INFORMATION TECHNOLOGY INSTITUTE

ARTIFICIAL INTELLIGENCE TRACK - COMPUTER VISION PROJECT

PLANT SEEDLINGS CLASSIFICATION

PREPARED BY:
ROLA HOSSAM
MUHAMMAD NASSER
FAWZIA WAGEEH

PRESENTATION CONTENT

1. Project Description
2. Project Importance
3. Problems w/ Plant Seedling Classification
4. Project Objective
5. Success Determination
6. Dataset Information
7. Model Selection
8. Codes & Demo
9. Models Comparison

PROJECT DESCRIPTION

CREATE A SYSTEM THAT CLASSIFIES DIFFERENT TYPES OF PLANT SEEDLINGS.

IMPORTANCE OF
PLANT SEEDLING CLASSIFICATION

PLANT SEEDLING CLASSIFICATION IS A CRUCIAL TASK IN AGRICULTURE AND FORESTRY

PROBLEMS WITH
PLANT SEEDLING CLASSIFICATION

TRADITIONAL METHODS OF MANUALLY IDENTIFYING PLANT SPECIES FROM THEIR SEEDLINGS ARE TIME-CONSUMING AND PRONE TO ERRORS.

PROJECT'S OBJECTIVE

THE SYSTEM WILL BE TRAINED ON A DATASET OF IMAGES OF PLANT SEEDLINGS, WITH EACH IMAGE LABELED ACCORDING TO ITS CORRESPONDING PLANT SPECIES.

DETERMINE SUCCESS...

THE PROJECT'S SUCCESS WILL BE DETERMINED BY ACHIEVING HIGH ACCURACY AND PERFORMANCE IN CLASSIFYING DIFFERENT PLANT SEEDLINGS.

DATA PREPROCESSING

THE DATASET WILL BE PREPROCESSED
TO REMOVE NOISE AND ENHANCE
THE FEATURES OF THE IMAGES
TO IMPROVE THE SYSTEM'S ACCURACY.

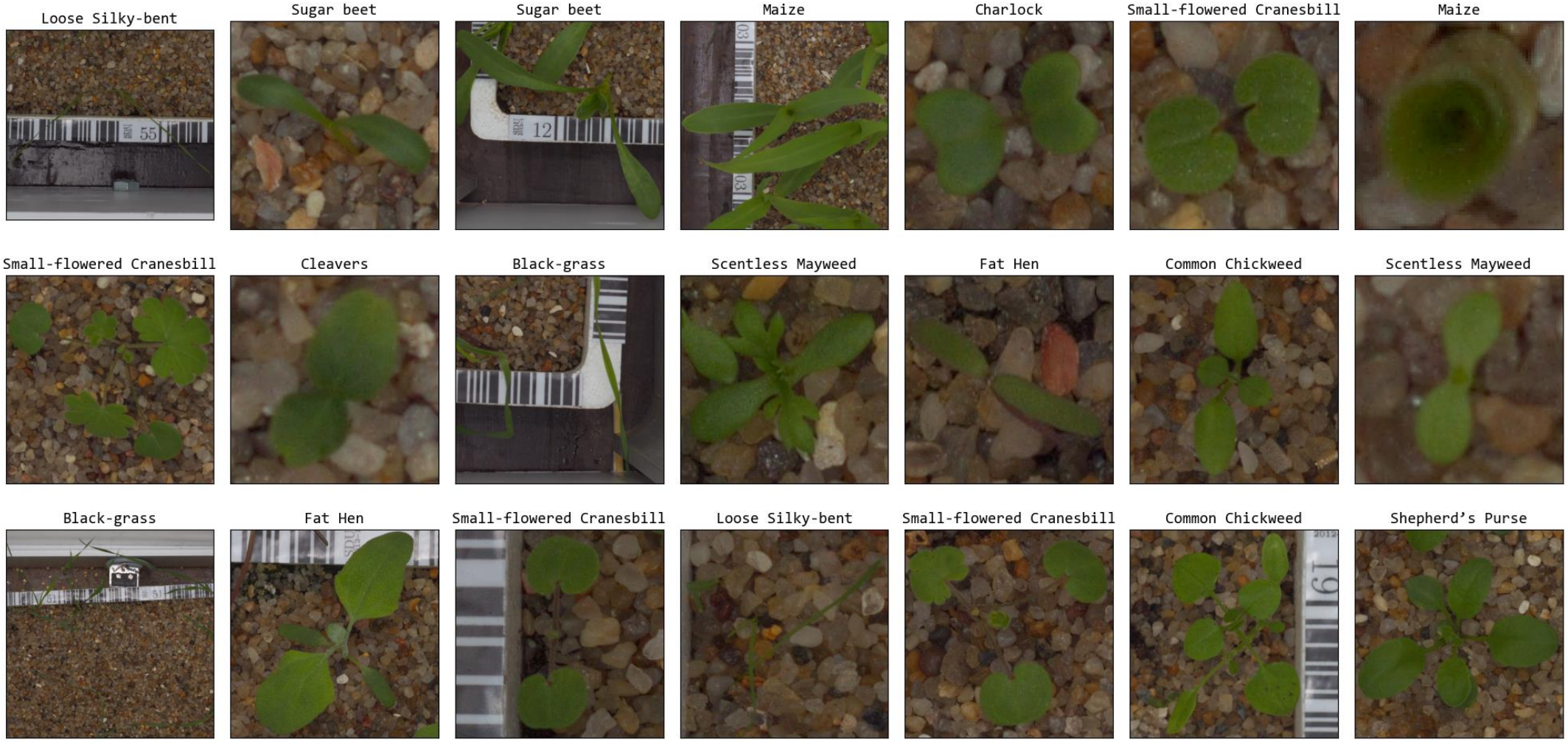
DATASET INFORMATION

CREDITS TO:

Computer Vision and Signal Processing Group
Department of Engineering - Aarhus University
DENMARK

