run(["git", "clone", repo_url, clone_dir])
os.chdir(clone_dir)

# Raw segmented images
classes = extract_folder(repo_url, clone_dir, "raw/segmented")
```

```
Folder 'raw/segmented' extracted successfully.
Directory 'Apple___Black_rot' contains 621 files.
Directory 'Tomato___Leaf_Mold' contains 952 files.
Directory 'Raspberry___healthy' contains 371 files.
Directory 'Tomato___Tomato_Yellow_Leaf_Curl_Virus' contains 5357 files.
Directory 'Apple___Apple_scab' contains 630 files.
Directory 'Tomato___Early_blight' contains 1000 files.
Directory 'Strawberry___healthy' contains 456 files.
Directory 'Apple___Cedar_apple_rust' contains 275 files.
```

```
Directory 'Soybean___healthy' contains 5090 files.
Directory 'Potato___Late_blight' contains 1000 files.
Directory 'Peach___Bacterial_spot' contains 2297 files.
Directory 'Pepper,_bell___healthy' contains 1478 files.
Directory 'Grape___Leaf_blight_(Isariopsis_Leaf_Spot)' contains 1076 files.
Directory 'Strawberry___Leaf_scorch' contains 1109 files.
Directory 'Tomato___Tomato_mosaic_virus' contains 373 files.
Directory 'Grape___healthy' contains 423 files.
Directory 'Tomato___Target_Spot' contains 1404 files.
Directory 'Squash___Powdery_mildew' contains 1835 files.
Directory 'Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot' contains 513
files.
Directory 'Blueberry___healthy' contains 1502 files.
Directory 'Tomato___Spider_mites Two-spotted_spider_mite' contains 1676 files.
Directory 'Peach___healthy' contains 360 files.
Directory 'Grape___Esca_(Black_Measles)' contains 1384 files.
Directory 'Corn_(maize)___Common_rust_' contains 1192 files.
Directory 'Corn_(maize)___Northern_Leaf_Blight' contains 985 files.
Directory 'Apple___healthy' contains 1645 files.
Directory 'Tomato___Septoria_leaf_spot' contains 1771 files.
Directory 'Orange___Haunglongbing_(Citrus_greening)' contains 5507 files.
Directory 'Cherry_(including_sour)___healthy' contains 854 files.
Directory 'Tomato___Bacterial_spot' contains 2127 files.
Directory 'Potato___Early_blight' contains 1000 files.
Directory 'Corn_(maize)___healthy' contains 1162 files.
Directory 'Cherry_(including_sour)___Powdery_mildew' contains 1052 files.
Directory 'Tomato___healthy' contains 1591 files.
Directory 'Potato___healthy' contains 152 files.
Directory 'Pepper,_bell___Bacterial_spot' contains 997 files.
Directory 'Tomato___Late_blight' contains 1909 files.
Directory 'Grape___Black_rot' contains 1180 files.
Total number of files: 54306

total number of classes : '38'.
['Apple___Black_rot', 'Tomato___Leaf_Mold', 'Raspberry___healthy',
'Tomato___Tomato_Yellow_Leaf_Curl_Virus', 'Apple___Apple_scab',
'Tomato___Early_blight', 'Strawberry___healthy', 'Apple___Cedar_apple_rust',
'Soybean___healthy', 'Potato___Late_blight', 'Peach___Bacterial_spot',
'Pepper,_bell___healthy', 'Grape___Leaf_blight_(Isariopsis_Leaf_Spot)',
'Strawberry___Leaf_scorch', 'Tomato___Tomato_mosaic_virus', 'Grape___healthy',
'Tomato___Target_Spot', 'Squash___Powdery_mildew',
'Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot', 'Blueberry___healthy',
'Tomato___Spider_mites Two-spotted_spider_mite', 'Peach___healthy',
'Grape___Esca_(Black_Measles)', 'Corn_(maize)___Common_rust_',
'Corn_(maize)___Northern_Leaf_Blight', 'Apple___healthy',
'Tomato___Septoria_leaf_spot', 'Orange___Haunglongbing_(Citrus_greening)',
'Cherry_(including_sour)___healthy', 'Tomato___Bacterial_spot',
'Potato___Early_blight', 'Corn_(maize)___healthy',
```

```
'Cherry_(including_sour)___Powdery_mildew', 'Tomato___healthy',
'Potato___healthy', 'Pepper,_bell___Bacterial_spot', 'Tomato___Late_blight',
'Grape___Black_rot']
```

[ ]:
```
extracted_folder = "raw/segmented"
my_dataset = CropDiseaseDataset(root_dir=extracted_folder, train=True,␣
 ↪gray_scale=False, segmented=True)
train_loader = DataLoader(my_dataset, batch_size=6, shuffle=True, num_workers=2)

def show_images(img):
    plt.imshow(transforms.functional.to_pil_image(img))
    plt.show()

images, labels = next(iter(train_loader))
show_images(utils.make_grid(images))
print(*[my_dataset.classes[l] for l in labels])
```




