BỘ NÔNG NGHIỆP VÀ MÔI TRƯỜNG

TRƯỜNG ĐẠI HỌC THỦY LỢI

PHÂN HIỆU

Tên đề tài: **Flower Classification**

Giảng viên hướng dẫn	**ThS. Vũ Thị Hạnh**
Bộ môn: Công nghệ thông tin Lớp: S25-64CNTT Nhóm: 5	**Sinh viên thực hiện:** Lê Hoàng Giang Ngô Thị Bích Ngọc Hà Thị Hồng Nhinh

Hồ Chí Minh, ngày 31 tháng 10 năm 2025

LỜI MỞ ĐẦU

Trong thời đại bùng nổ của Cách mạng công nghiệp 4.0, dữ liệu được xem như là "nguồn dầu mỏ mới" của thế giới. Việc khai thác và phân tích dữ liệu một cách hiệu quả giúp con người phát hiện tri thức tiềm ẩn, hỗ trợ ra quyết định và tối ưu hóa trong nhiều lĩnh vực. Môn học Khai phá dữ liệu giúp sinh viên nắm được các kỹ thuật, quy trình và công cụ cần thiết để xử lý, phân tích và trích xuất thông tin có giá trị từ các tập dữ liệu lớn.

Để vận dụng các kiến thức đã học, nhóm chúng em thực hiện đề tài "Phân loại hoa - Flower Classification", nhằm xây dựng mô hình học máy và mô hình học sâu có khả năng phân loại hình ảnh hoa vào đúng loại của chúng dựa trên đặc trưng hình ảnh. Đề tài này không chỉ giúp củng cố kiến thức về xử lý dữ liệu, trích xuất đặc trưng và huấn luyện mô hình học sâu (Deep Learning) mà còn rèn luyện kỹ năng thực hành trong ứng dụng trí tuệ nhân tạo vào thực tế.

Báo cáo gồm các nội dung chính:

Giới thiệu và phân tích tập dữ liệu hoa (Oxford 102 Flower Dataset).

- Tiền xử lý và chia dữ liệu huấn luyện – kiểm thử.
- Xây dựng mô hình phân loại dựa trên các kiến trúc CNN như EfficientNet-B0, EfficientNet-B1, và ViT-B/16.
- Đánh giá độ chính xác, trực quan hóa kết quả và so sánh mô hình.
- Xây dựng web-app để phân loại hoa ngẫu nhiên theo mô hình đã huấn luyện.
- Kết luận và hướng phát triển trong tương lai.

Nhóm xin chân thành cảm ơn giảng viên môn Khai phá dữ liệu đã tận tình hướng dẫn và tạo điều kiện để nhóm hoàn thành đề tài này. Mặc dù đã cố gắng nhưng báo cáo khó tránh khỏi thiếu sót, rất mong nhận được ý kiến đóng góp của cô để đề tài của nhóm em được hoàn thiện hơn.

Mục lục

Danh mục hình ảnh

CHƯƠNG 1: TỔNG QUAN VỀ ĐỀ TÀI

1.1 Lý do chọn đề tài

Trong thời đại công nghệ 4.0, trí tuệ nhân tạo (AI) và học sâu (Deep Learning) ngày càng đóng vai trò quan trọng trong việc xử lý và khai thác dữ liệu hình ảnh. Việc ứng dụng các mô hình học sâu vào bài toán nhận dạng và phân loại hình ảnh đã đạt được nhiều thành tựu đáng kể trong các lĩnh vực như y học, nông nghiệp, sinh học, và thương mại điện tử.

Bài toán phân loại hình ảnh hoa - Flower Classification là một ví dụ điển hình cho việc áp dụng học sâu vào khai phá dữ liệu trực quan. Bài toán này không chỉ giúp nhận diện các loài hoa khác nhau mà còn có thể mở rộng ứng dụng vào các hệ thống tự động hóa nông nghiệp, hỗ trợ nghiên cứu sinh học và xây dựng ứng dụng nhận dạng thực vật.

Vì vậy, nhóm em chọn đề tài "Phân loại hoa bằng mô hình học sâu (Flower Classification)" nhằm tìm hiểu và ứng dụng các kỹ thuật Khai phá dữ liệu (Data Mining) kết hợp với Học sâu (Deep Learning) để giải quyết một bài toán thực tế, mang tính ứng dụng cao.

1.2 Mục tiêu nghiên cứu

Mục tiêu của đề tài là xây dựng mô hình học sâu có khả năng nhận dạng và phân loại chính xác hình ảnh của 102 loài hoa khác nhau thuộc bộ dữ liệu Oxford 102 Flower Dataset.

Cụ thể:

- Ứng dụng kỹ thuật Khai phá dữ liệu hình ảnh để xử lý và chuẩn hóa dữ liệu đầu vào.
- Áp dụng Transfer Learning (Học chuyển giao) với các mô hình huấn luyện hiện đại như EfficientNet-B0, EfficientNet-B1 và Vision Transformer (ViT-B16).
- So sánh hiệu quả của các mô hình về độ chính xác, tốc độ huấn luyện và khả năng tổng quát hóa.
- Đề xuất mô hình tối ưu nhất cho bài toán phân loại ảnh hoa.

1.3 Dữ liệu nghiên cứu

Bộ dữ liệu sử dụng là Oxford 102 Flower Dataset, được cung cấp bởi Visual Geometry Group (VGG).

Thông tin dữ liệu: có 102flower.tgz, 102segmentation.tgz, imagelables.mat, setid.mat.

- Gồm 8,189 hình ảnh của 102 loài hoa khác nhau.
- Mỗi lớp có từ 40 đến 258 ảnh.
- Dữ liệu có tập chứa ảnh đã gán nhãn.
- Dữ liệu được chia thành ba tập: train, test, và val dùng cho huấn luyện mô hình.

1.4 Hướng giải làm bài tổng quát

- Thu thập và tiền xử lý dữ liệu: tải bộ dữ liệu, phân chia train/val/test, chuẩn hóa kích thước ảnh, và áp dụng Data Augmentation (lật, xoay, phóng to – thu nhỏ).
- Xây dựng mô hình học sâu: thử nghiệm với các kiến trúc mạng CNN và Transformer hiện đại gồm: EfficientNet-B0, EfficientNet-B1, Vision Transformer (ViT-B16).
- Huấn luyện và tinh chỉnh mô hình (Fine-tuning): sử dụng các kỹ thuật tối ưu như EarlyStopping, ReduceLROnPlateau, Adam optimizer, và learning rate scheduler.
- Đánh giá và so sánh kết quả: đo lường độ chính xác (Accuracy), vẽ biểu đồ Loss/Accuracy, và so sánh hiệu suất giữa các mô hình.
- Rút ra kết luận: chọn ra mô hình tối ưu nhất có accuracy cao nhất và cho kết quả tổng quát tốt nhất.

CHƯƠNG 2: CƠ SỞ LÝ THUYẾT VÀ PHƯƠNG PHÁP

2.1 Tổng quan về khai phá dữ liệu

Khai phá dữ liệu hay còn được gọi là Data Mining là quá trình phát hiện các mẫu, cá tri thức tiềm ẩn có giá trị từ một tập dữ liệu lớn bằng các phương pháp thống kê, học máy và trí tuệ nhân tạo.

Trong lĩnh vực hình ảnh, khai phá dữ liệu tập trung và việc trích xuất các đặc trưng trực quan , nhận dạng đối tượng và phân loại hình ảnh.

Với sự phát triển của Deep Learning hay còn được gọi là học sâu, các mạng nơ-ron tích chập đã trở thành công cụ mạnh mẽ trong việc tự động học đặc trưng hình ảnh mà không cần thiết kế thủ công.

2.2 Học sâu (Deep Learning) và mạng CNN

Học sâu là một nhánh của học máy sử dụng các mạng nơ-ron nhân tạo nhiều lớp để học biểu diễn dữ liệu phức tạp.