Download the Dataset:
https://vision.eng.au.dk/plant-seedlings-dataset/

DATASET SAMPLE



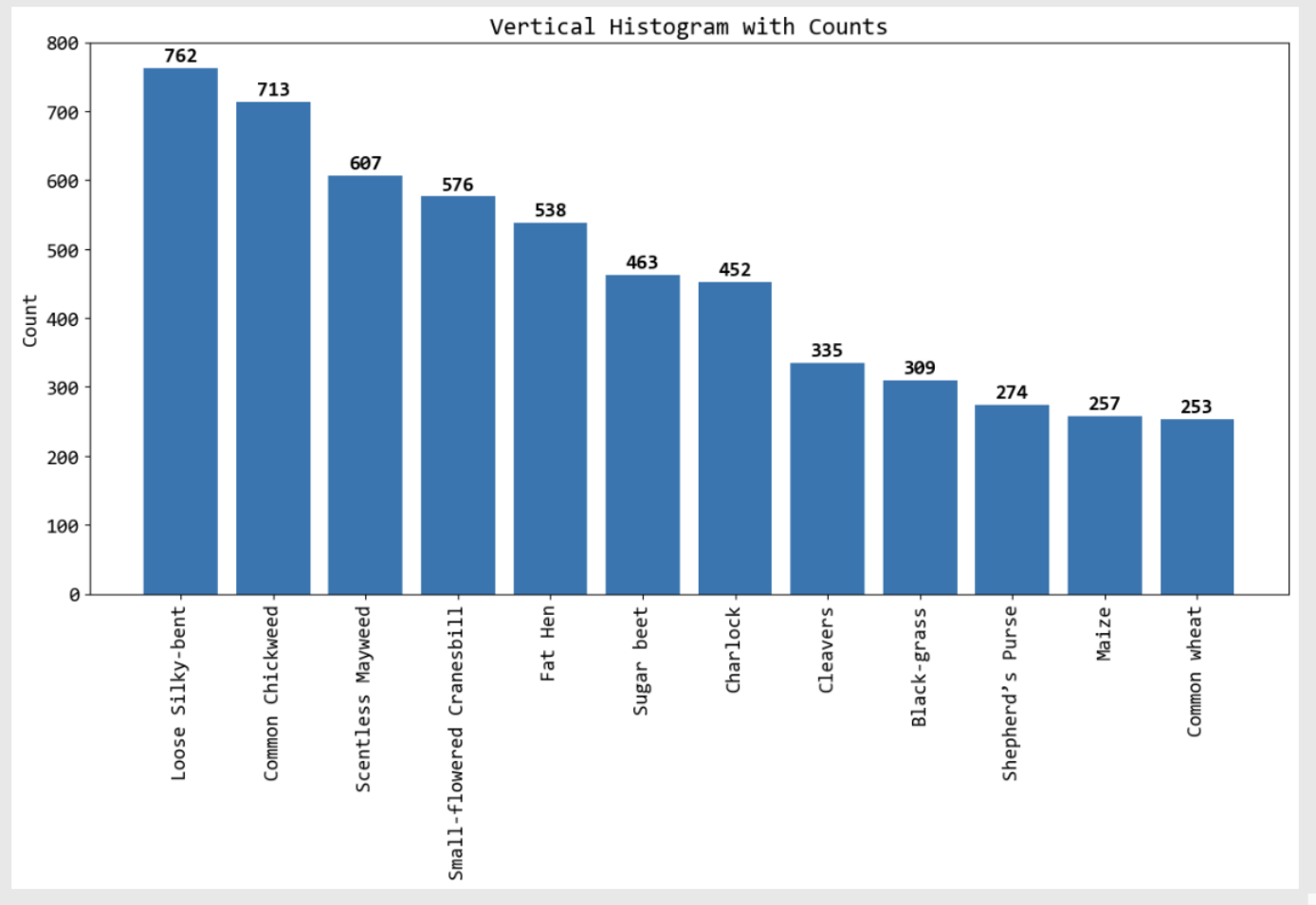
MODEL ONE
CONVOLUTION NEURAL NETWORK

MODEL #1: CNN

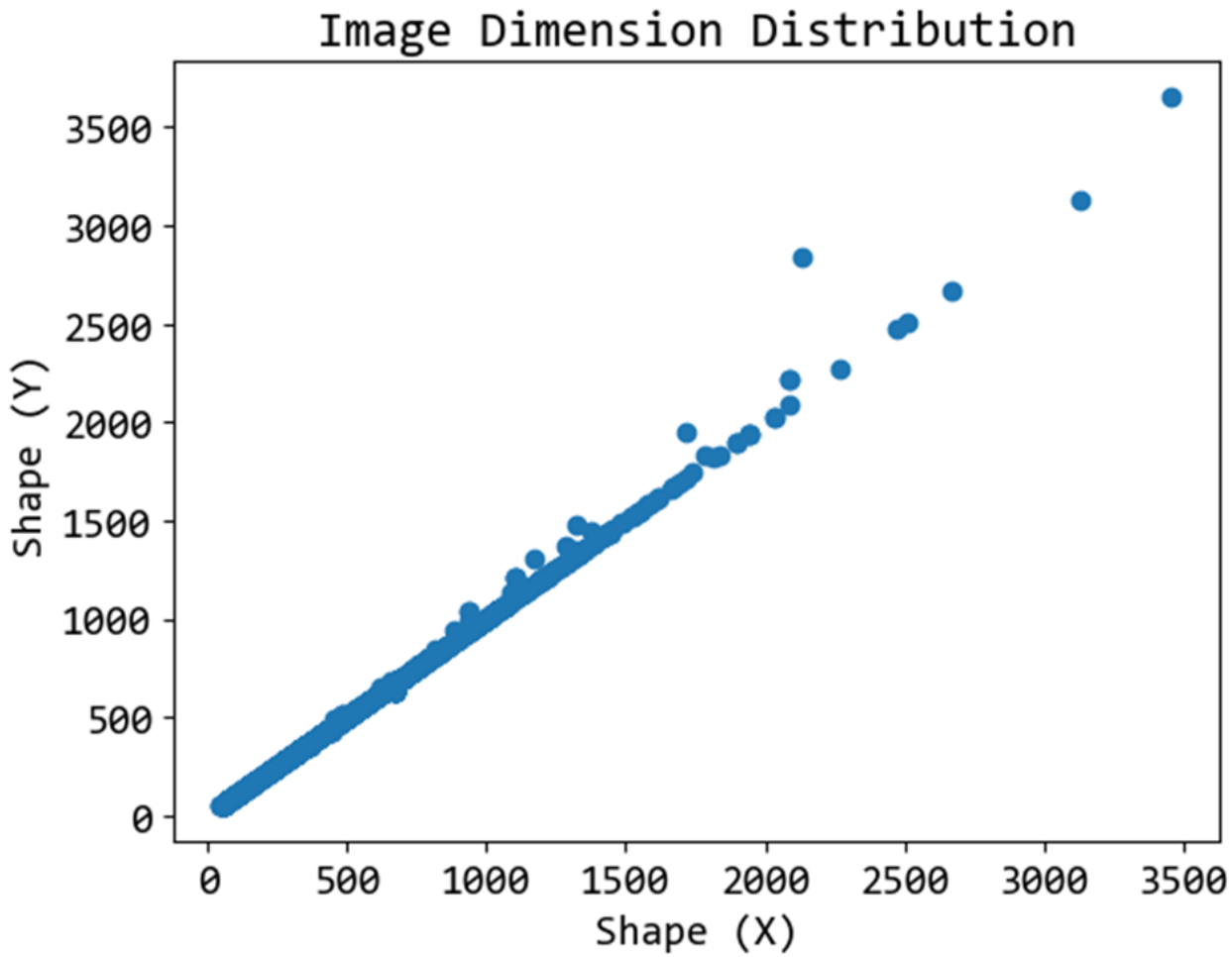
```
# Path to all images in training set. (* means include all folders and files.)
path = "/content/kaggle_data/train/**/*.png"
files = glob(path)

# Empty list to store the image data as numbers.
trainImg = []
# Empty list to store the labels of images
trainLabel = []
j = 1
num = len(files)

# Obtain images and resizing, obtain labels
for img in files:
    print(str(j) + "/" + str(num), end="\r")
    # Get image (with resizing to 128x128)
    trainImg.append(cv2.resize(cv2.imread(img), (128, 128)))
    trainLabel.append(img.split('/')[-2])  # Get image label (folder name
contains the class to which the image belong)
    j += 1

# Train images set
trainImg = np.asarray(trainImg)
# Train labels set
trainLabel = pd.DataFrame(trainLabel)
```

