Set-up environment

Firstly, install Transformers as well as Datasets.

```
!pip install -q git+https://github.com/huggingface/transformers.git
```

```
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.0/3.0 MB 45.1 MB/s eta 0:00:00
  Building wheel for transformers (pyproject.toml) ... done
```

Change the transformers version because the newest one is not stable for this model.

```
!pip install transformers==4.30.0
```

```
Collecting transformers==4.30.0
  Downloading transformers-4.30.0-py3-none-any.whl.metadata (113 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 113.6/113.6 kB 7.0 MB/s eta 0:00:00
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: huggingface-hub<1.0,>=0.14.1 in /usr/local/lib/python3
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (f
Collecting tokenizers!=0.11.3,<0.14,>=0.11.1 (from transformers==4.30.0)
  Downloading tokenizers-0.13.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_6
Requirement already satisfied: safetensors>=0.3.1 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.1
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-p
Downloading transformers-4.30.0-py3-none-any.whl (7.2 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.2/7.2 MB 47.0 MB/s eta 0:00:00
Downloading tokenizers-0.13.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.8/7.8 MB 59.7 MB/s eta 0:00:00
Installing collected packages: tokenizers, transformers
  Attempting uninstall: tokenizers
    Found existing installation: tokenizers 0.21.0
    Uninstalling tokenizers-0.21.0:
      Successfully uninstalled tokenizers-0.21.0
  Attempting uninstall: transformers
    Found existing installation: transformers 4.48.0.dev0
    Uninstalling transformers-4.48.0.dev0:
      Successfully uninstalled transformers-4.48.0.dev0
ERROR: pip's dependency resolver does not currently take into account all the package
sentence-transformers 3.2.1 requires transformers<5.0.0,>=4.41.0, but you have transf
Successfully installed tokenizers-0.13.3 transformers-4.30.0
```

```
!pip install -q datasets
```

```
                                        480.6/480.6 kB 14.0 MB/s eta 0:00:00
                                        116.3/116.3 kB 8.2 MB/s eta 0:00:00
                                        179.3/179.3 kB 10.8 MB/s eta 0:00:00
                                        134.8/134.8 kB 9.0 MB/s eta 0:00:00
                                        194.1/194.1 kB 6.5 MB/s eta 0:00:00
ERROR: pip's dependency resolver does not currently take into account all the package
gcsfs 2024.10.0 requires fsspec==2024.10.0, but you have fsspec 2024.9.0 which is inc
sentence-transformers 3.2.1 requires transformers<5.0.0,>=4.41.0, but you have transf
```

Load the image classfication dataset

Load the plant health condition dataset from hugging face.

```
from datasets import load_dataset

ds = load_dataset("timm/plant-pathology-2021")
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarnin
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public mo
  warnings.warn(
```

README.md: 100% 2.19k/2.19k [00:00<00:00, 19.4kB/s]

Resolving data files: 100% 30/30 [00:00<00:00, 38.48it/s]

Resolving data files: 100% 30/30 [00:00<00:00, 7.71it/s]

Downloading data: 100% 30/30 [06:16<00:00, 12.04s/files]

train-00000-of-00030.parquet: 100% 489M/489M [00:12<00:00, 40.9MB/s]

train-00001-of-00030.parquet: 100% 481M/481M [00:11<00:00, 42.6MB/s]

train-00002-of-00030.parquet: 100% 488M/488M [00:11<00:00, 42.5MB/s]

train-00003-of-00030.parquet: 100% 482M/482M [00:11<00:00, 42.8MB/s]

train-00004-of-00030.parquet: 100% 482M/482M [00:11<00:00, 41.8MB/s]

train-00005-of-00030.parquet: 100% 486M/486M [00:11<00:00, 41.9MB/s]

train-00006-of-00030.parquet: 100% 481M/481M [00:11<00:00, 42.4MB/s]

train-00007-of-00030.parquet: 100% 482M/482M [00:11<00:00, 42.0MB/s]

train-00008-of-00030.parquet: 100% 485M/485M [00:11<00:00, 41.8MB/s]

