ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO

BÀI TẬP MÔN NHẬN DẠNG

NHẬN DẠNG MẪU

Người thực hiện 20120547 – Võ Thành Phong

Thành phố Hồ Chí Minh – 4/2023

MỤC LỤC

I. Thông tin cá nhân:

Sinh viên: Võ Thành Phong

Mã số sinh viên: 20120547

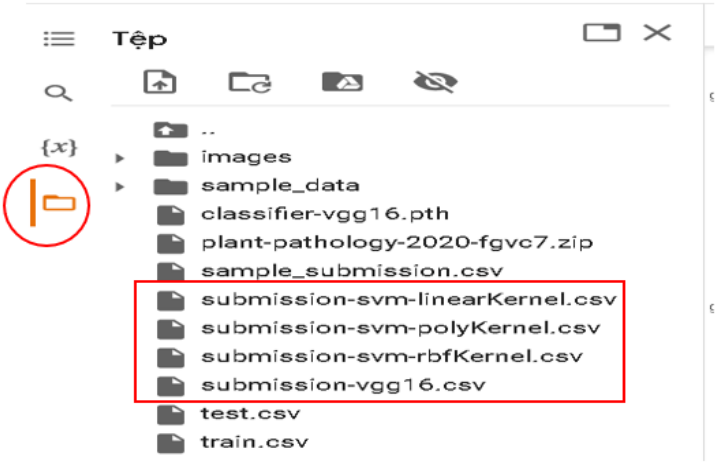
Email: 20120547@student.hcmus.edu.vn

II. Yêu cầu kĩ thuật:

- File Source code ở dạng .ipynb được khuyến khích chạy trên môi trường **google colab.**

- Do có sử dụng mô hình deep learning xử lý dữ liệu dạng hình ảnh, nên số lượng tham số rất lớn và thời gian training sẽ có thể tiêu tốn nhiều thời gian, do đó khi chạy file .ipynb trên google colab **cần liên kết thời gian chạy với GPU** của google colab cho việc training mô hình deep learning.

- Do tránh tình trạng đường dẫn cứng (đường dẫn tĩnh) trong file notebook nên việc tải file và xuất file, tải thư viện sẽ được chạy cục bộ trên phiên làm việc của notebook trên google colab, do đó sau khi chạy xong foile notebook trên google colab thì **trước khi tắt file google colab hãy vào mục folder bên phải để tải các file submission về máy, do khi tắt file notebook khỏi google colab thì lần mở lại sau sẽ không còn file đã xuất ra nữa mà phải chạy lại từ đầu file notebook đó một lần nữa trên google colab.**



III. Giới thiệu bài toán:

- Sự sinh trưởng và phát triển của cây trồng phụ thuộc vào giống cây, độ dinh dưỡng, các điều kiện thời tiết và nhiều yếu tố khác. Bên cạnh đó sâu bệnh cũng ảnh hưởng trực tiếp đến năng suất và có khả năng lây lan trên diện rộng. Đối với các loại cây trồng việc phát hiện sâu bệnh và xử lý kịp thời rất quan trọng.

- Khi quy mô canh tác ở mức lớn hay rất lớn, mắt người rất khó để phát hiện các dấu hiệu của sâu bệnh trong giai đoạn phát triển ban đầu dẫn đến sâu bệnh phát triển mạnh làm ảnh hưởng đến chất lượng sản phẩm. Việc không phát hiện bệnh kịp thời hoặc chuẩn đoán sai loại bệnh cho cây trồng không những không cứu được cây trồng mà còn phát sinh những chi phí dư thừa, không hiệu quả.

- Mục tiêu chính của bài toán là phát triển các mô hình dựa trên máy học để phân loại chính xác một hình ảnh lá nhất định từ bộ dữ liệu thử nghiệm cho một loại bệnh cụ thể và xác định một bệnh riêng lẻ từ nhiều triệu chứng bệnh trên một hình ảnh lá đơn lẻ.

IV. Giới thiệu và khám phá bộ dữ liệu:

- Bộ dữ liệu có tên là 'plant-pathology'.

- Bộ dữ liệu chứa các ảnh về các loại bệnh đang có trên cây trồng, cụ thể gồm có tập dữ liệu hình ảnh về các loại bệnh trên cây và tập dữ liệu đã được xác định được loại bệnh (train.csv và test.csv) bao gồm 4 loại: Combinations, Healthy, Rust, Scab.

- Khi giải nén từ định dạng .zip bộ dữ liệu bao gồm:

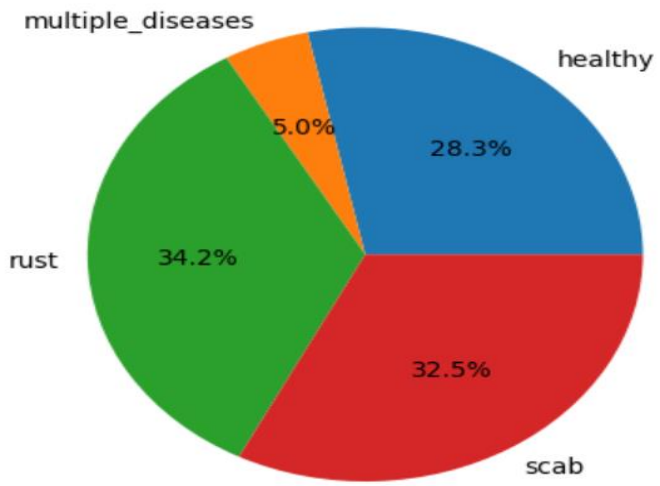
  + 1 folder images: tập hợp các ảnh lá cây ở định dạng .jpg, cho cả tập train và tập test.

  + 3 files csv: 'sample_submission.csv': mẫu file cho việc xuất kết quả; 'train.csv': file nhãn cho việc train dữ liệu; 'test.csv': file chứa ID của các ảnh để chuẩn đoán cho tập test.

- Số lượng các loại bệnh giảm dần theo thứ tự từ trái sang phải như sau: rust, scab, healthy, multiple_diseases.

- Minh họa:

```
#vẽ pie chart thống kê phần trăm của mỗi lớp
count_by_attr.plot(kind='pie',autopct='%1.1f%%', ylabel='');
```



V. Tổng quan quá trình thiết kế mô hình

- Các mô hình để nhận dạng trên bộ dữ liệu cây trồng plant-pathology của em sẽ bao gồm 2 mô hình như sau:

  + Xây dựng bộ trích xuất đặc trưng (feature extractor) kết hợp bộ phân lớp (classifier) SVM.

  + Mô hình mạng CNN mà cụ thể là kiến trúc VGG16.

- Quá trình xây dựng 2 mô hình này được chia làm 3 giai đoạn:

+ Xây dựng bộ trích xuất đặc trưng (feature extractor) kết hợp bộ phân lớp (classifier) Support Vector Machine (SVM).

+ Cải tiến bộ trích xuất đặc trưng và SVM.

+ Xây dựng kiến trúc VGG16.

VI. Xây dựng Feature Extractor và Classifier

1. Tổng quan

- Thông thường, đầu vào của các mô hình học máy cổ điển bao gồm cả support vector machine là một vector các đặc trưng (vector số) đại diện cho thông tin của một mẫu trong bộ huấn luyện.