```
Pepper,_bell___healthy Cherry_(including_sour)___Powdery_mildew Tomato___healthy
Grape___Black_rot Squash___Powdery_mildew Peach___Bacterial_spot
```

[11]:
```
# Define hyperparameters
learning_rate = 1e-3
num_epochs = 25

network = CNN(grayscale=False).to(device)
network.apply(initialize_parameters)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(network.parameters(), lr=learning_rate)

print(f'The model has {count_parameters(network):,} trainable parameters')
```

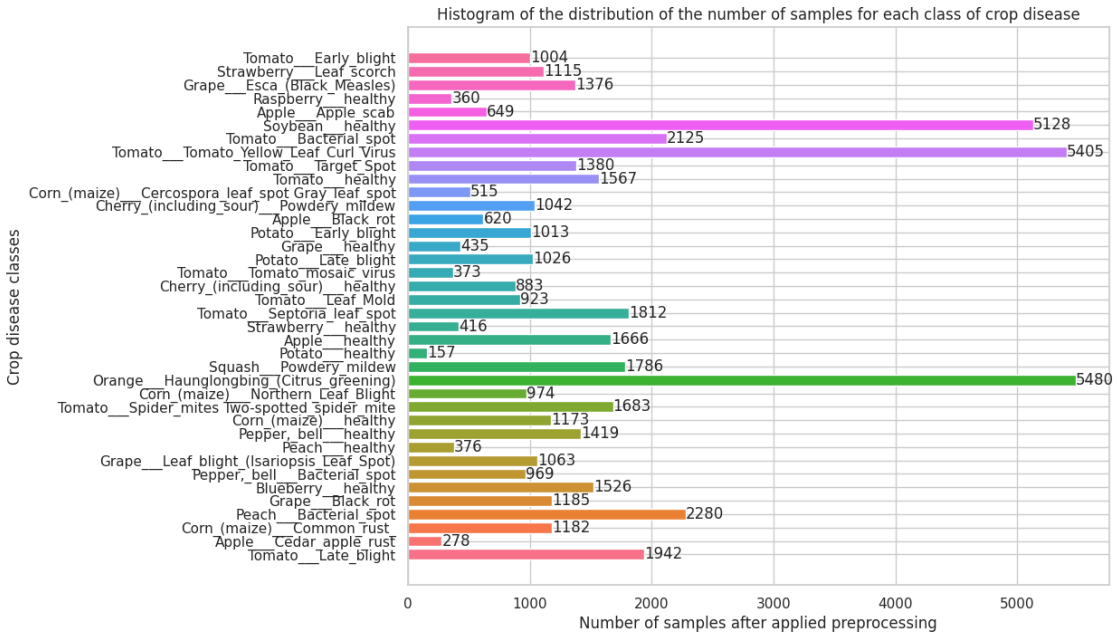
```
The model has 6,846,758 trainable parameters
```

[12]:
```
# Splitting the dataset
extracted_folder = "raw/segmented"
train_set = CropDiseaseDataset(root_dir=extracted_folder, train=True,␣
 ↪validation=False, gray_scale=False, segmented=True)
```

```
validation_set = CropDiseaseDataset(root_dir=extracted_folder, train=False,␣
↪validation=True, gray_scale=False, segmented=True)
test_set = CropDiseaseDataset(root_dir=extracted_folder, train=False,␣
↪validation=False, gray_scale=False, segmented=True)

trainloader = torch.utils.data.DataLoader(train_set, batch_size=64,␣
↪shuffle=True, num_workers=2)
validationloader = torch.utils.data.DataLoader(validation_set, batch_size=64,␣
↪shuffle=False, num_workers=2)
testloader = torch.utils.data.DataLoader(test_set, batch_size=64, shuffle=False,␣
↪num_workers=2)
```

[13]:
```
merged_dataset = ConcatDataset([train_set, validation_set, test_set])

# Create a single loader for the combined dataset
mergedloader = DataLoader(merged_dataset, batch_size=64, shuffle=True,␣
↪num_workers=2)
```

[ ]:
```
class_counts = defaultdict(int)
for images, labels in mergedloader:
    for label in labels:
        class_counts[label.item()] += 1

class_labels = sorted(class_counts.keys())
counts = [class_counts[label] for label in class_labels]
colors = sns.color_palette("husl", len(class_labels))

class_names = [train_set.classes[label] for label in class_labels]

sns.set(style="whitegrid")
plt.figure(figsize=(10, 8))
plt.barh(class_names, counts, color=colors)
plt.xlabel('Number of samples after applied preprocessing')
plt.ylabel('Crop disease classes')
plt.title('Histogram of the distribution of the number of samples for each class␣
↪of crop disease')

for i, (count, name) in enumerate(zip(counts, class_names)):
    plt.text(count, i, str(count), ha='left', va='center')

