Plant Disease Classification of Corn Crop Leaves

MSDS 634 Deep Learning Final Project

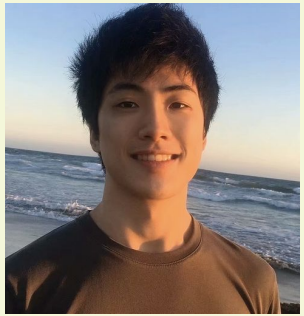
Amadeo Cabanela, Gaurav Goyal, Arios Tong, Jiaxuan Ouyang

Our Team

Amadeo Cabanela

Gaurav Goyal

Arios Tong



Jiaxuan Ouyang

Table of contents

01
Background

Problem + Project Motivation

Problem: The Challenge of Food Insecurity

Global Population

Projected to increase to 10 billion by 2050 (UN)

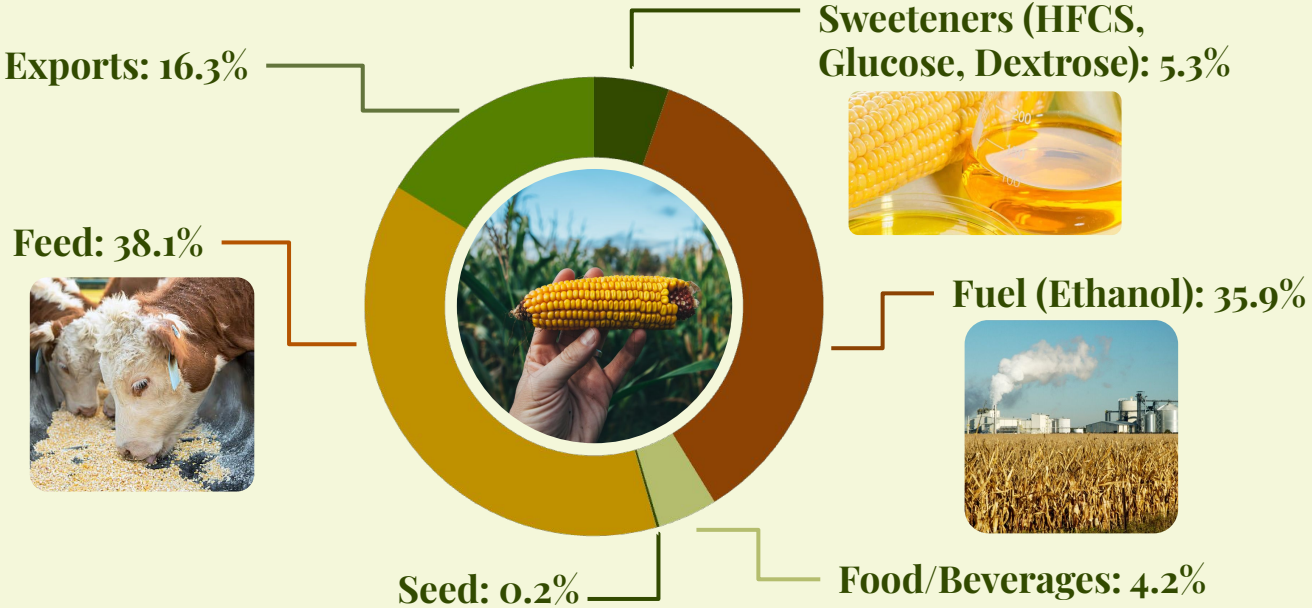
Crop Infectious Diseases

- Global average: **40%** potential yield crop reduction
- Developing world: up to **100%** yield crop reduction

The Importance of Corn

- Corn is the most widely produced feed grain in the United States [USDA]
- In 2021, American farmers harvested about 15 billion bushels of corn grain

Corn Use in the United States (2020-2021)