train-00009-of-00030.parquet: 100% 479M/479M [00:17<00:00, 29.7MB/s]

train-00010-of-00030.parquet: 100% 489M/489M [00:11<00:00, 42.2MB/s]

train-00011-of-00030.parquet: 100% 477M/477M [00:11<00:00, 42.1MB/s]

train-00012-of-00030.parquet: 100% 479M/479M [00:11<00:00, 41.4MB/s]

train-00013-of-00030.parquet: 100% 475M/475M [00:11<00:00, 42.2MB/s]

```
00030.parquet:  100%
train-00014-of-                                 482M/482M  [00:11<00:00,  42.0MB/s]
00030.parquet:  100%
train-00015-of-                                 480M/480M  [00:11<00:00,  41.1MB/s]
00030.parquet:  100%
train-00016-of-                                 488M/488M  [00:24<00:00,  18.1MB/s]
00030.parquet:  100%
train-00017-of-                                 484M/484M  [00:11<00:00,  41.6MB/s]
00030.parquet:  100%
train-00018-of-                                 484M/484M  [00:11<00:00,  41.9MB/s]
00030.parquet:  100%
train-00019-of-                                 486M/486M  [00:11<00:00,  42.7MB/s]
00030.parquet:  100%
```

Create smaller dataset for fine-tuning

I create a small image classification dataset by selecting about 1% from the huge plant health dataset, which should contains only about hundreds of images with only one type of health condition each.

```
small_train = ds['train'].train_test_split(test_size=0.01, seed=42)['test']
validation_test_split = ds["validation"].train_test_split(test_size=0.1, seed=42)["test"]
small_validation = validation_test_split['train']
small_test = validation_test_split['test']

# Filter dataset to include only images with a single label
def filter_single_label(example):
    return len(example["label_names"]) == 1  # Only keep examples with one label

small_train = small_train.filter(filter_single_label)
small_validation = small_validation.filter(filter_single_label)
small_test = small_test.filter(filter_single_label)

# Confirm sizes
print(f"Train examples: {len(small_train)}")
print(f"New Validation Size: {len(small_validation)}")
print(f"Test Size: {len(small_test)}")
```

```
Filter: 100%  168/168 [00:29<00:00,  5.65 examples/s]
Filter: 100%  93/93 [00:14<00:00,  6.52 examples/s]
Filter: 100%  94/94 [00:07<00:00, 12.99 examples/s]
Train examples: 149
New Validation Size: 87
Test Size: 90
```

The "labels" column should be converted to integer because it is in the form of "[1]".

```
def parse_label(label):
    return label[0]  # Extract the first element if it's a list

small_train = small_train.map(lambda x: {"labels": parse_label(x["labels"])})
small_validation = small_validation.map(lambda x: {"labels": parse_label(x["labels"])})
small_test = small_test.map(lambda x: {"labels": parse_label(x["labels"])})
```

```
Map: 100%  149/149 [00:00<00:00, 343.37 examples/s]
Map: 100%  87/87 [00:00<00:00, 349.82 examples/s]
Map: 100%  90/90 [00:00<00:00, 283.04 examples/s]
```

Show an example from the training dataset to show the plant leaf image and its information.

```
ex = small_train[50]
ex
```

```
{'image': <PIL.JpegImagePlugin.JpegImageFile image mode=RGB size=4000x2672>,
 'labels': 1,
 'label_names': ['frog_eye_leaf_spot'],
 'image_id': 'aa3a9ac8c8d6bd4a'}
```

```
ex_image = ex['image']
ex_image
```

Make sure the validation and test dataset do not contain health conditions that are in the training set.

```
# Flatten the lists in the column and get unique values
from itertools import chain
train_unique_names = set(chain.from_iterable(small_train['label_names']))
vali_unique_names = set(chain.from_iterable(small_validation['label_names']))
test_unique_names = set(chain.from_iterable(small_test['label_names']))

# Check for differences
valid_diff = vali_unique_names.difference(train_unique_names)
```

```
test_diff = test_unique_names.difference(train_unique_names)

# Print results
print("Labels in validation dataset but not in training dataset:", valid_diff)
print("Labels in test dataset but not in training dataset:", test_diff)
```

```
Labels in validation dataset but not in training dataset: set()
Labels in test dataset but not in training dataset: set()
```

