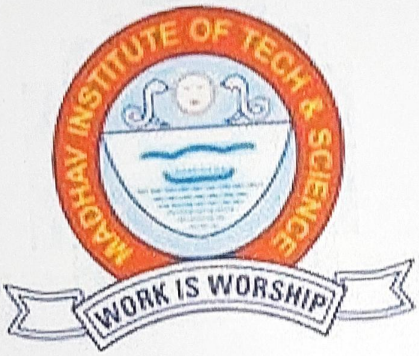
MADHAV INSTITUTE OF TECHNOLOGY & SCIENCE GWALIOR

(A Govt. Aided UGC Autonomous Institute Affiliated to RGPV, Bhopal)

NAAC Accredited with A++ Grade



Project Report

on

Plant Seed Classification

Submitted By:

Dhruv Agrawal(0901AI211024)

Gauri Tripathi(0901AI211030)

Faculty Mentor:

Prof. Gaurisha Sisodia

CENTRE FOR ARTIFICIAL INTELLIGENCE

MADHAV INSTITUTE OF TECHNOLOGY & SCIENCE

GWALIOR - 474005 (MP) est. 1957

JULY-DEC. 2023

MADHAV INSTITUTE OF TECHNOLOGY & SCIENCE GWALIOR

(A Govt. Aided UGC Autonomous Institute Affiliated to RGPV, Bhopal)

NAAC Accredited with A++ Grade

CERTIFICATE

This is certified that **Dhruv Agrawal** (0901AI211024) And **Gauri Tripathi (0901AI211030)** has submitted the project report titled **Plant seed classification** under the mentorship of **Prof. Gaurisha Sisodia** in partial fulfilment of the requirement for the award of degree of Bachelor of Technology in **Artificial Inteligence and Robotics** from Madhav Institute of Technology and Science, Gwalior.

Prof Gaurisha Sisodia

Faculty Mentor
Centre for Artificial Intelligence

23/11/25

Dr. R. R. Singh

Coordinator
Centre for Artificial Intelligence

MADHAV INSTITUTE OF TECHNOLOGY & SCIENCE GWALIOR

(A Govt. Aided UGC Autonomous Institute Affiliated to RGPV, Bhopal)

NAAC Accredited with A++ Grade

DECLARATION

I hereby declare that the work being presented in this project report, for the partial fulfilment of requirement for the award of the degree of Bachelor of Technology in **Artificial Intelligence and Robotics** at Madhav Institute of Technology & Science, Gwalior is an authenticated and original record of my work under the mentorship of **Prof. Gaurisha Sisodia** , Centre of Artificial Intelligence

I declare that I have not submitted the matter embodied in this report for the award of any degree or diploma anywhere else.

Dhruv Agrawal
0901AI211024
Gauri Tripathi
0901AI211030
Centre for Artificial Intelligence

MADHAV INSTITUTE OF TECHNOLOGY & SCIENCE GWALIOR

(A Govt. Aided UGC Autonomous Institute Affiliated to RGPV, Bhopal)

NAAC Accredited with A++ Grade

ACKNOWLEDGEMENT

The full semester project has proved to be pivotal to my career. I am thankful to my institute, **Madhav Institute of Technology and Science** to allow me to continue my disciplinary/interdisciplinary project as a curriculum requirement, under the provisions of the Flexible Curriculum Scheme (based on the AICTE Model Curriculum 2018), approved by the Academic Council of the institute. I extend my gratitude to the Director of the institute, **Dr. R. K. Pandit** and Dean Academics, **Dr. Manjaree Pandit** for this.

I would sincerely like to thank my department, **Centre for Artificial Intelligence,** for allowing me to explore this project. I humbly thank **Dr. R. R. Singh**, Coordinator, Centre for Artificial Intelligence, for his continued support during the course of this engagement, which eased the process and formalities involved.

I am sincerely thankful to my faculty mentors. I am grateful to the guidance of **Prof. Gaurisha Sisodia** Centre for Artificial Intelligence for his continued support and guidance throughout the project. I am also very thankful to the faculty and staff of the department.

Dhruv Agrawal
0901AI211024
Gauri Tripathi
0901IA211030
Centre for Artificial Intelligence

ABSTRACT

Agriculture is vital for human survival and remains a major driver of several economies around the world; more so in underdeveloped and developing economies. With increasing demand for food and cash crops, due to a growing global population and the challenges posedby climate change, there is a pressing need to increase farm outputs while incurring minimal costs. Previous machine vision technologies developed for selective weeding have faced the challenge of reliable and accurate weed detection. We present approaches for plant seedlings classification with a dataset that contains 4,275 images of approximately 960 unique plants belonging to 12 species at several growth stages. We compare the performances of two traditional algorithms and a Convolutional Neural Network (CNN), a deep learning techniquewidely applied to image recognition, for this task. Our findings show that CNN-driven seedling classification applications when used in farming automation has the potential to optimize crop yield and improve productivity and efficiency when designed appropriately

सार :