plt.show()
```

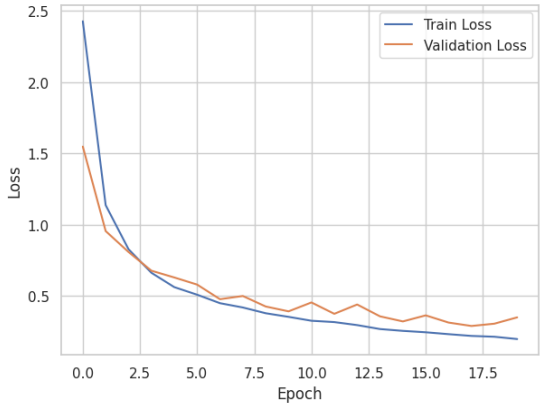


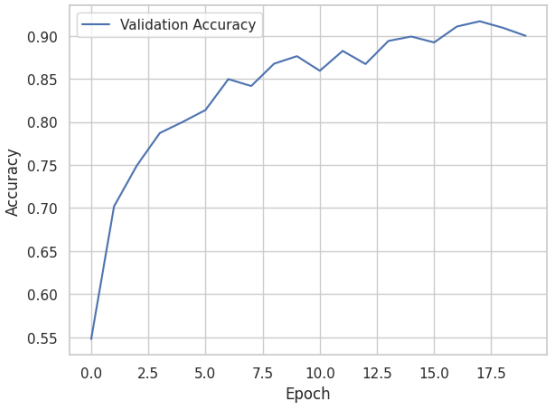
[ ]:
```
# Train the models
train_avg_loss, validation_avg_loss, validation_accuracy, train_accuracy =␣
↪train(network, num_epochs, trainloader, validationloader, criterion,␣
↪optimizer,validation_phase=True)
```

```
Epoch [1/20], Validation Loss: 1.5477, Validation Accuracy: 54.78%, Train Loss:
2.4259, Train Accuracy: 40.44%
| Epoch Time: 0m 33s
Epoch [2/20], Validation Loss: 0.9554, Validation Accuracy: 70.19%, Train Loss:
1.1370, Train Accuracy: 65.83%
| Epoch Time: 0m 31s
Epoch [3/20], Validation Loss: 0.8088, Validation Accuracy: 74.94%, Train Loss:
0.8282, Train Accuracy: 73.92%
| Epoch Time: 0m 31s
Epoch [4/20], Validation Loss: 0.6777, Validation Accuracy: 78.70%, Train Loss:
0.6631, Train Accuracy: 78.95%
| Epoch Time: 0m 31s
Epoch [5/20], Validation Loss: 0.6303, Validation Accuracy: 79.99%, Train Loss:
0.5627, Train Accuracy: 81.72%
| Epoch Time: 0m 32s
Epoch [6/20], Validation Loss: 0.5804, Validation Accuracy: 81.38%, Train Loss:
0.5094, Train Accuracy: 83.40%
| Epoch Time: 0m 33s
Epoch [7/20], Validation Loss: 0.4780, Validation Accuracy: 84.96%, Train Loss:
0.4500, Train Accuracy: 85.12%
| Epoch Time: 0m 31s
Epoch [8/20], Validation Loss: 0.5004, Validation Accuracy: 84.16%, Train Loss:
```

```
0.4199, Train Accuracy: 86.27%
| Epoch Time: 0m 31s
Epoch [9/20], Validation Loss: 0.4268, Validation Accuracy: 86.75%, Train Loss:
0.3796, Train Accuracy: 87.57%
| Epoch Time: 0m 33s
Epoch [10/20], Validation Loss: 0.3928, Validation Accuracy: 87.62%, Train Loss:
0.3542, Train Accuracy: 88.37%
| Epoch Time: 0m 31s
Epoch [11/20], Validation Loss: 0.4552, Validation Accuracy: 85.93%, Train Loss:
0.3269, Train Accuracy: 89.06%
| Epoch Time: 0m 32s
Epoch [12/20], Validation Loss: 0.3755, Validation Accuracy: 88.24%, Train Loss:
0.3175, Train Accuracy: 89.73%
| Epoch Time: 0m 32s
Epoch [13/20], Validation Loss: 0.4408, Validation Accuracy: 86.72%, Train Loss:
0.2964, Train Accuracy: 90.19%
| Epoch Time: 0m 32s
Epoch [14/20], Validation Loss: 0.3577, Validation Accuracy: 89.39%, Train Loss:
0.2687, Train Accuracy: 91.12%
| Epoch Time: 0m 31s
Epoch [15/20], Validation Loss: 0.3222, Validation Accuracy: 89.91%, Train Loss:
0.2559, Train Accuracy: 91.79%
| Epoch Time: 0m 31s
Epoch [16/20], Validation Loss: 0.3646, Validation Accuracy: 89.22%, Train Loss:
0.2461, Train Accuracy: 92.11%
| Epoch Time: 0m 31s
Epoch [17/20], Validation Loss: 0.3141, Validation Accuracy: 91.07%, Train Loss:
0.2329, Train Accuracy: 92.29%
| Epoch Time: 0m 31s
Epoch [18/20], Validation Loss: 0.2903, Validation Accuracy: 91.68%, Train Loss:
0.2206, Train Accuracy: 92.67%
| Epoch Time: 0m 31s
Epoch [19/20], Validation Loss: 0.3063, Validation Accuracy: 90.93%, Train Loss:
0.2148, Train Accuracy: 92.97%
| Epoch Time: 0m 31s
Epoch [20/20], Validation Loss: 0.3506, Validation Accuracy: 89.99%, Train Loss:
0.1986, Train Accuracy: 93.47%
| Epoch Time: 0m 33s
```

[ ]:
```
plot_training_results(train_avg_loss, validation_avg_loss, validation_accuracy,␣
↪is_validation=True)
```





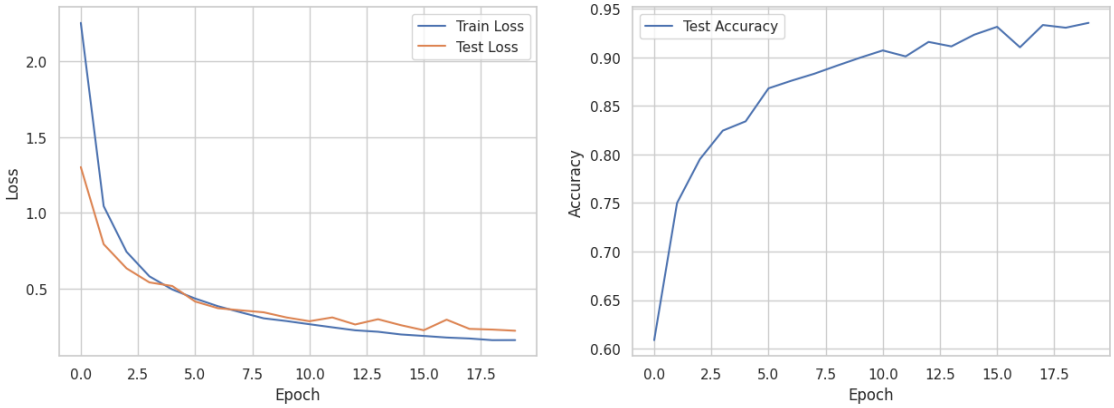
[14]:
```
# Combining train and validation datasets for testing the model
combined_train_set = ConcatDataset([train_set, validation_set])
combined_train_loader = torch.utils.data.DataLoader(combined_train_set,␣
↪batch_size=64, shuffle=True, num_workers=2)
```

[20]:
```
# Test the model
num_epochs = 20
network = CNN(grayscale=False).to(device)
network.apply(initialize_parameters)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(network.parameters(), lr=learning_rate)

