```
# Real-Time Crop Growth Forecasting Using Thermal and Visual Imaging

import cv2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from keras.utils import to_categorical
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
import tensorflow as tf
from google.colab import drive

# Mount Google Drive
drive.mount('/content/drive')

# Path to the image folder on Google Drive (Change 'your-folder-path'
to your actual folder path)
image_folder = '/content/drive/MyDrive/images/'
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

```
base_path = '/content/drive/MyDrive/images/'
print(os.listdir(base_path))
```

```
['leaf spot', 'hispa', 'Blast', 'leaf folder', 'BLB', 'healthy']
```

```
# Load and preprocess image dataset
def load_images_from_folder(folder, label):

    images = []
    labels = []
    for filename in os.listdir(folder):
        img_path = os.path.join(folder, filename)
        img = cv2.imread(img_path)
        if img is not None:
            img = cv2.resize(img, (128, 128))
            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            img = preprocess_image(img)
            images.append(img)
            labels.append(label)
    return images, labels
```

```
# Advanced image preprocessing techniques
def preprocess_image(img):
    img = gaussian_denoise(img)
    img = enhance_contrast(img)
    img = normalize_image(img)
    return img

# Gaussian noise reduction
def gaussian_denoise(img):
    return cv2.GaussianBlur(img, (5, 5), 0)

def enhance_contrast(img):
    lab = cv2.cvtColor(img, cv2.COLOR_RGB2LAB)
    l, a, b = cv2.split(lab)
    clahe = cv2.createCLAHE(clipLimit=3.0, tileGridSize=(8,8))
    cl = clahe.apply(l)
    limg = cv2.merge((cl,a,b))
    return cv2.cvtColor(limg, cv2.COLOR_LAB2RGB)

def normalize_image(img):
    return img / 255.0

# ORB feature detection
orb = cv2.ORB_create()
def detect_orb_features(image):
    gray = cv2.cvtColor((image * 255).astype(np.uint8),
cv2.COLOR_RGB2GRAY)
    keypoints, descriptors = orb.detectAndCompute(gray, None)
    return keypoints, descriptors

def visualize_orb(image):
    keypoints, _ = detect_orb_features(image)
    orb_img = cv2.drawKeypoints((image * 255).astype(np.uint8),
keypoints, None, color=(0,255,0), flags=0)
    plt.figure(figsize=(6,6))
    plt.title("ORB Feature Detection")
    plt.imshow(orb_img)
    plt.axis('off')
    plt.show()

image_folder = '/content/drive/MyDrive/images/'
if os.path.exists(image_folder):
    print("Folder exists")
    print(os.listdir(image_folder))  # This will list the contents of
the folder
else:
    print("Folder does not exist")
```

```
Folder exists
['leaf spot', 'hispa', 'Blast', 'leaf folder', 'BLB', 'healthy']
```

```
# Assuming subfolders are named by class/label
image_folder = '/content/drive/MyDrive/images/'
all_images = []
all_labels = []

# Ensure the path is correct
for label in os.listdir(image_folder):
    folder_path = os.path.join(image_folder, label)
    if os.path.isdir(folder_path):  # Check if it’s a folder
        imgs, lbls = load_images_from_folder(folder_path, label)
        all_images.extend(imgs)
        all_labels.extend(lbls)

X = np.array(all_images)
le = LabelEncoder()
y = le.fit_transform(all_labels)
y_cat = to_categorical(y)

X_train, X_test, y_train, y_test = train_test_split(X, y_cat,
test_size=0.2, random_state=42)

# CNN Model 1
model_cnn = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(128,128,3)),
    MaxPooling2D(2,2),
    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D(2,2),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(len(np.unique(y)), activation='softmax')
])
model_cnn.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
model_cnn.fit(X_train, y_train, epochs=10, validation_data=(X_test,
y_test))
```

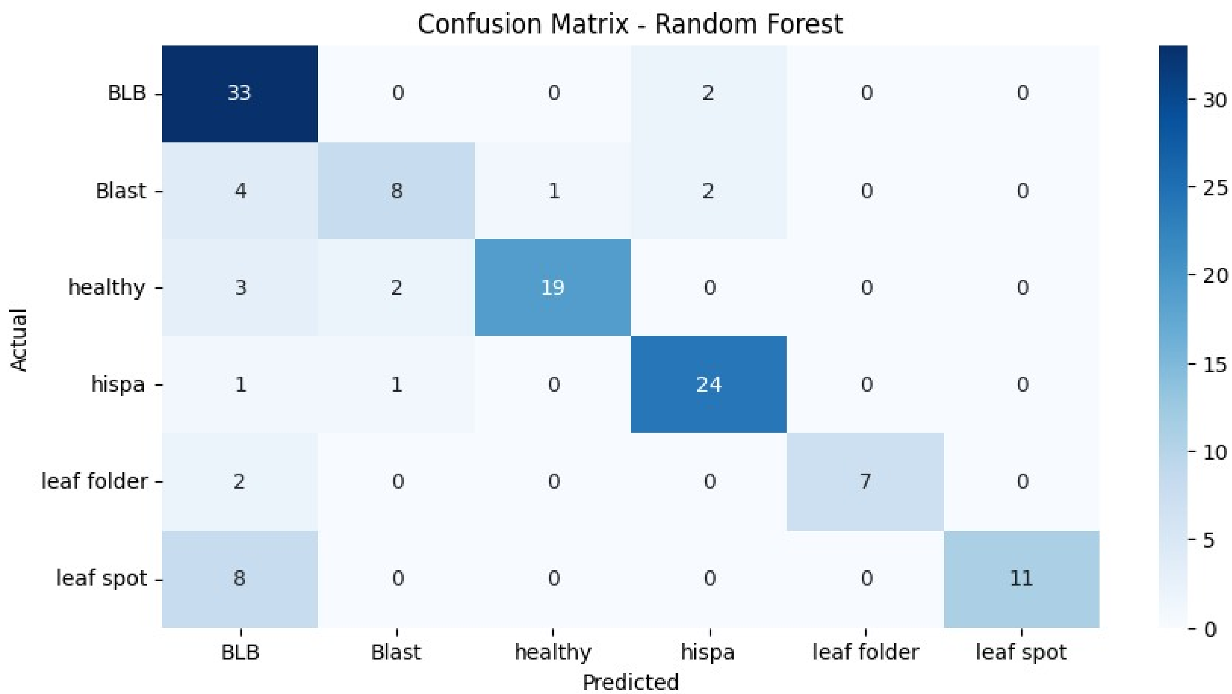
```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/
convolutional/base_conv.py:107: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)

Epoch 1/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 17s 860ms/step - accuracy: 0.3247 - loss:
2.4086 - val_accuracy: 0.3828 - val_loss: 1.5580
Epoch 2/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 13s 813ms/step - accuracy: 0.4818 - loss:
```

```
1.3451 - val_accuracy: 0.5078 - val_loss: 1.1360
Epoch 3/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 21s 908ms/step - accuracy: 0.6140 - loss:
1.0394 - val_accuracy: 0.6641 - val_loss: 0.9467
Epoch 4/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 20s 844ms/step - accuracy: 0.6527 - loss:
0.9096 - val_accuracy: 0.6953 - val_loss: 0.8045
Epoch 5/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 21s 914ms/step - accuracy: 0.7170 - loss:
0.8150 - val_accuracy: 0.7188 - val_loss: 0.7726
Epoch 6/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 20s 864ms/step - accuracy: 0.7140 - loss:
0.7622 - val_accuracy: 0.7734 - val_loss: 0.6828
Epoch 7/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 20s 891ms/step - accuracy: 0.7674 - loss:
0.5937 - val_accuracy: 0.7812 - val_loss: 0.7523
Epoch 8/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 20s 886ms/step - accuracy: 0.7962 - loss:
0.5664 - val_accuracy: 0.7656 - val_loss: 0.6997
Epoch 9/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 21s 910ms/step - accuracy: 0.8359 - loss:
0.4226 - val_accuracy: 0.7422 - val_loss: 0.7425
Epoch 10/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 21s 913ms/step - accuracy: 0.8456 - loss:
0.4354 - val_accuracy: 0.7109 - val_loss: 0.8614

<keras.src.callbacks.history.History at 0x7cb4ee1485d0>
```

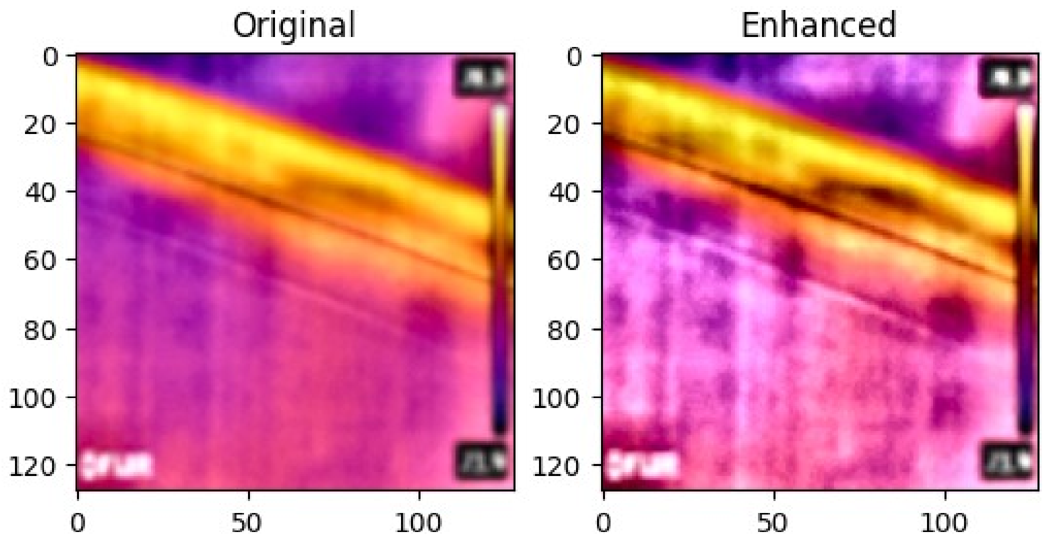
```
# Model 2: SVM with flattened image features
X_flat = X.reshape(len(X), -1)
Xf_train, Xf_test, yf_train, yf_test = train_test_split(X_flat, y,
test_size=0.2, random_state=42)
model_svm = SVC(kernel='linear')
model_svm.fit(Xf_train, yf_train)
yf_pred_svm = model_svm.predict(Xf_test)

# Model 3: Random Forest with flattened image features
model_rf = RandomForestClassifier(n_estimators=100, random_state=42)
model_rf.fit(Xf_train, yf_train)
yf_pred_rf = model_rf.predict(Xf_test)

# Evaluation
print("CNN Model Accuracy:", model_cnn.evaluate(X_test, y_test)[1])
print("SVM Accuracy:", accuracy_score(yf_test, yf_pred_svm))
print("Random Forest Accuracy:", accuracy_score(yf_test, yf_pred_rf))
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 211ms/step - accuracy: 0.7292 - loss:
0.7252
CNN Model Accuracy: 0.7109375
```

```
SVM Accuracy: 0.828125
Random Forest Accuracy: 0.796875

# Visualization
plt.figure(figsize=(10,5))
sns.heatmap(confusion_matrix(yf_test, yf_pred_rf), annot=True,
fmt='d', cmap='Blues', xticklabels=le.classes_,
yticklabels=le.classes_)
plt.title("Confusion Matrix - Random Forest")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```




```
# Example OpenCV enhancement visualization
sample_img = (X[0] * 255).astype(np.uint8)
enhanced = enhance_contrast(sample_img)
plt.subplot(1,2,1)
plt.title("Original")
plt.imshow(sample_img)
plt.subplot(1,2,2)
plt.title("Enhanced")
plt.imshow(enhanced)
plt.show()
```




```
# Feature detection visualization
visualize_orb(X[0])
```