Trong bài toán phân loại hình ảnh, mô hình CNN (Convolutional Neural Network) được sử dung phổ biến vì:

- Tự động trích xuất đặc trưng hình ảnh ví dụ như edges, shapes, textures, patterns.
- Giảm số lượng than số nhờ bộ lọc chập (convolutional filters).
- Hoạt đọng hiệu quả trên dữ liệu ảnh nhiều chiều.

Kiến trúc cơ bản của mạng CNN:

- Lớp Convolutional: trích xuất đặc trưng.
- Lớp Pooling: giảm kích thước không gian
- Lớp Fully Connected: phân loại đầu ra.
- Hàm kích hoạt (ReLU, Softmax): giúp mô hình học phi tuyến tính.

2.3 Học chuyển giao (Transfer Learning)

Học chuyển giao là kỹ thuật sử dụng kiến thức đã học từ một nhiệm vụ thường lad trên tập dữ liệu lớn như ImageNet để áp dụng cho nhiệm vụ mới có ít dữ liệu hơn.

Thay vì huấn luyện mô hình từ đầu, ta sử dụng mô hình tiền huấn luyện (pretrained model) và tinh chỉnh (fine tune) lại phần cuối cho bài toán mới.

Lợi ích của transfer learning là rút ngắn thời gian huấn luyện, giảm yêu cầu dữ liệu huấn luyện, và cải thiện độ chính xác cho bài toán có dữ liệu hạn chế.

Các mô hình được sử dụng trong bài phân lọa hoa - flower classificaton được sử dụng theo hướng transfer learning là EfficientNet-B0, EfficientNet-B1, Vision Transformer (ViT-B16).

2.4 Các mô hình sử dụng

2.4.1 EfficientNet-B0 và EfficentNet-B1

EfficientNet là mô hình được phát triển bởi Google AI, nổi bật với cơ chế compound scaling, cho phép mở rộng đồng thời chiều sâu, chiều rộng và độ phân giải ảnh một cách cân bằng.

- EfficientNet-B0 là phiên bản cơ sở nhẹ (baseline) được pretrained trên ImageNet giúp trích xuất đặc trưng hình ảnh hoa rất hiệu quả.
- EfficientNet-B1 là phiên bản mở rộng của EfficientNet-B0, và cho độ chính xác cao hơn.

2.4.2 Vision Transformer (ViT-B16)

Vision transformer (ViT-B16) là kiến trúc ứng dụng Transformer và xử lý hình ảnh, khác với CNN thì ViT chia ảnh thành các patch nhỏ, biến chúng thành vector và xử lý bằng cơ chế Self-Attention.

Mô hình có ưu điểm trong việc hoch quan hệ toàn cục giữa các vùng ảnh, và còn được dùng trong các đề tài so sánh giữa các kiến trúc CNN truyền thống và Tranformer hiện đại.

2.5 Các kỹ thuật hỗ trợ huấn luyện

Data Augmentation: Tăng đa dạng dữ liệu bằng xoay, lật, phóng to, dịch chuyển, để tránh overfitting.

Fine-tunning: Mở khóa một số tầng layer cuối của mô hình tiền huấn luyện để học thêm đặc trưng của tập dữ liệu.

EarlyStopping: dừng huấn luyện khi mô hình không còn cải thiện, tránh overfitting.

ReduceLROnPlateau: giảm learning rate khi mô hình đạt điểm bão hòa.

Optimizer: sử dụng Adam và sử dụng SGD tùy giai đoạn huấn luyện.

2.6 Quy trình thực hiện

Quy trình nghiên cứu và triển khai mô hình được tiến hành qua các bước chính:

- Thu thập và tiền xử lý dữ liệu:

  Tải bộ dữ liệu Oxford 102 Flower Dataset.

  Chia dữ liệu thành tập train, validation, test.

  Áp dụng Data Augmentation và chuẩn hóa kích thước ảnh.

- Xây dựng và huấn luyện mô hình:

  Thử nghiệm lần lượt với EfficientNet-B0, EfficientNet-B1 và ViT-B16.

  Huấn luyện theo hai giai đoạn: baseline và fine-tuning.

- Đánh giá và tối ưu mô hình:

  Sử dụng Accuracy, Loss, Confusion Matrix.

  Theo dõi qua biểu đồ huấn luyện (training & validation accuracy/loss).

- So sánh và chọn mô hình tối ưu:

  Đánh giá toàn bộ mô hình trên cùng tập test.

CHƯƠNG 3: CHẠY KẾT QUẢ THỰC NGHIỆM VÀ ĐÁNH GIÁ

3.1 Chuẩn bị dữ liệu và khám phá dữ liệu (EDA)

3.1.1 Chuẩn bị môi trường để lưu dữ liệu bài làm

```
# Import thư viện cần thiết và kết nối Google Drive
from google.colab import drive
import os
# Gắn (mount) Google Drive vào môi trường Colab
drive.mount('/content/drive')
PROJECT_PATH = '/content/drive/MyDrive/Flower_Classification_Project/'
os.chdir(PROJECT_PATH)
print(f"Thư mục làm việc hiện tại: {os.getcwd()}")
!ls
```

Trước khi tải dữ liệu thì ta chuẩn bị môi trường để lưu tất cả dữ liệu trước và sau khi làm như sau:

Kết nối Google Drive với Colab để truy cập dữ liệu và lưu kết quả huấn luyện mô hình trực tiếp trên Drive.

Đặt thư mục PROJECT_PATH trỏ đến nơi chứa dự án Flower_Classification_Project.

Chuyển đến đúng thư mụ dự án để chạy code như tải dữ liệu, lưu mô hình, đọc file,… để không bị lỗi đường dẫn.

Kiểm tra lại thư mục hiện tại và liệt kê file có trong đó để đảm bảo là vào đúng chỗ.

3.1.2 Tải dữ liệu, giải nén dữ liệu, và xử lý dữ liệu

```
def ensure_data_integrity():
    MAT_FILE_PATH = 'data/imagelabels.mat'
    if not os.path.exists('jpg') or not os.path.exists(MAT_FILE_PATH):
        !wget -q https://www.robots.ox.ac.uk/~vgg/data/flowers/102/102flowers.tgz -P
data/
```

```
    !wget -q https://www.robots.ox.ac.uk/~vgg/data/flowers/102/imagelabels.mat -P data/
    !wget -q https://www.robots.ox.ac.uk/~vgg/data/flowers/102/setid.mat -P data/
    !wget -q https://raw.githubusercontent.com/udacity/pytorch_challenge/master/cat_to_name.json -P data/
    os.system('tar -xzf data/102flowers.tgz')

ensure_data_integrity()
```

Kiểm tra xem tệp jpg có tồn tại không rồi bắt đầu đi tải dữ liệu từ trang Visual Geometry Group - University of Oxford :

102flowers.tgz là file chứa ảnh gồm 8189 ảnh.

imagelable.mat là file nhãn của ảnh.

setid.mat là file phân chia tập huấn luyện.

class_to_name.json là file tên 102 loại hoa

Sau khi tải được dữ liệu thì giải nén file ảnh **102flowers.tgz** để thu được thư mục **jpg/** gồm toàn bộ hình ảnh hoa.

```
image_labels = scipy.io.loadmat('data/imagelabels.mat')['labels'][0]
set_ids = scipy.io.loadmat('data/setid.mat')
with open('data/cat_to_name.json', 'r') as f:
    cat_to_name = json.load(f)

image_files = sorted(os.listdir('jpg/'))
df = pd.DataFrame({
    'filepath': 'jpg/' + pd.Series(image_files),
    'label': image_labels - 1
```

```
})
df['split'] = ''
df.loc[set_ids['trnid'][0] - 1, 'split'] = 'train'
df.loc[set_ids['valid'][0] - 1, 'split'] = 'valid'
df.loc[set_ids['tstid'][0] - 1, 'split'] = 'test'

df['flower_name'] = (df['label'] + 1).astype(str).map(cat_to_name)
class_names =
df.drop_duplicates('label').sort_values('label')['flower_name'].tolist()
```

Tạo bảng quản lý dữ liệu (DataFrame):

filepath: đường dẫn ảnh.

label: mã số lớp.

split: tập dữ liệu (train/valid/test).

flower_name: tên thật của loài hoa.