train_avg_loss, test_avg_loss, test_accuracy, train_accuracy = train(network,␣
↪num_epochs, combined_train_loader, testloader, criterion,␣
↪optimizer,validation_phase=False)
```

```
Epoch [1/20], Test Loss: 1.3019, Test Accuracy: 60.88%, Train Loss: 2.2548,
Train Accuracy: 43.15%
| Epoch Time: 0m 46s
Epoch [2/20], Test Loss: 0.7924, Test Accuracy: 75.00%, Train Loss: 1.0440,
Train Accuracy: 68.17%
| Epoch Time: 0m 46s
Epoch [3/20], Test Loss: 0.6328, Test Accuracy: 79.51%, Train Loss: 0.7420,
Train Accuracy: 76.63%
| Epoch Time: 0m 42s
Epoch [4/20], Test Loss: 0.5399, Test Accuracy: 82.44%, Train Loss: 0.5800,
Train Accuracy: 81.23%
| Epoch Time: 0m 42s
Epoch [5/20], Test Loss: 0.5161, Test Accuracy: 83.40%, Train Loss: 0.4931,
Train Accuracy: 84.05%
| Epoch Time: 0m 43s
Epoch [6/20], Test Loss: 0.4137, Test Accuracy: 86.80%, Train Loss: 0.4333,
Train Accuracy: 85.87%
```

