CODE EXPLANATION GUIDE

Smart AGRI

Team Captain: Shivam Gohri
Team: UIET Chandigarh_intelliJ

PEST DETECTION

Besides, minimizing the use of pesticides, Early Pest Detection can prevent significant amount of loss of crops and ensure high quality yield.
The techniques of Machine Vision and Image Processing can be effectively used for the same.
The images of leaves from the crop fields are taken and then processed to interpret the image contents by image processing methods.

PROPOSED METHODOLOGY:

- **IMAGE CAPTURING**
- **IMAGE PROCESSING**
- **FEATURE EXTRACTION**

Image Processing essentially consists of following steps:

IMAGE PROCESSING

- RBG TO GRAYSCALE
- FILTERING THE IMAGE
- IMAGE SEGMENTATION

1. RGB TO GRAY SCALE CONVERSION

RGB images require large storage space and takes more time to process hence RGB images captured by the camera are converted to gray scale images so they can be handled easily. The following equation shows how RGB images are converted to gray scale images.

*I(x,y)=0.2989B +0.5870G +0.1140*B*

```
def conversion():
    image = cv2.imread('/content/drive/My Drive/input image.PNG')
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    cv2.imwrite('gray_image.png',gray_image)
    cv2_imshow(image)
    cv2_imshow(gray_image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

2. FILTERING THE IMAGE

Filtering is nothing but eliminating the unwanted portion of the image, different types of filters are available we used Gaussian filter. Gray image is blurred using Gaussian Filter, it is a linear filter used to reduce the noise and blur the edges and reduce contrast.

```
def gaussian():
    image = cv2.imread('gray_image.png')
    cv2.getGaussianKernel(9,9)
    blur= cv2.GaussianBlur(image,(5,5),0)
    cv2.imwrite('blur.png',blur)
    cv2_imshow(blur)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

Input Image

Gray scale image

Blurred Image

3. IMAGE SEGMENTATION

To detect the pests from the images, the image background is calculated using morphological operators, so the resulting image will only have the objects with pixel values 1 and background pixel values 0.
Noise contains dew drops, dust and other visible parts of leaves. As only the object of interest was to be visible on the images, so the aim was to remove the noise to get better and effective results. The Erosion algorithm has been used to remove isolated noisy pixels and to smoothen object boundaries .

```
def segmentation():
    image = cv2.imread('blur.png')
    gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
    ret, thresh = cv2.threshold(gray,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
    cv2.imwrite('thresh_image.jpg',thresh)
    cv2_imshow(thresh)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

    kernel = np.ones((3,3),np.uint8)
    opening = cv2.morphologyEx(thresh,cv2.MORPH_OPEN,kernel, iterations = 3)
    cv2_imshow(thresh)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

After noise removal, the next goal was to enhance the detected pests after segmentation which was performed by using the dilation algorithm. Next goal is to count the number of pests on a leaf, for this label function of ndimage library is used, this function considers any non zero value as object and any 0 as background ane returns the number of objects in the input array.

```
sure_bg = cv2.dilate(opening,kernel,iterations=3)
cv2_imshow(sure_bg)
cv2.waitKey(0)
cv2.destroyAllWindows()
print ("No. of pests in the image: ")
labelarray, particle_count = ndimage.measurements.label(sure_bg)
print (particle_count)
return particle_count
```

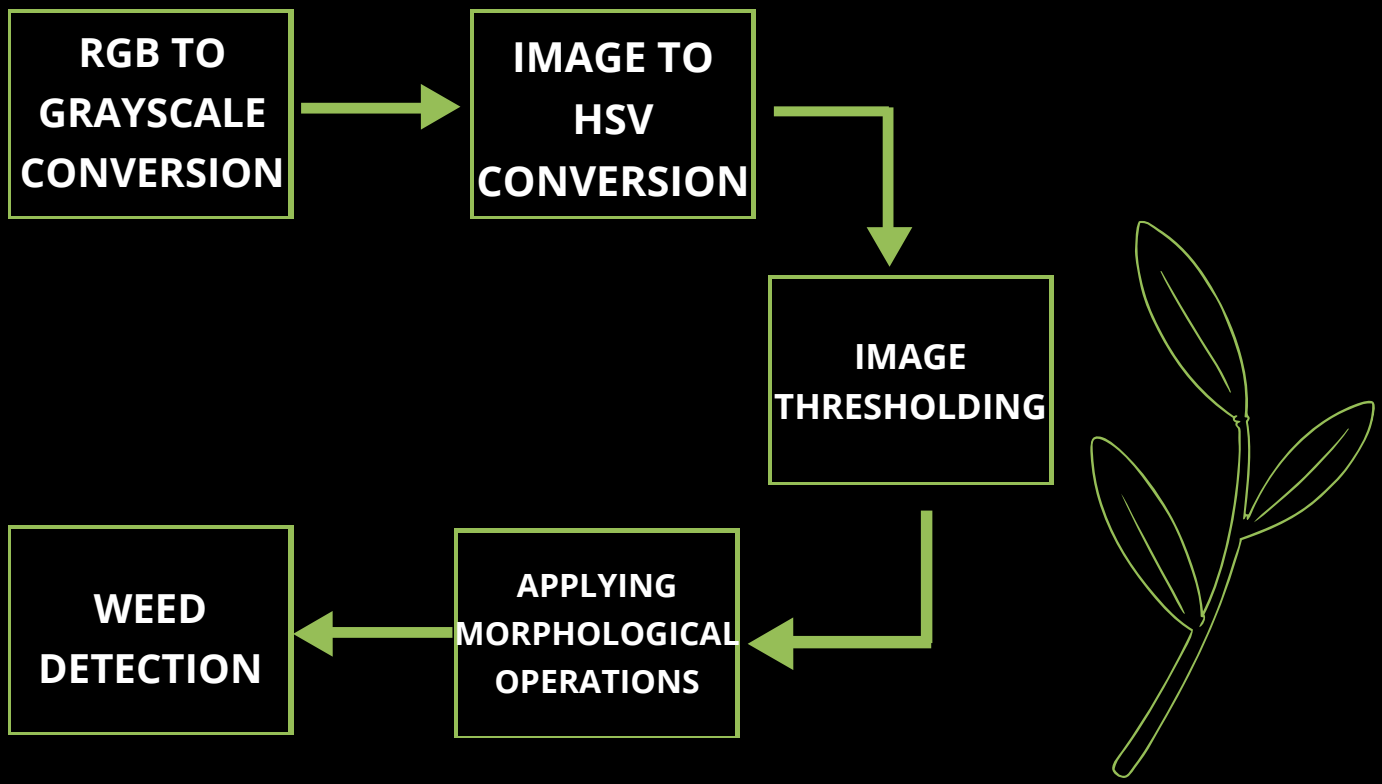
Image after erosion

Image after dilation

WEED DETECTION

- Similar to Pest Detection, Machine Vision and Image Processing can be effectively used for Weed Detection and elimination as well.
- The images of leaves from the crop fields are taken and then processed to interpret the image contents by image processing methods.
- Input of Weed areas is given to automatic weed elimination system which works on See and Spray Technology. For this, separate nozzles for spraying weedicides are paired with Surface Drip Irrigation (SDI) systems leading to a significant reduction in the amount of weedicides, hardware, chemical content in plants thereby resulting in efficient weeding..

ALGORITHM:



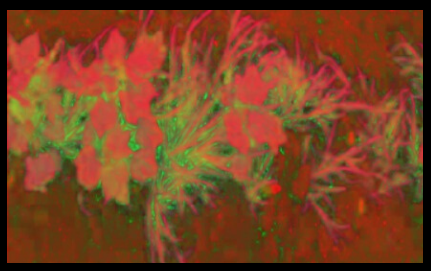
1. RGB TO GRAY SCALE CONVERSION

RGB images require large storage space and takes more time to process hence RGB images captured by the camera are converted to gray scale images so they can be handled easily. The following equation shows how RGB images are converted to gray scale images.
*I(x,y)=0.2989B +0.5870G +0.1140*B*

```
img = cv2.imread('/content/drive/My Drive/weed3.PNG')
cv2_imshow(img)
gray_image = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
cv2_imshow(gray_image)
```

Input Image

Gray scale image



HSV Image

2. IMAGE TO HSV CONVERSION

The R, G, and B components of an object's color in a digital image are all correlated with the amount of light hitting the object, and therefore with each other, image descriptions in terms of those components make object discrimination difficult, hence, image is converted to HSV format. 'Hue' represents the color, 'saturation' represents the amount to which that respective color is mixed with white and 'value' represents the amount to which that respective color is mixed with black.HSV is used to separate image luminance from color information, hence it is the most suitable color space for color based image segmentation.

```
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
cv2_imshow(hsv)
```

3. IMAGE THRESHOLDING

We convert our hsv image to a binary image, this is done to select the area of interest, any pixel having value greater than the threshold is set as 1 and with value less than threshold is set as 0.

```
mask = cv2.inRange(hsv,(36, 0, 0), (86, 255, 255) )
cv2_imshow(mask)
arr = numpy.ones([5,5],numpy.uint8)
```

4. APPLYING MORPHOLOGICAL OPERATIONS

Morphological operation i.e, erosion is applied on the binary image to eliminate the irrelevent or hidden details from the image. Dilation is then applied on the eroded image to create an enhanced version of the eroded image.

```
img_erosion = cv2.erode(mask, arr, iterations=1)
cv2_imshow(img_erosion)
img_dilation = cv2.dilate(img_erosion, arr, iterations=1)
n_white_pix = numpy.sum(img_dilation == 255)
```

5. WEED DETECTION

Number of white pixels in the dilated image is calculated and are compared with a fixed threshold value, if they number of white pixels are more than threshold value that means weed is present.

```
n_white_pix = numpy.sum(img_dilation == 255)
print('Number of white pixels:', n_white_pix)
if n_white_pix >= 10000:
    asd=1
else:
    asd=0
```

Image in binary format

Image after erosion

Image after dialation

CROP DISEASE DETECTION

- With the use of Deep Convolutional Networks, a plant disease recognition model can be developed based on leaf image classification.

- Neural Networks, with their outstanding ability to derive meaning from complex or imperfect data, can be applied for extracting patterns and detecting trends that are too difficult to notice by humans or computer techniques.

DATASET DESCRIPTION

The dataset consisted of 20,639 images of healthy and unhealthy pepper bell, potato and tomato plants divided into 15 classes of classified on the basis of different plant diseases.

- Pepper__bell___Bacterial_spot
- Pepper__bell___healthy
- Potato___Early_blight
- Potato___healthy
- Potato___Late_blight
- Tomato__Target_Spot
- Tomato__Tomato_mosaic_virus
- Tomato__Tomato_YellowLeaf__Curl_Virus
- Tomato_Bacterial_spot
- Tomato_Early_blight
- Tomato_healthy
- Tomato_Late_blight
- Tomato_Leaf_Mold
- Tomato_Septoria_leaf_spot
- Tomato_Spider_mites_Two_spotted_spider_mite

Potato healthy

Potato Early Blight

Pepper bell bacterial spot

Pepper bell healthy

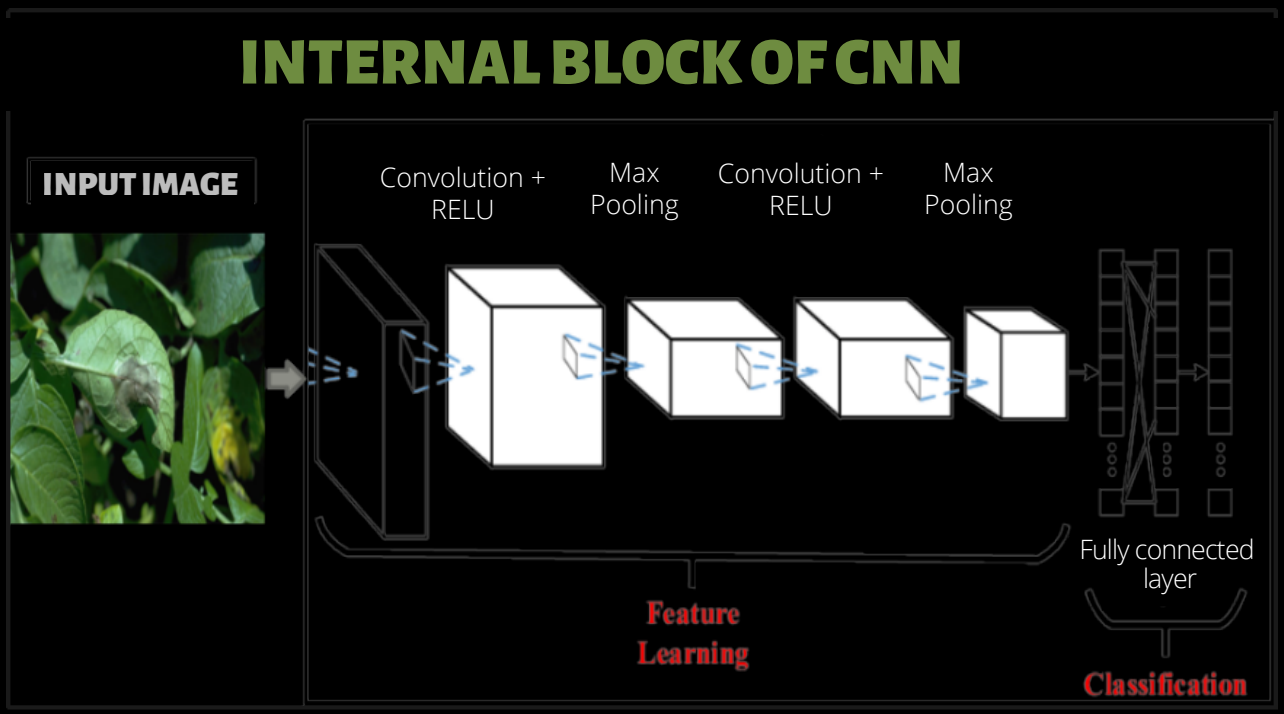
Tomato mosaic virus

Tomato healthy

DEEP LEARNING MODEL USED:

Convolution Neural Networks (CNN) deep learning algorithm was used, it is well known for its widely used in applications of image and video recognition and also in recommender systems and Natural Language Processing(NLP) Main steps to build a CNN:

1. Convolution operation
2. ReLU layer
3. Pooling layer
4. Flattening
5. Fully connected layer



CODE:

READING AND PREPROCESSING THE DATA:

Data is read from the folder and images are added to image list and plant folder is added to the label list