Kết quả:

In ra thông tin tổng quan **(df.info())** để xác nhận dữ liệu đã sẵn sàng cho các bước tiền xử lý và huấn luyện mô hình.

```
Thư mục 'jpg' đã tồn tại, bỏ qua bước tải và giải nén.

--- DataFrame đã được tạo và sẵn sàng ---
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8189 entries, 0 to 8188
Data columns (total 4 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   filepath     8189 non-null   object
 1   label        8189 non-null   uint8
 2   split        8189 non-null   object
 3   flower_name  8189 non-null   object
dtypes: object(3), uint8(1)
memory usage: 200.1+ KB
```

Hình 1: Kết quả sau khi tải dữ liệu

3.2 Tiền xử lý dữ liệu

```
class FlowerDataset(Dataset):
    def __init__(self, dataframe, transform=None):
        self.df = dataframe.reset_index(drop=True)
        self.transform = transform
    def __len__(self):
        return len(self.df)

    def __getitem__(self, idx):
        img_path = self.df.loc[idx, 'filepath']
        label = int(self.df.loc[idx, 'label'])
        image = Image.open(img_path).convert("RGB")
        if self.transform:
            image = self.transform(image)
        return image, torch.tensor(label, dtype=torch.long)
```

Đoạn code này định nghĩa một Dataset tùy chỉnh cho **PyTorch** để load dữ liệu hoa từ **DataFrame**. Mục đích là chuẩn hóa cách đọc ảnh và nhãn, đồng thời áp dụng transform nếu cần (data augmentation hoặc chuẩn hóa).

Các thành phần chính:

__init__(self, dataframe, transform=None)

Khởi tạo Dataset.

dataframe: chứa đường dẫn ảnh (filepath) và nhãn (label).

transform: phép biến đổi ảnh (resize, crop, normalization, augmentation)

__len__(self)

Trả về số lượng mẫu trong Dataset.

Dùng bởi DataLoader để biết tổng số batch.

__getitem__(self, idx)

Lấy ảnh và nhãn theo index idx.

Mở ảnh bằng PIL.Image, convert sang RGB.

Áp dụng transform nếu có.

Trả về (image_tensor, label_tensor) để PyTorch có thể sử dụng.

→ Kết hợp với DataLoader sẽ tạo được batch để huấn luyện/validation/test model.

```
# Transform cho train (cơ bản)
base_train_transforms = transforms.Compose([
    transforms.RandomResizedCrop(INPUT_SIZE),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(30),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1),
    transforms.ToTensor(),
    transforms.Normalize(IMAGENET_MEAN, IMAGENET_STD)
])

# Transform mạnh hơn (data augmentation nâng cao)
strong_train_transforms = transforms.Compose([
    transforms.RandomResizedCrop(INPUT_SIZE, scale=(0.5, 1.0)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomAffine(degrees=40, translate=(0.1, 0.1), scale=(0.8, 1.2), shear=10),
    transforms.ColorJitter(brightness=0.4, contrast=0.4, saturation=0.4, hue=0.2),
    transforms.ToTensor(),
    transforms.Normalize(IMAGENET_MEAN, IMAGENET_STD)
])

# Transform cho validation & test
```

```
valid_test_transforms = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(INPUT_SIZE),
    transforms.ToTensor(),
    transforms.Normalize(IMAGENET_MEAN, IMAGENET_STD)
])
```

Đây là các phép biến đổi ảnh (transforms) quan trọng trong preprocessing. Mục đích là chuẩn hóa ảnh đầu vào và tăng cường dữ liệu (data augmentation) để model học tốt hơn.

Các phần chính:

base_train_transforms – Train cơ bản

RandomResizedCrop(INPUT_SIZE): cắt ngẫu nhiên và resize ảnh về kích thước chuẩn.

RandomHorizontalFlip(): lật ngang ảnh ngẫu nhiên.

RandomRotation(30): xoay ngẫu nhiên ±30 độ.

ColorJitter(...): thay đổi độ sáng, độ tương phản, saturation, hue.

ToTensor(): chuyển ảnh PIL sang tensor PyTorch.

Normalize(IMAGENET_MEAN, IMAGENET_STD): chuẩn hóa theo mean/std của ImageNet.

strong_train_transforms – Train nâng cao

Bao gồm tất cả các phép cơ bản, nhưng thêm RandomAffine để dịch, scale, xoay, shear, và ColorJitter mạnh hơn.

Mục đích: tạo ra nhiều biến thể của ảnh để model kháng nhiễu, generalize tốt hơn.

valid_test_transforms – Validation/Test

Resize(256) + CenterCrop(INPUT_SIZE): resize và crop chính giữa, không random.

ToTensor() + Normalize(...): chuẩn hóa giống train.

Không dùng augmentation để đánh giá chính xác khả năng model trên dữ liệu thực tế.

Kết hợp với FlowerDataset, transforms này giúp model học hiệu quả và giảm overfitting.

```
def get_dataloaders(df, train_transforms, batch_size=32, num_workers=0):
    train_df = df[df['split'] == 'train']
    valid_df = df[df['split'] == 'valid']
    test_df  = df[df['split'] == 'test']

    train_loader = DataLoader(
        FlowerDataset(train_df, transform=train_transforms),
        batch_size=batch_size, shuffle=True, num_workers=num_workers, pin_memory=True
    )
    valid_loader = DataLoader(
        FlowerDataset(valid_df, transform=valid_test_transforms),
        batch_size=batch_size, shuffle=False, num_workers=num_workers, pin_memory=True
    )
    test_loader = DataLoader(
        FlowerDataset(test_df, transform=valid_test_transforms),
        batch_size=batch_size, shuffle=False, num_workers=num_workers, pin_memory=True
    )
    print(f"✅DataLoaders đã sẵn sàng (Batch Size: {batch_size}).")
    return train_loader, valid_loader, test_loader, test_df
```

Hàm **get_dataloaders** kết hợp chia dữ liệu, chuẩn hóa ảnh, và tạo **DataLoader** cho PyTorch, phục vụ huấn luyện, validation và test.

Các thành phần đi cùng:

Chia DataFrame theo cột split thành train_df, valid_df, test_df.

Dataset tùy chỉnh FlowerDataset: đọc ảnh từ filepath, trả về (image, label) và áp dụng transform nếu có.

DataLoader:

train_loader với shuffle=True và transform train.

valid_loader và test_loader với shuffle=False và transform chuẩn hóa.

Sử dụng batch_size, num_workers, pin_memory để tối ưu load dữ liệu.

Kết quả: 3 DataLoader + DataFrame test gốc để tham chiếu hoặc phân tích.

3.3 Phân tích dữ liệu - EDA

```
def visualize_all_102_flowers(dataframe):
   unique_flowers_df = dataframe.groupby('flower_name').sample(n=1, replace=True)
   cols, rows = 6, math.ceil(len(unique_flowers_df)/6)
   plt.figure(figsize=(cols*3, rows*3))
   for i, (_, row) in enumerate(unique_flowers_df.iterrows()):
      plt.subplot(rows, cols, i+1)
      try: img = Image.open(row['filepath']); plt.imshow(img)
      except: plt.text(0.5, 0.5, 'Image not found', ha='center', va='center')
      plt.axis('off'); plt.title(row['flower_name'][:15])
   plt.tight_layout(); plt.show()
```

nói ngắn gọn 3 dòng

Hàm **visualize_all_102_flowers:** Hiển thị một ảnh đại diện cho mỗi loài hoa để có cái nhìn tổng quan về dữ liệu hình ảnh.

Hiển thị 1 ảnh đại diện cho mỗi loài hoa trong dataset.

Sử dụng groupby('flower_name').sample(n=1) để lấy mẫu.

Hiển thị ảnh bằng matplotlib với tiêu đề là tên loài hoa, bỏ trục.