Make use the "labels" number is mapped to the label names as ClassLabel class. This is required for the pre-trained ViT model.

```
from datasets import Dataset, ClassLabel

class_names = list(train_unique_names)
class_label = ClassLabel(names=class_names)
small_train = small_train.cast_column('labels', class_label)
small_validation = small_validation.cast_column('labels', class_label)
small_test = small_test.cast_column('labels', class_label)
```

```
Casting the dataset: 100%  149/149 [00:00<00:00, 384.44 examples/s]
Casting the dataset: 100%  87/87 [00:00<00:00, 117.90 examples/s]
Casting the dataset: 100%  90/90 [00:00<00:00, 364.08 examples/s]
```

Check the train set after all the modification as required.

```
# Show the data structure
print(small_train)
print(small_train[0])
```

```
Dataset({
    features: ['image', 'labels', 'label_names', 'image_id'],
    num_rows: 149
})
{'image': <PIL.JpegImagePlugin.JpegImageFile image mode=RGB size=4000x2672 at 0x7E965
```

Load model and processor

Load the model from the hugging face and set all the necrssary parameters.

```
from transformers import AutoImageProcessor, AutoModelForImageClassification
from torch.optim import AdamW

# Load the image processor and model
processor = AutoImageProcessor.from_pretrained("google/vit-base-patch16-224")
model = AutoModelForImageClassification.from_pretrained(
```

```
    "google/vit-base-patch16-224",
    num_labels=len(class_names), # Adjust the number of labels
    ignore_mismatched_sizes=True
)

# Define the optimizer
optimizer = AdamW(model.parameters(), lr=5e-5)  # Learning rate can be adjusted
```

```
The cache for model files in Transformers v4.22.0 has been updated. Migrating your ol
     0/0 [00:00<?, ?it/s]
/usr/local/lib/python3.10/dist-packages/huggingface_hub/file_download.py:797: FutureW
  warnings.warn(
preprocessor_config.json: 100%                                  160/160 [00:00<00:00, 9.61kB/s]
config.json: 100%                                               69.7k/69.7k [00:00<00:00, 855kB/s]
/usr/local/lib/python3.10/dist-packages/huggingface_hub/file_download.py:797: FutureW
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/huggingface_hub/file_download.py:797: FutureW
  warnings.warn(
model.safetensors: 100%                                         346M/346M [00:01<00:00, 219MB/s]
Some weights of ViTForImageClassification were not initialized from the model checkpc
- classifier.bias: found shape torch.Size([1000]) in the checkpoint and torch.Size([6
- classifier.weight: found shape torch.Size([1000, 768]) in the checkpoint and torch.
You should probably TRAIN this model on a down-stream task to be able to use it for p
```

The preprocessing step prepares the raw image data for use with the ViT model. This ensures the images are:

- Resized to match the model's expected input dimensions.
- Normalized to match the model's training conditions.
- Converted into tensors for compatibility with the model.

```
# Preprocess function
def preprocess(example):
    # Ensure pixel_values are tensors
    example["pixel_values"] = processor(example["image"], return_tensors="pt")["pixel_val
    return example

# Apply preprocessing to train and validation sets
train_set = small_train.map(preprocess, batched=False)
val_set = small_validation.map(preprocess, batched=False)
test_set = small_test.map(preprocess, batched=False)
```

```
Map: 100%                                                       149/149 [00:58<00:00,  4.26 examples/s]
Map: 100%                                                       87/87 [00:27<00:00,  5.62 examples/s]
Map: 100%                                                       90/90 [00:27<00:00,  5.95 examples/s]
```

Custom Batching: The dataset may not directly provide tensors in the required shape, so collate_fn ensures consistent formatting.

Efficiency: DataLoader handles data loading in parallel, which is faster than manually iterating over the dataset.