- Load the data images and labels into two lists after resizing them (128x 128).
- The labels are the directories names.

MODEL #1: CNN (contd.)



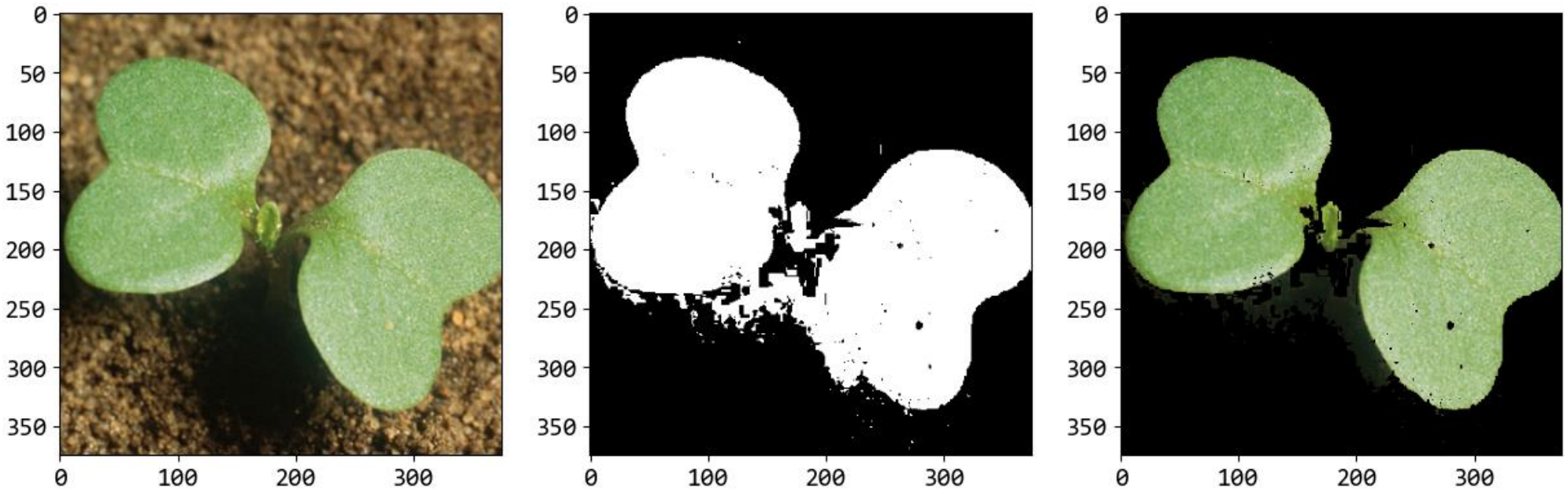
MODEL #1: CNN (contd.)



MODEL #1: CNN (contd.)

```
import cv2
def plot_mask(image, colormin, colormax):
    hsv_p1 = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)
    mask = cv2.inRange(hsv_p1, colormin , colormax)
    result = cv2.bitwise_and(image, image, mask=mask)
    plt.figure(figsize=(15,10))
    plt.subplot(1, 3, 1)
    plt.imshow(image)
    plt.grid(False)
    plt.subplot(1, 3, 2)
    plt.imshow(mask, cmap="gray")
    plt.grid(False)
    plt.subplot(1, 3, 3)
    plt.imshow(result)
    plt.grid(False)
    return plt.show()

colormin = (36, 25, 25)
colormax = (70, 255,255)

p1 = cv2.imread('charlock-young.jpg')
p1 = cv2.cvtColor(p1, cv2.COLOR_BGR2RGB)

plot_mask(p1, colormin, colormax)
```

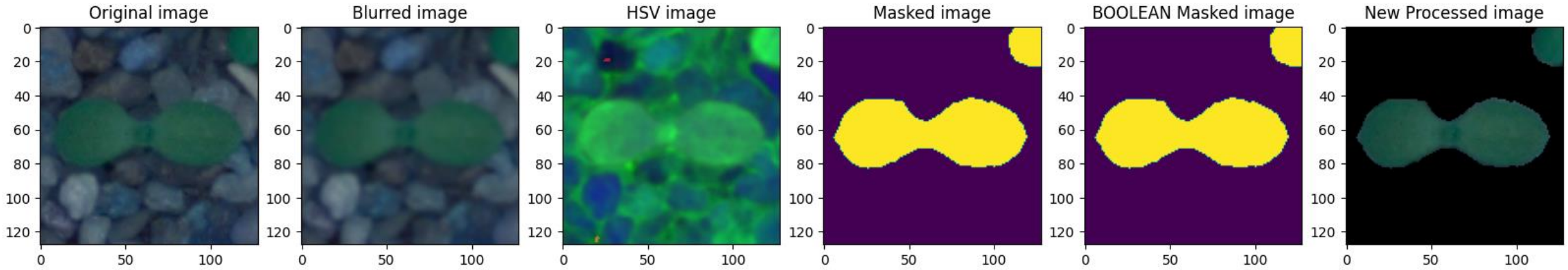
MODEL #1: CNN (contd.)