कृषि मानव अस्तित्व के लिए महत्वपूर्ण है और दुनिया भर की कई अर्थव्यवस्थाओं का एक प्रमुख चालक बना हुआ है; अविकसित और विकासशील अर्थव्यवस्थाओं में अधिक। बढ़ती वैश्विक आबादी और जलवायु परिवर्तन से उत्पन्न चुनौतियों के कारण खाद्य और नकदी फसलों की बढ़ती मांग के साथ, न्यूनतम लागत ों को उठाते हुए कृषि उत्पादन को बढ़ाने की आवश्यकता है। चयनात्मक निराई के लिए विकसित पिछली मशीन दृष्टि प्रौद्योगिकियों को विश्वसनीय और सटीक खरपतवार का पता लगाने की चुनौती का सामना करना पड़ा है।हम एक डेटासेट के साथ पौधों के रोपाई वर्गीकरण के लिए दृष्टिकोण प्रस्तुत करते हैं जिसमें कई विकास चरणों में 12 प्रजातियों से संबंधित लगभग 960 अद्वितीय पौधों की 4,275 छवियां शामिल हैं। हम इस कार्य के लिए दो पारंपरिक एल्गोरिदम और एक कन्वोल्यूशनल न्यूरल नेटवर्क (सीएनएन) के प्रदर्शन की तुलना करते हैं, जो छवि पहचान के लिए व्यापक रूप से लागू एक गहरी सीखने की तकनीक है। हमारे निष्कर्ष बताते हैं कि सीएनएन-संचालित अंकुर वर्गीकरण अनुप्रयोगों का उपयोग जब खेती स्वचालन में किया जाता है तो फसल को अनुकूलित करने की क्षमता होती है।

TABLE OF CONTENTS

Chapter 1: INTRODUCTION

1.1 AIM :To use deep learning in solving the real world agriculture related problems .

1.1.1 Contextual Background

Plants continue to serve as a source of food and oxygen for all life on earth. In continents like Africa, where agriculture is predominant, proper automation of the farming process would help optimize crop yield and ensure continuous productivity and sustainability . The transformation of the agricultural sector by use of smart farming methods can power economic growth in many countries. According to [2], there is a strong link between increased productivity and economic prosperity.

1.1.2 Problem Statement

In this work, we explore the performance of traditional computer vision methods on this task and show that a Deep Convolutional Neural Network (CNN) does the best job at classifying plant seedlings. In computer vision, CNNs have been known to be powerful visual models that yield hierarchies of features enabling accurate segmentation. They are also known to perform predictions relatively faster than other algorithms while maintaining competitive performance at the same time

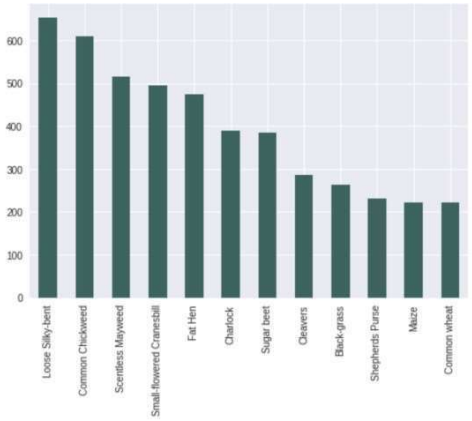


Figure 1: Bar graph showing the distribution of the different classes of plants

1.2 Objectives and Scope

One major reason for reduction in crop yield is weed invasion on farmlands. Weeds generally have no useful value in terms of food, nutrition or medicine yet they have accelerated growth and parasitically compete with actual crops for nutrients and space. Inefficient processes such as hand weeding has led to significant losses and increasing costs due to manual labour [3]. Precision agriculture, with the goal of defining systems that support decision-making in farm management in order to optimize returns onoutputs while preserving resources, and weed control systems have been developed aiming at optimizing yields and costs while minimizing environmental challenges;some robotic systems have been used to do this [4]. The robots and the vision machines need to be able to precisely and reliably detect a weed from the useful plants. Machine vision technologies developed for selective weeding face a challenge of reliable and accurate weed detection. It's not easy to identify the weeds due to unclear crops boundaries, with varying rocky or sandy backgrounds, and as a result, traditional classification methods are likely to fail on this task [5].

1.3 Features

- **Data Handling:**
  - The code loads training and testing images from specified directories using Image Data Generator. Training images are resized to (80, 80) pixels, converted to RGB, and normalized.
- **Model Architecture**:
  - The code utilizes a pre-trained VGG19 model for transfer learning.
  - Additional layers are added to the VGG19 model to create a custom model (NewModel) for the specific classification task.
  - The top layers of the VGG19 model are excluded, and new layers, including dense layers, are added for fine-tuning.

- **Fine-Tuning and Freezing Layers:**
  - The code freezes the layers of the pre-trained VGG19 model up to the last five layers, allowing the model to retain knowledge from the pre- training.

- **Model Compilation and Training:**
  - The model is compiled with the Adam optimizer and categorical cross-entropy loss function.
  - The model is trained using the fit_generator method with a specified number of epochs.

- **Model Evaluation:**
  - The trained model is used to make predictions on the test data.
  - Predictions are converted back to class names and stored in a DataFrame (pred).
- **Plotting and Visualization:**
  - Training history, including accuracy and loss over epochs, is plotted and saved as an image (combined_plot.png).
  - The architecture of the original VGG19 model and the modified model (NewModel) is visualized and saved as images.

- **HTML Display:**
  - The code generates HTML code to display images and plots in the notebook using base64 encoding.

- **Additional Information:**
  - The code sets constants such as the number of epochs, batch size, and the target shape for image resizing.
  - The code uses K-Fold cross-validation (KFold) but does not show the specific use in the provided snippet.
  - The model is compiled with the Adam optimizer and trained using the fit_generator method.