```
directory_root = "/content/drive/My Drive/PlantVillage"
print(len(listdir(directory_root)))
for image in plant_disease_folder_list:
     image_directory = f"{directory_root}/{plant_folder}/{image}"
     if image_directory.endswith(".jpg") == True or image_directory.endswith(".JPG") ==
True:       image_list.append(image_directory)
            label_list.append(plant_folder)
img_info = pd.DataFrame({'image_path':image_list,'label':label_list})
print(img_info.head())
```

PRE PROCESSING THE RAW DATA INTO USABLE FORMAT:

Numeric labels starting from 0 are assigned to the species, images are shuffled so that the test/train/validation data represent the overall distribution of data. Images are resized to make all the images of the same size as CNN accepts input of the same size.

```
#new column (empty)
img_info['labels_integer'] = None
#index of new column
index_labels_integer = img_info.columns.get_loc("labels_integer")
#index of species column
index_species = img_info.columns.get_loc("label")
#to assign numeric labels starting with 0 for the first species
k = 0
for i in range(len(img_info)):
    if i == 0:
        img_info.iloc[i, index_labels_integer] = k #here, k == 0
    if i > 0:
        if img_info.iloc[i-1, index_species] == img_info.iloc[i, index_species]:
            img_info.iloc[i, index_labels_integer] = k
        else:
            k += 1
            img_info.iloc[i, index_labels_integer] = k
img_info = shuffle(img_info)
list_vectors = []

for image_path in img_info.image_path:
    #read as rgb array
    img = Image.open(image_path)
    size = (64, 64)
    img = img.resize(size, PIL.Image.ANTIALIAS)
    img_array = np.array(img)
    #append image vector to list
    list_vectors.append(img_array)
```

NORMALIZING THE DATA:

Data is normalized in the range [0,1], this is done to ensure that each input parameter (pixel in this case) has a similar distribution ([0,1], in thins case). This makes convergence while training the network.