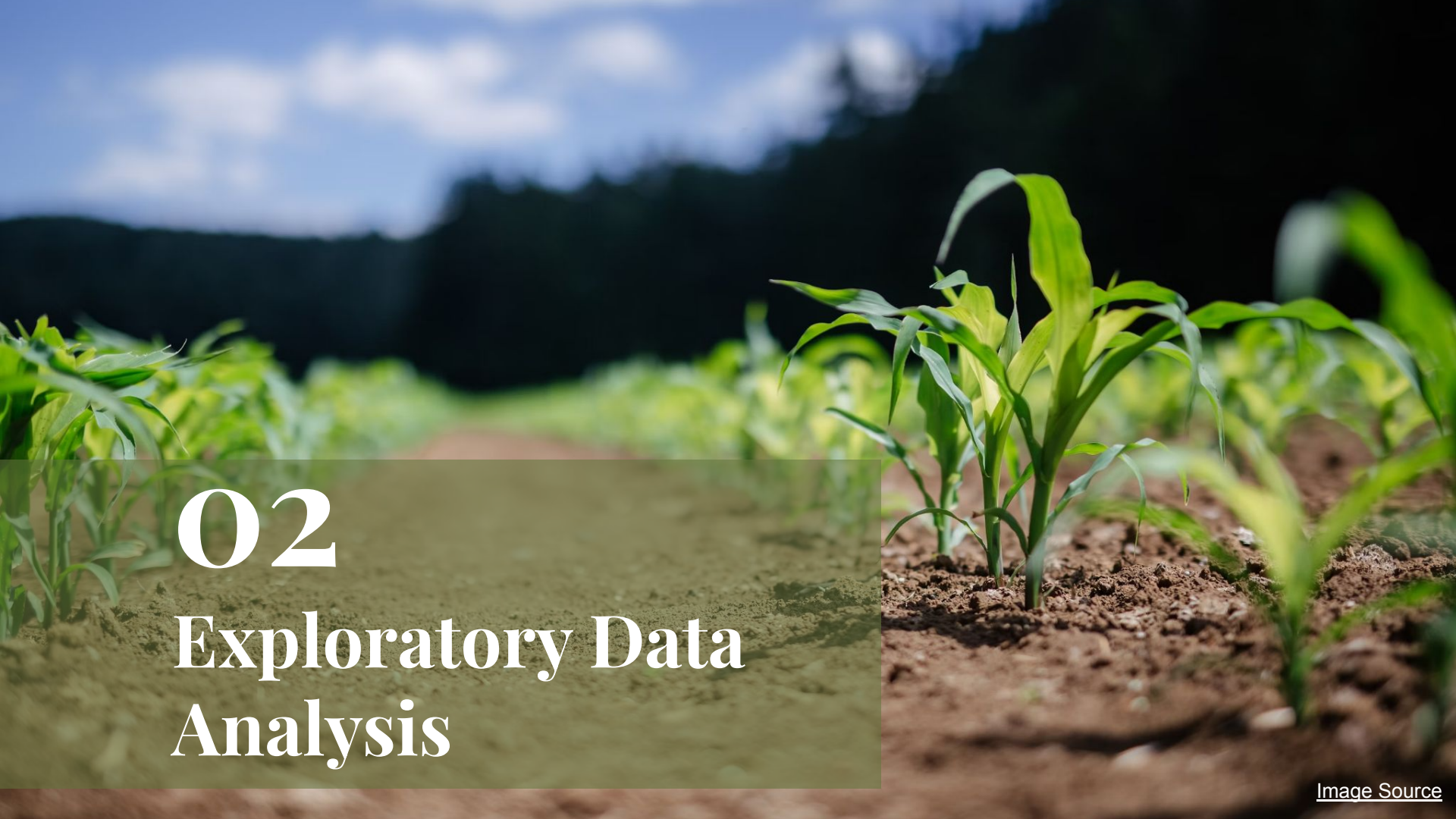


Project Solution

Potential solution: Leverage computer vision and deep learning to programmatically identify crop diseases and address them more quickly and accurately.