1.4 Feasibility And System Requirement

Feasible Aspects:

1. **Transfer Learning:**
    - Transfer learning using pre-trained models like VGG19 is a common and feasible approach for image classification tasks. It allows leveraging knowledge learned from large datasets.
2. **Data Preprocessing**:
    - The code includes necessary preprocessing steps such as image resizing, conversion to RGB, and normalization, which are essential for training deep learning models.
3. **Model Architecture**:
    - The model architecture, with additional layers for fine-tuning, is reasonable for image classification tasks. It allows the model to learn specific features relevant to plant seedling classification.
4. **Training and Evaluation:**
    - The model is compiled with an appropriate optimizer and loss function. Training and evaluation steps are included, and the model's performance is assessed using accuracy and loss metrics.
5. **Visualization:**
    - The code provides visualizations of the model architecture and training history, aiding in understanding and debugging.

1.4.1 Hardware

Hardware Requirements:

1. GPU (Graphics Processing Unit):

- Deep learning models, especially those based on convolutional neural networks (CNNs) like VGG19, can benefit significantly from GPU acceleration. The code assumes access to a GPU for faster training. In

particular, training large models like VGG19 can be computationally intensive, and a GPU can significantly speed up the process.

2. Sufficient RAM:
    - Training deep learning models can be memory-intensive. Ensure that your system has enough RAM to handle the size of the dataset and the computational requirements of the model.

1.4.2 Software

Software Requirements:

1. **Operating System:**
    - The code does not explicitly specify the operating system, but it appears to be designed to run in an environment that supports the specified Python libraries (TensorFlow, NumPy, Pandas, Matplotlib, Seaborn, etc.). The code includes paths with directory structures compatible with Unix-like systems (e.g., Linux).
2. **Python:**
    - The code is written in Python, so a Python interpreter is required. It's likely that the code is intended to run with a version of Python 3.x.
3. **Python Libraries:**
    - The code relies on several Python libraries, including TensorFlow, NumPy, Pandas, Matplotlib, Seaborn, PIL (Pillow), and scikit-learn. Ensure that these libraries are installed in your Python environment.
4. **Jupyter Notebook:**
    - The code appears to be designed for execution in a Jupyter Notebook environment. It uses Jupyter-specific display functions (display(HTML(...))) and assumes access to the IPython display capabilities.
5. **Kaggle Environment:**
    - The code includes paths ('/kaggle/input/...') that are consistent with the directory structure of a Kaggle kernel. It suggests that the code might have been developed and tested on the Kaggle platform.

Chapter 2. Literature review

2.1 CNN

Convolutional Neural Networks (CNNs) have emerged as a groundbreaking technology in the field of computer vision and image recognition. CNNs are specialized neural networks designed to process and analyze visual data, making themparticularly well-suited for tasks such as image classification, object detection, and segmentation. The architecture of CNNs is inspired by the visual processing in the human brain, featuring convolutional layers that automatically learn hierarchical representations from the input data.

Key Points:

- CNNs have demonstrated exceptional performance in various image-related tasks, surpassing traditional methods in accuracy and efficiency.
- Their ability to automatically learn hierarchical features through convolutional layers makes them robust to variations in scale, orientation, and position.
- Convolutional Neural Networks (CNNs) are a class of deep neural networks specifically designed for tasks involving images and spatial data. They have been particularly successful in computer vision applications, including image recognition, object detection, and image segmentation. CNNs are inspired by the organization and functionality of the visual cortex in animals, and they excel at automatically learning hierarchical features from raw input data.

  Here are the key components and concepts of Convolutional Neural Networks:

1. Convolutional Layers:

The fundamental building block of CNNs is the convolutional layer. These layers apply convolutional operations to input data using filters (also called kernels). Each filter scans over the input data, performing element-wise multiplications and summations to produce feature maps. These feature maps capture different aspects orpatterns in the input, allowing the network to learn hierarchical representations.

2. Pooling Layers:

Pooling layers are used to downsample the spatial dimensions of the input data. Max pooling is a common technique where the maximum value from a group of neighboring pixels is retained, reducing the spatial resolution and computational complexity. Pooling helps make the representation more robust and invariant to small translations and distortions in the input data.

3. Activation Functions:

Activation functions introduce non-linearities into the network, allowing it to learn complex mappings between input and output. Common activation functions used in CNNs include Rectified Linear Unit (ReLU), which replaces negative values with zero, promoting sparse activations and efficient learning.

4. Fully Connected Layers:

After several convolutional and pooling layers, CNNs typically end with one or more fully connected layers. These layers combine the learned features and make final predictions. In image classification tasks, the output layer often employs a softmax activation function to produce class probabilities.

5. Stride and Padding:

Stride determines the step size of the filter as it moves across the input data during convolution. Padding involves adding extra pixels around the input to prevent information loss at the edges. These parameters influence the spatial dimensions of the feature maps.

6. Hierarchical Feature Learning:

CNNs automatically learn hierarchical representations of features. Lower layers capture simple patterns like edges and textures, while deeper layers combine these patterns to form more complex and abstract features, enabling the network to recognize objects and scenes.

7. Transfer Learning:

CNNs, especially pre-trained models on large datasets, can be used for transfer learning. By leveraging the knowledge gained from one task (e.g., ImageNet classification), the pre-trained model can be fine-tuned for a different task with a smaller dataset.

8. Applications:

CNNs find applications in various computer vision tasks, including image classification, object detection, image segmentation, and facial recognition. They havealso been successful in natural language processing tasks when applied to the analysis of sequential data.