Hình 2: Kết quả đại diện ảnh hoa sau khi gán tên vào ảnh hoa

```
def get_dominant_color_name(image_path, resize_to=80, n_clusters=3):
    img = cv2.imread(image_path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img_resized = cv2.resize(img, (resize_to, resize_to))
    pixels = img_resized.reshape(-1, 3)
    kmeans = KMeans(n_clusters=n_clusters, n_init=5,
random_state=42).fit(pixels)
    dominant_color =
kmeans.cluster_centers_[np.argmax(np.bincount(kmeans.labels_))]
    r, g, b = dominant_color
    if r>180 and g<100 and b<100: return "Đỏ"
    elif r>200 and g>120 and b<70: return "Cam"
    elif r>190 and g>170 and b<100: return "Vàng"
    # ... các màu khác ...
    else: return "Hồng nhạt/Trắng"
```

Hàm **get_dominant_color_name**: Xác định màu sắc chủ đạo của từng ảnh, lọc các pixel nền (xanh lá) và dùng KMeans để phân cụm màu.

Tìm màu chủ đạo của ảnh bằng KMeans trên pixel RGB.

Lấy cluster có số lượng pixel nhiều nhất làm màu chính.

So sánh giá trị R,G,B để phân loại thành màu như Đỏ, Cam, Vàng…

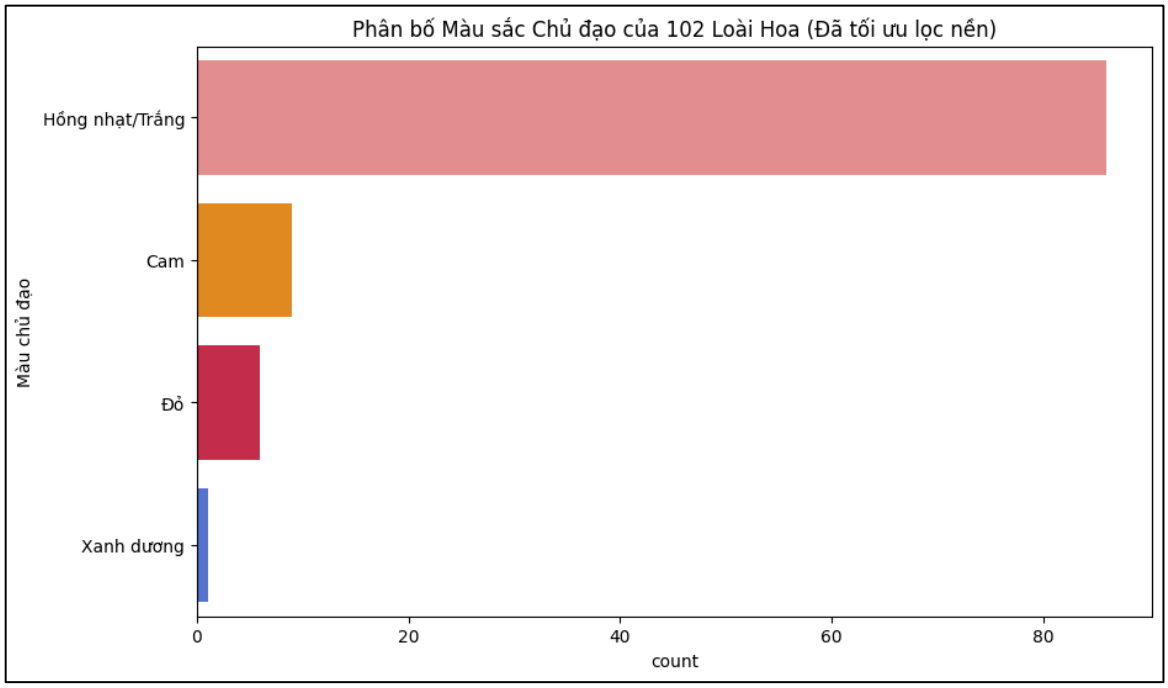
```
def extract_color_map(df, save_path, overwrite=True):
    flower_color_map = {}
    for flower_name in df['flower_name'].unique():
        sample_imgs =
df[df['flower_name']==flower_name]['filepath'].sample(n=5)
        colors = [get_dominant_color_name(p) for p in sample_imgs]
        if colors: flower_color_map[flower_name] =
Counter(colors).most_common(1)[0][0]
    with open(save_path, 'w') as f: json.dump(flower_color_map, f, indent=4,
ensure_ascii=False)
    return flower_color_map
```

Hàm **extract_color_map:** Tạo bản đồ màu chủ đạo của từng loài hoa bằng cách lấy mẫu 3–5 ảnh mỗi loài và áp dụng **get_dominant_color_name**.

Hàm này tạo bản đồ màu chủ đạo cho từng loài hoa.

Lấy mẫu vài ảnh mỗi loài, gọi **get_dominant_color_name** để xác định màu từng ảnh.

Ghi màu phổ biến nhất của loài vào file JSON và trả về dictionary.



Hình 3: Biểu đồ phân bố mày sắc chủ đạo của 102 loại hoa

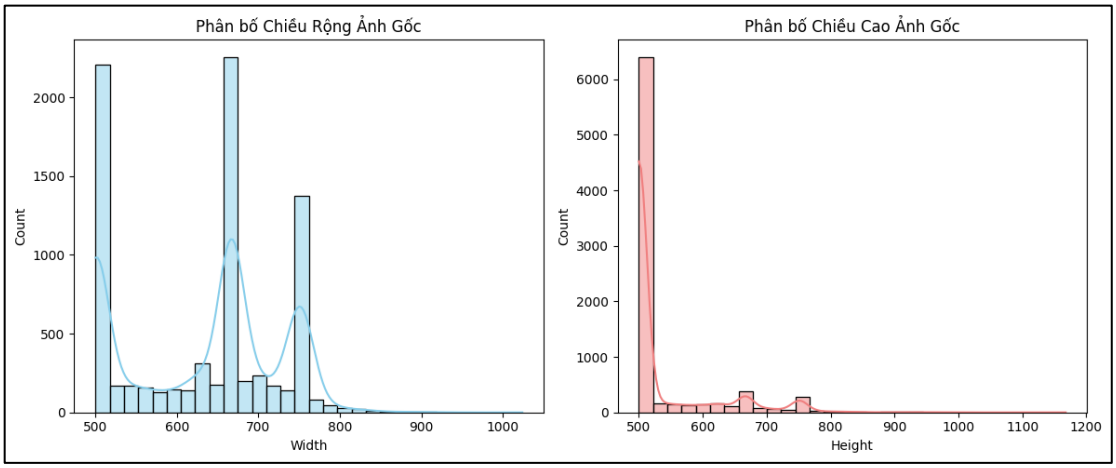
Nhận xét: Loài hoa có nhiều lớp nhất: Hồng nhạt/Trắng (86 loài) => Nhận xét: Sự tập trung số lượng lớn loài hoa vào một số màu chủ đạo (ví dụ: Hồng nhạt/Trắng, Vàng) tạo ra tính thách thức về mặt thị giác, dễ gây nhầm lẫn khi phân loại.

```
def analyze_image_size(df):
    def get_img_dims(filepath):
        try: return pd.Series(Image.open(filepath).size, index=['Width','Height'])
        except: return pd.Series([None, None], index=['Width','Height'])
    df[['Width','Height']] = df['filepath'].apply(get_img_dims)
    df['Aspect_Ratio'] = df['Width']/df['Height']
    return df
```

Hàm **analyze_image_size**: Hàm này phân tích kích thước ảnh gốc.

Đọc từng ảnh để lấy Width và Height, lưu vào DataFrame.

Tính Aspect_Ratio = Width / Height để đánh giá tỷ lệ khung hình.