Task: Multiclass image classification of corn crop leaves

Dataset: PlantVillage dataset



02

Exploratory Data Analysis

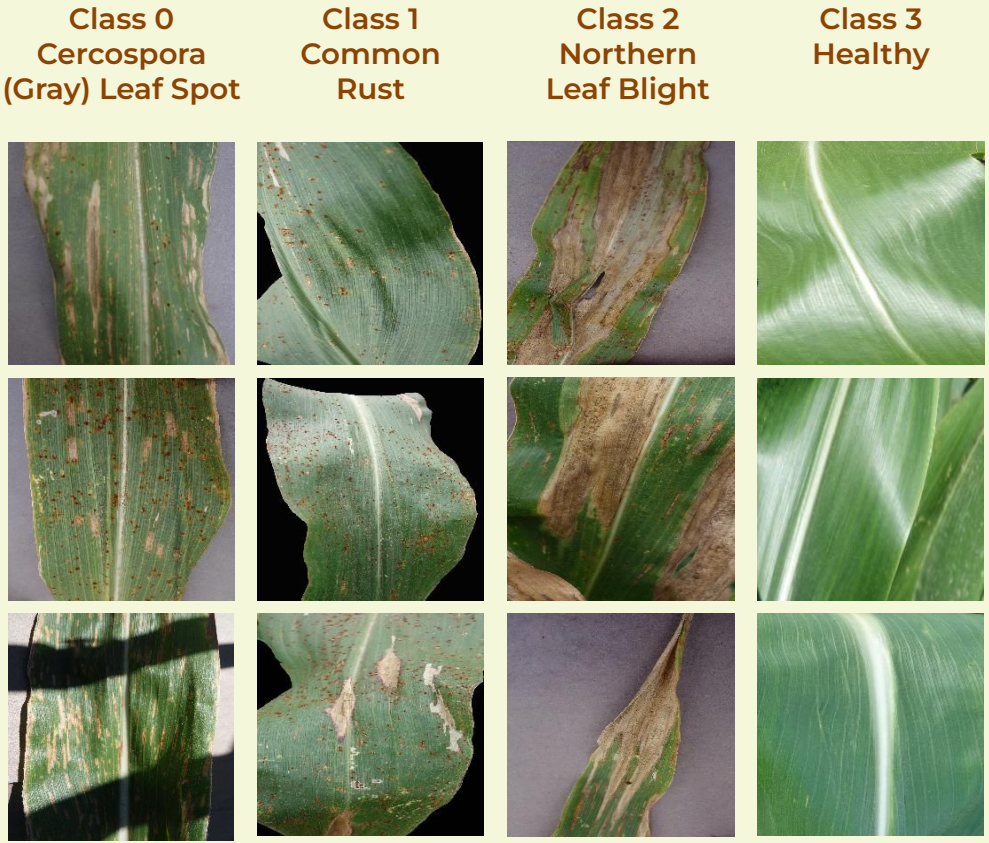
Image Source

Corn PlantVillage Dataset

1. Corn Subset of PlantVillage dataset
   a. Available through Kaggle

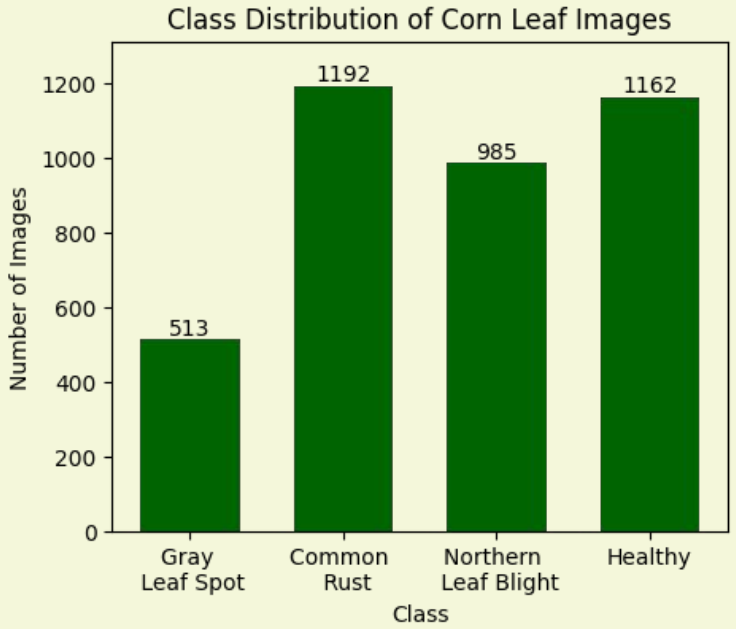
2. Data collection process
   a. Single leaf images
   b. Taken by technicians during field trials of crops infected with diseases
   c. Gray/black background or outdoors
   d. Various lighting conditions

3. Potential challenges:
   a. Shadows
   b. Distinguishing similar looking green leaves



EDA Insights

1. **Imbalanced dataset**
   a. Gray leaf Spot: 513
   b. Common Rust: 1192
   c. Northern Leaf Blight: 985
   d. Healthy: 1162
   e. Total: 3852
2. **Sizes:**
   a. Average width: 256 pixels
   b. Average height: 256 pixels
3. **Format:** .JPG
4. **Average RGB color channel values for train set:**
   a. Red: 0.43613595
   b. Green: 0.4974372
   c. Blue: 0.3781651
5. **Standard Deviation of color channels for train set:**
   a. Red: 0.21189487
   b. Green: 0.22010513
   c. Blue: 0.21154968



03
Data Preprocessing

Image Source

Preprocessing Steps

Step 1: Normalized the images using train mean and standard deviation of RGB color channels

```
# Define the transformations
transform = transforms.Compose([
    transforms.Resize((256, 256)),  # Resize images to 256x256
    transforms.ToTensor(),  # Convert images to PyTorch tensors
    transforms.Normalize(mean=[0.43613595, 0.4974372, 0.3781651],
                         std=[0.21189487, 0.22010513, 0.21154968]),  # Normalize using mean and std
])
```

Step 2: Saved preprocessed 256x256 images to numpy files

- Significantly improves memory efficiency during model training
- Leverages vectorization from numpy

Step 3: Some models expected 224x224 inputs

- Resized images from 256 to 224 using linear interpolation from OpenCV to retain information
- Saved preprocessed 224x224 image arrays to numpy files (good memory efficiency)