2.2 VG19

- The VGG (Visual Geometry Group) architecture, specifically VGG19, is a widely recognized deep CNN architecture that has made significant contributions to image classification tasks. Developed by the Visual Geometry Group at the University of Oxford, VGG19 is characterized by its simplicity and uniform architecture, comprising 19 layers, including convolutional and fully connected layers. The repeated use of small kernel sizes in convolutional layers allows VGG19 to capture intricate features in the input images.
- VGG19 gained prominence for its straightforward design, making it easy to understand and implement.
- The stacking of small convolutional filters contributes to the network's ability to learn complex features with fewer parameters.
- Despite its success, VGG19 is computationally expensive and may face challenges in terms of memory usage and training time
- VGG19 is a deep convolutional neural network architecture that belongs to the VGG family, developed by the Visual Geometry Group at the University of Oxford. The term "VGG" stands for Visual Geometry Group. The VGG architectures are known for their simplicity and uniform structure, making them easy to understand and implement. VGG19 specifically is characterized by its depth, consisting of 19 layers, and it has been widely used for image classification tasks.

Here are the key features of the VGG19 architecture:

1. Layer Configuration:

VGG19 has a straightforward architecture with a sequential stacking of convolutional layers. The network comprises a series of convolutional layers, each followed by a max-pooling layer for spatial down sampling. The depth of VGG19 is a result of stacking multiple convolutional layers.

2. Small Convolutional Filters:

One distinctive feature of VGG architectures, including VGG19, is the use of small 3x3 convolutional filters. The repeated use of these small filters is shown to be effective in capturing complex features and patterns in the input data. Convolutionallayers with 3x3 filters are used multiple times before spatial pooling is applied, contributing to the hierarchical feature learning.

3. Uniform Architecture:

VGG19 maintains a uniform architecture throughout the network. The convolutional layers are followed by max-pooling layers, and the fully connected layers are placed at the end of the network. The simplicity and uniformity of the architecture make it easy to understand and modify.

4. Fully Connected Layers:

After the convolutional and pooling layers, VGG19 has three fully connected layers, followed by a softmax activation layer for classification. The fully connected layers at the end of the network are responsible for combining high-level features learned by the convolutional layers and making final predictions.

5. Model Depth:

VGG19's depth, with 19 layers, was considered deep at the time of its introduction, contributing to its effectiveness in learning hierarchical representations. However, deeper architectures have since been developed, such as ResNet and DenseNet.

2.3 Transfer Learning

Transfer learning is a machine learning technique where a model trained on one task is adapted for a second related task. In the context of deep learning, transfer learning involves taking a pre-trained neural network model and using it as a starting point for a new but related task. Instead of training a deep neural network from scratch, transfer learning leverages the knowledge gained from a source task to improve the learning process on a target task.

Transfer learning is a machine learning technique where a model trained on one task is adapted for a second related task. In the context of deep learning, transfer learning involves taking a pre-trained neural network model and using it as a starting point for a new but related task. Instead of training a deep neural network from scratch, transfer learning leverages the knowledge gained from a source task to improve the learning process on a target task.

Here's a breakdown of the key concepts in transfer learning:

Source Task:

In transfer learning, there is a source task for which a model is pre-trained on a large dataset. This source task is usually a generic task with a large and diverse dataset. Examples include image classification on ImageNet, language modeling on a large corpus, or even tasks like object detection

1. **Pre-Trained Model:**
    - The model trained on the source task is referred to as the pre-trained model. This model has already learned useful features and patterns from the source task data, capturing general representations that are transferable to other tasks.
2. **Target Task:**
    - The target task is the specific task for which transfer learning is applied. It is typically a related task to the source task but may have a smaller dataset or slightly different characteristics.
3. **Fine-Tuning or Feature Extraction:**
    - There are two main approaches to transfer learning: fine-tuning and feature extraction.

- Fine-Tuning: In fine-tuning, the pre-trained model is further trained on the target task with the target dataset. This involves updating the weights of the model to adapt to the specifics of the new task.
- Feature Extraction: In feature extraction, the pre-trained model is used as a fixed feature extractor. The early layers of the model, which capture more generic features, are frozen, and only the later layers are trained on the target task.

4. **Benefits of Transfer Learning:**
    - Data Efficiency: Transfer learning allows the model to benefit from the knowledge gained on a large dataset, even when the target dataset is small.
    - Faster Convergence: Training a model from scratch can be time-consuming. Transfer learning often leads to faster convergence since the model starts with pre-learned features.
    - Improved Generalization: The pre-trained model has already learned generic features, which can improve the model's ability to generalize to new tasks.
5. **Domains of Transfer Learning:**
    - Transfer learning is widely used in various domains, including computer vision, natural language processing, and speech recognition. In computer vision, for example, a model pre-trained on a large image dataset can be adapted for specific image classification tasks.
6. **Popular Pre-Trained Models:**
    - In computer vision, popular pre-trained models include VGG, ResNet, Inception, and MobileNet. In natural language processing, models like BERT and GPT are commonly used for transfer learning.