Reproducibility: Ensures the same data preprocessing logic is applied consistently across all data splits (training, validation, testing).

```
import torch
from torch.utils.data import DataLoader

# Create PyTorch DataLoader
def collate_fn(batch):
    pixel_values = torch.stack([torch.tensor(x["pixel_values"]) for x in batch])
    labels = torch.tensor([x["labels"] for x in batch])
    return {"pixel_values": pixel_values, "labels": labels}

train_loader = DataLoader(train_set, batch_size=8, shuffle=True, collate_fn=collate_fn)
val_loader = DataLoader(val_set, batch_size=8, shuffle=False, collate_fn=collate_fn)
test_loader = DataLoader(test_set, batch_size=8, shuffle=False, collate_fn=collate_fn)
```

Train the model

Run train loop to get train, validation, and test loss for each epoch and generate information for each epoch.

```
from tqdm import tqdm
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import torch

# Set device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

# Initialize lists to store metrics for visualization
train_losses = []
val_losses = []
test_losses = []
test_accuracies = []

# Training loop with integrated visualization
num_epochs = 10
for epoch in range(num_epochs):
    print(f"Epoch {epoch + 1}/{num_epochs}")

    # Training phase
    model.train()
    total_train_loss = 0
```

```
    for batch in tqdm(train_loader, desc="Training"):
        optimizer.zero_grad()
        pixel_values = batch["pixel_values"].to(device)
        labels = batch["labels"].to(device)

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

        total_train_loss += loss.item()

    avg_train_loss = total_train_loss / len(train_loader)
    train_losses.append(avg_train_loss)

    # Validation phase
    model.eval()
    total_val_loss = 0
    with torch.no_grad():
        for batch in tqdm(val_loader, desc="Validation"):
            pixel_values = batch["pixel_values"].to(device)
            labels = batch["labels"].to(device)

            outputs = model(pixel_values=pixel_values, labels=labels)
            loss = outputs.loss

            total_val_loss += loss.item()

    avg_val_loss = total_val_loss / len(val_loader)
    val_losses.append(avg_val_loss)

    # Test phase
    total_test_loss = 0
    correct = 0
    total = 0
    true_labels = []
    predicted_labels = []
    with torch.no_grad():
        for batch in tqdm(test_loader, desc="Testing"):
            pixel_values = batch["pixel_values"].to(device)
            labels = batch["labels"].to(device)

            outputs = model(pixel_values=pixel_values, labels=labels)
            loss = outputs.loss
            total_test_loss += loss.item()

            logits = outputs.logits
            predictions = torch.argmax(logits, dim=-1)

            correct += (predictions == labels).sum().item()
            total += labels.size(0)

            true_labels.extend(labels.cpu().numpy())
            predicted_labels.extend(predictions.cpu().numpy())
```