```
| Epoch Time: 0m 42s
Epoch [7/20], Test Loss: 0.3694, Test Accuracy: 87.58%, Train Loss: 0.3824,
Train Accuracy: 87.54%
| Epoch Time: 0m 43s
Epoch [8/20], Test Loss: 0.3554, Test Accuracy: 88.30%, Train Loss: 0.3420,
Train Accuracy: 88.88%
| Epoch Time: 0m 42s
Epoch [9/20], Test Loss: 0.3423, Test Accuracy: 89.14%, Train Loss: 0.3026,
Train Accuracy: 90.14%
| Epoch Time: 0m 43s
Epoch [10/20], Test Loss: 0.3081, Test Accuracy: 89.95%, Train Loss: 0.2845,
Train Accuracy: 90.70%
| Epoch Time: 0m 44s
Epoch [11/20], Test Loss: 0.2834, Test Accuracy: 90.70%, Train Loss: 0.2635,
Train Accuracy: 91.45%
| Epoch Time: 0m 44s
Epoch [12/20], Test Loss: 0.3084, Test Accuracy: 90.09%, Train Loss: 0.2432,
Train Accuracy: 92.20%
| Epoch Time: 0m 42s
Epoch [13/20], Test Loss: 0.2618, Test Accuracy: 91.58%, Train Loss: 0.2233,
Train Accuracy: 92.75%
| Epoch Time: 0m 43s
Epoch [14/20], Test Loss: 0.2966, Test Accuracy: 91.12%, Train Loss: 0.2138,
Train Accuracy: 93.09%
| Epoch Time: 0m 43s
Epoch [15/20], Test Loss: 0.2572, Test Accuracy: 92.32%, Train Loss: 0.1959,
Train Accuracy: 93.64%
| Epoch Time: 0m 43s
Epoch [16/20], Test Loss: 0.2239, Test Accuracy: 93.13%, Train Loss: 0.1861,
Train Accuracy: 94.06%
| Epoch Time: 0m 43s
Epoch [17/20], Test Loss: 0.2936, Test Accuracy: 91.02%, Train Loss: 0.1756,
Train Accuracy: 94.41%
| Epoch Time: 0m 44s
Epoch [18/20], Test Loss: 0.2328, Test Accuracy: 93.32%, Train Loss: 0.1692,
Train Accuracy: 94.54%
| Epoch Time: 0m 43s
Epoch [19/20], Test Loss: 0.2282, Test Accuracy: 93.04%, Train Loss: 0.1582,
Train Accuracy: 94.95%
| Epoch Time: 0m 44s
Epoch [20/20], Test Loss: 0.2204, Test Accuracy: 93.54%, Train Loss: 0.1586,
Train Accuracy: 94.99%
| Epoch Time: 0m 43s
```

[21]:
```
plot_training_results(train_avg_loss, test_avg_loss, test_accuracy,␣
↪is_validation=False)
```



Examining the model

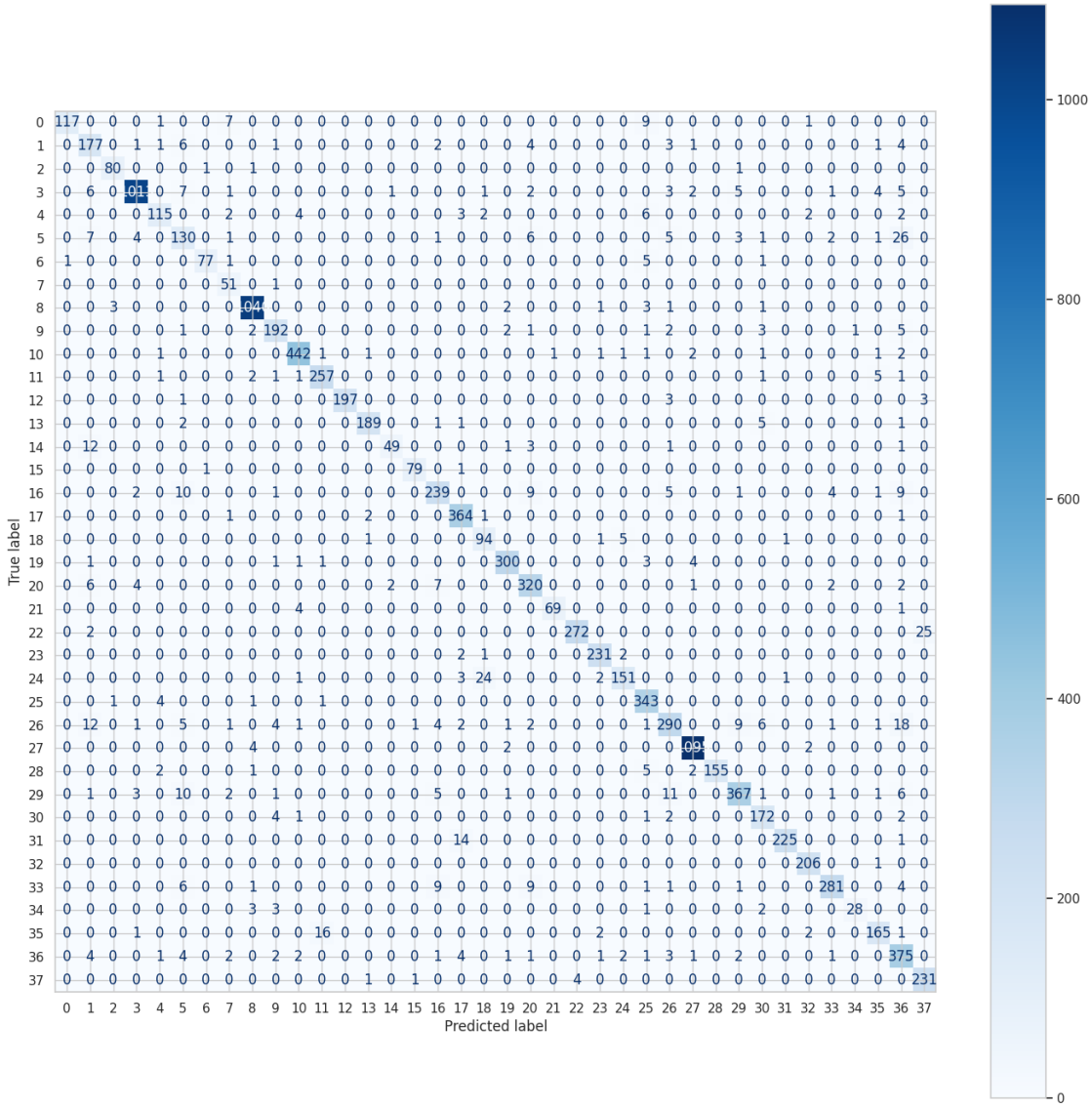
[22]:
```
network.load_state_dict(torch.load('model.pt'))
```

[22]:
```
<All keys matched successfully>
```

[28]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
images, labels, probs, corrects = get_predictions(network, testloader, device)
pred_labels = torch.argmax(probs, 1)

print(f"There are {len(corrects)} correct predictions.")
```

```
There are 10863 correct predictions.
```