Chapter 3. Preliminary design

3.1 Model Architecture

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, None, None, 3)]   0

 block1_conv1 (Conv2D)       (None, None, None, 64)    1792

 block1_conv2 (Conv2D)       (None, None, None, 64)    36928

 block1_pool (MaxPooling2D)  (None, None, None, 64)    0

 block2_conv1 (Conv2D)       (None, None, None, 128)   73856

 block2_conv2 (Conv2D)       (None, None, None, 128)   147584

 block2_pool (MaxPooling2D)  (None, None, None, 128)   0

 block3_conv1 (Conv2D)       (None, None, None, 256)   295168

 block3_conv2 (Conv2D)       (None, None, None, 256)   590080

 block3_conv3 (Conv2D)       (None, None, None, 256)   590080

 block3_conv4 (Conv2D)       (None, None, None, 256)   590080

 block3_pool (MaxPooling2D)  (None, None, None, 256)   0

 block4_conv1 (Conv2D)       (None, None, None, 512)   1180160

 block4_conv2 (Conv2D)       (None, None, None, 512)   2359808

 block4_conv3 (Conv2D)       (None, None, None, 512)   2359808

 block4_conv4 (Conv2D)       (None, None, None, 512)   2359808

 block4_pool (MaxPooling2D)  (None, None, None, 512)   0

 block5_conv1 (Conv2D)       (None, None, None, 512)   2359808
   block5_conv2 (Conv2D)       (None, None, None, 512)   2359808

   block5_conv3 (Conv2D)       (None, None, None, 512)   2359808

   block5_conv4 (Conv2D)       (None, None, None, 512)   2359808

   block5_pool (MaxPooling2D)  (None, None, None, 512)   0

   global_average_pooling2d (  (None, 512)               0
   GlobalAveragePooling2D)

   dense (Dense)               (None, 1024)              525312

   dense_1 (Dense)             (None, 1024)              1049600

   dense_2 (Dense)             (None, 512)               524800

   dense_3 (Dense)             (None, 12)                6156

 =================================================================
 Total params: 22130252 (84.42 MB)
 Trainable params: 2105868 (8.03 MB)
 Non-trainable params: 20024384 (76.39 MB)
 _________________________________________________________________
```

1. VGG19 Base Model:

```
VGG19_MODEL = VGG19(weights='imagenet', include_top=False)
```

The code imports the VGG19 model from Keras's applications module. This model is pre-trained on the ImageNet dataset and is used as the base model.

2. Global Average Pooling and Dense Layers:

```
x=VGG19_MODEL.output
x=GlobalAveragePooling2D()(x)
x=Dense(1024,activation='relu')(x) #we add dense layers so that the model can learn more complex
tions and classify for better results.
x=Dense(1024,activation='relu')(x) #dense layer 2
x=Dense(512,activation='relu')(x) #dense layer 3
preds=Dense(len(class_name_num), activation='softmax')(x) #final layer with softmax activation
```

- The output from the VGG19 base model is passed through a global average pooling layer. This layer reduces the spatial dimensions of the output.
- Subsequently, there are three dense (fully connected) layers. These layers are added to enable the model to learn more complex representations and classify the input data.
- The number of neurons in these dense layers is gradually reduced, leading to a final dense layer with the number of neurons equal to the number of classes in your classification task.
- The activation function used in the dense layers is ReLU (Rectified Linear Unit), except for the final layer where softmax activation is used for multi-class classification.

3. Creating the New Model:

```
NewModel=Model(inputs=VGG19_MODEL.input,outputs=preds)
NewModel.summary()
```

- The Model class from Keras is used to create a new model by specifying the inputs and outputs. The input is set as the input of the VGG19 base model, and the output is set as the final dense layer.

4. Freezing Layers:

```
for layer in NewModel.layers[:-5]:
    layer.trainable=False

NewModel.summary()
```

- This code freezes the layers of the VGG19 base model up to the last five layers. Freezing means that the weights of these layers won't be updated during training. Thepurpose is to retain the knowledge learned from ImageNet and fine-tune the model onthe specific task.

5. Model Summary and Visualization:

```
NewModel.summary()
```

- This code prints a summary of the architecture of the new model, including the layer types, output shapes, and the number of parameters.

6. Model Compilation:

```
NewModel.compile(optimizer='Adam',loss='categorical_crossentropy',metrics=['accuracy'])
```

- The model is compiled with the Adam optimizer, categorical crossentropy as the loss function (commonly used for multi-class classification), and accuracy as the metric to monitor during training.

7. Data Generator:

```
train_datagen=ImageDataGenerator(preprocessing_function=preprocess_input)
train_generator=train_datagen.flow_from_directory(
    pathToTrainData,
    target_size=(80,80),
    color_mode='rgb',
    batch_size=32,
    class_mode='categorical',
    shuffle=True
);
```

- An ImageDataGenerator is set up for data augmentation during training. It generates batches of augmented images from the specified directory.

8. Model Training:

```
step_size_train=train_generator.n//train_generator.batch_size
history = NewModel.fit_generator(generator=train_generator,steps_per_epoch=step_size_train,epochs=1
0);
```

- The model is trained using the generator created from the training data. The training history, including loss and accuracy over epochs, is stored in the history variable.

9. Model Evaluation and Prediction:

```
predictions = NewModel.predict(
    X_test,
    batch_size=None,
    verbose=0,
    steps=None,
    callbacks=None,
    max_queue_size=10,
    workers=1,
    use_multiprocessing=False
)

predictions=pd.DataFrame(predictions)
```

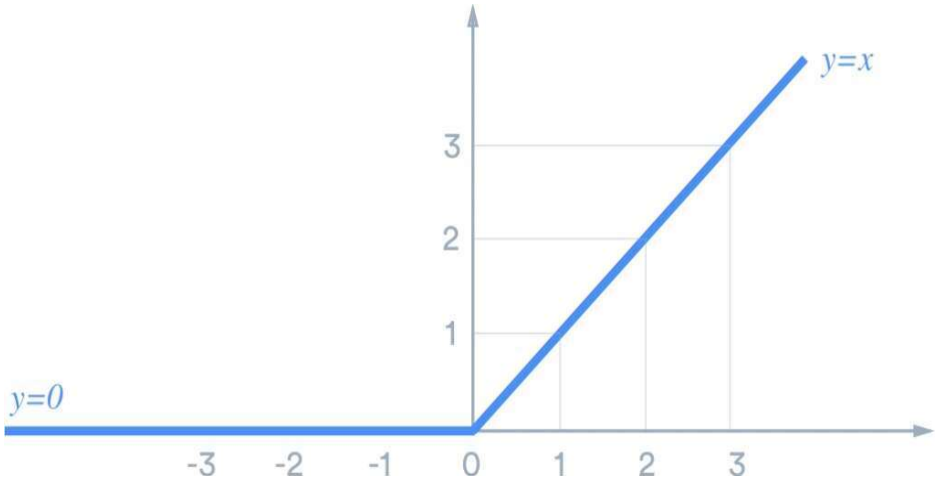
- The trained model is used to make predictions on the test data (X_test).

3.2 Activation Function

- **Relu:**

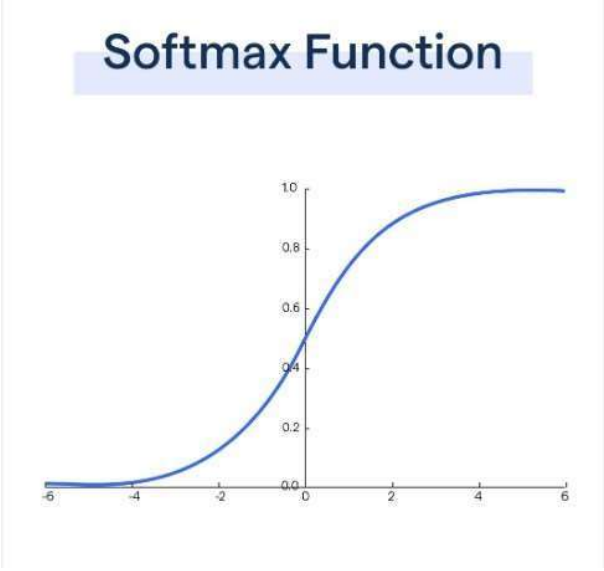
  The Rectified Linear Unit, or ReLU for short, is one of the many activation functions available to you for deep learning. What makes the ReLU activation function stand out is its simplicity while being an incredibly powerful function.

  While the name rectified linear unit may sound complex, the function is anything but. At its core, the ReLU function applies a very straightforward rule: if the input is greater than zero, it leaves it unchanged; otherwise, it sets itto zero.



- **Softmax**

The softmax function is often used as the last activation function of a neural network to normalize the output of a network to a probability distribution over predicted output classes, based on Luce's choice axiom.



3.3 CODE:

```
!pip3 install -q seaborn tensorflow pillow scikit-learn pydot graphviz
# Standard Libraries
import  os
import warnings

# External Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from PIL import Image
from numpy import array, asarray
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score, confusion_matrix

# TensorFlow and Keras
from tensorflow.keras import Sequential
from tensorflow.keras.layers import (
   Dense,
   Flatten,
   Conv2D,
   MaxPooling2D,
   GlobalAveragePooling2D,
)
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications.vgg19 import VGG19
from tensorflow.keras.utils import plot_model
```

```
from tensorflow.keras.applications.mobilenet import preprocess_input

# Other Utilities
import base64
from IPython.display import HTML, display

# Set seed and constants
np.random.seed(42)
kf = KFold(n_splits=5)
epochs = 20
batch_size = 32

# Suppress warnings
warnings.filterwarnings("ignore")

pathToTrainData='/kaggle/input/plant-seedlings-classification/train'
pathToTestData ='/kaggle/input/plant-seedlings-classification/test'

training_img_list = list()
testing_img_list = list()

shape_sum       = 0
class_name_num = dict()
train_avg_shape = 80

for dirname, _, filenames in os.walk(pathToTrainData):
    for filename in filenames:
        img_data = Image.open(os.path.join(dirname, filename))

        resizedImage = img_data.resize((train_avg_shape, train_avg_shape))
        resizedImage = resizedImage.convert('RGB')
        resizedImage = asarray(resizedImage)/255
```

```
        class_label = dirname.split('/')[-1]
        training_img_list.append([resizedImage, class_label])
        shape_sum += np.max(img_data.size)
        class_name_num[class_label] = len(class_name_num)-1



for dirname, _, filenames in os.walk(pathToTestData):
    for filename in filenames:
        img_data = Image.open(os.path.join(dirname, filename))


        resizedImage = img_data.resize((train_avg_shape, train_avg_shape))
        resizedImage = resizedImage.convert('RGB')
        resizedImage = asarray(resizedImage)/255


        testing_img_list.append([resizedImage,filename])