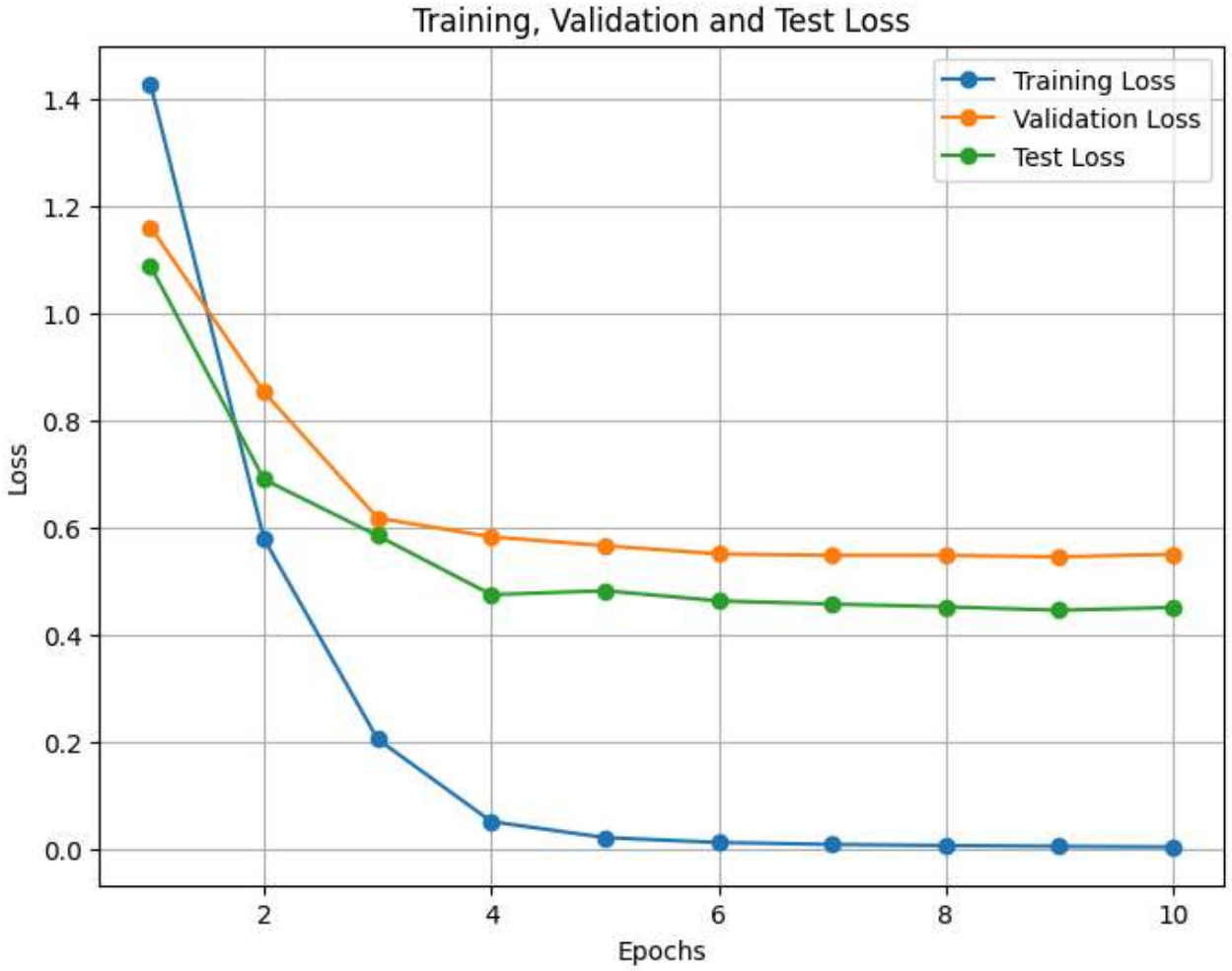
```
    avg_test_loss = total_test_loss / len(test_loader)
    test_accuracy = correct / total
    test_losses.append(avg_test_loss)
    test_accuracies.append(test_accuracy)

    # Log epoch results
    print(f"Epoch {epoch + 1}/{num_epochs}")
    print(f"Train Loss: {avg_train_loss:.4f}")
    print(f"Validation Loss: {avg_val_loss:.4f}")
    print(f"Test Loss: {avg_test_loss:.4f}, Test Accuracy: {test_accuracy:.4f}")
```