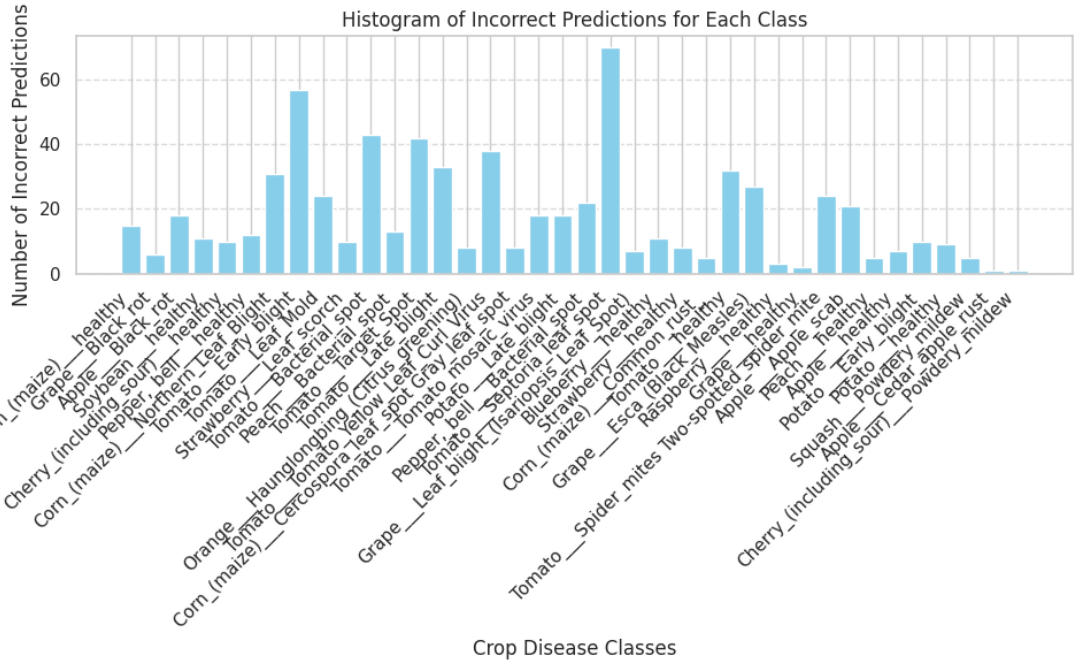
[29]:
```
plot_confusion_matrix(labels, pred_labels)
```



[30]:
```
# Now, extract and sort incorrect examples
incorrect_examples = []
for image, label, prob, correct in zip(images, labels, probs, corrects):
    if not correct:
        incorrect_examples.append((image, label, prob))

incorrect_examples.sort(reverse=True, key=lambda x: torch.max(x[2], dim=0).
 ↪values)
print(f"There are {len(incorrect_examples)} incorrect predictions.")
```

```
There are 685 incorrect predictions.
```

[31]:
```
incorrect_counts = {}

# Count the occurrences of each class label in the incorrect examples
for _, label, _ in incorrect_examples:
    if label.item() not in incorrect_counts:
        incorrect_counts[label.item()] = 1
    else:
        incorrect_counts[label.item()] += 1

class_names = [train_set.classes[label] for label in incorrect_counts.keys()]

# Plotting the histogram
plt.figure(figsize=(10, 6))
plt.bar(class_names, incorrect_counts.values(), color='skyblue')
plt.xlabel('Crop Disease Classes')
plt.ylabel('Number of Incorrect Predictions')
plt.title('Histogram of Incorrect Predictions for Each Class')
plt.xticks(rotation=45, ha='right')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```



[32]:
```
data = {'Class Name': class_names, 'Incorrect Counts': list(incorrect_counts.
 ↪values())}
df1 = pd.DataFrame(data)
```

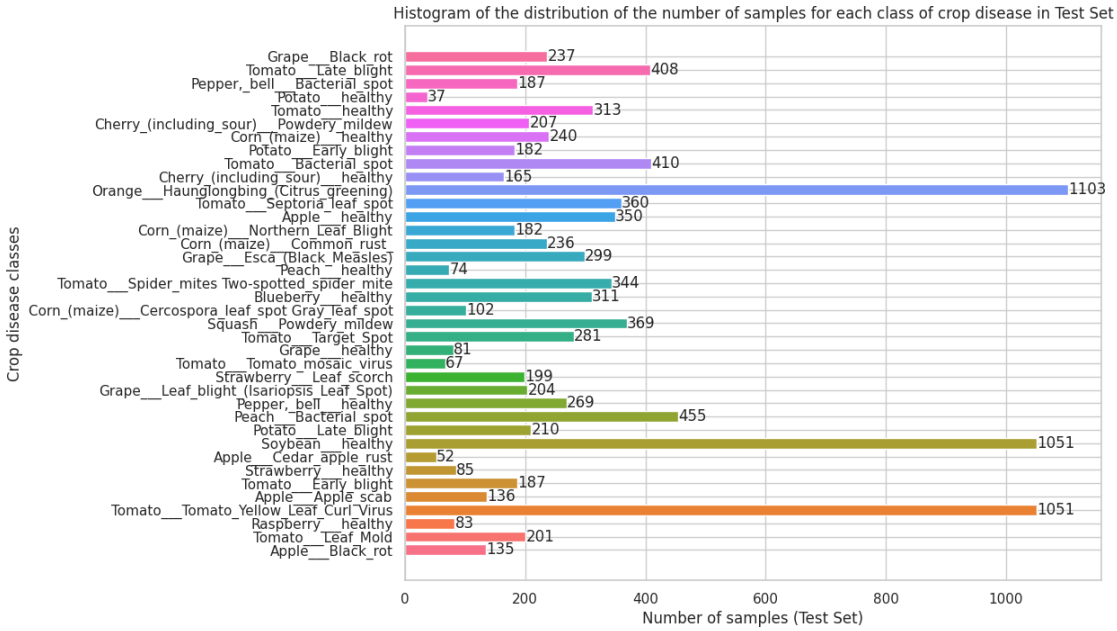
```
df1.head(5)
```

[32]:

	Class Name	Incorrect Counts
0	Corn_(maize)___healthy	15
1	Grape___Black_rot	6
2	Apple___Black_rot	18
3	Soybean___healthy	11
4	Cherry_(including_sour)___healthy	10

[33]:
```
class_counts = defaultdict(int)
for images, labels in testloader:
    for label in labels:
        class_counts[label.item()] += 1