04
Model Implementation

Image Source

Baseline Model

Shallow Neural Network

- 1 hidden layer with 128 neurons

```
# Define the shallow neural network model
class ShallowNN(nn.Module):
    def __init__(self):
        super(ShallowNN, self).__init__()
        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(3 * 256 * 256, 128)
        self.fc2 = nn.Linear(128, 4)

    def forward(self, x):
        x = self.flatten(x)
        x = self.fc1(x)
        x = nn.ReLU()(x)
        x = self.fc2(x)
        return x

# Instantiate the model
model = ShallowNN()

# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters())
```

CNN

Conventional Neural Network

- 3 hidden layer with 512 neurons

```
class CNNModel(nn.Module):
    def __init__(self, num_classes=4):
        super(CNNModel, self).__init__()
        self.conv1 = nn.Conv2d(3, 48, kernel_size=3, padding=1)
        self.relu1 = nn.ReLU()
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv2 = nn.Conv2d(48, 64, kernel_size=3, padding=1)
        self.relu2 = nn.ReLU()
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
        self.relu3 = nn.ReLU()
        self.pool3 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(128 * 32 * 32, 512)
        self.relu4 = nn.ReLU()
        self.dropout = nn.Dropout(0.2)
        self.fc2 = nn.Linear(512, num_classes)

    def forward(self, x):
        x = self.pool1(self.relu1(self.conv1(x)))
        x = self.pool2(self.relu2(self.conv2(x)))
        x = self.pool3(self.relu3(self.conv3(x)))
        x = self.flatten(x)
        x = self.dropout(self.relu4(self.fc1(x)))
        x = self.fc2(x)
        return x
```

ResNet

Residual Network

- Pre-trained 50 layer deep learning model

```
# Load pre-trained ResNet50
model = models.resnet50(pretrained=True)

# Freeze all layers in the network
for param in model.parameters():
    param.requires_grad = False

# Replace the final fully connected layer
num_features = model.fc.in_features
model.fc = nn.Linear(num_features, 4)  # Adjust the number
of output classes to 4

# Move the model to GPU if available
device = torch.device("cuda" if torch.cuda.is_available()
else "cpu")
model.to(device)

# Define the loss function and optimizer
criterion =
nn.CrossEntropyLoss(weight=torch.tensor(class_weights,
dtype=torch.float))
# Only optimize the parameters of the final layer
optimizer = optim.Adam(model.fc.parameters(), lr=0.001)
```

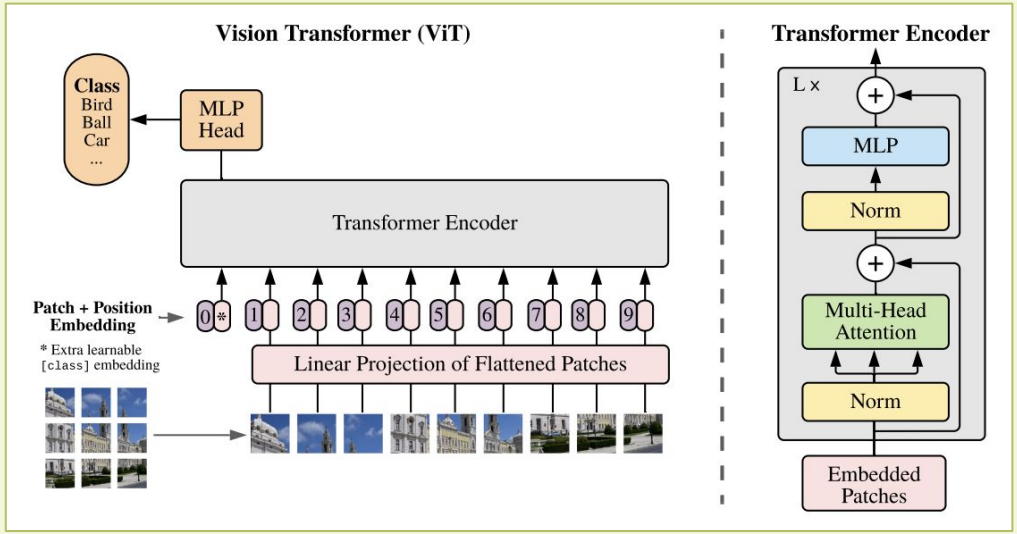
Vision Transformer (ViT)

How it was developed:

- Developed by Google Brain
- Research Paper: “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale” by Dosovitskiy et. al. (2021)
- Trained different variations/sizes on datasets like ImageNet-21k and CIFAR-10/100

Why we chose to use ViT

- Combines concepts we learned in class: NLP, Computer Vision, Transformers
- Long-range and global context – subtle disease-related patterns across different image regions
- Transfer learning – leverage pre-trained visual knowledge for classification task with limited labeled data
- Robustness to Spatial Variations – leaf orientation, background clutter



How it works:

- Applies transformers to sequences of image patches instead of words
  - Patch embeddings – split image into fix-sized patches
  - Positional embeddings, Transformer Encode, Global Average Pooling, classification token

Vision Transformer (ViT)

How we applied the transfer learning and fine-tuning:

- Loaded ViT small model (12 encoder blocks) using torchvision.models:
  `models.vit_b_16(pretrained=True)`
- Replaced the classification head with 4 outputs (instead of original 10)
- Froze all layers using
  - `param.requires_grad = False`
- Unfroze the last 2 encoder layers before the classification head
  - `param.requires_grad = True`
- Verified frozen/unfrozen parameters:
  - Printed `param.requires_grad` for all layers
  - Inspected model summary output

Model summary output:

```
==========================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================
VisionTransformer                        [1, 4]                    768
├─Conv2d: 1-1                            [1, 768, 14, 14]          (590,592)
├─Encoder: 1-2                           [1, 197, 768]             151,296
│    └─Dropout: 2-1                      [1, 197, 768]             --
│    └─Sequential: 2-2                   [1, 197, 768]             --
│    │    └─EncoderBlock: 3-1            [1, 197, 768]             (7,087,872)
│    │    └─EncoderBlock: 3-2            [1, 197, 768]             (7,087,872)
│    │    └─EncoderBlock: 3-3            [1, 197, 768]             (7,087,872)
│    │    └─EncoderBlock: 3-4            [1, 197, 768]             (7,087,872)
│    │    └─EncoderBlock: 3-5            [1, 197, 768]             (7,087,872)
│    │    └─EncoderBlock: 3-6            [1, 197, 768]             (7,087,872)
│    │    └─EncoderBlock: 3-7            [1, 197, 768]             (7,087,872)
│    │    └─EncoderBlock: 3-8            [1, 197, 768]             (7,087,872)
│    │    └─EncoderBlock: 3-9            [1, 197, 768]             (7,087,872)
│    │    └─EncoderBlock: 3-10           [1, 197, 768]             (7,087,872)
│    │    └─EncoderBlock: 3-11           [1, 197, 768]             7,087,872
│    │    └─EncoderBlock: 3-12           [1, 197, 768]             7,087,872
│    └─LayerNorm: 2-3                    [1, 197, 768]             (1,536)
├─Sequential: 1-3                        [1, 4]                    --
│    └─Linear: 2-4                       [1, 4]                    3,076
==========================================================
Total params: 85,801,732
Trainable params: 14,178,820
Non-trainable params: 71,622,912
Total mult-adds (M): 172.47
==========================================================
Input size (MB): 0.60
Forward/backward pass size (MB): 104.09
Params size (MB): 229.21
Estimated Total Size (MB): 333.89
==========================================================
```

05
Methods

Metrics, experiment setup,
model building techniques

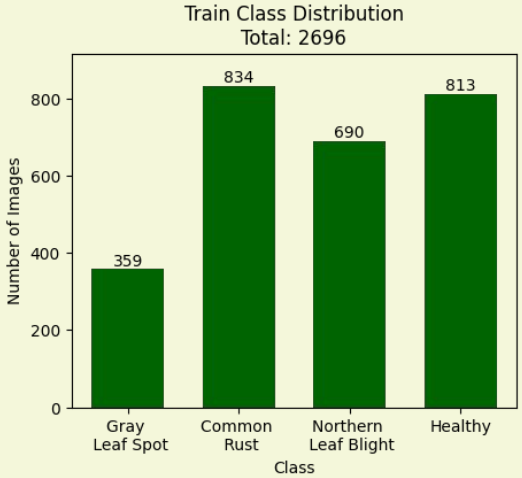
Image Source

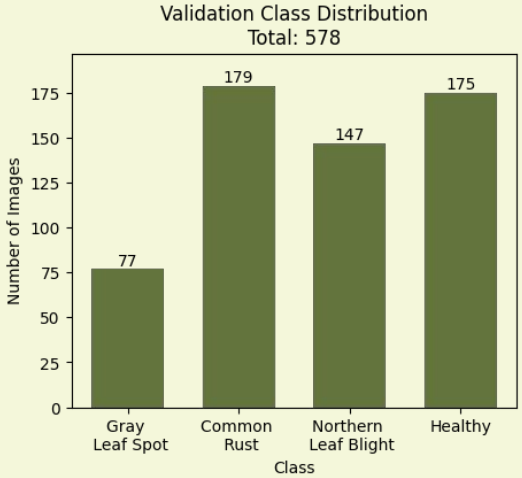
Dataset Splits

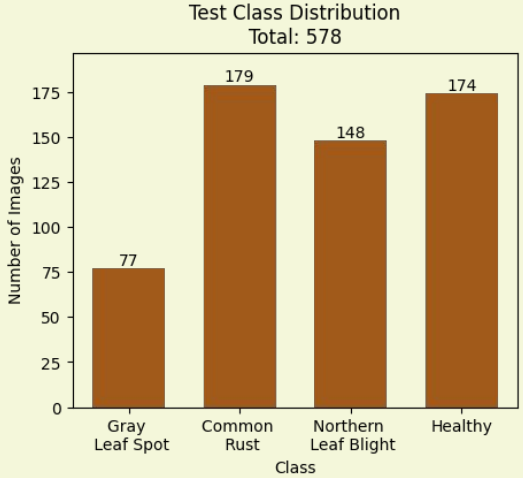
We split our data into train, validation, and test sets using stratified sampling.

Split Ratio:

- 70% Train
- 15% Validation
- 15% Test







Experiment Setup & Design

Train All Models → Select Best Model → Evaluate on Test

Iterative process:

- Train on train set
- Evaluate on val set
- Tune hyperparameters
- Use early stopping
- Use regularization (e.g. dropout, weight_decay)

Primary metric:

- F1 Score

Secondary metrics:

- Precision
- Recall

- Select best model with highest validation F1 score

- Evaluate final model on test set
- Interpret final model performance/analyze results

Other Experiment Methods

Addressing Data Imbalance

- Calculated class weights and passed to loss function `CrossEntropyLoss`
- Instead of image augmentation due to limited computational resources and running time

```
# Calculate class frequencies and total number of samples
unique_classes, class_counts = np.unique(train_labels, return_counts=True)
total_samples = len(train_labels)

# Calculate class weights
class_weights_dict = {cls: 1 - (count / total_samples) for cls, count in zip(unique_classes, class_counts)}
class_weights = list(class_weights_dict.values())

# Define loss function and optimizer
criterion = nn.CrossEntropyLoss(weight=torch.tensor(class_weights, dtype=torch.float))
```

Model Checkpointing

- Saved model weights and results to save time

Transfer Learning

- Loaded some pre-trained models with `torchvision.models` subpackage
- Froze bottom layers to retain pre-trained knowledge
- Unfroze top layers close to the classification head

Frameworks and Resources Used

- Primary Deep Learning Library: `PyTorch`
  - `Torchvision` – package for model architectures and image processing for computer vision
  - `torchvision.models` – subpackage for pre-trained models
  - `torchinfo` – print model architecture summary
- Resizing images: `OpenCV`
- Primary Data Saving and Loading Library: `NumPy`
- Data Storage: Google Drive
- Modeling Environment: Google Colab Pro
  - Used “High-RAM” setting
  - GPU Hardware:
    - A100 (more powerful but costly)
    - T4 TPU (less costly)
  - System RAM: 51 GB
  - GPU RAM: 15 GB

06
Results

Image Source

Experiment Results

Model	Config	Train F1 Score	Val F1 Score	Val Precision	Val Recall
Baseline (Shallow NN)	1 hidden layer with 128 neurons	0.7795	0.7553	0.7375	0.7993
CNN	3 hidden layer with 512 neurons	0.9274	0.9585	0.9511	0.9389
Resnet50	50 layer, model prebuilt	0.9575	0.9364	0.9468	0.9391
Vision Transformer	12 encoder blocks (2 unfrozen)	0.9989	**0.9689**	0.9689	0.9689

Final Model: Vision Transformer

Why we chose this model:

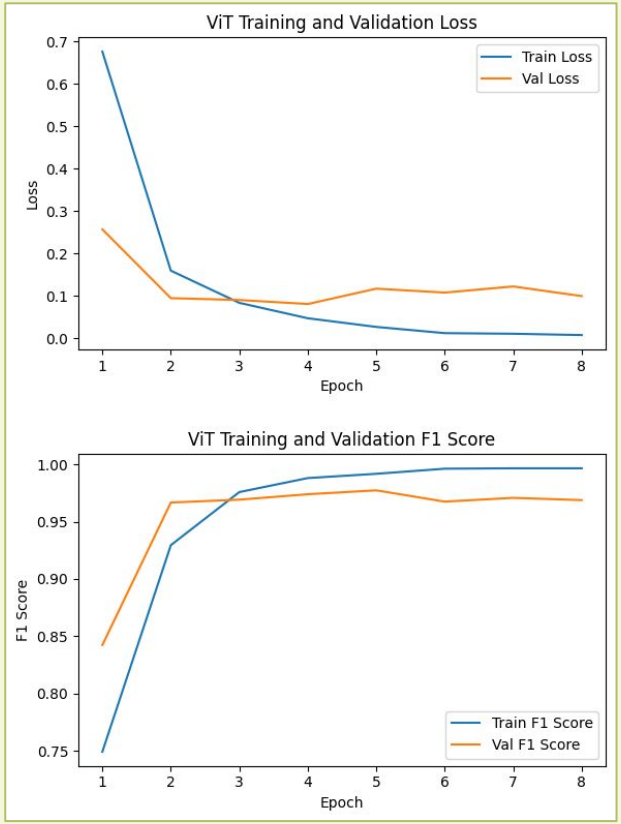
- Highest validation F1 score
- We ran 3 ViT experiments, but this version had the smoothest train/val curves

Training details:

- Trained for 20 epochs, early stopping with 2 epochs patience
- Unfroze last 2 encoder blocks (out of 12)
- Batch size = 256
- Learning rate = 1e-3

Test Results

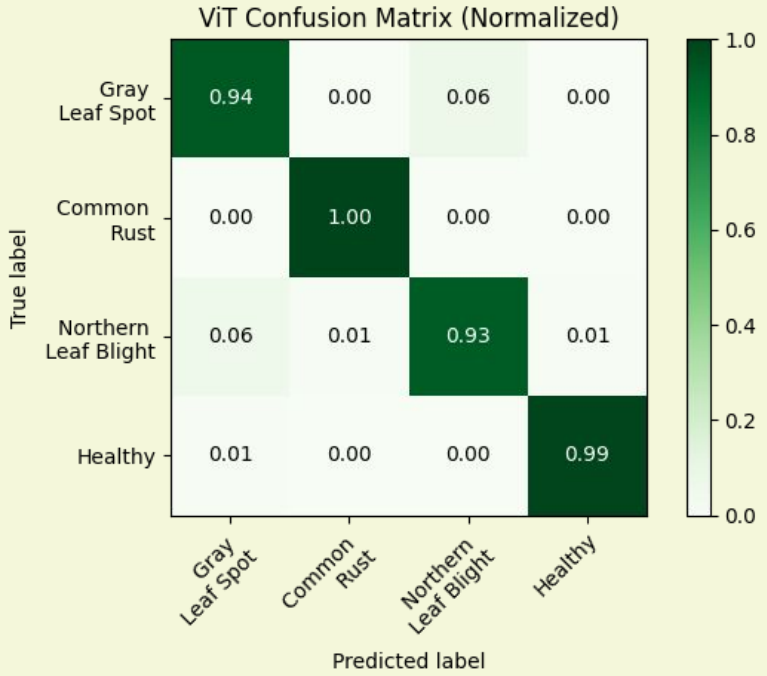
- Test F1 Score: 0.9707
- Test Precision: 0.9713
- Test Recall: 0.9706



Final Model Performance

Classification Report

	precision	recall	f1-score	support
0	0.88	0.94	0.91	77
1	0.99	1.00	1.00	179
2	0.96	0.93	0.94	148
3	0.99	0.99	0.99	174
accuracy			0.97	578
macro avg	0.96	0.96	0.96	578
weighted avg	0.97	0.97	0.97	578



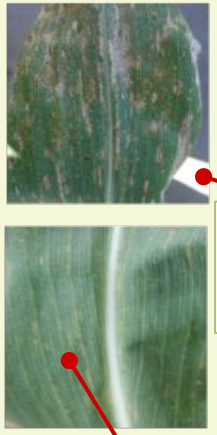
Final Model: Misclassification Analysis

The model misclassified images in the test set below. Here are the True and Predicted labels:

True: Northern Leaf Blight (Class 2)
Predicted: Gray Leaf Spot (Class 0)

Confusion with light brown streaks/patches

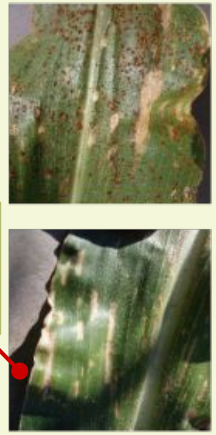
True: Northern Leaf Blight (Class 2)
Predicted: Healthy (Class 3)



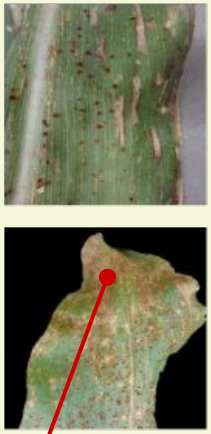
Shadows and highlights maybe confusing model

Looks smooth like a healthy leaf

True: Northern Leaf Blight (Class 2)
Predicted: Common Rust (Class 1)