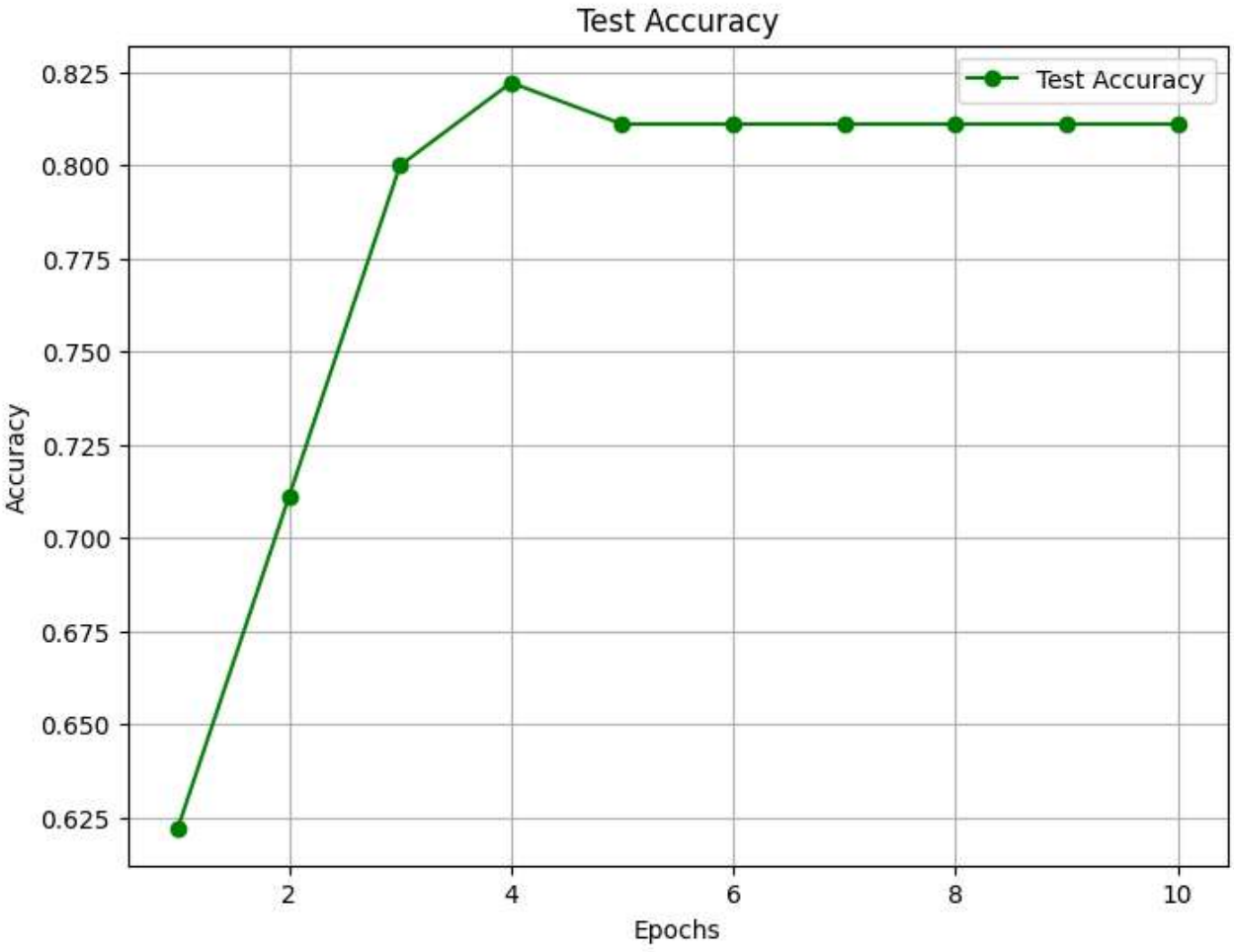
```
Epoch 1/10
Training: 100%|██████████| 19/19 [05:40<00:00, 17.92s/it]
Validation: 100%|██████████| 11/11 [01:16<00:00,  6.98s/it]
Testing: 100%|██████████| 12/12 [01:17<00:00,  6.44s/it]
Epoch 1/10
Train Loss: 1.4292
Validation Loss: 1.1620
Test Loss: 1.0892, Test Accuracy: 0.6222
Epoch 2/10
Training: 100%|██████████| 19/19 [05:56<00:00, 18.78s/it]
Validation: 100%|██████████| 11/11 [01:18<00:00,  7.10s/it]
Testing: 100%|██████████| 12/12 [01:20<00:00,  6.73s/it]
Epoch 2/10
Train Loss: 0.5795
Validation Loss: 0.8538
Test Loss: 0.6910, Test Accuracy: 0.7111
Epoch 3/10
Training: 100%|██████████| 19/19 [05:48<00:00, 18.37s/it]
Validation: 100%|██████████| 11/11 [01:18<00:00,  7.13s/it]
Testing: 100%|██████████| 12/12 [01:19<00:00,  6.58s/it]
Epoch 3/10
Train Loss: 0.2063
Validation Loss: 0.6189
Test Loss: 0.5860, Test Accuracy: 0.8000
Epoch 4/10
Training: 100%|██████████| 19/19 [05:46<00:00, 18.24s/it]
Validation: 100%|██████████| 11/11 [01:16<00:00,  6.91s/it]
Testing: 100%|██████████| 12/12 [01:22<00:00,  6.91s/it]