- Khi làm việc với dữ liệu dạng ảnh, mỗi ảnh được đại diện bởi một ma trận các pixel (gọi là ma trận nếu ảnh là ảnh xám chỉ có 1 kênh màu) hay tensor các pixel (gọi là tensor nếu ảnh là ảnh màu là sự tập hợp của nhiều kênh màu đơn lẻ). Do đó trước khi đi vào một mô hình học máy nói chung hay SVM nói riêng thì ảnh phải được trích xuất ra một vector các đặc trưng để làm đầu vào cho SVM.

- Có nhiều cách trích xuất đặc trưng thông dụng như: thuật toán HOG, thuật toán shift, …. Hay đơn giản chỉ là flatten ma trận pixel ban đầu của ảnh thành một 1D vector.

- Bằng cách thực nghiệm, việc flatten ma trận pixel để làm vector đặc trưng trên bộ dữ liệu này là không hiệu quả khi các ảnh đều có màu sắc nền và màu sắc các vật thể tương đối giống nhau làm cho các vector pixel sau khi làm phẳng sẽ không mang quá nhiều ý nghĩa phân loại. Do đó cần thiết có một bộ trích xuất đặc trưng hợp lý cho việc tìm ra các đặc trưng phù hợp làm đầu vào cho SVM.

- Thuật toán HOG là thuật toán trích xuất đặc trưng bằng cách chia ảnh thành dạng lưới gồm nhiều ô nhỏ và sau đó tính toán phương và độ lớn của các gradient trên từng ô, cuối cùng tạo một vector histogram theo các phương gradient trên mỗi ô đó. Việc dùng HOG cũng không mang lại hiệu quả do làm mô hình trở nên rất over fitting vì độ phức tạp khi nó cho ra tận gần 100000 tham số cho mỗi vector. (Quá trình trích xuất HOG được em thử nghiệm và phát hiện điều này dẫn tới over fitting rất nặng, em sẽ không trình bày trong file .ipynb nộp lên moodle nhưng dưới đây sẽ là các hình ảnh minh họa overfitting).

```
def create_dataset_with_HOG(file_path, df,data,train=True):

    count=1
    for i in range(len(df)):
        imgpath = file_path + '/' + df.loc[i, 'image_id'] + '.jpg'
        main_img = cv2.imread(imgpath)
        center=crop_center_image(main_img)
        #rz_img=cv2.resize(main_img, (512,512))
        #Preprocessing

        # Convert to HSV color space
        hsv = cv2.cvtColor(center, cv2.COLOR_BGR2HSV)
        # Define range of leaf's color in HSV
        lower_green = np.array([36, 25, 25])
        upper_green = np.array([95, 255, 255])
        lower_yellow = np.array([20, 25, 25])
        upper_yellow = np.array([35, 255, 255])
        lower_brown = np.array([10, 25, 25])
        upper_brown = np.array([20, 255, 255])
        # Threshold the HSV image to get only leaf's colors
        mask_green = cv2.inRange(hsv, lower_green, upper_green)
        mask_yellow = cv2.inRange(hsv, lower_yellow, upper_yellow)
        mask_brown = cv2.inRange(hsv, lower_brown, upper_brown)
        mask_gy = cv2.bitwise_or(mask_green, mask_yellow)
        mask = cv2.bitwise_or(mask_gy, mask_brown)
        # Apply the mask to the original image
        result = cv2.bitwise_and(center, center, mask=mask)
        img = cv2.cvtColor(result, cv2.COLOR_BGR2RGB)
        img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)

        # Khởi tạo HOG descriptor
        H = feature.hog(img, orientations=9, pixels_per_cell=(8, 8),
                        cells_per_block=(1, 1), transform_sqrt=True, block_norm="L2")
```

```
X_train2, X_valid2, y_train2, y_valid2=create_dataset_with_HOG(file_path, train_df, data)
```

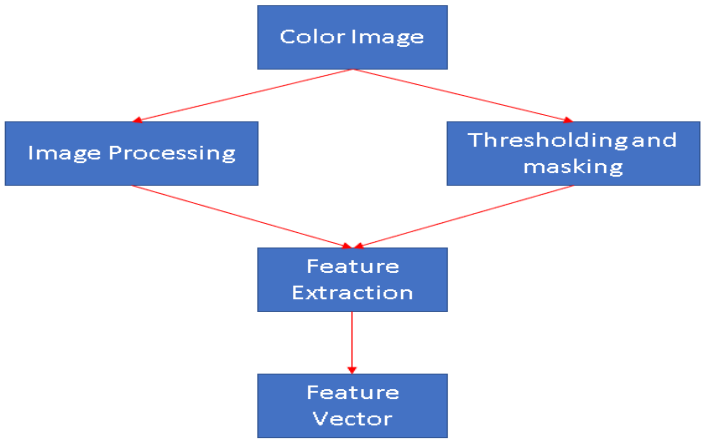
```
500 photos have been preprocessed
======================================
1000 photos have been preprocessed
======================================
1500 photos have been preprocessed
======================================
All photos have been preprocessed
======================================
```

```
for i in range(len(kers2)):
    print('with '+kers2[i]+':')
    print('Train accuracy: ', train_acc_scores2[i])
    print('Valid accuracy: ', valid_acc_scores2[i])
    print('----------------------------')
```

```
with linear SVM:
Train accuracy:  0.9993131868131868
Valid accuracy:  0.4273972602739726
----------------------------
with rbf SVM:
Train accuracy:  0.9993131868131868
Valid accuracy:  0.3589041095890411
----------------------------
with poly SVM:
Train accuracy:  0.9993131868131868
Valid accuracy:  0.29863013698630136
----------------------------
```

- Quá trình trích xuất đặc trưng mà em lựa chọn sẽ là: **Trích xuất Color Feautres và Texture Features của ảnh.**

2. Xây dựng Feature Extractor trích xuất color và texture features



- Đầu tiên giả sử một ảnh màu A sẽ trả qua 2 giai đoạn là ‘Image Processing’ và ‘Thresholding and masking’ trước khi thực sự được trích xuất các đặc trưng về màu và về cấu trúc ảnh.

- Trong giai đoạn Image Processing:

  + Ảnh màu A ban đầu sẽ được giảm kích thước nhằm tăng tốc độ tính toán về sau khi chạy thuật toán Haralick.

  + Ảnh màu A ban đầu sẽ được chuyển từ hệ màu RGB sang hệ màu Gray Scale (thang màu xám), tạm gọi là ảnh xám B.