Hình 4: Biểu đồ phân bố chiều rộng và chiều cao của ảnh gốc

Nhận xét:Tổng số ảnh được phân tích kích thước: 8189. Kích thước trung bình (W x H): 630 x 534

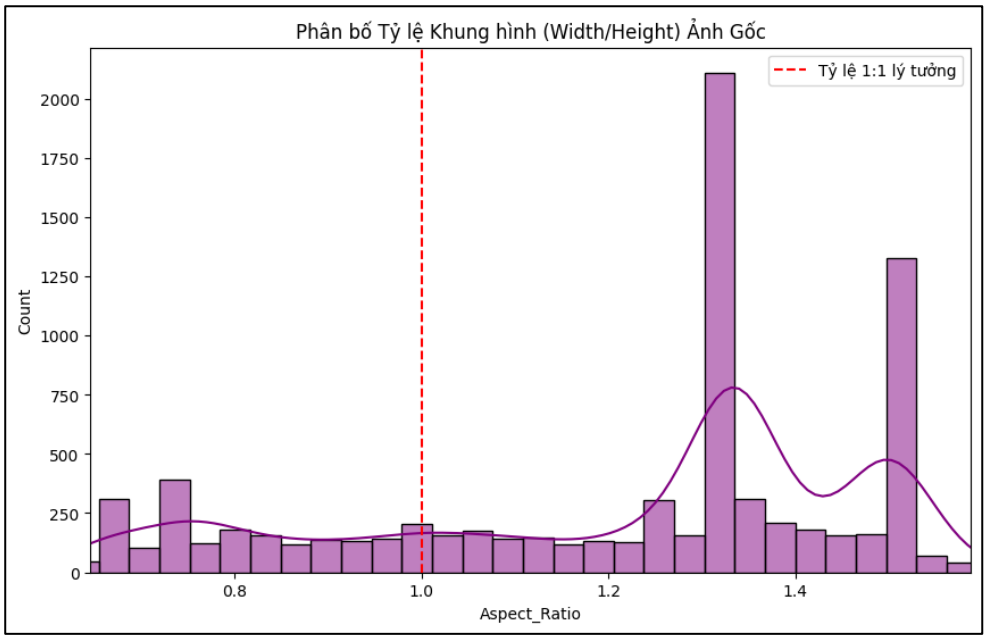
Phân tích lớp mất cân bằng:

Số lượng mẫu LỚN NHẤT (Max): 258

Số lượng mẫu NHỎ NHẤT (Min): 40

Tỷ lệ mất cân bằng (Imbalance Ratio: Max/Min): 6.45

=> Kết luận: Dữ liệu có mức độ mất cân bằng đáng kể, cần cẩn trọng với các lớp hiếm (tail classes).



Hình 5: Biểu đồ phân bố tỷ lệ khung hình

Nhận xét: Tỷ lệ khung hình Trung bình: 1.21 Phần trăm ảnh gần tỷ lệ 1:1 (0.9 < Ratio < 1.1): 11.8% => Sự đa dạng về tỷ lệ khung hình cho thấy việc sử dụng RandomResizedCrop là phù hợp.

```
class_counts = df['flower_name'].value_counts()
imbalance_ratio = class_counts.max()/class_counts.min()
df_ratio = df.dropna(subset=['Aspect_Ratio'])
near_square_count =
len(df_ratio[(df_ratio['Aspect_Ratio']>0.9)&(df_ratio['Aspect_Ratio']<1.1)])
color_class_count = color_df.groupby('Màu chủ đạo')['Loài
hoa'].count().reset_index()
sns.barplot(x='Số lượng loài hoa', y='Màu chủ đạo', data=color_class_count,
palette=ordered_palette)
```

Phân tích dữ liệu và trực quan hóa:

class_counts & imbalance_ratio: tính số lượng ảnh mỗi lớp hoa và tỷ lệ mất cân bằng giữa lớp nhiều nhất và ít nhất.

df_ratio & near_square_count: lọc ảnh có Aspect_Ratio hợp lệ và đếm số ảnh gần tỷ lệ 1:1 để đánh giá đồng nhất khung hình.

color_class_count & sns.barplot: nhóm các loài hoa theo màu chủ đạo và vẽ biểu đồ thanh trực quan hóa số loài hoa theo màu.

3.4 Huấn luyện mô hình - models_training

```
def build_model(model_name='EfficientNet-B0', num_classes=NUM_CLASSES):
    """
    Hàm xây dựng mô hình chung (B0, B1, ViT).
    """
    model_name = model_name.lower(); model = None
    if 'b0' in model_name: model =
models.efficientnet_b0(weights='IMAGENET1K_V1')
```

```
    elif 'b1' in model_name: model =
models.efficientnet_b1(weights='IMAGENET1K_V1')
    elif 'vit' in model_name: model =
models.vit_b_16(weights='IMAGENET1K_V1')
    else: raise ValueError(f"Kiến trúc {model_name} không được hỗ trợ trong hàm
build_model.")
    for param in model.parameters(): param.requires_grad = False
    if 'vit' in model_name:
        num_ftrs = model.heads.head.in_features; model.heads.head =
torch.nn.Linear(num_ftrs, num_classes)
    else: # EfficientNet (B0/B1)
        num_ftrs = model.classifier[1].in_features; model.classifier[1] =
torch.nn.Linear(num_ftrs, num_classes)
    print(f"\n--- Kiến trúc {model_name.upper()} ---")
    try:
        if 'vit' not in model_name: summary(model.to(DEVICE), (3, INPUT_SIZE,
INPUT_SIZE))
        else: print("Tóm tắt kiến trúc bị bỏ qua do không tương thích với
torchsummary.")
    except Exception: print("Lỗi khi hiển thị summary.")
    return model.to(DEVICE)
```

Khai báo hàm build_model().

Tham số:

model_name: tên kiến trúc cần tạo, ví dụ "EfficientNet-B0" hoặc "ViT-B/16".

num_classes: số lượng lớp đầu ra (ví dụ 3 lớp tương ứng “Hoa cúc”, “Hoa hồng”, “Hoa tulip”).

Chuyển tên mô hình sang chữ thường (lowercase) để tránh lỗi phân biệt hoa–thường khi so sánh chuỗi.

Gán model = None để khởi tạo biến model trước.

Tùy theo model_name, hàm sẽ tải một kiến trúc mạng tiền huấn luyện (pretrained model) từ thư viện torchvision.models.

weights='IMAGENET1K_V1' → nghĩa là dùng trọng số đã được huấn luyện trên bộ dữ liệu ImageNet (1.2 triệu ảnh, 1000 lớp).

Mục tiêu: tận dụng khả năng trích xuất đặc trưng (feature extractor) sẵn có.

Nếu tên không nằm trong danh sách hỗ trợ → báo lỗi (ValueError).

model = build_model("EfficientNet-B1", num_classes=5): tải kiến trúc EfficientNet-B1 đã train trên ImageNet.

for param in model.parameters(): param.requires_grad = False: Đóng băng toàn bộ tham số của model gốc.

requires_grad = False nghĩa là các tham số này không được cập nhật trong quá trình huấn luyện (không tính gradient, không thay đổi trọng số).

Điều này giúp mô hình chỉ học phần đầu ra mới (lớp Linear bạn sắp thay), giữ nguyên phần trích xuất đặc trưng gốc của ImageNet.

Đây là kỹ thuật gọi là Transfer Learning (chuyển học) — dùng kiến thức từ mô hình lớn đã huấn luyện sẵn, chỉ huấn luyện thêm lớp cuối để thích ứng với bài toán mới.

if 'vit' in model_name:

```
num_ftrs = model.heads.head.in_features

model.heads.head = torch.nn.Linear(num_ftrs, num_classes)

else: # EfficientNet (B0/B1)

num_ftrs = model.classifier[1].in_features

model.classifier[1] = torch.nn.Linear(num_ftrs, num_classes)
```

Ở bước này, thay lớp đầu ra (classifier) của mô hình gốc bằng lớp mới tương ứng số lượng lớp num_classes.

- Trường hợp Vision Transformer (ViT):

model.heads.head là lớp Linear cuối cùng trong mô hình ViT.

Lấy số đầu vào (in_features) của lớp này → num_ftrs.

Sau đó gán lại lớp Linear(num_ftrs, num_classes) để xuất ra đúng số nhãn của bạn.