class_labels = sorted(class_counts.keys())
counts = [class_counts[label] for label in class_labels]
colors = sns.color_palette("husl", len(class_labels))


class_names = [train_set.classes[label] for label in class_labels]

sns.set(style="whitegrid")
plt.figure(figsize=(10, 8))
plt.barh(class_names, counts, color=colors)
plt.xlabel('Number of samples (Test Set)')
plt.ylabel('Crop disease classes')
plt.title('Histogram of the distribution of the number of samples for each class␣
 ↪of crop disease in Test Set')

for i, (count, name) in enumerate(zip(counts, class_names)):
    plt.text(count, i, str(count), ha='left', va='center')

plt.show()
```

```
/usr/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning: os.fork()
was called. os.fork() is incompatible with multithreaded code, and JAX is
multithreaded, so this will likely lead to a deadlock.
  self.pid = os.fork()
/usr/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning: os.fork()
was called. os.fork() is incompatible with multithreaded code, and JAX is
multithreaded, so this will likely lead to a deadlock.
  self.pid = os.fork()
```



[34]:
```
data = {'Class Name': class_names, 'Counts': counts}
df2 = pd.DataFrame(data)
df2.head(5)
```

[34]:

	Class Name	Counts
0	Apple___Black_rot	135
1	Tomato___Leaf_Mold	201
2	Raspberry___healthy	83
3	Tomato___Tomato_Yellow_Leaf_Curl_Virus	1051
4	Apple___Apple_scab	136

[35]:
```
merged_df = pd.merge(df1, df2, on='Class Name', suffixes=('_incorrect', '_test'))
merged_df.head(3)
```

[35]:

	Class Name	Incorrect Counts	Counts
0	Corn_(maize)___healthy	15	240
1	Grape___Black_rot	6	237
2	Apple___Black_rot	18	135

[36]:
```
# Calculate the rate of success for each class in percentage
merged_df['Success Rate (%)'] = ((merged_df['Counts'] - merged_df['Incorrect␣
 ↪Counts']) / merged_df['Counts']) * 100
merged_df
```

[36]:

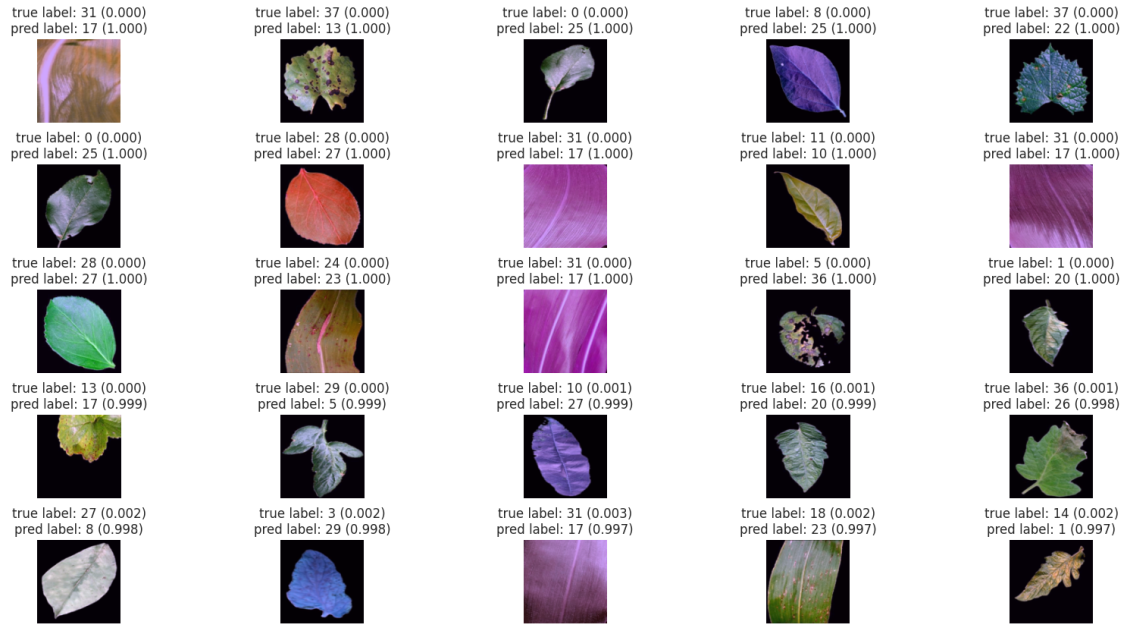
	Class Name	Incorrect Counts	\
0	Corn_(maize)___healthy	15	

1	Grape___Black_rot	6
2	Apple___Black_rot	18
3	Soybean___healthy	11
4	Cherry_(including_sour)___healthy	10
5	Pepper,_bell___healthy	12
6	Corn_(maize)___Northern_Leaf_Blight	31
7	Tomato___Early_blight	57
8	Tomato___Leaf_Mold	24
9	Strawberry___Leaf_scorch	10
10	Tomato___Bacterial_spot	43
11	Peach___Bacterial_spot	13
12	Tomato___Target_Spot	42
13	Tomato___Late_blight	33
14	Orange___Haunglongbing_(Citrus_greening)	8
15	Tomato___Tomato_Yellow_Leaf_Curl_Virus	38
16	Corn_(maize)___Cercospora_leaf_spot Gray_leaf_...	8
17	Tomato___Tomato_mosaic_virus	18
18	Potato___Late_blight	18
19	Pepper,_bell___Bacterial_spot	22
20	Tomato___Septoria_leaf_spot	70
21	Grape___Leaf_blight_(Isariopsis_Leaf_Spot)	7
22	Blueberry___healthy	11
23	Strawberry___healthy	8
24	Corn_(maize)___Common_rust_	5
25	Tomato___healthy	32
26	Grape___Esca_(Black_Measles)	27
27	Raspberry___healthy	3
28	Grape___healthy	2
29	Tomato___Spider_mites Two-spotted_spider_mite	24
30	Apple___Apple_scab	21
31	Peach___healthy	5
32	Apple___healthy	7
33	Potato___Early_blight	10
34	Potato___healthy	9
35	Squash___Powdery_mildew	5
36	Apple___Cedar_apple_rust	1
37	Cherry_(including_sour)___Powdery_mildew	1

	Counts	Success Rate (%)
0	240	93.750000
1	237	97.468354
2	135	86.666667
3	1051	98.953378
4	165	93.939394
5	269	95.539033
6	182	82.967033
7	187	69.518717