- Trong giai đoạn Thresholding and masking:

  + Ảnh màu A ban đầu sẽ được lọc nền bằng kĩ thuật mặt nạ màu masking, chỉ giữ lại hình ảnh là lá từ ảnh ban đầu, cố gắng đưa những phần không là lá về màu nền đen. Ta tạm gọi ảnh sau khi qua bước này là ảnh C.

```
#Preprocessing

# Convert to HSV color space
hsv = cv2.cvtColor(main_img, cv2.COLOR_BGR2HSV)
# Define range of leaf's color in HSV
lower_green = np.array([36, 25, 25])
upper_green = np.array([95, 255, 255])
lower_yellow = np.array([20, 25, 25])
upper_yellow = np.array([35, 255, 255])
lower_brown = np.array([10, 25, 25])
upper_brown = np.array([20, 255, 255])
# Threshold the HSV image to get only leaf's colors
mask_green = cv2.inRange(hsv, lower_green, upper_green)
mask_yellow = cv2.inRange(hsv, lower_yellow, upper_yellow)
mask_brown = cv2.inRange(hsv, lower_brown, upper_brown)
mask_gy = cv2.bitwise_or(mask_green, mask_yellow)
mask = cv2.bitwise_or(mask_gy, mask_brown)
# Apply the mask to the original image
result = cv2.bitwise_and(main_img, main_img, mask=mask)
img = cv2.cvtColor(result, cv2.COLOR_BGR2RGB)
gs = cv2.cvtColor(resized_img,cv2.COLOR_BGR2GRAY)
```

Minh họa masking trên ảnh 'Test_8' của bộ dữ liệu

- Trích xuất Color Features:

+ Ảnh sẽ được dùng cho việc trích xuất các giá trị đặc trưng về màu như sau: Đầu tiên sẽ là việc tách riêng 3 kênh màu red, green, blue. Sau đó loại những pixel nào có màu trắng còn sót lại, chỉ giữ lại những pixel có màu. Cuối cùng tính **6 giá trị đặc trưng** về màu mà về sau sẽ kết hợp thêm với các giá trị đặc trưng về cấu trúc để tạo thành vector đặc trưng hoàn chỉnh. Sáu giá trị này bao gồm: **'red_mean', 'blue_mean', 'green_mean':** lần lượt là trung bình của tất cả các pixel trong các kênh: màu đỏ, màu xanh dương, màu xanh lá cây; **'red_std', 'blue_std', 'green_std':** lần lượt là độ lệch chuẩn của tất cả các pixel trong các kênh: màu đỏ, màu xanh dương, màu xanh lá cây.

```
#Color features
r_channel = img[:,:,0]
g_channel = img[:,:,1]
b_channel = img[:,:,2]
r_channel[r_channel == 255] = 0 #loại bỏ các pixel có nền trắng và chỉ giữ lại các pixel có màu.
g_channel[g_channel == 255] = 0 #loại bỏ các pixel có nền trắng và chỉ giữ lại các pixel có màu.
b_channel[b_channel == 255] = 0 #loại bỏ các pixel có nền trắng và chỉ giữ lại các pixel có màu.

red_mean = np.mean(r_channel)
green_mean = np.mean(g_channel)
blue_mean = np.mean(b_channel)

red_std = np.std(r_channel)
green_std = np.std(g_channel)
blue_std = np.std(b_channel)
```

- Trích xuất Texture Features:

+ **Thuật toán Haralick**: Hay còn được gọi là Ma trận Xác suất xám (Gray-level co-occurrence matrix - GLCM) do nhà khoa học máy tính nổi tiếng Robert Haralick đề xuất, gồm các bước thực hiện sau:

*Bước 1: Chọn hướng của các cặp pixel cùng xuất hiện. Hướng này có thể được chọn từ 4 hướng cơ bản (trên, dưới, trái, phải) hoặc 8 hướng (bao gồm các hướng chéo).

*Bước 2: Tạo ra Ma trận Xác suất xám (GLCM) bằng cách đếm số lần một cặp pixel cùng xuất hiện với nhau trong hình ảnh.

*Bước 3: Chuẩn hóa GLCM bằng cách chia toàn bộ ma trận cho tổng số lần xuất hiện của tất cả các cặp pixel cùng xuất hiện.

*Bước 4: Tính toán các đặc trưng của hình ảnh từ GLCM đã được chuẩn hóa. Các đặc trưng này có thể bao gồm độ đồng nhất, độ đa dạng, độ phân tán, độ lệch chuẩn và độ đối xứng. (Em sẽ không đi sâu vào công thức tính của từng đặc trưng do có thư viện hỗ trợ tính sẵn mà em sẽ chỉ trình bày ý nghĩa của những đặc trưng được lựa chọn)

+ Ảnh xám B ở bước ‘Image Processing’ sẽ được áp dụng thuật toán Haralick để tính toán các đặc trưng về cấu trúc thông qua sự hỗ trợ của **thư viện mahotas**. Hàm haralick trong thư viện mahotas sẽ trả về 13 đặc trưng về cấu trúc, và em sẽ chỉ lựa chọn **4 đặc trưng** cần thiết cho bài như sau:

* **Độ tương phản (Contrast)**: Thể hiện mức độ khác biệt giữa các giá trị màu sắc trong hình ảnh. Mỗi đối tượng trong một ảnh có độ tương phản khác biệt so với phần còn lại của hình ảnh có thể được dễ dàng phát hiện bằng cách sử dụng các ngưỡng.

* **Độ đối xứng (Correlation)**: Thể hiện mức độ tương quan giữa các giá trị pixel trong hình ảnh. Những phần có màu gần giống nhau sẽ có độ đối xứng cho các cặp pixel gần giống nhau. Những phần có màu hoàn toàn khác biệt thì độ đối xứng thấp.

* **Độ phân tán (Inverse Difference Moment)**: Thể hiện sự đồng nhất hoặc độ đa dạng trong các giá trị xám liên tiếp trong hình ảnh. Giá trị độ phân tán nằm trong khoảng [0,1]. Giá trị càng gần 0, thể hiện sự đồng nhất cao, tức là các giá trị xám liên tiếp trong hình ảnh có xu hướng giống nhau. Ngược lại, giá trị càng gần 1, thể hiện sự đa dạng cao, tức là các giá trị xám liên tiếp trong hình ảnh có xu hướng khác nhau.

* **Độ mịn (Entropy)**: Thể hiện mức độ phức tạp của hình ảnh, nếu một bức ảnh có độ phức tạp càng cao thì giá trị entropy càng cao và ngược lại độ phức tạp càng thấp thì giá trị entropy càng thấp.

```
#libary for feature extraction
import mahotas as mt
```

```
#Texture features
textures = mt.features.haralick(gs)
ht_mean = textures.mean(axis=0)
contrast = ht_mean[1]
correlation = ht_mean[2]
inverse_diff_moments = ht_mean[4]
entropy = ht_mean[8]
```

Cuối cùng kết hợp 6 giá trị đặc trưng về màu cùng với 4 giá trị đặc trưng về cấu trúc tạo thành một vector đặc trưng gồm 10 thành phần cho một bức ảnh sau khi kết thúc bước Feature Extraction, kết hợp tất cả các vector của các ảnh trong bộ huấn luyện tạo thành ma trận X làm đầu vào cho bộ phân lớp SVM.

```
    vector = [red_mean,green_mean,blue_mean,red_std,green_std,blue_std,
              contrast,correlation,inverse_diff_moments,entropy]

    df_temp = pd.DataFrame([vector],columns=columns_name)
    data = pd.concat([data, df_temp], axis=0)

#bước tạo vector chứa các nhãn
if train==True:
    y_train=np.array([np.argmax(df.loc[i,df.columns[1:]].values) for i in range(len(df))])

#Chuyển dataframe sang máng numpy
X=data.to_numpy()

#split thành tập train và tập validation
if train==True:
    X_train, X_valid, y_train, y_valid = train_test_split(X,y_train,test_size=0.2, random_state=82)
    return X_train, X_valid, y_train, y_valid
else:
    return X
```

```
X_train[:5]
```

```
array([[ 76.53727535, 108.29877196,  51.72869448,  42.47305149,
         45.21811019,  34.38883969,  58.36673995,   0.98196846,
          0.35743997,  11.29094523],
       [102.88661716, 135.53114626,  85.72956874,  37.10103482,
         38.77472273,  42.94918524,  91.58283355,   0.95941375,
          0.32209159,  11.58554523],
       [ 80.55060561, 102.5577034 ,  49.03025698,  48.25469757,
         51.50718025,  33.46620037,  67.4918383 ,   0.98208693,
          0.34320828,  11.58619197],
       [106.62841475, 125.41067029,  64.30971519,  43.27806645,
         47.62519819,  31.67316686,  34.50613217,   0.98361506,
          0.40089009,  10.89457083],
       [ 72.58660249, 105.46394624,  67.89748669,  52.85907441,
         64.34239843,  51.22056172,  35.85309774,   0.99006176,
          0.36156225,  11.3090454 ]])
```

5 hàng đầu tiên của tập X_train

3. Xây dựng bộ phân lớp SVM

a. Lý thuyết về Support vector machine

- Khoảng cách từ một điểm tới một siêu phẳng:

+ Trong không gian 2 chiều: Ta có khoảng cách từ một điểm (x_0, y_0) tới một đường thẳng $w_1x + w_2y + b = 0$ được xác định bởi:

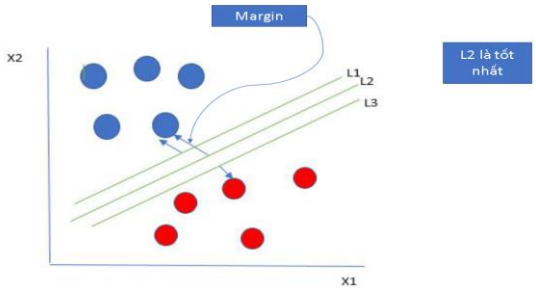
$$\frac{|w_1x_0 + w_2y_0 + b|}{\sqrt{w_1^2 + w_2^2}}$$

+ Trong không gian 3 chiều: Ta có khoảng cách từ một điểm (x_0, y_0, z_0) tới một mặt phẳng $w_1x + w_2y + w_3z + b = 0$ được xác định bởi:

$$\frac{|w_1x_0 + w_2y_0 + w_3z_0 + b|}{\sqrt{w_1^2 + w_2^2 + w_3^2}}$$

- Khi bỏ trị tuyệt đối ở tử số của 2 công thức trên đi thì tử số có thể âm hoặc dương và nhờ vào tính chất có dấu này mà ta có thể biết được một điểm đang nằm về phía nào của đường thẳng hay mặt phẳng đang xét. Để không mất tính tổng quát thì một đường thẳng trong không gian hai chiều hay một mặt phẳng trong không gian 3 chiều hay một mặt phân chia nào đó trong không gian nhiều chiều hơn đều sẽ được gọi chung là một **hyperplane**.

- Giả sử xét bài toán phân loại nhị phân trong không gian 2 chiều. Ta có hai lớp mà chúng tách biệt tuyến tính với nhau. Khi đó chúng ta cần tìm một đường thẳng phân chia sao cho khoảng cách gần nhất từ một điểm dữ liệu của mỗi lớp đến đường thẳng là lớn nhất, và khoảng cách lớn nhất này gọi là margin.



Link ảnh: https://www.geeksforgeeks.org/support-vector-machine-algorithm/

- Bài toán tối ưu cho SVM:

+ Giả sử các điểm xanh thuộc lớp 1 và các điểm đỏ thuộc lớp -1, mặt phân chia có dạng: $w_1x_1 + w_2x_2 + b = 0$, khi đó margin sẽ được tính:

$$\text{margin} = \min_n \frac{y_n(\mathbf{w}^T\mathbf{x}_n + b)}{||\mathbf{w}||_2}$$

+ Vậy việc tối ưu cần làm sẽ là tìm w và b sao cho margin này là lớn nhất:

$$(\mathbf{w}, b) = \arg\max_{\mathbf{w},b} \left\{ \min_n \frac{y_n(\mathbf{w}^T\mathbf{x}_n + b)}{||\mathbf{w}||_2} \right\}$$

- **Kernel trong SVM:**

+ Trong thực tế rất khó để bộ dữ liệu có thể đạt được trạng thái phân biệt tuyến tính ngay từ đầu, do đó thay vì các điểm dữ liệu khi ở chiều không gian cũ sẽ khó để tìm được một hyperplane phân tách tuyến tính được chúng thì chúng ta sẽ dùng các hàm được gọi là **hàm kernel** có tác dụng chuyển đổi các điểm dữ liệu sang không gian mới mà ở đó ta hy vọng sẽ tìm được một hyperplane phân tách tuyến tính được các lớp.

+ Sau đây em sẽ trình bày ba loại kernel thông dụng mà em sẽ sử dụng trong bài làm lần này:

* **Linear kernel:** Tích vô hướng của hai vector.

$$k(\mathbf{x}, \mathbf{z}) = \mathbf{x}^T \mathbf{z}$$

* **Rbf kernel:** Dựa trên phép tính Gaussian giữa 2 vector.

$$k(\mathbf{x}, \mathbf{z}) = \exp(-\gamma ||\mathbf{x} - \mathbf{z}||_2^2), \quad \gamma > 0$$

Trong đó gamma sẽ là một hyperparameter của kernel, nếu gamma càng cao thì độ phức tạp của hyperplane tìm được sẽ càng cao và ngược lại.

* **Poly kernel:** Kernel bậc cao khi áp dụng cho các trường hợp không còn là tổ hợp tuyến tính.

$$k(\mathbf{x}, \mathbf{z}) = (r + \gamma \mathbf{x}^T \mathbf{z})^d$$

Trong đó thì d sẽ là bậc của đa thức.

b. Cài đặt các mô hình SVM cho các loại kernel

- Việc cài đặt sẽ được hỗ trợ bằng thư viện sklearn

- Trước khi đưa dữ liệu vào SVM thì dữ liệu cần được chuẩn hóa nhằm mục đích có một phân phối đẹp hơn và tăng tính ổn định cho mô hình.

```
#thực hiện bước feature scaling
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_valid = scaler.transform(X_valid)
```

- Xây dựng ba bộ phân lớp SVM cho ba kernel:

```
from sklearn import svm
```

```
C = 100  # SVM regularization parameter
lin_svm = svm.SVC(kernel='linear', gamma=1.5, tol=0.001, C=C, probability=True).fit(X_train, y_train)
rbf_svm = svm.SVC(kernel='rbf', decision_function_shape='ovr',
                  degree=3, gamma=1.5, tol=0.001, C=C, probability=True).fit(X_train, y_train)
poly_svm = svm.SVC(kernel='poly', degree=3, C=C, tol=0.001, probability=True).fit(X_train, y_train)
```

+ kernel: chỉ định loại kernel được dùng.

+ gamma: hypeparameter xác định mức độ phức tạp của hyperplane.

+ tol: mức sai số chấp nhận, khi đạt ngưỡng này thì mô hình sẽ dừng lại mà không tiếp tục học nữa.

+ C: mức regularization cho các tham số của mô hình.