MODEL #1: CNN (contd.)

```
trainImg_new = []
sets = []; getEx = True

for i in trainImg:
    # Blurred image
    blurr = cv2.GaussianBlur(i,(5,5),0)
    # HSV image
    hsv = cv2.cvtColor(blurr,cv2.COLOR_BGR2HSV)

    #Green Parameters
    sensitivity = 35
    lower  = np.array([60 - sensitivity, 100, 50])
    upper = np.array([60 + sensitivity, 255, 255])

    #Masked image
    mask = cv2.inRange(hsv,lower,upper)
    struc = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(5,5))
    mask = cv2.morphologyEx(mask,cv2.MORPH_CLOSE,struc)

    #Boolean image
    boolean = mask>0
    new = np.zeros_like(i,np.uint8)
    new[boolean] = i[boolean]
    trainImg_new.append(new)

    if getEx:
        f = plt.figure(figsize=(20, 20))
        f.add_subplot(1,6,1);plt.imshow(i);plt.title('Original image') # Original image
        f.add_subplot(1,6,2);plt.imshow(blurr);plt.title('Blurred image') # Blurred image
        f.add_subplot(1,6,3);plt.imshow(hsv);plt.title('HSV image') # HSV image
        f.add_subplot(1,6,4);plt.imshow(mask);plt.title('Masked image') # Masked image
        f.add_subplot(1,6,5);plt.imshow(boolean);plt.title('BOOLEAN Masked image') # BOOLEAN Masked image
        f.add_subplot(1,6,6);plt.imshow(new);plt.title('New Processed image') # New Processed image
        getEx = False

trainImg_new = np.asarray(trainImg_new)
```

MODEL #1: CNN (contd.)



MODEL #1: CNN (contd.)

```
trainImg_new = trainImg_new.astype('float32') / 255.0
```

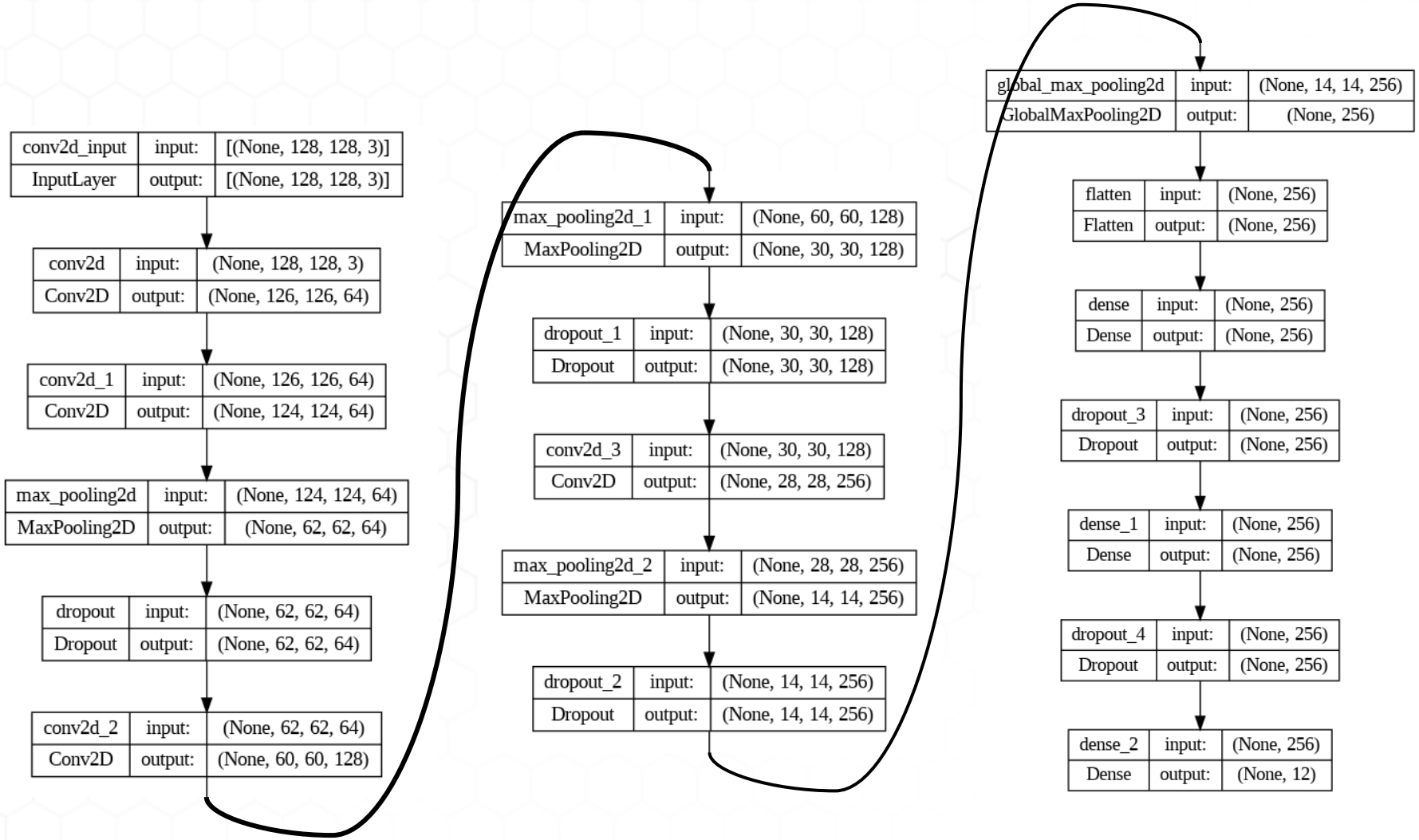
MODEL #1: CNN (contd.)

```
X_train,X_test,y_train,y_test = train_test_split(trainImg_new,convertedlabels,test_size=0.3,random_state=38,stratify=convertedlabels)
X_val,X_test_new,y_val,y_test_new = train_test_split(X_test,y_test,test_size=0.5,random_state=38,stratify=y_test)
```

MODEL #1: CNN (contd.)

```
model = Sequential()

model.add(Conv2D(filters=64, kernel_size=(3, 3), input_shape=(128, 128, 3), activation='relu'))
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPool2D((2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(filters=128, kernel_size=(3, 3), activation='relu'))
model.add(MaxPool2D((2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(filters=256, kernel_size=(3, 3), activation='relu'))
model.add(MaxPool2D((2, 2)))
model.add(Dropout(0.25))

model.add(GlobalMaxPool2D())

model.add(Flatten())

model.add(Dense(256, activation='relu'))
model.add(Dropout(0.25))

model.add(Dense(256, activation='relu'))
model.add(Dropout(0.25))

model.add(Dense(classes, activation='softmax'))

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

model.summary()
```

MODEL #1: CNN (contd.)



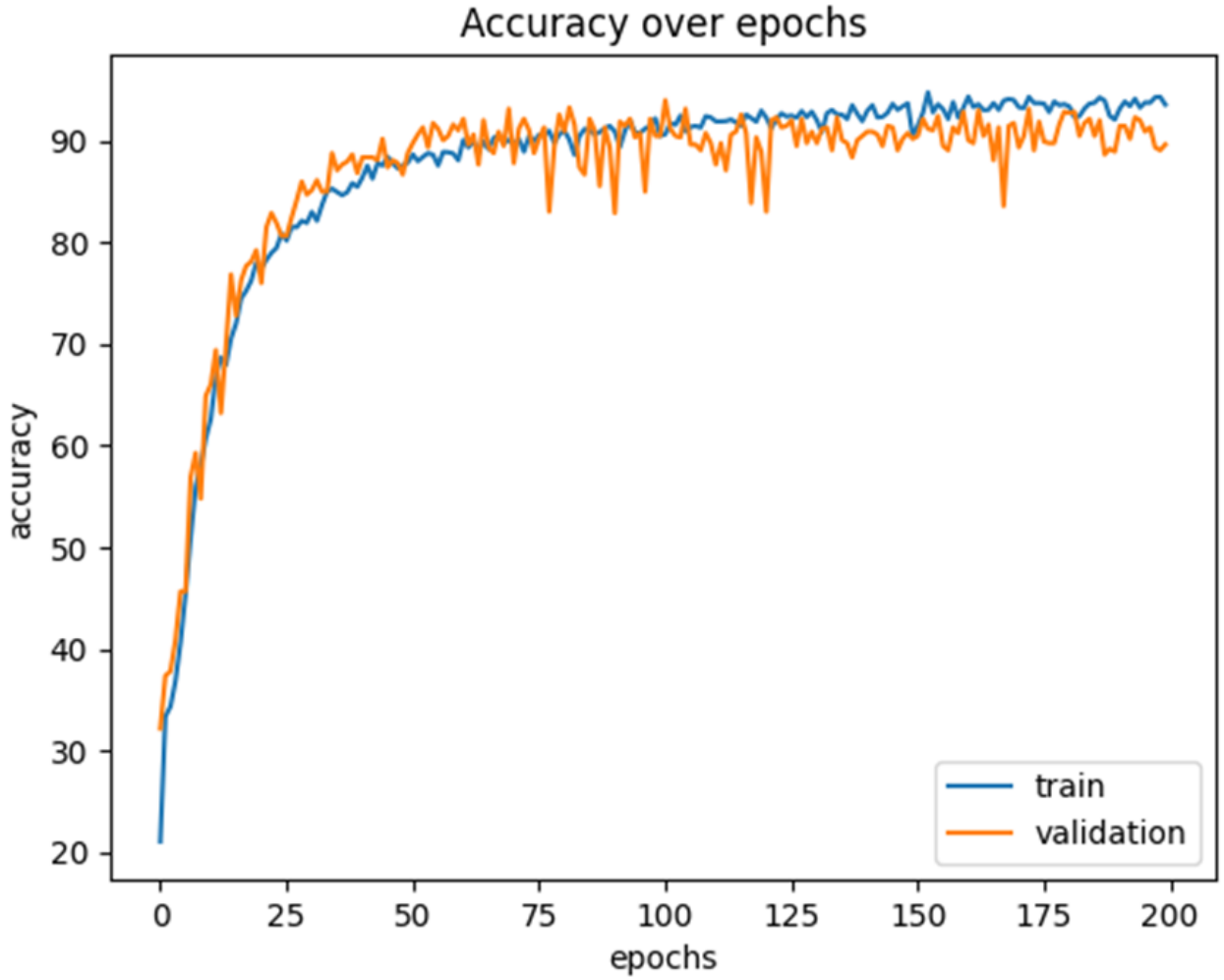
MODEL #1: CNN (contd.)

```
generator = ImageDataGenerator(rotation_range = 180,
                               zoom_range = 0.2,
                               width_shift_range = 0.2,
                               height_shift_range = 0.2,
                               horizontal_flip = True,
                               vertical_flip = True)
generator.fit(X_train)
```

MODEL #1: CNN (contd.)

```
history = model.fit(generator.flow(X_train,y_train,batch_size=64),epochs=200, verbose=2,shuffle=True,validation_data=(X_val,y_val))
pd.DataFrame(history.history)
```

MODEL #1: CNN (contd.)

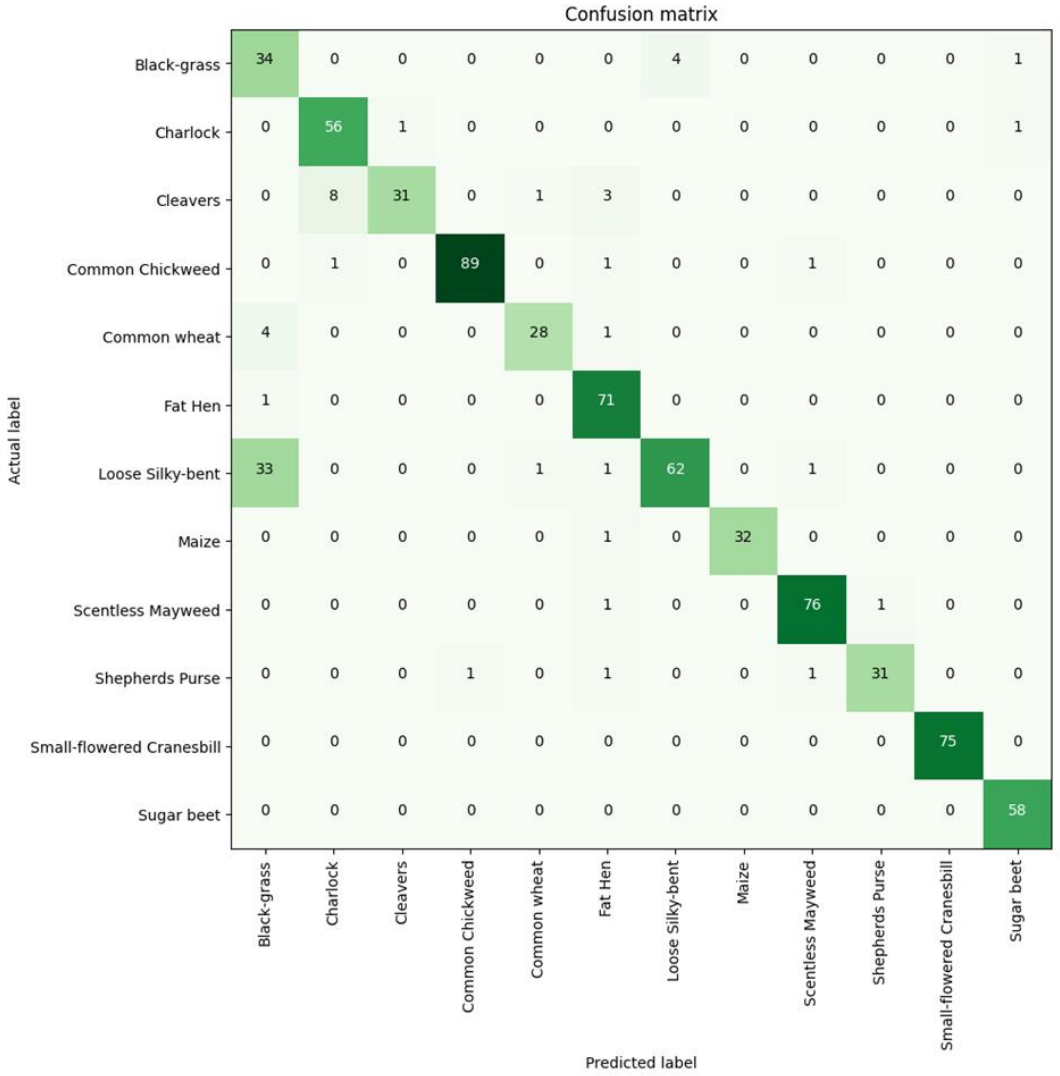


MODEL #1: CNN (contd.)

```
model.evaluate(X_test_new, y_test_new)
print('Test loss:', scores[0])
print('Test accuracy:', scores[1])
```

```
23/23 [==============================] - 1s 30ms/step - loss: 0.2907 - accuracy: 0.9018
Test loss: 0.29073983430862427
Test accuracy: 0.9018232822418213
```

MODEL #1: CNN (contd.)



MODEL #1: CNN (contd.)

```
score, acc = model.evaluate(X_test_new,y_test_new)
score1, acc1 = model.evaluate(X_train,y_train)
print('Test score:', score,'   Test accuracy:', acc)
print('Train score:', score1,'   Train accuracy:',acc1)
```

```
23/23 [==============================] - 0s 18ms/step - loss: 0.2907 - accuracy: 0.9018
104/104 [==============================] - 2s 24ms/step - loss: 0.1579 - accuracy: 0.9320
Test score: 0.290739834308624 27    Test accuracy: 0.9018232822418213
Train score: 0.1579451858997345    Train accuracy: 0.9320300817489624
```

MODEL #1: CNN (contd.)

```
test_images_path = "/wild_data/*.png"
test_images = glob(test_images_path)
test_images_arr = []
test_files = []

for img in test_images:
    i = cv2.resize(cv2.imread(img), (128, 128))
    test_files.append(img.split('/')[-1])
    # Blurred image
    blurr = cv2.GaussianBlur(i,(5,5),0)
    # HSV image
    hsv = cv2.cvtColor(blurr,cv2.COLOR_BGR2HSV)
    #Green Parameters
    sensitivity = 35
    lower  = np.array([60 - sensitivity, 100, 50])
    upper = np.array([60 + sensitivity, 255, 255])
    #Masked image
    mask = cv2.inRange(hsv,lower,upper)
    struc = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(5,5))
    mask = cv2.morphologyEx(mask,cv2.MORPH_CLOSE,struc)
    #Boolean image
    boolean = mask>0
    new = np.zeros_like(i,np.uint8)
    new[boolean] = i[boolean]
    test_images_arr.append(new)

test_X = np.asarray(test_images_arr)
# Normalization of the Image Data
test_X = test_X.astype('float32') / 255
```

- Downloaded another wild seedlings dataset that has no effects applied on it.
- Applied the preprocessing steps on it, and tested one of the images.
- The code outputs the correct class.

```
1 predictions = loaded_model.predict(test_X)
```

```
1/1 [==============================] - 0s 21ms/step
```

```
1 max_idx = np.argmax(predictions)
2 print(max_idx)
```

```
9
```

```
1 labels.classes_[max_idx]
```

```
'Shepherds Purse'
```

MODEL TWO
EFFICIENT NET

MODEL #2: EFFICIENT NET

EFFICIENTNET IS A FAMILY OF CNN ARCHITECTURES DESIGNED TO ACHIEVE STATE-OF-THE-ART

PERFORMANCE

WHILE BEING COMPUTATIONALLY EFFICIENT.

MODEL #2: EFFICIENT NET (contd.)

EFFICIENTNET BALANCES MODEL DEPTH, WIDTH, AND RESOLUTION TO ACHIEVE OPTIMAL PERFORMANCE, USING A COMPOUND SCALING METHOD THAT UNIFORMLY SCALES THESE DIMENSIONS.

THE AUTHORS OBSERVED THAT SIMPLY SCALING UP EXISTING CNN ARCHITECTURES BY INCREASING THE DEPTH, WIDTH, OR INPUT RESOLUTION CAN LEAD TO DIMINISHING RETURNS OR EVEN DEGRADE THE PERFORMANCE.

MODEL #2: EFFICIENT NET (contd.)

EFFICIENTNET FOLLOWS A THREE-STEP PROCESSES

1. Base Network Architecture:
   - The base architecture of EfficientNet is inspired by the widely-used convolutional neural network called MobileNetV2. MobileNetV2 introduces the concept of inverted residual blocks and depthwise separable convolutions, which are computationally efficient operations. EfficientNet adopts this base architecture as its foundation.

2. Compound Scaling:
   - EfficientNet scales the base architecture by applying a compound scaling method to increase the network's depth, width, and resolution simultaneously. The scaling coefficients are determined using a heuristic method that ensures a balanced increase in these dimensions. The authors introduce a scaling parameter called "phi" to control the network's size. A higher phi value leads to a larger and more powerful network, while a lower value results in a smaller and faster network.

MODEL #2: EFFICIENT NET (contd.)

EFFICIENTNET FOLLOWS A THREE-STEP PROCESSES

3. Neural Architecture Search (NAS):
    - The authors of EfficientNet used Neural Architecture Search to find the optimal scaling coefficients for the model.
    - The search process used a large-scale dataset (ImageNet) and a measure called "model efficiency" that considers both accuracy and computational cost.

By following this three-step process, EfficientNet achieves superior performance compared to previous CNN architectures while maintaining computational efficiency. EfficientNet has been shown to outperform other models on various benchmark datasets, such as ImageNet, CIFAR-100, and Pascal VOC, with significantly fewer parameters and computational resources.

MODEL #2: EFFICIENT NET (contd.)

In [6]:

```
# Using a tensorflow ImageDataGenerator for generating the training data
# This will handle the io along with any augmentations to be done to the imag
es

from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import efficientnet

train_generator_eff_net = ImageDataGenerator(
    preprocessing_function = efficientnet.preprocess_input,
    validation_split = 0.1
)

test_generator_eff_net = ImageDataGenerator(
    preprocessing_function = efficientnet.preprocess_input
)
```

- **Data Augmentation:**
  - Horizontal Flipping-Shearing-Scaling-Translation-Rotation-Brightness Shift

MODEL #2: EFFICIENT NET (contd.)

```
train = train_generator_eff_net.flow_from_dataframe(
    dataframe=train_df,
    x_col="Images",
    y_col="Labels",
    target_size=(224, 224),
    color_mode="rgb",
    class_mode="categorical",
    batch_size=32,
    shuffle=True,
    seed=42,
    subset='training',
    rotation_range=32,
    zoom_range=0.2,
    width_shift_range=0.2,
    height_shift_range=0.2,
    sheer_range=0.2,
    brightness_range=[0.5,1.5],
    horizontal_flip=True,
    fill_mode="nearest"
```

We used the train generator to specify the values for each technique

MODEL #2: EFFICIENT NET (contd.)

```
test = test_generator_eff_net.flow_from_dataframe(
    dataframe=test_df,
    x_col="Images",
    y_col="Labels",
    target_size=(224, 224),
    color_mode="rgb",
    class_mode="categorical",
    batch_size=32,
    rotation_range=32,
    zoom_range=0.2,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shuffle=False
)
```

MODEL #2: EFFICIENT NET (contd.)

```
basemodel = efficientnet.EfficientNetB0(
                                        include_top=False,
                                        weights='imagenet',
                                        input_shape=(224,224,3),
                                        pooling='avg',
                                       )
# basemodel.summary()
basemodel.trainable = False
```

```
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import CategoricalCrossentropy
from tensorflow.keras.metrics import CategoricalAccuracy

# building the Predictor layers
x = Dense(256, activation='relu')(basemodel.output)
x = Dense(128, activation='relu')(x)
x = Dropout(0.4)(x)

outputs = Dense(12, activation='softmax')(x)

efficientnet = Model(inputs=basemodel.inputs, outputs=outputs)
```

MODEL #2: EFFICIENT NET (contd.)

EFFICIENTNET MODEL STAGES (BLOCKS)

1. **Initial** convolutional layer, batch normalization, and activation functions. It applies a 3x3 convolution with 32 filters to the input images.
2. **Depthwise** separable convolution, which applies a Depthwise convolution followed by a pointwise convolution. It expands the number of channels and applies batch normalization and activation.
3. **Repeat** the Depthwise separable convolution with increased number of filters and introduces a squeeze-and-excitation (se) module. The SE module helps the network to focus on important channels by learning channel-wise attention weights.
4. **Expand** the network by repeating the Depthwise separable convolution and se module. The number of filters keeps increasing, allowing the network to capture more complex features.

MODEL #2: EFFICIENT NET (contd.)

EFFICIENTNET MODEL STAGES (BLOCKS) (contd.)

5. **Continue** the pattern of Depthwise separable convolution and se module, further expanding the network and increasing the number of filters.
6. **Top Convolutional Layers**: After the last block, a 1x1 convolution is applied to reduce the number of channels to 1920. Batch normalization and activation are applied before the global average pooling layer.
7. **Global Average Pooling**: This layer calculates the average value for each channel across the spatial dimensions, resulting in a fixed-length feature vector.
8. **Top Dropout and Predictions**: A dropout layer is added to prevent overfitting, followed by a fully connected dense layer with 1000 units for classification into 1000 classes (as the model was pretrained on ImageNet).

MODEL #2: EFFICIENT NET (contd.)

```
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import CategoricalCrossentropy
from tensorflow.keras.metrics import CategoricalAccuracy

# building the Predictor layers
x = Dense(256, activation='relu')(basemodel.output)
x = Dense(128, activation='relu')(x)
x = Dropout(0.4)(x)

outputs = Dense(12, activation='softmax')(x)

efficientnet = Model(inputs=basemodel.inputs, outputs=outputs)
```

```
efficientnet.compile(
    optimizer=Adam(),
    loss=CategoricalCrossentropy(),
    metrics=[CategoricalAccuracy()]
)
```

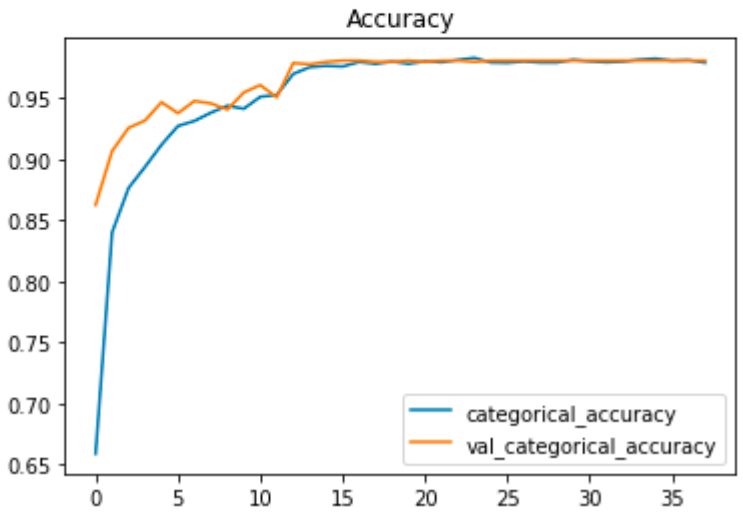
MODEL #2: EFFICIENT NET (contd.)

```
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckpoint

# training
results = efficientnet.fit(
    train,
    validation_data = validation,
    batch_size = 32,
    epochs = 40,
    callbacks = [
        EarlyStopping(
            monitor="val_loss",
            patience=5,
            restore_best_weights=True
        ),
        ReduceLROnPlateau(patience=2),
        ModelCheckpoint(
        str(CHECKPOINTS),
        monitor="val_loss",
        save_best_only=True)
    ]
)
```

MODEL #2: EFFICIENT NET (contd.)

In [13]:

```
pd.DataFrame(results.history)[['categorical_accuracy', 'val_categorical_accuracy']].plot()
plt.title("Accuracy")
plt.show()
```

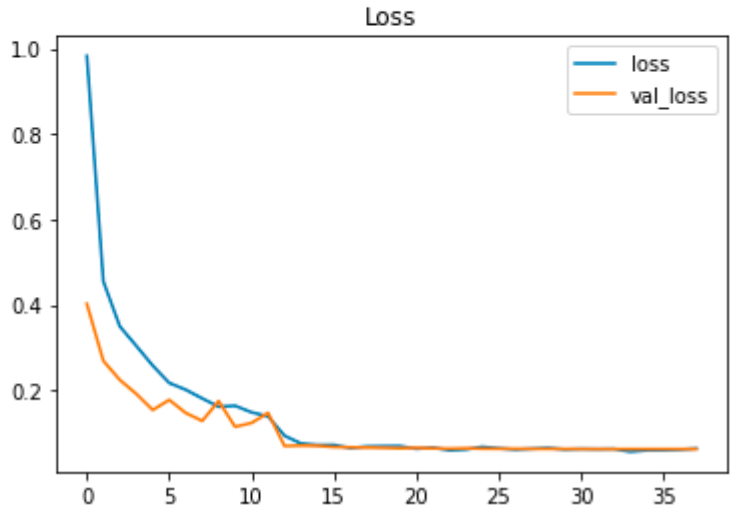


Activate Windo

MODEL #2: EFFICIENT NET (contd.)

In [14]:

```
pd.DataFrame(results.history)[['loss', 'val_loss']].plot()
plt.title("Loss")
plt.show()
```



MODEL #2: EFFICIENT NET (contd.)