```
8      201          88.059701
9      199          94.974874
10     410          89.512195
11     455          97.142857
12     281          85.053381
13     408          91.911765
14    1103          99.274705
15    1051          96.384396
16     102          92.156863
17      67          73.134328
18     210          91.428571
19     187          88.235294
20     360          80.555556
21     204          96.568627
22     311          96.463023
23      85          90.588235
24     236          97.881356
25     313          89.776358
26     299          90.969900
27      83          96.385542
28      81          97.530864
29     344          93.023256
30     136          84.558824
31      74          93.243243
32     350          98.000000
33     182          94.505495
34      37          75.675676
35     369          98.644986
36      52          98.076923
37     207          99.516908
```

[39]:
```
N_IMAGES = 25

plot_most_incorrect(incorrect_examples, N_IMAGES)
```


true label: 31 (0.000)
pred label: 17 (1.000)
true label: 37 (0.000)
pred label: 13 (1.000)
true label: 0 (0.000)
pred label: 25 (1.000)
true label: 8 (0.000)
pred label: 25 (1.000)
true label: 37 (0.000)
pred label: 22 (1.000)
true label: 0 (0.000)
pred label: 25 (1.000)
true label: 28 (0.000)
pred label: 27 (1.000)
true label: 31 (0.000)
pred label: 17 (1.000)
true label: 11 (0.000)
pred label: 10 (1.000)
true label: 31 (0.000)
pred label: 17 (1.000)
true label: 28 (0.000)
pred label: 27 (1.000)
true label: 24 (0.000)
pred label: 23 (1.000)
true label: 31 (0.000)
pred label: 17 (1.000)
true label: 5 (0.000)
pred label: 36 (1.000)
true label: 1 (0.000)
pred label: 20 (1.000)
true label: 13 (0.000)
pred label: 17 (0.999)
true label: 29 (0.000)
pred label: 5 (0.999)
true label: 10 (0.001)
pred label: 27 (0.999)
true label: 16 (0.001)
pred label: 20 (0.999)
true label: 36 (0.001)
pred label: 26 (0.998)
true label: 27 (0.002)
pred label: 8 (0.998)
true label: 3 (0.002)
pred label: 29 (0.998)
true label: 31 (0.003)
pred label: 17 (0.997)
true label: 18 (0.002)
pred label: 23 (0.997)
true label: 14 (0.002)
pred label: 1 (0.997)

[ ]: