SEMINAR ON : The Project "Detection of Tea Leaf Disease Using Deep Learning"

Jenish Ali Borah

Roll No: PS-211-811-0011
Supervised by: Dr.Hasin A. Ahmed
Department of Computer Science
Gauhati University,Jalukbari,Ghy-14

June 27, 2023

What I'll Discuss Today

Introduction

Introduction

Crop diseases have a significant economic impact on societies, affecting plant function, growth, and overall agricultural activities. Since tea is a widely used beverage and a perennial crop, illnesses that affect its production are common

Tea leaf illnesses must be manually inspected, which is time-consuming, labor-intensive, and expensive.

Automating the detection process through the use of machine learning techniques can reduce the need for manual inspections, save time, and increase the effectiveness of disease detection in the tea sector.

Tea Leaf Diseases

Tea Leaf Diseases

Few common tea leaf diseases :

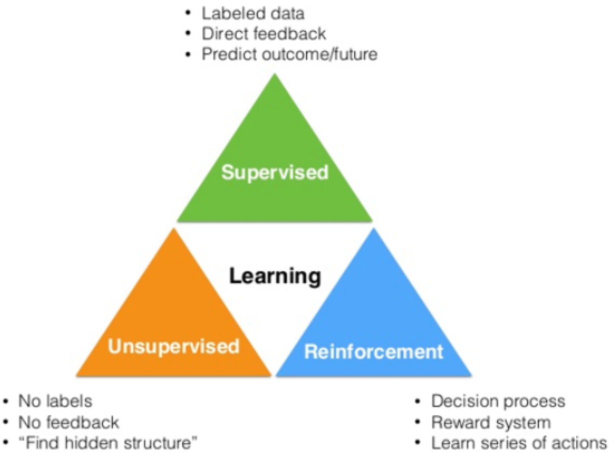
- Gray blightt
- Brown blight
- Copper blight
- Blister blight
- White spot
- Anthracnose
- Bird's eyespot
- Brown rot
- Black rot

Machine Learning

Machine Learning

- It is a branch of artificial intelligence that enables computers to learn from data and make predictions or decisions without being explicitly programmed.
- It involves algorithms and statistical models that allow systems to automatically learn and improve from experience.

Types of Machine Learning



Convolutional Neural Network

Convolutional Neural Network

- CNNs are a type of deep learning algorithm specifically designed for image analysis and pattern recognition.
- They automatically learn and extract meaningful features from images.

CNNs have shown remarkable success in various computer vision tasks, including the detection and classification of diseases in tea leaves.

Convolutional Neural Network

Variations of CNN

There are various variations of CNN, Some of them are:

- VGGNet
- ResNet
- InceptionNet
- NASNet
- EfficientNet

What I'll Discuss Today

Aim of the work

Motivation

- Use deep learning to improve the tea industry's ability to avoid and recognise diseases that afflict tea plants, resulting in better crop management and reduce economic losses.

Aim of the work

- To develop an accurate and efficient system for automated identification of tea leaf diseases using machine learning techniques.
- Investigate the possibilities of transfer learning by utilising pre-trained models such as NASNetMobile,VGG16 and InceptionV3 to increase disease detection capabilities and efficiency.

What I'll Discuss Today

Literature Review

Following are some of the works done on Agricultre activities using Machine Learning:

Study	Paper Name	Approach	Dataset	Accuracy
Hemnath et al.	Deep Learning based leaf disease detection in crops using images for agricultre application	Hyperparameter tuning in DenseNet-121, ResNet-50, VGG-16, Inception V4	Plant Village	99.81% (DenseNet-121)
Narayanan et al.	Banana plant disease classification using hybrid CNN	CNN and SVM	3500 images of banana plants	99%
Jadhav et al.	Identification of plant disease using CNN	AlexNet and GoogleNet	Soybean plant species	98.75% (AlexNet)
Anh et al.	Deep Learning models for multi-leaf diseases	MobileNet	Part of Plant Village	95.58%

What I'll Discuss Today

Dataset description

Data collection

"The tea sickness dataset" contains unhealthy and a class of healthy tea leaves.

Description

The dataset contains tea leaves showing 7 common diseases of tea:

1. Red leaf spot;
2. Algal leaf spot;
3. Bird's eyespot;
4. Gray blight;
5. White spot;
6. Anthracnose;
7. Brown blight.

Each class contains almost 100 images.

Data Preprocessing

Rescalling

The Rescale option is used as 1./255. It gurantess that the pixel values are within appropriate range to train the models

Data Loading and spliting

- The dataset is loaded using "flow_from_directory" function.
- Set Class mode: "Categorical".
- Set the batch size.
- Spliting data into training and validation subset into 80:20.

Data Preprocessing

Data Augmentation

Data Augmentation is applied to generate additional images artificially to incrase the size of the dataset.

Data Augmentation Techniques

- Rotation
- Flipping
- cropping
- zooming etc.

What I'll Discuss Today

Experimental Work

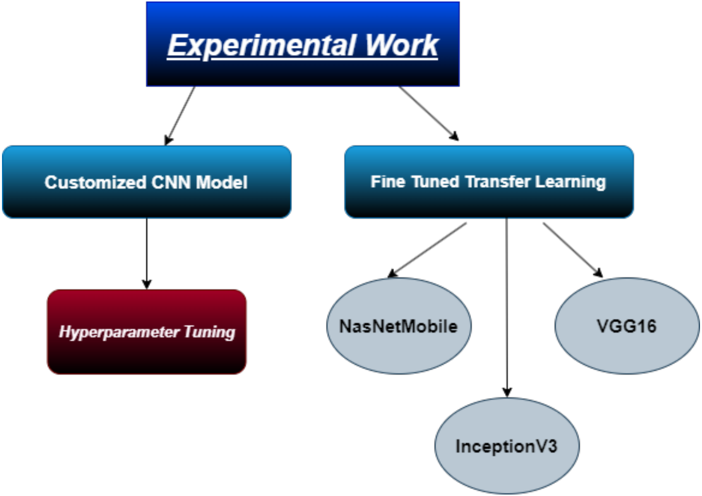


Figure: General Methodology

Propsed CNN Model with Layers View

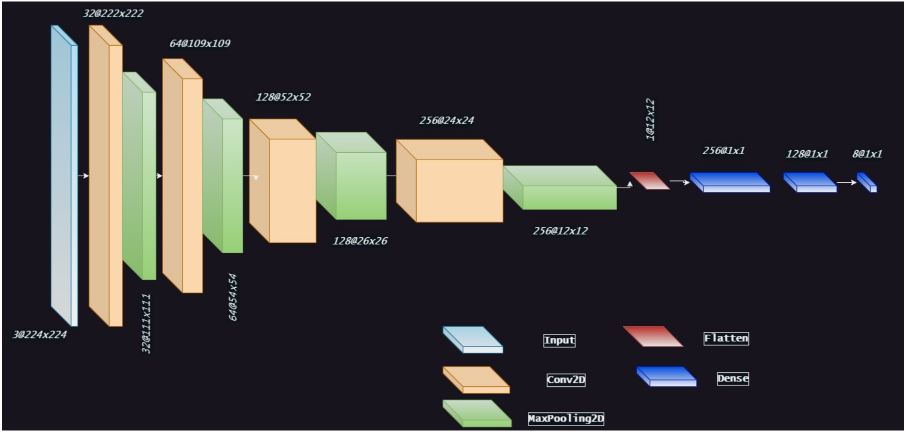


Figure: Convolutional Neural Network (CNN) architecture for tea leaf disease detection.

Layers Description

Description

Applying the model to input photos with the shape (224, 224, 3)
The algorithm used to get the output shapes is:

$$(input\_shape - kernel\_size + 2 * padding)/stride + 1. \tag{1}$$

Layer 1: Conv2D

- Number of filters: 32
- Filter size: (3, 3)
- Activation function: ReLU
- Input shape: (224, 224, 3)
- Output shape: (222, 222, 32)

Layers Description

Layer 2: MaxPooling2D

- Pool size: (2, 2)
- Input shape: (222, 222, 32)
- Output shape: (111, 111, 32)

Layer 3: Conv2D

- Number of filters: 64
- Filter size: (3, 3)
- Activation function: ReLU
- Input shape: (111, 111, 32)
- Output shape: (109, 109, 64)

Layers Description

Layer 4: MaxPooling2D

- Pool size: (2, 2)
- Input shape: (109, 109, 64)
- Output shape: (54, 54, 64)

Layer 5: Conv2D

- Number of filters: 128
- Filter size: (3, 3)
- Activation function: ReLU
- Input shape: (54, 54, 64)
- Output shape: (52, 52, 128)

Layers Description

Layer 6: MaxPooling2D

- Pool size: (2, 2)
- Input shape: (52, 52, 128)
- Output shape: (26, 26, 128)

Layer 7: Conv2D

- Number of filters: 256
- Filter size: (3, 3)
- Activation function: ReLU
- Input shape: (26, 26, 128)
- Output shape: (24, 24, 256)

Layers Description

Layer 8: MaxPooling2D

- Pool size: (2, 2)
- Input shape: (24, 24, 256)
- Output shape: (12, 12, 256)

Layer 9: Flatten

- Input shape: (12, 12, 256)
- Output shape: (36864,)

Layers Description

Layer 10: Dense

- Number of neurons: 256
- Activation function: ReLU
- Input shape: (36864,)
- Output shape: (256,)

Layer 11: Dense

- Number of neurons: 128
- Activation function: ReLU
- Input shape: (256,)
- Output shape: (128,)

Layers Description

Layer 12: Dense(Output Layer)

- Number of neurons: 8
- Activation function: Softmax
- Input shape: (128,)
- Output shape: (8,)

Hyperparameter Tuning

Hyperparameters

- These are specified parameters that influence the behavior and generalization capabilities of a model.
- Hyperparameters include learning rate, batch size, number of layers, filter size, activation functions, and regularization approaches.

Hyperparameter Tuning

- It is crucial to find the optimal combination of hyperparameter values that maximizes the model's performance.
- I have utilized the trial and error method for hyperparameter tuning.

Proposed Model Workflow

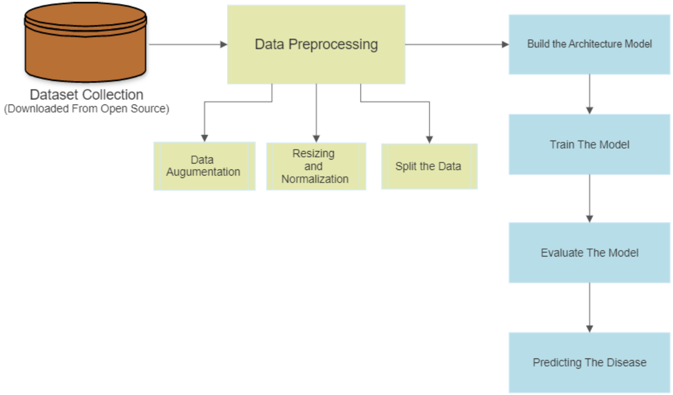


Figure: General Workflow of the Propsed Model.

Transfer Learning

Transfer Learning

- It is a machine learning technique that leverages the knowledge and expertise of pre-trained models to solve new or related tasks.
- For leaf disease detection, transfer learning involves transferring learned features from a pre-trained model to a new model specifically designed for disease detection.

Popular Transfer Learning Models

- VGG-16, ResNet, DenseNet, and Inception are examples of widely used transfer learning models.

Transfer Learning

Why Transfer Learning

- It improved generalization, and the ability to train successful models with less data. It helps overcome the challenges of training deep learning models from scratch by utilizing the learned representations from pre-trained models.
- Transfer learning can significantly enhance learning even when the dataset is small.

Transfer Learning

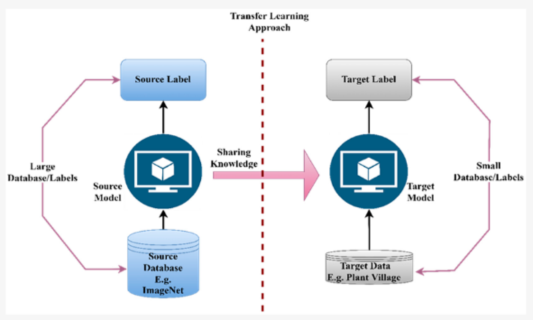


Figure: Basic idea behind Transfer learning.

Transfer Learning: Base model as NASNetMobile

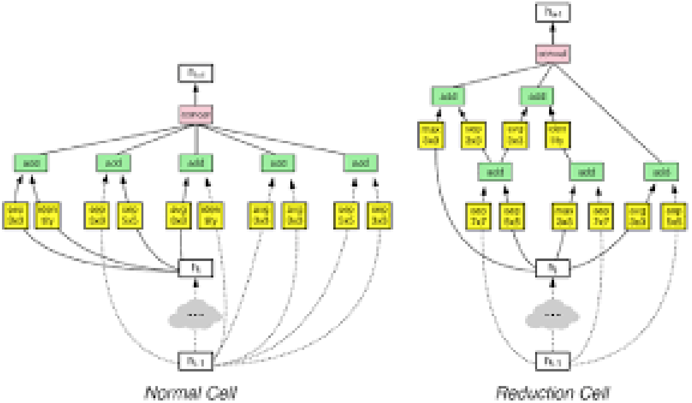


Figure: NASNetMobile Model Architecture

Transfer Learning: Base model as NASNetMobile

```
base_model = tf.keras.applications.NASNetMobile(
    input_shape=(224, 224, 3),
    include_top=False)

# Freeze all but the last few layers of the base model
for layer in base_model.layers[:-10]:
    layer.trainable = False

# Add some layers on top of the pre-trained model
x = base_model.output
x = tf.keras.layers.GlobalAveragePooling2D()(x)
x = tf.keras.layers.Dense(512, activation='relu')(x)
x = tf.keras.layers.Dropout(0.5)(x)
x = tf.keras.layers.Dense(8, activation='softmax')(x)
# Create the final model
model = tf.keras.models.Model(inputs=base_model.input, outputs=x)
```

Figure: Code for Customized NASNetMobile

Transfer Learning: Base model as VGG16

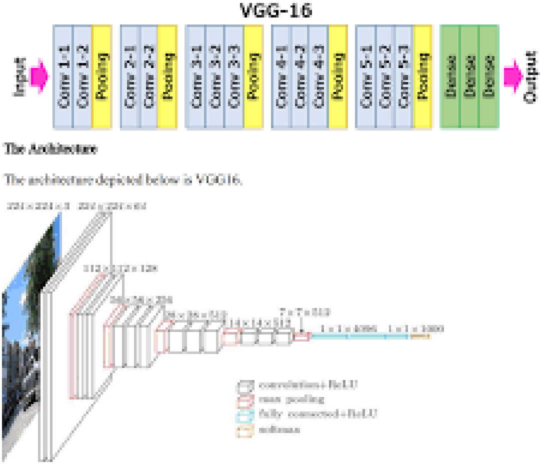


Figure: VGG16 Model Architecture

Transfer Learning: Base model as VGG16

```
# Step 1: Choose a pre-trained model
base_model = VGG16(weights='imagenet', include_top=False,
                   input_shape=(224, 224, 3))

# Step 2: Load the pre-trained model
model = Sequential()
model.add(base_model)

# Step 3: Freeze initial layers
for layer in base_model.layers:
    layer.trainable = False

# Step 4: Add new classification layers
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(8, activation='softmax'))
```

Figure: Code for Customized VGG16

Transfer Learning: Base model as Inception V3

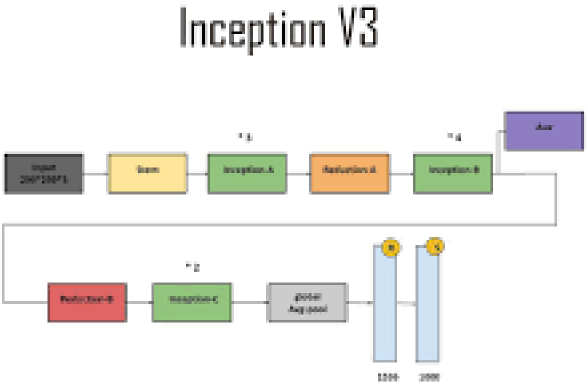


Figure: Inception V3 Model Architecture

Transfer Learning: Base model as Inception V3

```
# Load the InceptionV3 base model without the top layers
base_model = InceptionV3(weights='imagenet', include_top=False,
          input_shape=(224, 224, 3))

# Freeze the layers of the base model
for layer in base_model.layers:
    layer.trainable = False

# Add custom classification layers on top of the base model
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(256, activation='relu')(x)
predictions = Dense(8, activation='softmax')(x)
# Create the model
model = Model(inputs=base_model.input, outputs=predictions)
```

Figure: Code for Customized Inception V3

What I'll Discuss Today

Compile and Train the model

Compile code snippet

```
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

Train code snippet

```
# Train the model
history = model.fit(
    train_data,
    epochs=50,
    batch_size=batch_size,
    validation_data=val_data
)
```

What I'll Discuss Today

Graphical User Interface

Tools used

The GUI was developed using the following tools and libraries:

- Flask
- HTML
- CSS
- JavaScirpt
- Matplotlib
- NumPy
- OpenCv
- TensorFlow

Graphical User Interface Snapshots

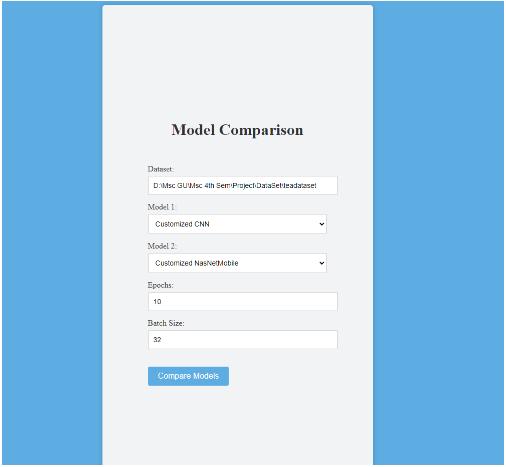


Figure: Home Page View.

Graphical User Interface Snapshots

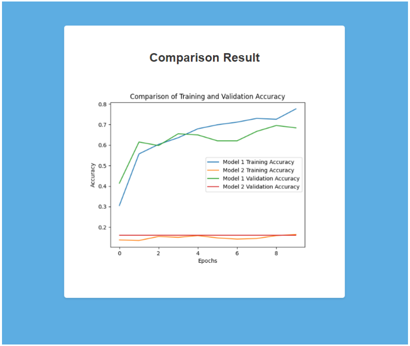


Figure: Display of comparison based on selected attribute

Graphical User Interface Snapshots

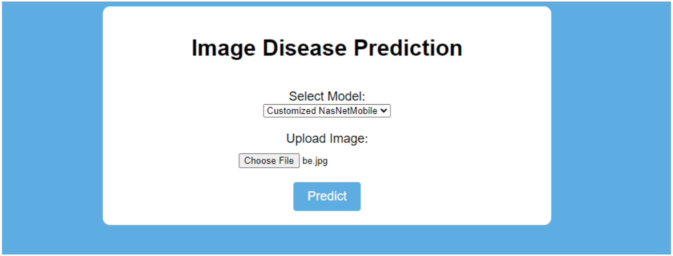


Figure: View of Selecting a model and Upload a image for prediction

Graphical User Interface Snapshots

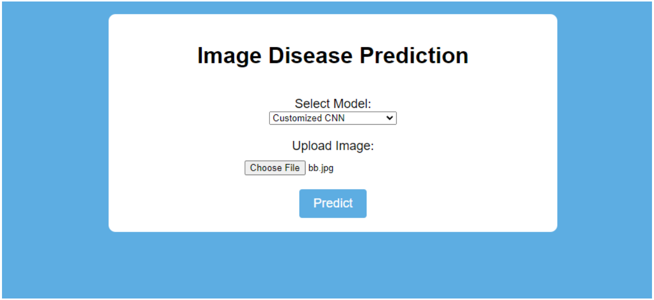


Figure: View of Selecting a model and Upload a image for prediction

Graphical User Interface Snapshots

Figure: Display of the predicted result

Graphical User Interface Snapshots

Figure: Display of the predicted result

What I'll Discuss Today

Training and Validation Loss of customized CNN

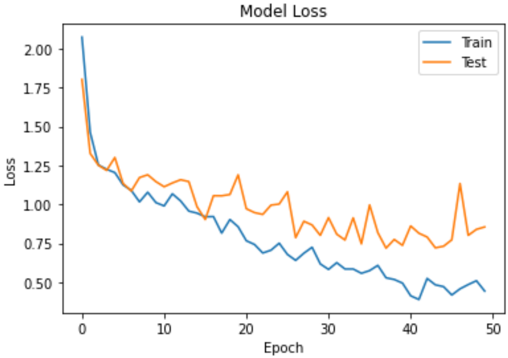


Figure: Model loss graph against customized CNN model

Training and Validation Accuracy of customized CNN

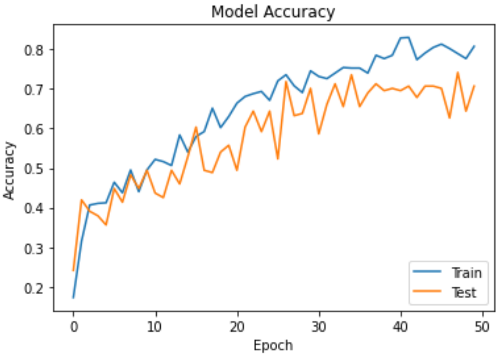


Figure: Model accuracy graph against customized CNN model

Training and Validation loss of customized NASNetMobile model

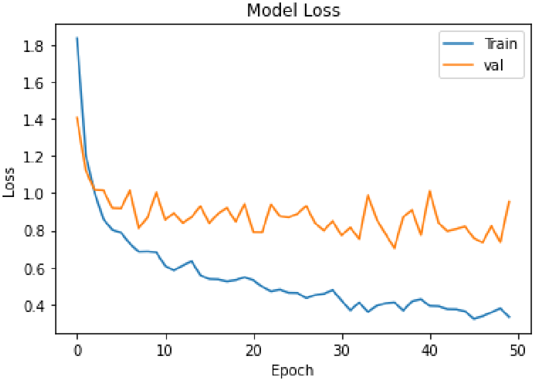


Figure: Model loss graph against customized NASNetMobile

Training and Validation Accuracy of customized NASNetMobiole model

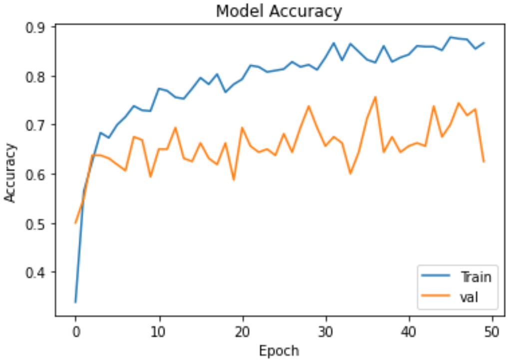


Figure: Model accuracy graph against customized NASNetMobile

Training and Validation loss of customized VGG16 model

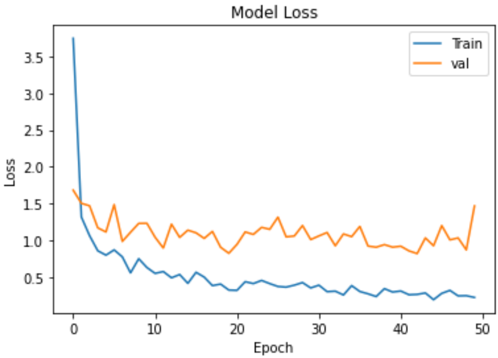


Figure: Model loss graph against customized VGG16

Training and Validation Accuracy of customized VGG16 model

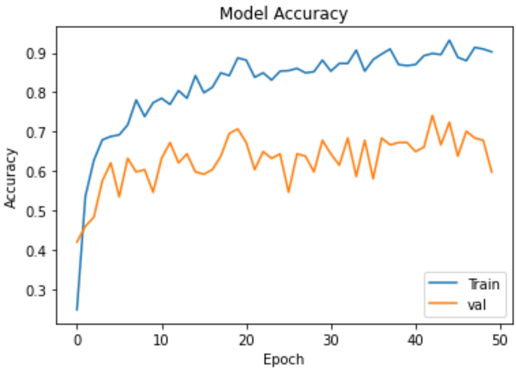


Figure: Model accuracy graph against customized VGG16

Training and Validation loss of customized Inception V3 model

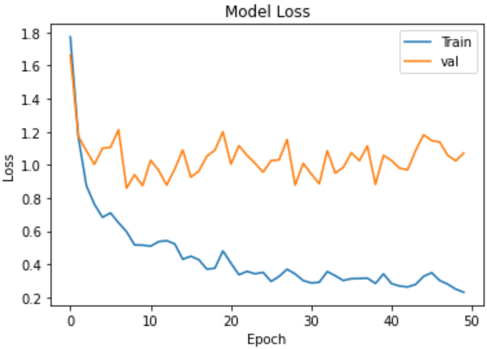


Figure: Model loss graph against customized Inception V3

Training and Validation Accuracy of customized Inception V3 model

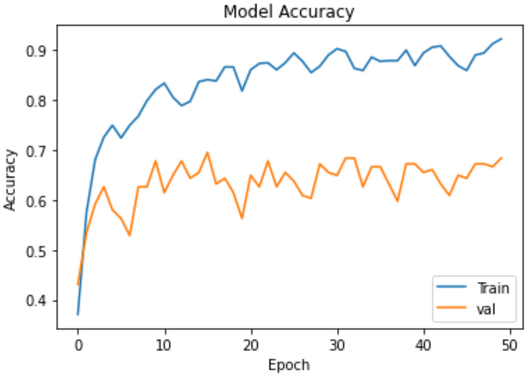


Figure: Model accuracy graph against customized Inception V3

Predicted and true labels on Test Data

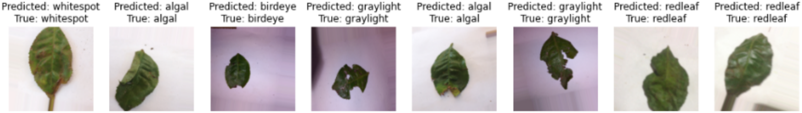


Figure: Predicted and True class of Images by Customized CNN model

Predicted and true labels on Test Data

Figure: Predicted and True class of Images by Customized NasNetMobile model

Predicted and true labels on Test Data

Figure: Predicted and True class of Images by Customized VGG16 model

Predicted and true labels on Test Data

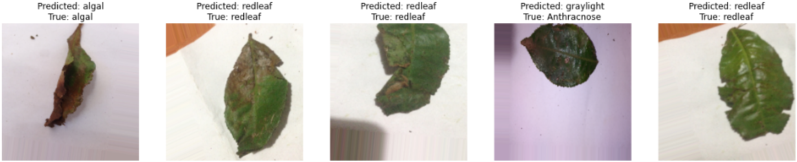


Figure: Predicted and True class of Images by Customized Inception V3 model

Models Comparsion Result

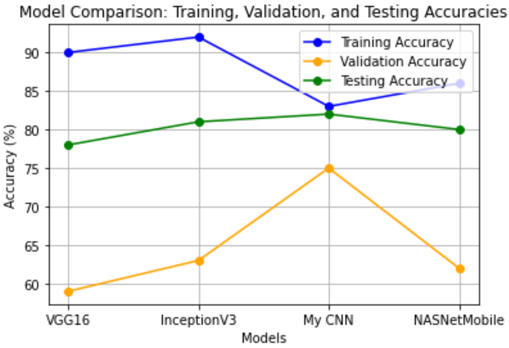


Figure: Comparison Graph of the models

What I'll Discuss Today

Conclusion

Conclusion

- In a performance evaluation of various pre-trained architectures and a CNN-based model, the CNN-based model outperformed NasNetMobile, VGG-16, and Inception V3.
- The customized CNN-based model was found to be easier to train due to its lower number of trainable parameters and lower computing complexity compared to the other pre-trained architectures.

Conclusion

Conclusion

- The customized CNN-based model is considered better suited for tea leaf disease detection due to its superior performance and ease of training.
- The proposed model achieved an 83% training accuracy and an 82% testing accuracy, indicating its effectiveness in detecting tea leaf diseases.

Future Work

Future Work- Experimental

- I will work on different pre-trained models with layer customization to get better accuracy.
- I will try differnt methods in my CNN model to get best combined hyparperemeter.
- Mainly, I Will try to get a better Dataset to work on.

I'm almost done with a thorough research report, and I'll communicate with a publication soon.

thank you