X_test = np.zeros((len(testing_img_list), train_avg_shape, train_avg_shape, 3),
dtype='float32')
for i,img in enumerate(testing_img_list):
    X_test[i] = testing_img_list[i][0]
VGG19_MODEL = VGG19(weights='imagenet', include_top=False)
# Specify the path to save the plot
plot_path = '/kaggle/working/VGG19Original.png'
# Save the plot to the specified path
plot_model(VGG19_MODEL, to_file=plot_path, show_shapes=True,
show_layer_names=True)
# Check if the file exists before reading
if os.path.exists(plot_path):
    with open(plot_path, "rb") as img_file:
        img_data = img_file.read()
    img_base64 = base64.b64encode(img_data).decode("utf-8")
```

```
    html_code = f"
    <div style="background-color:white; border-radius:2px; border:#000000 solid; padding: 15px; font-size:100%; text-align:center;">
        <img src="{img_base64}" style="display: block; margin: 0 auto;">
    </div>
    '''
    display(HTML(html_code))
else:
    print(f"Error: File '{plot_path}' not found.")


VGG19_MODEL.summary()


print(f"VGG19 Model Layers Count :{len(VGG19_MODEL.layers)}")


x=VGG19_MODEL.output
x=GlobalAveragePooling2D()(x)
x=Dense(1024,activation='relu')(x) #we add dense layers so that the model can learn more complex functions and classify for better results.
x=Dense(1024,activation='relu')(x) #dense layer 2
x=Dense(512,activation='relu')(x) #dense layer 3
preds=Dense(len(class_name_num), activation='softmax')(x) #final layer with softmax activation
NewModel=Model(inputs=VGG19_MODEL.input,outputs=preds)
NewModel.summary()


print(f"New Model layers count :{len(NewModel.layers)}")


for layer in NewModel.layers[:-5]:
    layer.trainable=False


NewModel.summary()
```

```
plot_path = '/kaggle/working/NewModel.png'

# Save the plot to the specified path

plot_model(NewModel, to_file=plot_path, show_shapes=True,
show_layer_names=True)

# Check if the file exists before reading

if os.path.exists(plot_path):

    with open(plot_path, "rb") as img_file:

        img_data = img_file.read()

    img_base64 = base64.b64encode(img_data).decode("utf-8")


    html_code = f"

    <div style="background-color:white; border-radius:2px; border:#000000 solid;
padding: 15px; font-size:100%; text-align:center;">

        <img src="{img_base64}" style="display: block; margin:
0 auto;">

    </div>

    "

    display(HTML(html_code))

else:

    print(f"Error: File '{plot_path}' not found.")


train_datagen=ImageDataGenerator(preprocessing_function=preprocess_input)

train_generator=train_datagen.flow_from_directory(

    pathToTrainData,

    target_size=(80,80),

    color_mode='rgb',

    batch_size=32,

    class_mode='categorical',

    shuffle=True

)


NewModel.compile(optimizer='Adam',loss='categorical_crossentropy',metrics=['accu
racy'])
```

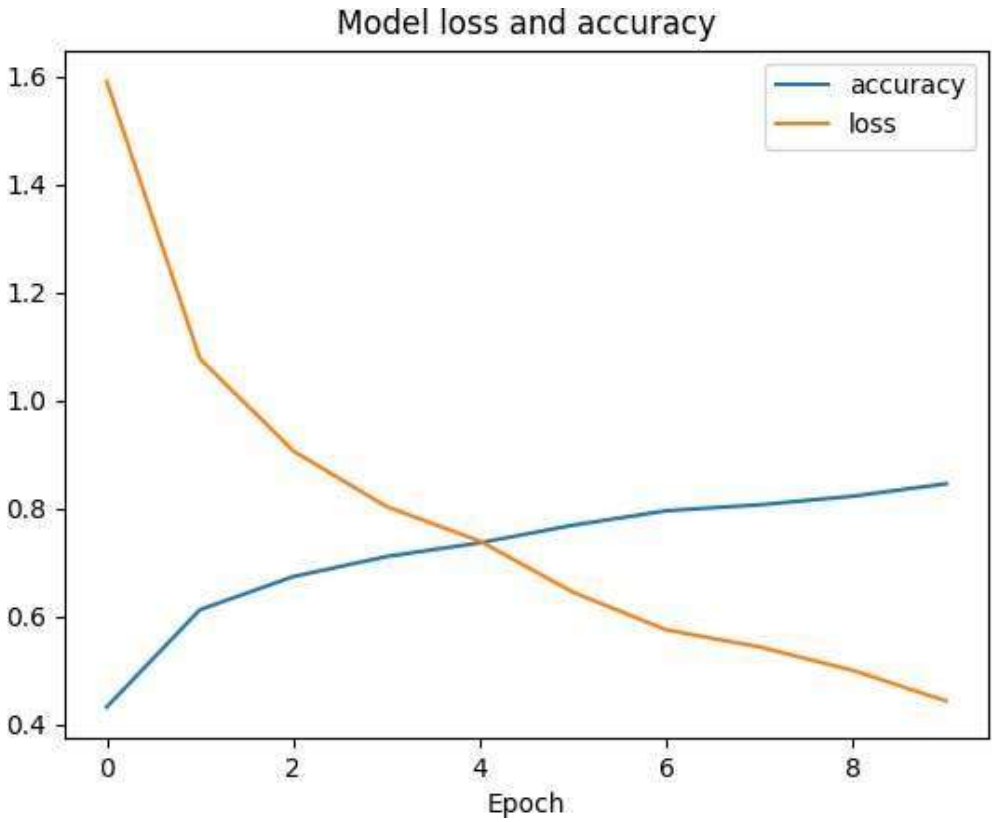
```
step_size_train=train_generator.n//train_generator.batch_size

history =
NewModel.fit_generator(generator=train_generator,steps_per_epoch=step_size_train,
epochs=10);


plt.plot(history.history['accuracy'])

plt.plot(history.history['loss'])

plt.title('Model loss and accuracy')

plt.xlabel('Epoch')

plt.legend(['accuracy','loss'], loc='upper right')

plt.savefig('combined_plot.png')

plt.close()

with open("combined_plot.png", "rb") as img_file:

   img_data = img_file.read()

img_base64 = base64.b64encode(img_data).decode("utf-8")

html_code = f'''