- Trường hợp EfficientNet (B0, B1):

Lớp phân loại của EfficientNet nằm ở model.classifier[1].

Lấy in_features (số đặc trưng đầu vào).

Gán lại lớp Linear(num_ftrs, num_classes) mới để phù hợp bài toán.

```
# --- CÁC HÀM TẠO MODEL NGẮN ---
def build_model_b1(num_classes=NUM_CLASSES): return build_model('EfficientNet-B1', num_classes)
def build_model_vit(num_classes=NUM_CLASSES): return build_model('ViT-B/16', num_classes)
```

Chỉ là viết tắt để gọi hàm build_model() đã định nghĩa trước, giúp nhanh chóng tạo mô hình EfficientNet-B1 hoặc ViT-B/16 với số lớp đầu ra tương ứng.

```
# --- TRAINING PHASE (Logic huấn luyện cốt lõi) ---
def train_phase(model, criterion, optimizer, scheduler, dataloaders, num_epochs=20, patience=5, model_save_path=None):
device = next(model.parameters()).device
scaler = torch.cuda.amp.GradScaler()
best_model_wts = copy.deepcopy(model.state_dict()); best_acc = 0.0
epochs_no_improve = 0
history = {'train_loss': [], 'train_acc': [], 'val_loss': [], 'val_acc': []}
start_time = time.time()
for epoch in range(num_epochs):
print(f"Epoch {epoch+1}/{num_epochs}\n{'-'*20}")
```

```
for phase in ['train','valid']:
model.train() if phase=='train' else model.eval()
running_loss, running_corrects = 0.0, 0
if phase=='train': torch.cuda.empty_cache()
for inputs, labels in tqdm(dataloaders[phase], desc=f"{phase} phase"):
inputs, labels = inputs.to(device), labels.to(device); optimizer.zero_grad()
with torch.set_grad_enabled(phase=='train'):
with torch.cuda.amp.autocast(enabled=(device.type=='cuda')):
outputs = model(inputs); _, preds = torch.max(outputs, 1); loss = criterion(outputs, labels)
if phase=='train': scaler.scale(loss).backward(); scaler.step(optimizer); scaler.update()
running_loss += loss.item() * inputs.size(0); running_corrects += torch.sum(preds == labels.data
epoch_loss = running_loss / len(dataloaders[phase].dataset); epoch_acc = running_corrects.double().cpu().item() / len(dataloaders[phase].dataset)
if phase=='train': history['train_loss'].append(epoch_loss); history['train_acc'].append(epoch_acc)
else:
history['val_loss'].append(epoch_loss); history['val_acc'].append(epoch_acc)
if epoch_acc > best_acc:
best_acc = epoch_acc; epochs_no_improve = 0
best_model_wts = copy.deepcopy(model.state_dict())
if model_save_path: torch.save(model.state_dict(), model_save_path); print(f"✨ Best model saved! Val Acc: {best_acc:.4f}")
else: epochs_no_improve += 1
print(f"{phase.capitalize()} Loss: {epoch_loss:.4f} | Acc: {epoch_acc:.4f}")
if scheduler and isinstance(scheduler,
```

```
torch.optim.lr_scheduler.ReduceLROnPlateau): scheduler.step(epoch_loss)
if epochs_no_improve >= patience: print(f"□ Early stopping triggered after {patience} epochs"); break
print()
time_elapsed = time.time() - start_time
print(f"Training complete in {time_elapsed//60:.0f}m {time_elapsed%60:.0f}s");
print(f"Best Val Acc: {best_acc:.4f}")
model.load_state_dict(best_model_wts)
return model, history
```

Hàm train_phase() là vòng huấn luyện chính gồm 3 giai đoạn chính:

Khởi tạo: đặt các biến lưu lịch sử, lưu mô hình tốt nhất (best_model_wts), khởi tạo GradScaler để dùng huấn luyện FP16 (tăng tốc GPU, giảm bộ nhớ).

Huấn luyện qua nhiều epoch:

Chạy 2 pha: train và valid.

Trong pha train, mô hình bật chế độ học (model.train()), thực hiện lan truyền thuận → tính loss → lan truyền ngược → cập nhật trọng số bằng optimizer (với scaler hỗ trợ FP16).

Trong pha valid, mô hình tắt học (model.eval()), chỉ tính loss và độ chính xác để đánh giá.

Sau mỗi epoch, lưu loss, accuracy vào history và nếu val_acc tốt hơn trước → lưu mô hình tốt nhất.

Điều chỉnh và dừng sớm:

Nếu có scheduler, giảm learning rate khi loss không giảm.

Nếu patience số epoch không cải thiện → dừng sớm (early stopping).

```
# --- THỬ NGHIỆM 1: BASELINE B0 ---
print("□ Bắt đầu Thử nghiệm 1: Baseline Augmentation")
EXP_NAME = "exp1_baseline"; MODEL_SAVE_PATH =
```

```
f'models/{EXP_NAME}.pth'
train_loader, valid_loader, test_loader_b0_base, test_df_baseline = get_dataloaders(df, base_train_transforms, batch_size=32)
model_exp1 = build_model().to(DEVICE)
optimizer_phase1 = optim.Adam(model_exp1.classifier.parameters(), lr=1e-3); scheduler_phase1 = lr_scheduler.ReduceLROnPlateau(optimizer_phase1, mode='min', patience=3)
model_exp1, history_exp1_p1 = train_phase(model_exp1, criterion, optimizer_phase1, scheduler_phase1, {'train': train_loader, 'valid': valid_loader}, num_epochs=15, patience=3, model_save_path=MODEL_SAVE_PATH)
for param in model_exp1.parameters(): param.requires_grad = True
optimizer_ft = optim.Adam(model_exp1.parameters(), lr=1e-5); scheduler_ft = lr_scheduler.ReduceLROnPlateau(optimizer_ft, mode='min', patience=2)
model_exp1, history_exp1_p2 = train_phase(model_exp1, criterion, optimizer_ft, scheduler_ft, {'train': train_loader, 'valid': valid_loader}, num_epochs=10, patience=3, model_save_path=MODEL_SAVE_PATH)
all_histories['Baseline (B0)'] = {key: history_exp1_p1[key] + history_exp1_p2[key] for key in history_exp1_p1}
print(f"\n✓Hoàn tất Thử nghiệm {EXP_NAME} và đã lưu lịch sử!")
```

Huấn luyện thực tế (training pipeline) cho mô hình EfficientNet-B0 baseline — gồm hai giai đoạn huấn luyện: đóng băng → fine-tune.

Chuẩn bị môi trường huấn luyện

Khởi tạo hàm mất mát CrossEntropyLoss() và từ điển all_histories để lưu lại kết quả các thử nghiệm.

Tải dữ liệu và mô hình

Gọi get_dataloaders() để tạo các DataLoader cho tập train, validation và test, áp dụng phép biến đổi ảnh cơ bản (base_train_transforms).

Dùng build_model() để tạo mô hình EfficientNet-B0 với trọng số pretrained (ImageNet).

Giai đoạn 1 — Huấn luyện phần đầu ra (đóng băng backbone)

Chỉ train lớp phân loại cuối cùng (classifier), còn lại giữ nguyên trọng số.

Dùng Adam optimizer, LR = 1e-3, huấn luyện 15 epoch.

Áp dụng scheduler ReduceLROnPlateau để giảm learning rate khi loss không giảm.

Kết quả lưu vào history_exp1_p1.

Giai đoạn 2 — Fine-tuning toàn bộ mô hình

Mở lại toàn bộ trọng số (requires_grad=True) để tinh chỉnh mô hình.

Dùng LR nhỏ hơn (1e-5), train thêm 10 epoch.