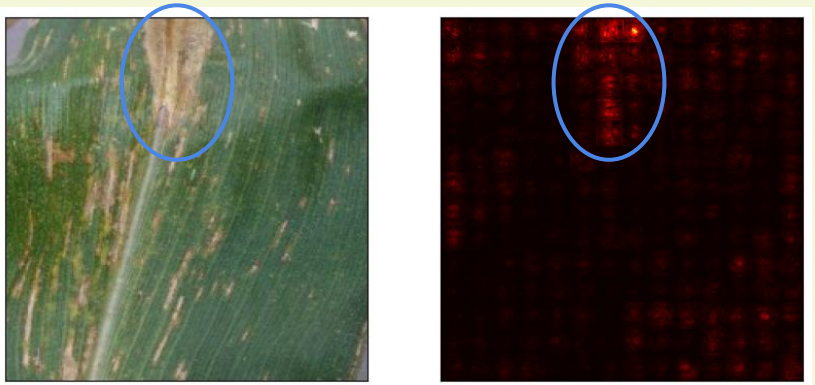
True: Gray Leaf Spot (Class 0)
Predicted: Northern Leaf Blight (Class 2)



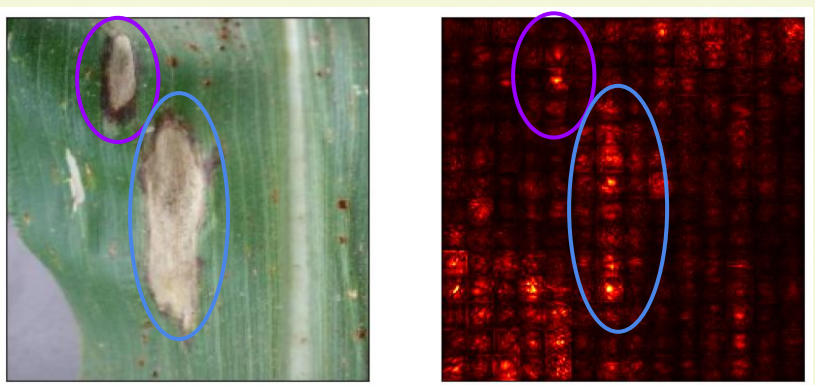
Confusion with light brown streaks/patches

Final Model Saliency Maps

Gray Leaf Spot (Class 0)
Identified light brown patches



Northern Leaf Blight (Class 2)
Identified light brown patches with dark outlines



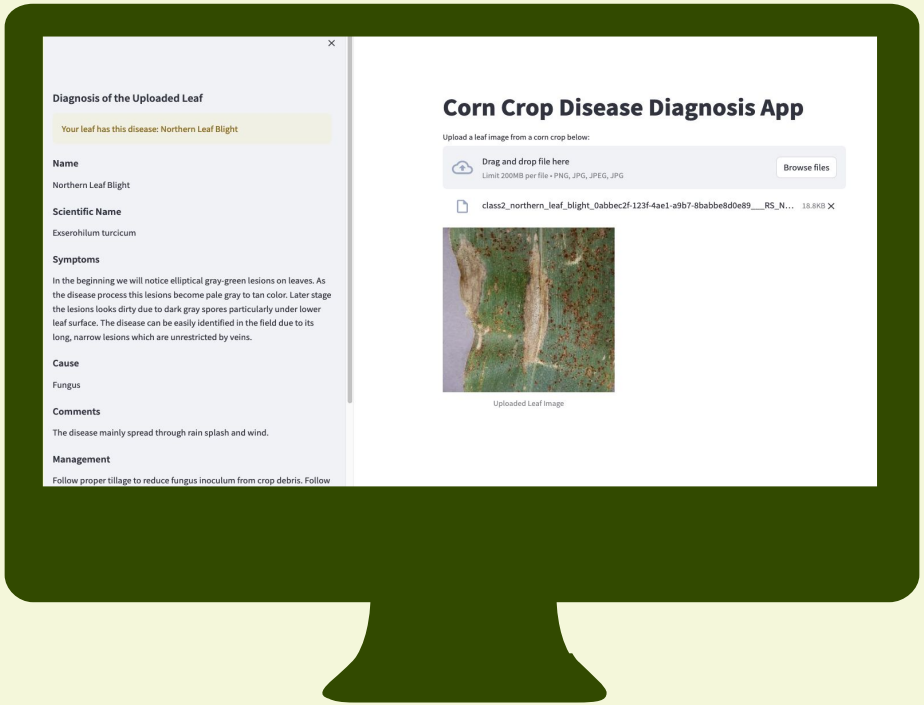
07

Web App Demo

Image Source

Demo

We created a web application to verify our model works and simulate how a crop grower might benefit from this project.



Thanks!