Epoch 4/10
Train Loss: 0.0526
Validation Loss: 0.5840
Test Loss: 0.4761, Test Accuracy: 0.8222
Epoch 5/10
Training: 100%|██████████| 19/19 [05:42<00:00, 18.01s/it]
Validation: 100%|██████████| 11/11 [01:17<00:00,  7.06s/it]
Testing: 100%|██████████| 12/12 [01:18<00:00,  6.58s/it]
Epoch 5/10
Train Loss: 0.0227
Validation Loss: 0.5673
Test Loss: 0.4834, Test Accuracy: 0.8111
Epoch 6/10
Training: 100%|██████████| 19/19 [05:42<00:00, 18.03s/it]
Validation: 100%|██████████| 11/11 [01:15<00:00,  6.87s/it]
Testing: 100%|██████████| 12/12 [01:18<00:00,  6.52s/it]
Epoch 6/10
Train Loss: 0.0136
Validation Loss: 0.5522
Test Loss: 0.4645, Test Accuracy: 0.8111
```

```
Epoch 7/10
Training: 100%|██████████| 19/19 [05:41<00:00, 17.96s/it]
Validation: 100%|██████████| 11/11 [01:15<00:00,  6.86s/it]
Testing: 100%|██████████| 12/12 [01:19<00:00,  6.62s/it]
Epoch 7/10
Train Loss: 0.0100
Validation Loss: 0.5493
Test Loss: 0.4586, Test Accuracy: 0.8111
Epoch 8/10
Training: 100%|██████████| 19/19 [05:37<00:00, 17.75s/it]
```