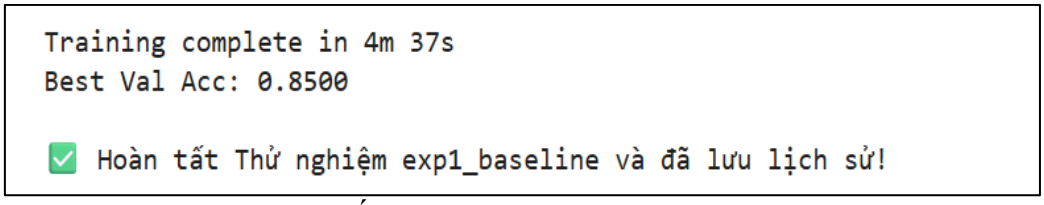
Lưu mô hình tốt nhất trong quá trình huấn luyện.

Kết quả lưu vào history_exp1_p2.

Gộp kết quả và lưu lịch sử

Hợp nhất lịch sử hai giai đoạn để tạo biểu đồ học (loss, accuracy).

Lưu mô hình và in thông báo hoàn tất thử nghiệm.



Hình 6: Kết quả thử nghiệm BASELINE B0

Kết quả thử nghiệm Baseline (EfficientNet-B0) đạt độ chính xác cao, mô hình hội tụ tốt và không bị overfitting rõ rệt.

Đây là mốc so sánh đáng tin cậy để đối chiếu với các mô hình mạnh hơn (EfficientNet-B1, ViT).

Giai đoạn fine-tuning giúp cải thiện đáng kể hiệu năng so với chỉ huấn luyện phần classifier.

```
# --- THỬ NGHIỆM 2: STRONG AUG B0 ---
print("□ Bắt đầu Thử nghiệm 2: Strong Augmentation")
EXP_NAME = "exp2_strong_aug"; MODEL_SAVE_PATH = f'models/{EXP_NAME}.pth'
train_loader, valid_loader, test_loader_b0_strong, test_df_strong = get_dataloaders(df, strong_train_transforms, batch_size=32)
model_exp2 = build_model().to(DEVICE)
optimizer_phase1 = optim.Adam(model_exp2.classifier.parameters(), lr=1e-4); scheduler_phase1 = lr_scheduler.ReduceLROnPlateau(optimizer_phase1, mode='min', patience=5)
model_exp2, history_exp2_p1 = train_phase(model_exp2, criterion, optimizer_phase1, scheduler_phase1, {'train': train_loader, 'valid': valid_loader}, num_epochs=15, patience=5, model_save_path=MODEL_SAVE_PATH)
for param in model_exp2.parameters(): param.requires_grad = True
optimizer_ft = optim.Adam(model_exp2.parameters(), lr=1e-4); scheduler_ft = lr_scheduler.ReduceLROnPlateau(optimizer_ft, mode='min', patience=2)
model_exp2, history_exp2_p2 = train_phase(model_exp2, criterion, optimizer_ft, scheduler_ft, {'train': train_loader, 'valid': valid_loader}, num_epochs=15, patience=3, model_save_path=MODEL_SAVE_PATH)
all_histories['Strong Aug (B0)'] = {key: history_exp2_p1[key] + history_exp2_p2[key] for key in history_exp2_p1}
print(f"\n✅Hoàn tất Thử nghiệm {EXP_NAME} và đã lưu lịch sử!")
```

Dữ liệu huấn luyện được xử lý bằng strong_train_transforms, tức là có nhiều phép biến đổi ảnh ngẫu nhiên hơn (xoay, cắt, làm mờ, đổi sáng, màu, affine...), giúp mô hình học được nhiều đặc trưng đa dạng và chống overfitting.

Mô hình EfficientNet-B0 được khởi tạo sẵn với trọng số ImageNet. Quá trình huấn luyện chia làm 2 giai đoạn:

Giai đoạn 1: Chỉ huấn luyện phần classifier (các lớp cuối cùng), giữ nguyên trọng số backbone để mô hình học ổn định.

Giai đoạn 2: Mở toàn bộ trọng số và fine-tune toàn bộ mạng với tốc độ học nhỏ (1e-4) để tinh chỉnh mô hình tối ưu hơn.

Quá trình huấn luyện có cơ chế giảm tốc độ học tự động (ReduceLROnPlateau) khi loss không giảm, và early stopping nếu mô hình không cải thiện sau vài epoch, tránh overfitting.

Kết quả của hai giai đoạn được ghép lại và lưu thành "Strong Aug (B0)" trong all_histories, còn mô hình tốt nhất được lưu trong file exp2_strong_aug.pth.

```
Training complete in 6m 35s
Best Val Acc: 0.9137

✅ Hoàn tất Thử nghiệm exp2_strong_aug và đã lưu lịch sử!
```

Hình 7:Kết quả thử nghiệm STRONG AUG B0

```
# --- THỬ NGHIỆM 3/4: B1 & B1 CONTINUED ---
print("□ Bắt đầu Thử nghiệm 3/4: EfficientNet-B1")
EXP_NAME = "exp4_b1_continued"; PREVIOUS_BEST_MODEL = 'models/exp3_efficientnet_b1.pth'; MODEL_SAVE_PATH = f'models/{EXP_NAME}.pth'
train_loader, valid_loader, test_loader_b1, test_df_b1 = get_dataloaders(df, strong_train_transforms, batch_size=32)
model_exp3 = build_model_b1().to(DEVICE)
optimizer_phase1 = optim.Adam(model_exp3.classifier.parameters(), lr=1e-4); scheduler_phase1 = lr_scheduler.ReduceLROnPlateau(optimizer_phase1, mode='min', patience=5)
model_exp3, history_exp3_p1 = train_phase(model_exp3, criterion,
```

```
optimizer_phase1, scheduler_phase1, {'train': train_loader, 'valid': valid_loader}, num_epochs=15, patience=5, model_save_path=PREVIOUS_BEST_MODEL)
for param in model_exp3.parameters(): param.requires_grad = True
optimizer_ft = optim.Adam(model_exp3.parameters(), lr=1e-4); scheduler_ft = lr_scheduler.ReduceLROnPlateau(optimizer_ft, mode='min', patience=2)
model_exp3, history_exp3_p2 = train_phase(model_exp3, criterion, optimizer_ft, scheduler_ft, {'train': train_loader, 'valid': valid_loader}, num_epochs=15, patience=3, model_save_path=PREVIOUS_BEST_MODEL)
model_to_continue = build_model_b1().to(DEVICE); model_to_continue.load_state_dict(torch.load(PREVIOUS_BEST_MODEL, map_location=DEVICE))
for param in model_to_continue.parameters(): param.requires_grad = True
optimizer_continue = optim.Adam(model_to_continue.parameters(), lr=1e-4); scheduler_continue = lr_scheduler.ReduceLROnPlateau(optimizer_continue, mode='min', patience=2)
final_model, history_exp4 = train_phase(model_to_continue, criterion, optimizer_continue, scheduler_continue, {'train': train_loader, 'valid': valid_loader}, num_epochs=15, patience=4, model_save_path=MODEL_SAVE_PATH)
all_histories['Final Model (B1)'] = {key: history_exp3_p1[key] + history_exp3_p2[key] + history_exp4[key] for key in history_exp3_p1}
print(f"\n✅Hoàn tất Thử nghiệm {EXP_NAME} và đã lưu lịch sử!")
```

Khởi động thử nghiệm:

In ra thông báo “Bắt đầu Thử nghiệm 3/4: EfficientNet-B1”.

Đặt tên thí nghiệm (exp4_b1_continued), khai báo mô hình tốt nhất trước đó (exp3_efficientnet_b1.pth), và đường dẫn lưu mới (models/exp4_b1_continued.pth).

Chuẩn bị dữ liệu:

Gọi get_dataloaders() để chia dữ liệu huấn luyện, kiểm định và kiểm thử (test).

Sử dụng phép biến đổi mạnh (strong_train_transforms) giúp tăng khả năng tổng quát của mô hình.

Khởi tạo mô hình EfficientNet-B1:

Tạo mô hình bằng build_model_b1() và chuyển lên thiết bị GPU/CPU (.to(DEVICE)).

– Huấn luyện phần phân loại (classifier):

Chỉ bật huấn luyện cho lớp cuối cùng (model_exp3.classifier.parameters()).