```
X = np.stack((list_vectors))
Y =  img_info['labels_integer']
X = X/255
Y_one_hot = keras.utils.to_categorical(Y, num_classes=15)
np.savez("x_images_arrayscnn", X)
np.savez("y_numeric_labelscnn", Y_one_hot)
x_npz = np.load("x_images_arrayscnn.npz")
X = x_npz['arr_0']

y_npz = np.load("y_numeric_labelscnn.npz")
Y_one_hot = y_npz['arr_0']
```

SPLITTING THE DATA FOR TRAINING TESTING AND VALIDATION:

80% data is used for training, that is, model is trained on this data in order to be generalized to other data later on. 10% data is used as testing data, predictions are made on this data. 10% of data is used to cross validate to get more accurate results.

```
split_train = 0.8 #train 0.8, validate 0.1, test 0.1
split_val = 0.9
index_train = int(split_train*len(X))
index_val = int(split_val*len(X))

X_train = X[:index_train]
X_val = X[index_train:index_val]
X_test = X[index_val:]

Y_train = Y_one_hot[:index_train]
Y_val = Y_one_hot[index_train:index_val]
Y_test = Y_one_hot[index_val:]
print(X_train.shape, X_val.shape, X_test.shape, Y_train.shape, Y_val.shape, Y_test.shape)
```

BUILDING THE CNN MODEL:

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(5, 5), strides=(1, 1), input_shape=input_shape))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Conv2D(64, (5, 5)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.7))
model.add(Flatten())
model.add(Dense(1000))
model.add(Activation('softmax'))
model.add(Dropout(0.7))
model.add(Dense(num_classes, activation='softmax'))
```

32 features should be extracted from input image

ReLU-Rectified linear activation to fire up the neurons

Flatten layer 4D array to 1D array

Dropout makes some neurons to relax

Softmax activation for probability outputs

CNN shrinks the parameters and learns features and stores valuable information output shape is decreasing after every layer.

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 60, 60, 32)        2432
_________________________________________________________________
activation_1 (Activation)    (None, 60, 60, 32)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 30, 30, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 26, 26, 64)        51264
_________________________________________________________________
activation_2 (Activation)    (None, 26, 26, 64)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 13, 13, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 13, 13, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 10816)             0
_________________________________________________________________
dense_1 (Dense)              (None, 1000)              10817000
_________________________________________________________________
activation_3 (Activation)    (None, 1000)              0
_________________________________________________________________
dropout_2 (Dropout)          (None, 1000)              0
_________________________________________________________________
dense_2 (Dense)              (None, 15)                15015
=================================================================
Total params: 10,885,711
Trainable params: 10,885,711
```

START TRAINING CNN WITH PARAMETERS:

```
model.compile(optimizer=keras.optimizers.Adam(lr=0.0003, beta_1=0.9, beta_2=0.999, epsilon=None, decay=1e-8,
amsgrad=False),loss='categorical_crossentropy',metrics=['accuracy'])
```

Adam optimizer is used with learning rate 0.0003, loss function 'categorical_crossentropy' is used. CNN model is trained using the validation data parameter with batch size 200 and 200 epochs.

```
results = model.fit(X_train, Y_train, epochs=200, batch_size=200, validation_data=(X_val,
Model evaluate(X_test, Y_test)
```

```
Epoch 194/200
 - 1s - loss: 2.4053 - acc: 0.8438 - val_loss: 2.4092 - val_acc: 0.9167
Epoch 195/200
 - 1s - loss: 2.4598 - acc: 0.8438 - val_loss: 2.4082 - val_acc: 0.9167
Epoch 196/200
 - 1s - loss: 2.3904 - acc: 0.8438 - val_loss: 2.4072 - val_acc: 0.9167
Epoch 197/200
 - 1s - loss: 2.4197 - acc: 0.8438 - val_loss: 2.4062 - val_acc: 0.9167
Epoch 198/200
 - 1s - loss: 2.4016 - acc: 0.8438 - val_loss: 2.4051 - val_acc: 0.9167
Epoch 199/200
 - 1s - loss: 2.3940 - acc: 0.8438 - val_loss: 2.4041 - val_acc: 0.9167
Epoch 200/200
 - 1s - loss: 2.4242 - acc: 0.8438 - val_loss: 2.4031 - val_acc: 0.9167
12/12 [==============================] - 0s 4ms/step
```

LINK TO DATASET :

bit.ly/intellij_dataset

LINK TO ALL CODES :

bit.ly/intellij_codes

LINK TO Guide Books :

bit.ly/intellij_guides

Thank you