Visualize the loss trend and show the confusion matrix.

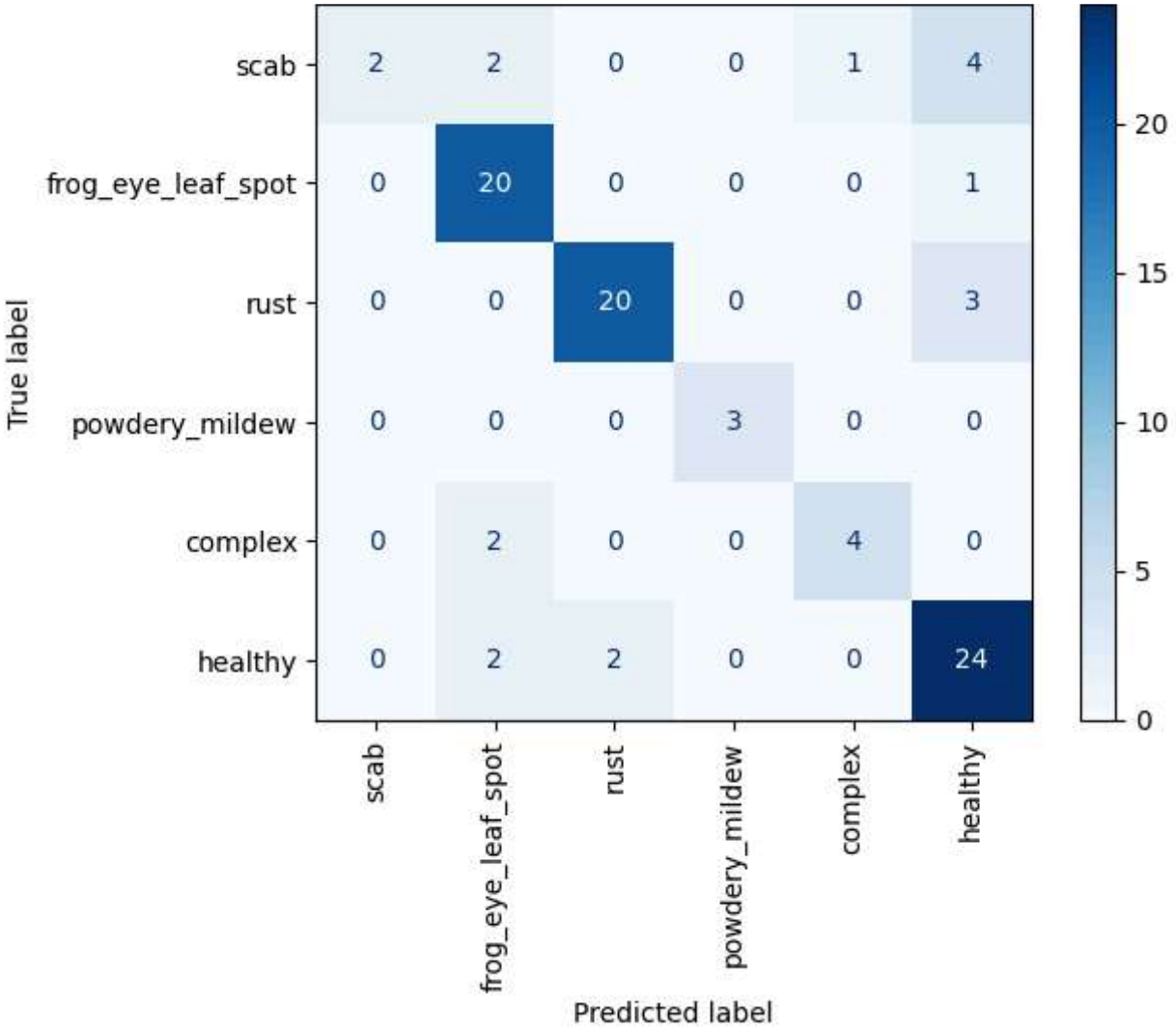
```
# Visualization
epochs = range(1, num_epochs + 1)

# Plot training and validation loss
plt.figure(figsize=(8, 6))
plt.plot(epochs, train_losses, label="Training Loss", marker="o")
plt.plot(epochs, val_losses, label="Validation Loss", marker="o")
plt.plot(epochs, test_losses, label="Test Loss", marker="o")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Training, Validation and Test Loss")
plt.legend()
plt.grid()
plt.show()

# Plot test accuracy
plt.figure(figsize=(8, 6))
plt.plot(epochs, test_accuracies, label="Test Accuracy", marker="o", color="green")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("Test Accuracy")
plt.legend()
plt.grid()
plt.show()

# Plot confusion matrix for the last epoch
cm = confusion_matrix(true_labels, predicted_labels)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_names)

plt.figure(figsize=(8, 6))
disp.plot(cmap="Blues", values_format="d")
plt.title("Confusion Matrix")
# Rotate x-axis labels
plt.xticks(rotation=90)
plt.show()
```






```
<Figure size 800x600 with 0 Axes>
```

Confusion Matrix


scab
frog_eye_leaf_spot
rust
powdery_mildew
complex
healthy
True label
2 2 0 0 1 4
0 20 0 0 0 1
0 0 20 0 0 3
0 0 0 3 0 0
0 2 0 0 4 0
0 2 2 0 0 24
scab
frog_eye_leaf_spot
rust
powdery_mildew
complex
healthy
Predicted label
20
15
10
5
0

Another way to report the result

- Evaluate Model Performance: Use the test dataset (test_loader) to generate predictions and compare them with true labels.
- Classification Report: Summarize the performance of the classification model using metrics like precision, recall, and F1-score.

```
from sklearn.metrics import classification_report

# Gather all predictions and labels
all_predictions = []
all_labels = []

with torch.no_grad():
    for batch in test_loader:
        pixel_values = batch["pixel_values"].to(device)
        labels = batch["labels"].to(device)

        logits = model(pixel_values=pixel_values).logits
        predictions = torch.argmax(logits, dim=-1)

        all_predictions.extend(predictions.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())

# Generate classification report
print(classification_report(all_labels, all_predictions, target_names=class_names))
```

```
                    precision    recall  f1-score   support

              scab       1.00      0.22      0.36         9
 frog_eye_leaf_spot       0.77      0.95      0.85        21
              rust       0.91      0.87      0.89        23
    powdery_mildew       1.00      1.00      1.00         3
           complex       0.80      0.67      0.73         6
```