<div style="background-color:white; border-radius:2px; border:#000000 solid;
padding: 15px; font-size:100%; text-align:center;">

   <img src="{img_base64}" style="display: block; margin: 0
auto;">

</div>

'''

display(HTML(html_code))


predictions = NewModel.predict(

   X_test,

   batch_size=None,

   verbose=0,

   steps=None,

   callbacks=None,

   max_queue_size=10,

   workers=1,

   use_multiprocessing=False
```

```
)

predictions=pd.DataFrame(predictions)

inverse_label_map = dict()
for k,v in train_generator.class_indices.items():
    inverse_label_map[v] = k
pred_label_num = predictions.idxmax(axis=1)
pred_label_num_new = list()

for x in pred_label_num:
    y = inverse_label_map[x]
    pred_label_num_new.append(y)

pred_label_num_new = pd.DataFrame(pred_label_num_new)
print(pred_label_num_new[0])

testImages = pd.DataFrame(testing_img_list)
pred=pd.DataFrame()
pred["file"] = testImages[1]
pred["species"] = pred_label_num_new[0]
pred.head()
```

Chapter 4. Final Analysis and design

4.1 result Analysis

We attempt to use a CNN for this problem. CNNs have been widely used for diverse image classification tasks. We use two types of input sets; a first case where we pass in the original image pixels and a second case where we performed OpenCV preprocessing of the input image data as in the baseline. The neural network architecture has 6 convolutional layers.
Each is followed with a rectified linear unit (ReLU). The first two convolutional layers have 64 filters, the next has 128 while the last one has 256. Each convolutional layer has zero padding. After each pair of convolutional layer, we have a max pooling layer for dimensionality reduction and a 10% dropout to prevent over-fitting. At the end of the six convolutional layers are 3 fully connected layers. The last fully connected layer has a softmaxactivation function which outputs probability distribution for each of the 12 classes. We use Adam optimizer with a batch size of 32 for each step and a weighted cross-entropy loss, to handle the imbalanced number of pixels for each class.



4.2 Applications

An efficient deep learning model for seedlings classification can help farmers optimize crop yields and significantly reduce losses. The model can detect and differentiate a weed from other plants in the wild . The proposed system can be extended to work with robotic arms for performing actual weeding operation in large farmlands.

4.3 Problems Faced

1. Training a model with a more inclusive dataset. For instance, using plant seedlings that are more prevalent in African agriculture or other parts of the underdeveloped/developing world other than that of Danish agriculture as provided in our dataset

Testing out the model using images with multiple plants in a scene. Although the advantageof weeding during plant seedlings early stage is to minimize the challenges that come with overlapping, it would be insightful to see how well the model identifies different classes of plants and potentially predicting the ratio of the classes present.

4.4 Limitation

1. **Overfitting:**
    - Fine-tuning a pre-trained model on a small dataset can lead to overfitting. The model might memorize the limited training examples rather than generalizing well to new, unseen data.
2. **Limited Adaptability:**
    - Pre-trained models may not adapt well to unique characteristics of the target dataset. Fine-tuning may be needed to adjust the model to specific features of the new task.
3. **Domain Shift:**
    - If the distribution of data in the target task differs significantly from the source task, the pre-trained model may not perform well. This is known as domain shift.

4. **Computational Resources:**
    - Fine-tuning a large pre-trained model requires significant computational resources. For resource-constrained environments, training a model from scratch on a smaller dataset might be more practical.

4.5 Conclusion

We believe that with promising results in classifying plant seedlings, we will be able to completely automate the process of weed control in large farms and thereby reducing costsand manual labour while improving crop yield and productivity

References:

1. Corral, A. Díaz, M. Monagas, and E. García, Agricultural Policies and Their Impact on Poverty Reduction in Developing Countries: Lessons Learned from Three Water Basins in Cape Verde, Sustainability, vol. 9, no. 10, p. 1841, 2017.

2. The Role of International Institutions in Economic Development and Poverty Reduction in the Developing World, Food and agriculture Organization of the United Nations. Rome, 2018. URL http://www.fao.org/3/I9900EN/i9900en.pdf

3. Y. Gharde, P. Singh, R. Dubey, and P. Gupta, Assessment of yield and economic losses in agriculture due to weeds in India, Crop Protection, vol. 107, pp. 12–18, 2018

T. Giselsson, R. Jørgensen, P. Jensen, M. Dyrmann, and H. Midtiby, A public image database for benchmark of plant seedling classification algorithms, CoRR, abs/1711.05458, 2017

J. S. Sepp Hochreiter, "Long Short-Term Memory.," *Neural Computation,* vol. 9, no. 8, pp. 1735-1780, 1997.

J. M. a. IlyaSutskever, Training Recurrent Neural Networks, Toronto : Department of Computer Science, University of Toronto.