Model Evaluation

In [17]:

```
# Testing the model
results = efficientnet.evaluate(test)
```

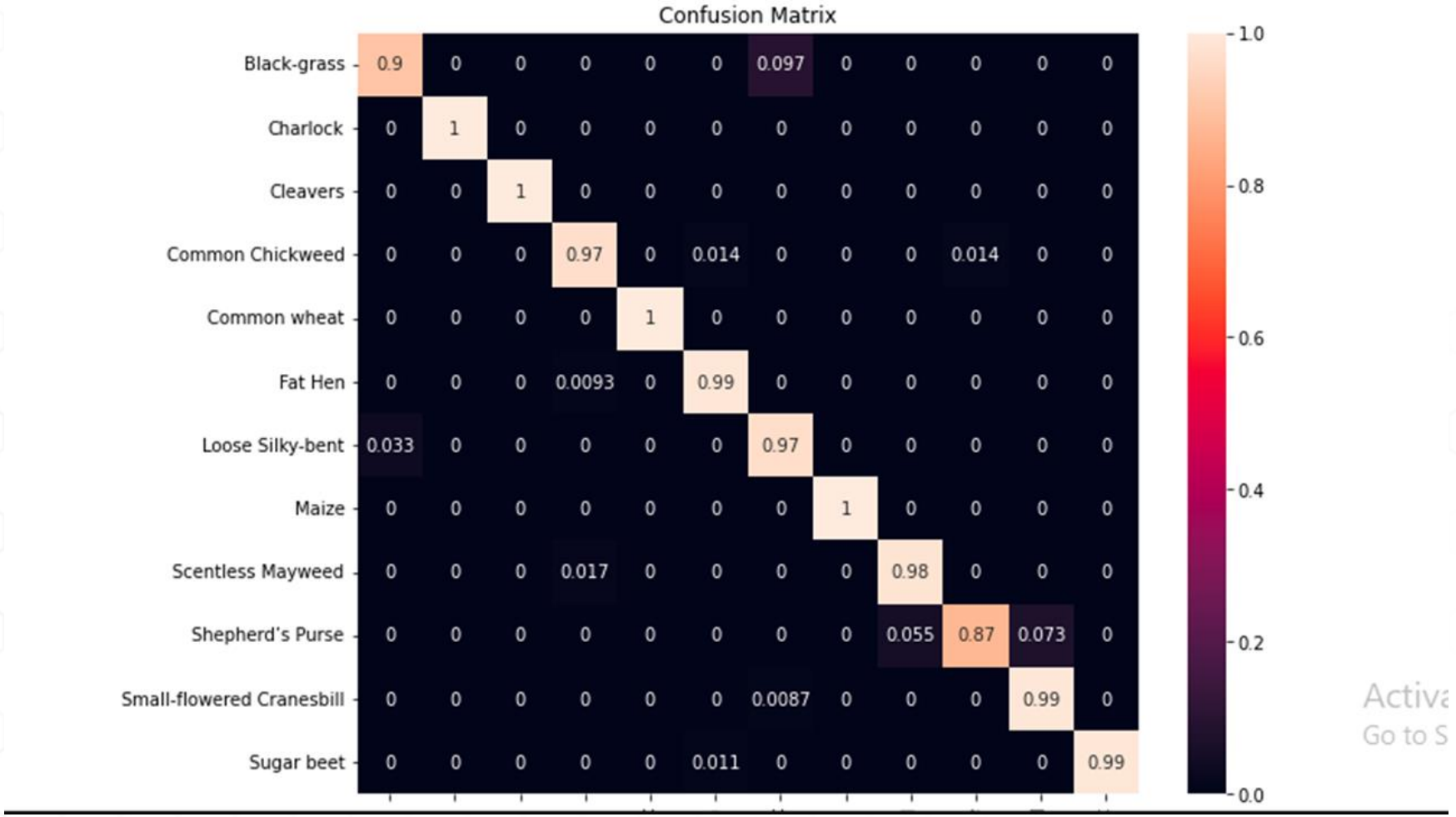
```
35/35 [==============================] - 20s 569ms/step - loss: 0.0706 - categorical_accuracy: 0.9
756
```

MODEL #2: EFFICIENT NET (contd.)

	precision	recall	f1-score	support
Black-grass	0.92	0.90	0.91	62
Charlock	1.00	1.00	1.00	90
Cleavers	1.00	1.00	1.00	67
Common Chickweed	0.98	0.97	0.98	143
Common wheat	1.00	1.00	1.00	51
Fat Hen	0.97	0.99	0.98	108
Loose Silky-bent	0.95	0.97	0.96	152
Maize	1.00	1.00	1.00	51
Scentless Mayweed	0.98	0.98	0.98	121
Shepherd's Purse	0.96	0.87	0.91	55
Small-flowered Cranesbill	0.97	0.99	0.98	115
Sugar beet	1.00	0.99	0.99	93
accuracy			0.98	1108
macro avg	0.98	0.97	0.97	1108
weighted avg	0.98	0.98	0.98	1108

Activa
Go to S

MODEL #2: EFFICIENT NET (contd.)



MODEL #2: EFFICIENT NET (contd.)

SHEPHERDS PURSE

```
In [39]: predictions = loaded_model.predict(test_predict)

In [43]: np.argmax(predictions)

Out[43]: 9

In [ ]:
```

CLASS (9) = Correct

MODELS COMPARISON

EFFICIENTNET	CNN
• Uses EfficientNetb0 as the base model, which is a pre-trained CNN architecture designed to achieve high performance while being computationally efficient.	• CNN architecture built from scratch.
• Has 4 layers (3 dense and 1 dropout).	• Has 13 layers (7 conv2d, 3 dense, and 3 dropout).
• Uses global average pooling to generate a fixed-length feature vector.	• Uses global max pooling and flattening.
• Has 12 output classes.	• Has 12 output classes
• Uses the ADAM optimizer.	• CNN model uses the categorical cross-entropy loss function and the ADAM optimizer.
• Uses transfer learning and fine-tunes the pre-trained EfficientNetb0 model by adding a few custom layers on top.	• Stand-alone architecture that is trained from scratch.
• Uses a pre-trained model that is specifically designed to be computationally efficient	• Has more layers and may be more computationally expensive.
• Has a fixed input size of (224, 224, 3)	• Has a fixed input size of (128, 128, 3).
• Uses the "avg" pooling method to generate a fixed-length feature vector	• Uses the "max" pooling method.
• Has fewer parameters than CNN model, as it only adds a few custom layers on top of the pre-trained efficientnetb0 model	• Has more layers and more parameters to learn.

THANK YOU!

PREPARED BY:
ROLA HOSSAM
MUHAMMAD NASSER
FAWZIA WAGEEH

FIND NOTEBOOKS AT
https://github.com/MuhammadNYoussef/plant-seedlings-classification-cv