+ probability: True: sẽ dự đoán xác suất cho tất cả các lớp cần phân lớp (cần thiết trong yêu cầu bài này); False: không dự đoán xác suất.

c. Đánh giá mô hình

```
train_acc_scores, valid_acc_scores=[], []
for s in [lin_svm, rbf_svm, poly_svm]:
    train_acc_scores.append(accuracy_score(y_train, s.predict(X_train)))
    valid_acc_scores.append(accuracy_score(y_valid, s.predict(X_valid)))
```

```
kers=['linear SVM', 'rbf SVM', 'poly SVM']
for i in range(len(kers)):
    print('with '+kers[i]+':')
    print('Train accuracy: ', train_acc_scores[i])
    print('Valid accuracy: ', valid_acc_scores[i])
    print('----------------------------')
```

```
with linear SVM:
Train accuracy:  0.49793956043956045
Valid accuracy:  0.5150684931506849
----------------------------
with rbf SVM:
Train accuracy:  0.9993131868131868
Valid accuracy:  0.5095890410958904
----------------------------
with poly SVM:
Train accuracy:  0.6675824175824175
Valid accuracy:  0.4712328767123288
----------------------------
```

- Nhận xét chung:

+ Mô hình SVM đạt độ chính xác thấp kể cả trên tập train và tập test.

+ Điều này là do việc dùng mô hình SVM cần có các giá trị số cho các thuộc tính làm đầu vào, việc trích xuất thuộc tính chỉ liên quan đến phần chiếc lá bị bệnh ở trung tâm bức ảnh gặp khó khăn với bộ dữ liệu này.

+ **Lý do gặp khó khăn là do:**

* Bức ảnh không tồn tại duy nhất một chiếc lá mà có rất nhiều chiếc lá khác xung quanh cũng như rất nhiều chiếc lá trong bức ảnh rõ nét hoặc mờ khác nhau từ vào việc nó ở xa hay gần trong lúc chụp.

* Nếu chiếc lá độc lập trên một nền có màu hoàn toàn thì việc tập trung trích xuất các đặc trưng số cho chiếc lá là khả thi hơn nhiều.

* Lá bị bệnh đôi khi có chi tiết về bệnh rất ít cũng như màu sắc đốm bệnh gần tương đồng với màu lá cũng gây khó khăn rất nhiều cho việc lấy được thông tin đúng khi trích xuất các đặc trưng về màu sắc cũng như cấu trúc chiếc lá.

+ **Nếu có thời gian xin thầy giành chút ít thời gian ghé thăm mô hình của em được áp dụng trên một bộ dataset ảnh lá cây khác, đó chính là bộ dataset khá nổi tiếng có tên 'PlantVillage'. Tại đây em sẽ minh chứng được những khó khăn mà mình vừa nêu trên do mô hình khi áp dụng trên tập dataset 'PlantVillage' đạt accuracy là 85%.**

+ Link notebook cho mô hình áp dụng trên PlantVillage:

https://colab.research.google.com/drive/1sKs5bw21cDzcNQMa_W0wvRzp3jvF99Vl?usp=sharing

VII. Cải tiến bộ trích xuất đặc trưng và SVM

1. Cải tiến bộ trích xuất đặc trưng

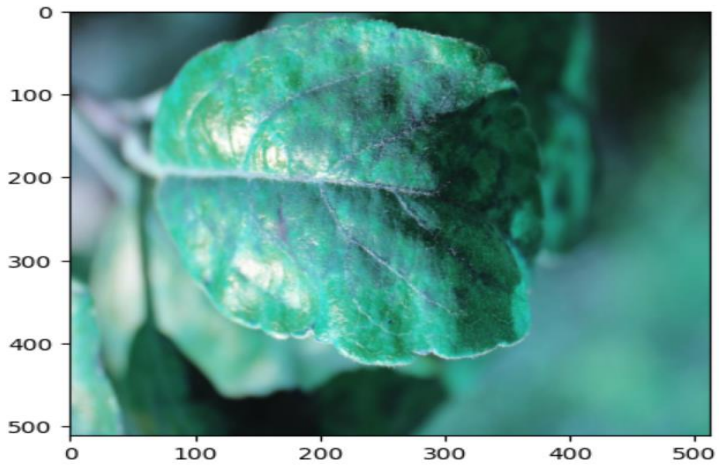
- Nếu khó khăn liên quan đến việc bức ảnh có quá nhiều nhiễu, màu nền khó phân tách với vật thể lá, hay độ mờ/nét của lá trong ảnh, ... Thì việc khắc phục chắc chắn sẽ là tập trung vào vấn đề lấy được đoạn hình ảnh quan trọng về lá cây bị bệnh.

- Nhận thấy tất cả các tấm ảnh đều sẽ có một lá ở trung tâm, lá này sẽ chứa tình trạng cho tất cả lá khác nếu cùng có mặt trong ảnh, vậy ta sẽ **cố gắng lấy thông tin của chỉ trên chiếc lá ở trung tâm ảnh này**.

- Các bước thực hiện:

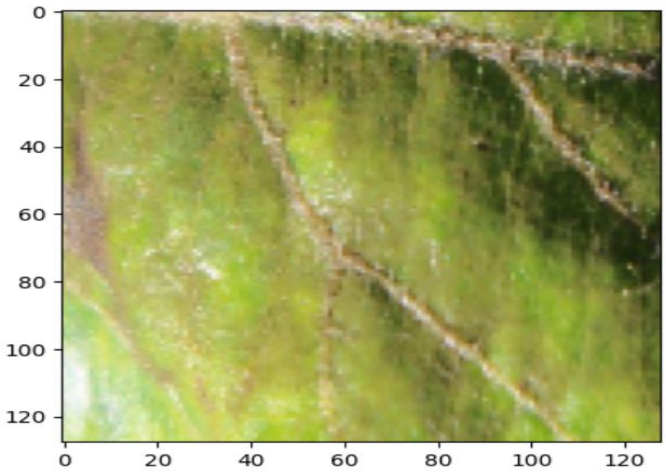
+ Cắt phần ảnh ở trung tâm bức ảnh:

* Trước khi cắt cần giảm kích thước ảnh ban đầu về 512x512, do việc giảm kích thước sẽ giúp phần cắt có một kích thước vừa phải không quá lớn nhưng vẫn sẽ giữ được nhiều thông tin nhất có thể so với khi nếu giữ nguyên kích thước gốc ban đầu thì việc cắt sao cho giữ lại nhiều thông tin thì cần kích thước lớn. Kích thước 512 là do nên sử dụng kích thước ảnh theo số mũ của cơ số 2.



Ảnh 'Train_424.jpg' sau khi resize và được hiện ở chế độ GrayScale

* Sau khi giảm kích thước thì cắt phần ảnh trung tâm với kích thước 128x128.



Phần trung tâm được cắt của ảnh 'Train_424.jpg' và được hiện ở chế độ màu RGB

+ Ảnh trung tâm được cắt sẽ được dùng để trích xuất đặc trưng.

- Hình ảnh tổng thể cho quá trình cải tiến:

```python
def crop_center_image(image):
    image=cv2.resize(image,(512,512))
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    M = cv2.moments(gray)
    cx = int(M["m10"] / M["m00"])
    cy = int(M["m01"] / M["m00"])

    # Tính toán kích thước và vị trí của vùng ảnh bạn muốn cắt ra từ ảnh gốc
    w = 128
    h = 128
    x = cx - w // 2
    y = cy - h // 2

    # Kiểm tra xem vùng ảnh cắt ra có vượt quá kích thước của ảnh gốc hay không
    if x < 0:
        x = 0
    if y < 0:
        y = 0
    if x + w > gray.shape[1]:
        w = gray.shape[1] - x
    if y + h > gray.shape[0]:
        h = gray.shape[0] - y

    # Cắt vùng ảnh từ ảnh gốc và chuyển sang grayscale
    center = image[y:y+h, x:x+w]
    return center
```

```python
def create_dataset_with_crop_center(file_path, df,train=True):
    """
    Parameters:
    ----------
    csv_train_file_path: đường dẫn đến file csv chứa kết quả nhãn cho các mẫu tập train.
    file_path: đường dẫn đến nơi lưu các ảnh.
    df: dataframe chứa thông tin từ file .csv
    train: biến boolean để xác định tạo dataset cho việc train hay huấn luyện.
    """

    columns_name = ['mean_r','mean_g','mean_b','stddev_r','stddev_g','stddev_b',
                    'contrast','correlation', 'inverse_difference_moments', 'entropy']
    data = pd.DataFrame([], columns=columns_name)

    for i in range(len(df)):
        imgpath = file_path + '/' + df.loc[i, 'image_id'] + '.jpg'
        main_img = cv2.imread(imgpath)
        center=crop_center_image(main_img)
        rgb_img = cv2.cvtColor(center, cv2.COLOR_BGR2RGB)

        gs = cv2.cvtColor(rgb_img,cv2.COLOR_RGB2GRAY)


        #Color features
        r_channel = rgb_img[:,:,0]
        g_channel = rgb_img[:,:,1]
        b_channel = rgb_img[:,:,2]
        r_channel[r_channel == 255] = 0 #Loại bỏ các pixel có nền trắng và chỉ giữ lại các pixel có màu.
        g_channel[g_channel == 255] = 0 #Loại bỏ các pixel có nền trắng và chỉ giữ lại các pixel có màu.
        b_channel[b_channel == 255] = 0 #Loại bỏ các pixel có nền trắng và chỉ giữ lại các pixel có màu.

        red_mean = np.mean(r_channel)
        green_mean = np.mean(g_channel)
        blue_mean = np.mean(b_channel)
```

```python
        #Texture features
        textures = mt.features.haralick(gs)
        ht_mean = textures.mean(axis=0)
        contrast = ht_mean[1]
        correlation = ht_mean[2]
        inverse_diff_moments = ht_mean[4]
        entropy = ht_mean[8]

        vector = [red_mean,green_mean,blue_mean,red_std,green_std,blue_std,
                  contrast,correlation,inverse_diff_moments,entropy]

        df_temp = pd.DataFrame([vector],columns=columns_name)
        data = pd.concat([data, df_temp], axis=0)

    #Ước tạo vector chứa các nhãn
    if train==True:
        y_train=np.array([np.argmax(df.loc[i,df.columns[1:]].values) for i in range(len(df))])

    #Chuyển dataframe sang mảng numpy
    X=data.to_numpy()

    #split thành tập train và tập validation
    if train==True:
        X_train, X_valid, y_train, y_valid = train_test_split(X,y_train,test_size=0.2, random_state=82)
        return X_train, X_valid, y_train, y_valid
    else:
        return X
```

2. Cải tiến SVM

- Các đặc điểm của hình ảnh đã rõ ràng hơn do đó cần giảm tham số gamma để tránh overfitting.

```
C = 100  # SVM regularization parameter
lin_svm = svm.SVC(kernel='linear', gamma=0.1, tol=0.001, C=C, probability=True).fit(X_train, y_train)
rbf_svm = svm.SVC(kernel='rbf', decision_function_shape='ovr',
                  degree=3, gamma=0.1, tol=0.001, C=C, probability=True).fit(X_train, y_train)
poly_svm = svm.SVC(kernel='poly', degree=3, C=C, tol=0.001, probability=True).fit(X_train, y_train)
```

3. Đánh giá

- Trên tập Train và Validation:

```
train_acc_scores, valid_acc_scores=[], []
for s in [lin_svm, rbf_svm, poly_svm]:
    train_acc_scores.append(accuracy_score(y_train, s.predict(X_train)))
    valid_acc_scores.append(accuracy_score(y_valid, s.predict(X_valid)))
```

```
kers=['linear SVM', 'rbf SVM', 'poly SVM']
for i in range(len(kers)):
    print('with '+kers[i]+':')
    print('Train accuracy: ', train_acc_scores[i])
    print('Valid accuracy: ', valid_acc_scores[i])
    print('---------------------------')
```

```
with linear SVM:
Train accuracy:  0.6016483516483516
Valid accuracy:  0.6219178082191781
---------------------------
with rbf SVM:
Train accuracy:  0.8660714285714286
Valid accuracy:  0.6356164383561644
---------------------------
with poly SVM:
Train accuracy:  0.7307692307692307
Valid accuracy:  0.6164383561643836
---------------------------
```

- Sau khi cải tiến, accuracy trên tập validation đạt đến xấp xỉ 63%.

- Tạo file submission trên tập test.

```
X_test=create_dataset_with_crop_center(file_path, test_df, train=False)
```

```
X_test = scaler.transform(X_test)
```

```
probs_rbf = rbf_svm.predict_proba(X_test)
softmax_rbf = np.exp(probs_rbf) / np.sum(np.exp(probs_rbf), axis=1, keepdims=True)
probs_linear = lin_svm.predict_proba(X_test)
softmax_linear = np.exp(probs_linear) / np.sum(np.exp(probs_linear), axis=1, keepdims=True)
probs_poly = poly_svm.predict_proba(X_test)
softmax_poly = np.exp(probs_poly) / np.sum(np.exp(probs_poly), axis=1, keepdims=True)
```

```
def create_submission(softmax, test_df):
    submission_df = softmax.copy()
    submission_df=pd.DataFrame(submission_df,columns=['healthy','multiple_diseases','rust','scab'])
    submission_df['image_id'] = list(test_df.image_id.values)
    submission_df = submission_df[['image_id','healthy','multiple_diseases','rust','scab']]
    return submission_df
```

```
submission_rbf_df = create_submission(softmax_rbf,test_df)
submission_linear_df = create_submission(softmax_linear,test_df)
submission_poly_df = create_submission(softmax_poly,test_df)
```

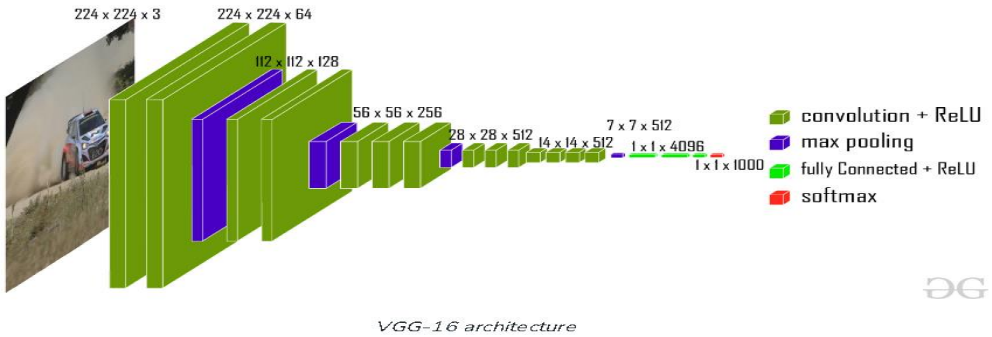
```
submission_rbf_df.to_csv('submission-svm-rbfKernel.csv', index=False)
submission_linear_df.to_csv('submission-svm-linearKernel.csv', index=False)
submission_poly_df.to_csv('submission-svm-polyKernel.csv', index=False)
```

VIII. Xây dựng kiến trúc VGG16

1. Sơ lược về lý thuyết kiến trúc VGG16

- VGG được xây dựng hầu như dựa trên tất cả các đặc điểm của mạng thần kinh tích chập (CNN) gồm các lớp chính như lớp tích chập (Convolutional layers), lớp kết nối đầy đủ (Fullyconnected layers), lớp pooling (pooling layer giúp chắt lọc thông tin). Bên cạnh đó VGG được xây dựng với bộ lọc (filters) tích chập rất nhỏ. VGG có 2 phiên bản là VGG-16 và VGG-19 tương ứng với 16 và 19 lớp tích chập (convolutional layers).

- Mạng VGG - 16 bao gồm 13 lớp convolution đều có kernel 3x3 (sau mỗi lớp convolution là max-pooling downside xuống 0.5) và 3 lớp fully connected.



VGG-16 architecture

Link ảnh: https://www.geeksforgeeks.org/vgg-16-cnn-model/

- **Lớp tích chập (Convolution layer)**: Ảnh được chuyển qua một một chồng các lớp tích chập với bộ lọc có kích thước 3x3. Stride của tích chập và padding đầu vào được cố định là 1 pixel cho các lớp tích chập 3 x 3, điều này đảm bảo rằng độ phân giải không gian được giữ nguyên sau khi tích chập. Năm lớp max-pooling 2x2 với giá trị stride là 2.

- **Lớp ẩn (Hidden Layers)**: Các lớp ẩn trong mạng VGG đều sử dụng hàm kích hoạt ReLU để lọc các giá trị nhỏ hơn 0.

- **Lớp kết nối đầy đủ (Fully-Connected Layers)**: Sau các lớp convolution và pooling thì dữ liệu được flatten và cho vào lớp fully connected. Mô hình có 3 lớp FC như sau: hai lớp đầu tiên có 4096 đơn vị mỗi lớp và lớp thứ ba chứa 1000 đơn vị cho mỗi lớp.

- Ngoài trong kiến trúc VGG16 còn áp dụng ResNet tại một số điểm (dùng skip connection để nối tắt qua một vài lớp).

2. Data Augmention

- Dữ liệu là yếu tố rất quan trọng đối với việc xây dựng mô hình học sâu, do đó khi số lượng dữ liệu quá ít và có thể làm giảm độ chính xác của mô hình thì việc tìm cách tăng số lượng dữ liệu là đều bắt buộc.

- Tuy nhiên việc tìm kiếm dữ liệu là không dễ dàng và có thể tốn chi phí, do đó một trong nhưng phương pháp rất dễ thực hiện và được áp dụng rất thường xuyên là data augmentation.

- Data augmentation cho hình ảnh là cách tận dụng những hình ảnh sẵn có sau đó sử dụng những kỹ thuật biến đổi hình học, thay đổi không gian màu, thêm nhiễu, xóa ngẫu nhiên…

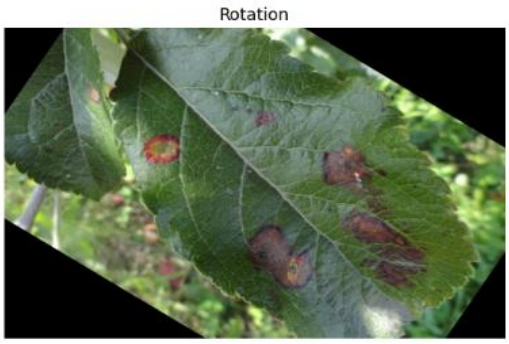
- Các kĩ thuật tăng cường phổ biến như là: lật ảnh, xoay ảnh, cắt ảnh, thêm nhiễu, đổi hệ màu, …

3. Cài đặt

- Cài đặt cho việc tăng cường dữ liệu:

```
# data_transforms cho việc tăng cường dữ liệu
training_transforms = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225)),
    transforms.Resize((224,224),antialias=True),
    transforms.RandomHorizontalFlip(),
    transforms.RandomVerticalFlip(),
    transforms.RandomRotation(degrees=30)]
)
valid_transforms = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225)),
    transforms.Resize((224,224),antialias=True)]
)
```

- Minh họa trên ảnh ‘Test_82.jpg’:



- Tạo một lớp dataset tùy chỉnh cho việc đọc dữ liệu:

```
class MyDataset(Dataset):
    def __init__(self, file_path, df, transforms=None, training=True):
        self.df = df
        self.transforms = transforms
        self.training = training
        self.Path_to_image=file_path
    def __len__(self):
        return len(self.df)
    def __getitem__(self, idx):
        img_path = self.Path_to_image + '/' + self.df.loc[idx, 'image_id'] + '.jpg'
        img = Image.open(img_path).convert('RGB')
        if self.training:
            labels = np.argmax(self.df.loc[idx,self.df.columns[1:]].values)
            labels =torch.as_tensor(labels, dtype=torch.int8)
            if self.transforms:
                img = self.transforms(img)
            return img, labels
        else:
            if self.transforms:
                img = self.transforms(img)
            return img
        return img
```

- Xây dựng model: Bằng cách tạo một khung model vgg16 có sẵn do torch.vision cung cấp sau đó training model và lưu lại bộ tham số tối ưu nhất cho model.

- Link đến mô hình đã được pre-trained: https://drive.google.com/file/d/1zIP10ACKPHbbb2Ke6a0XNbKIEFWrOL5r/view?usp=sharing

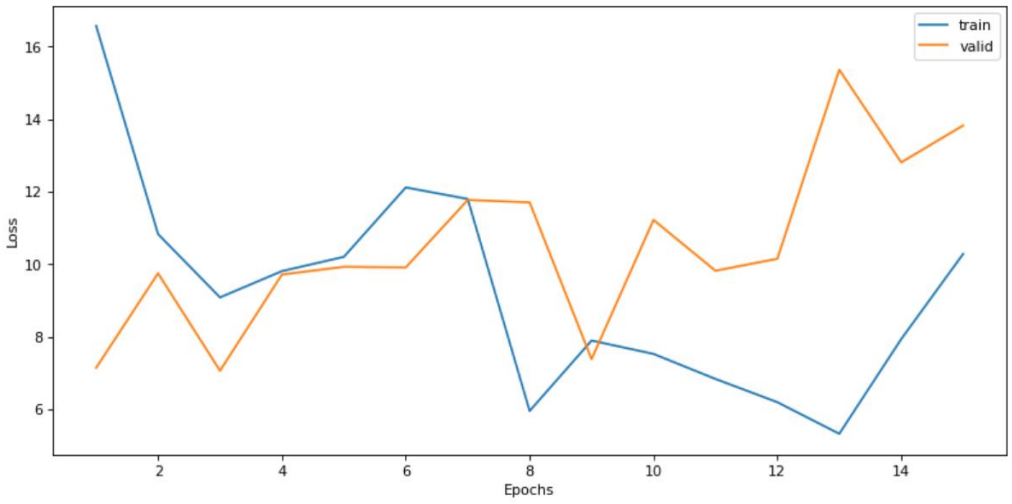
- Kết quả training 15 epochs của mô hình:

```
Epoch 1/15:
Training loss: 16.564  Accuracy: 0.941
Valid loss: 7.146     Accuracy: 0.976
Model save with 7.146 loss at Epoch 1

============================================================
Complete epoch 1/15     Training loss: 16.564     Trainning accuracy: 0.941   Valid loss: 7.146    Valid accurac
y: 0.976
============================================================
Epoch 2/15:
Training loss: 10.829  Accuracy: 0.956
Valid loss: 9.749     Accuracy: 0.976

============================================================
Complete epoch 2/15     Training loss: 10.829     Trainning accuracy: 0.956   Valid loss: 9.749    Valid accurac
y: 0.976
============================================================
Epoch 3/15:
Training loss: 9.085  Accuracy: 0.970
Valid loss: 7.067     Accuracy: 0.981
Model save with 7.067 loss at Epoch 3

============================================================
Complete epoch 3/15     Training loss: 9.085     Trainning accuracy: 0.970   Valid loss: 7.067    Valid accurac
y: 0.981

============================================================
Epoch 4/15:
Training loss: 9.809  Accuracy: 0.957
Valid loss: 9.715     Accuracy: 0.964

============================================================
Complete epoch 4/15     Training loss: 9.809     Trainning accuracy: 0.957   Valid loss: 9.715    Valid accurac
y: 0.964
============================================================
Epoch 5/15:
Training loss: 10.202  Accuracy: 0.959
Valid loss: 9.928     Accuracy: 0.964

============================================================
Complete epoch 5/15     Training loss: 10.202     Trainning accuracy: 0.959   Valid loss: 9.928    Valid accurac
y: 0.964
============================================================
Epoch 6/15:
Training loss: 12.115  Accuracy: 0.958
Valid loss: 9.911     Accuracy: 0.969

============================================================
Complete epoch 6/15     Training loss: 12.115     Trainning accuracy: 0.958   Valid loss: 9.911    Valid accurac
y: 0.969
============================================================

Epoch 7/15:
Training loss: 11.797  Accuracy: 0.960
Valid loss: 11.769     Accuracy: 0.966

============================================================
Complete epoch 7/15     Training loss: 11.797     Trainning accuracy: 0.960   Valid loss: 11.769    Valid accura
cy: 0.966
============================================================
Epoch 8/15:
Training loss: 5.959  Accuracy: 0.975
Valid loss: 11.702     Accuracy: 0.973

============================================================
Complete epoch 8/15     Training loss: 5.959     Trainning accuracy: 0.975   Valid loss: 11.702    Valid accurac
y: 0.973
============================================================
Epoch 9/15:
Training loss: 7.897  Accuracy: 0.963
Valid loss: 7.385     Accuracy: 0.973

============================================================
Complete epoch 9/15     Training loss: 7.897     Trainning accuracy: 0.963   Valid loss: 7.385    Valid accurac
y: 0.973
============================================================
```

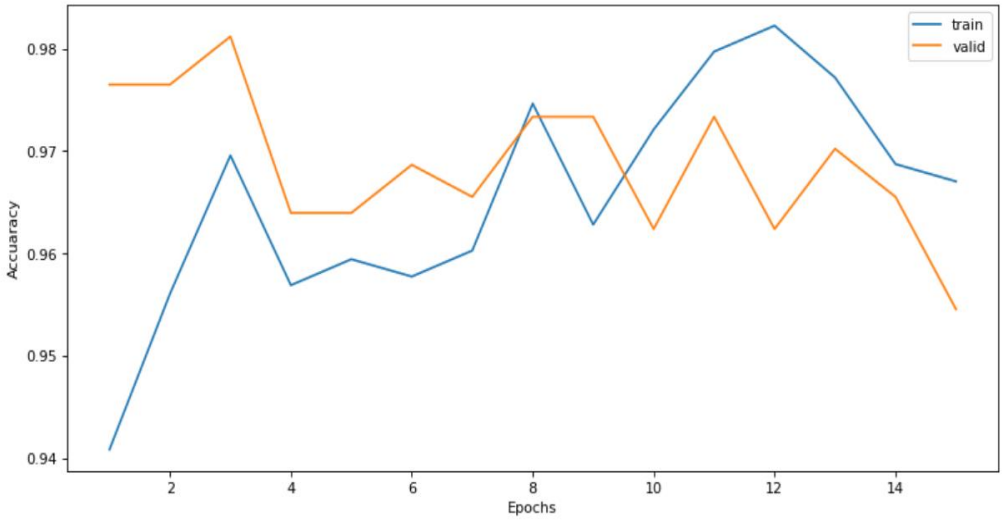
```
Epoch 10/15:
Training loss: 7.529  Accuracy: 0.972
Valid loss: 11.220     Accuracy: 0.962

============================================================
Complete epoch 10/15     Training loss: 7.529     Trainning accuracy: 0.972   Valid loss: 11.220    Valid accura
cy: 0.962
============================================================
Epoch 11/15:
Training loss: 6.841  Accuracy: 0.980
Valid loss: 9.819     Accuracy: 0.973

============================================================
Complete epoch 11/15     Training loss: 6.841     Trainning accuracy: 0.980   Valid loss: 9.819    Valid accurac
y: 0.973
============================================================
Epoch 12/15:
Training loss: 6.199  Accuracy: 0.982
Valid loss: 10.150     Accuracy: 0.962

============================================================
Complete epoch 12/15     Training loss: 6.199     Trainning accuracy: 0.982   Valid loss: 10.150    Valid accura
cy: 0.962

============================================================
Complete epoch 13/15     Training loss: 5.330     Trainning accuracy: 0.977   Valid loss: 15.356    Valid accura
cy: 0.970
============================================================
Epoch 14/15:
Training loss: 7.937  Accuracy: 0.969
Valid loss: 12.805     Accuracy: 0.966

============================================================
Complete epoch 14/15     Training loss: 7.937     Trainning accuracy: 0.969   Valid loss: 12.805    Valid accura
cy: 0.966
============================================================
Epoch 15/15:
Training loss: 10.283  Accuracy: 0.967
Valid loss: 13.821     Accuracy: 0.955

============================================================
Complete epoch 15/15     Training loss: 10.283     Trainning accuracy: 0.967   Valid loss: 13.821    Valid accur
acy: 0.955
============================================================
```

4. Đánh giá

- Biểu đồ hàm chi phí:



- Biểu đồ Accuracy:



IX. Tổng kết kết quả thu được

- Mô hình SVM kết hợp bộ trích xuất đặc trưng đã qua cải tiến:

+ Linear kernel:

Train accuracy: 0.6016.

Validation accuracy: 0.6219.

+ Rbf kernel:

Train accuracy: 0.86607.

Validation accuracy: 0.63562.

+ Poly kernel:

Train accuracy: 0.7308.

Validation accuracy: 0.6164.

- Mô hình học sâu VGG16:

Train loss: 10.283.

Train accuracy: 0.967.

Validation loss: 13.821.

Validation accuracy: 0.955.

X. Tài liệu tham khảo

Leaf Disease Detection using Support Vector Machine: https://ieeexplore.ieee.org/document/9182128

VGG16: https://medium.com/@mygreatlearning/everything-you-need-to-know-about-vgg16-7315defb5918

SVM sklearn: http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html