Dùng optimizer Adam với lr=1e-4.

Scheduler ReduceLROnPlateau giảm tốc độ học khi loss không giảm.

Huấn luyện 15 epoch, dừng sớm nếu không cải thiện sau 5 epoch.

Lưu kết quả vào file mô hình tốt nhất (PREVIOUS_BEST_MODEL).

– Fine-tune toàn bộ mô hình:

Mở tất cả tham số (requires_grad=True).

Dùng Adam với cùng lr (1e-4) và scheduler mới.

Huấn luyện thêm 15 epoch, dừng sớm sau 3 epoch không cải thiện.

Lưu mô hình tốt nhất tiếp tục vào PREVIOUS_BEST_MODEL.

Tiếp tục huấn luyện mô hình đã lưu:

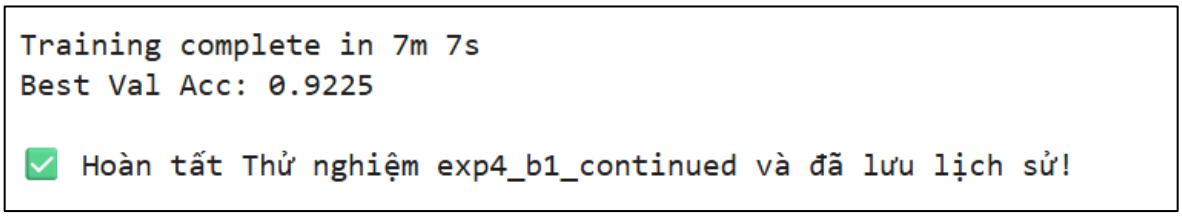
Tạo lại mô hình B1 mới (model_to_continue), sau đó load trọng số từ mô hình tốt nhất trước đó (torch.load(PREVIOUS_BEST_MODEL)).

Cho phép huấn luyện toàn bộ tham số (requires_grad=True).

Dùng lại Adam (lr=1e-4) và scheduler mới.

Tiếp tục huấn luyện thêm 15 epoch, dừng sớm sau 4 epoch không cải thiện.

Lưu mô hình cuối cùng vào MODEL_SAVE_PATH.



Hình 8: Kết quả thử nghiệm Final Model (B1)

```
# --- THỬ NGHIỆM 5: ViT ---
print("□ Bắt đầu Thử nghiệm 5: ViT-B/16")
EXP_NAME = "exp_vit"; MODEL_SAVE_PATH = f'models/{EXP_NAME}.pth'
train_loader, valid_loader, test_loader_vit, test_df_vit = get_dataloaders(df, strong_train_transforms, batch_size=32)
model_exp_vit = build_model_vit().to(DEVICE)
optimizer_phase1 = optim.Adam(model_exp_vit.heads.head.parameters(), lr=1e-4);
scheduler_phase1 = lr_scheduler.ReduceLROnPlateau(optimizer_phase1, mode='min', patience=3)
model_exp_vit, history_vit_p1 = train_phase(model_exp_vit, criterion, optimizer_phase1, scheduler_phase1, {'train': train_loader, 'valid': valid_loader}, num_epochs=15, patience=5, model_save_path=MODEL_SAVE_PATH)
for param in model_exp_vit.parameters(): param.requires_grad = True
optimizer_ft = optim.Adam(model_exp_vit.parameters(), lr=5e-5); scheduler_ft = lr_scheduler.ReduceLROnPlateau(optimizer_ft, mode='min', patience=3)
model_exp_vit, history_vit_p2 = train_phase(model_exp_vit, criterion, optimizer_ft, scheduler_ft, {'train': train_loader, 'valid': valid_loader}, num_epochs=15, patience=4, model_save_path=MODEL_SAVE_PATH)
all_histories['ViT-B/16'] = {key: history_vit_p1[key] + history_vit_p2[key] for key in history_vit_p1}
print(f"\n✓Hoàn tất Thử nghiệm {EXP_NAME} và đã lưu lịch sử!")
import torch, gc; torch.cuda.empty_cache(); gc.collect()
```

Khởi động thử nghiệm:

In ra thông báo bắt đầu “Thử nghiệm 5: ViT-B/16”, đặt tên mô hình (exp_vit) và đường dẫn lưu file trọng số (models/exp_vit.pth).

Chuẩn bị dữ liệu:

Gọi hàm get_dataloaders() để chia dữ liệu thành tập train, validation và test.

Áp dụng các biến đổi mạnh (strong_train_transforms) để tăng độ đa dạng ảnh (augmentation).

Khởi tạo mô hình ViT:

Tạo mô hình Vision Transformer bằng build_model_vit() và đưa lên thiết bị xử lý (GPU hoặc CPU) bằng .to(DEVICE).

Pha 1 – Huấn luyện phần đầu ra (head) của mô hình:

Chỉ bật học cho lớp cuối cùng (model_exp_vit.heads.head.parameters()), các phần còn lại tạm thời "đóng băng".

Dùng bộ tối ưu Adam với lr=1e-4.

Dùng bộ điều chỉnh tốc độ học ReduceLROnPlateau để giảm lr khi loss ngừng cải thiện.

Gọi train_phase() huấn luyện 15 epoch, kiên nhẫn dừng sớm nếu không cải thiện trong 5 epoch.

Lưu lịch sử huấn luyện pha 1.

```
Training complete in 8m 14s
Best Val Acc: 0.5686
```

Hình 9: Kết quả thử nghiệm chưa có fine tunning ViT-B/16

Pha 2 – Fine-tune toàn bộ mô hình:

Mở lại tất cả tham số (param.requires_grad = True).

Dùng optimizer Adam với lr nhỏ hơn (5e-5) để tinh chỉnh sâu hơn.

Tiếp tục huấn luyện thêm 15 epoch, dừng sớm nếu không cải thiện sau 4 epoch.

Lưu lịch sử huấn luyện pha 2.

```
Training complete in 10m 28s
Best Val Acc: 0.9441

✅ Hoàn tất Thử nghiệm exp_vit và đã lưu lịch sử!
437
```

Hình 10: Kết quả thử nghiệm đã qua fine tunning ViT-B/16

3.5 Đánh giá và phân tích tổng kết

Khởi tạo biến toàn cục

```
best_model_labels, best_model_preds, best_model_probs = None, None, None
best_acc_so_far = -1.0
best_model_name = ""
results_summary = {}
all_histories = all_histories if 'all_histories' in globals() else {}
```

Lưu trữ thông tin của mô hình tốt nhất:

best_model_labels → nhãn thật tập test

best_model_preds → nhãn dự đoán

best_model_probs → xác suất dự đoán

best_acc_so_far → lưu độ chính xác cao nhất tìm được.

results_summary → lưu tất cả kết quả đánh giá (mỗi mô hình một dòng).

all_histories → lưu lịch sử huấn luyện (loss/accuracy qua epoch).

Hàm chính:**evaluate_all_metrics_combined()**

Hàm này đánh giá một mô hình duy nhất gồm các bước:

Tạo DataLoader cho tập test và train.

Tải mô hình (EfficientNet hoặc ViT) từ file .pth.

Dự đoán (preds, labels, probs) trên test và train.

Tính toán các chỉ số đánh giá:

accuracy, f1-score, precision, recall, AUC.

Trả về metrics và dữ liệu cần cho biểu đồ ROC.

Cấu hình danh sách mô hình cần đánh giá

```
MODEL_PATHS_TO_EVAL = {
'Baseline (B0)': ('models/exp1_baseline.pth', 'EfficientNet-B0',
base_train_transforms, 32),
...
}
```

Để xác định đường dẫn file mô hình, loại mô hình, phép biến đổi dữ liệu và batch size.

Vòng lặp đánh giá tất cả mô hình

```
for exp_name, (path, model_type, train_aug_func, batch_size) in MODEL_PATHS_TO_EVAL.items():
metrics, labels, preds, probs = evaluate_all_metrics_combined(...)
```

Gọi hàm đánh giá cho từng mô hình.

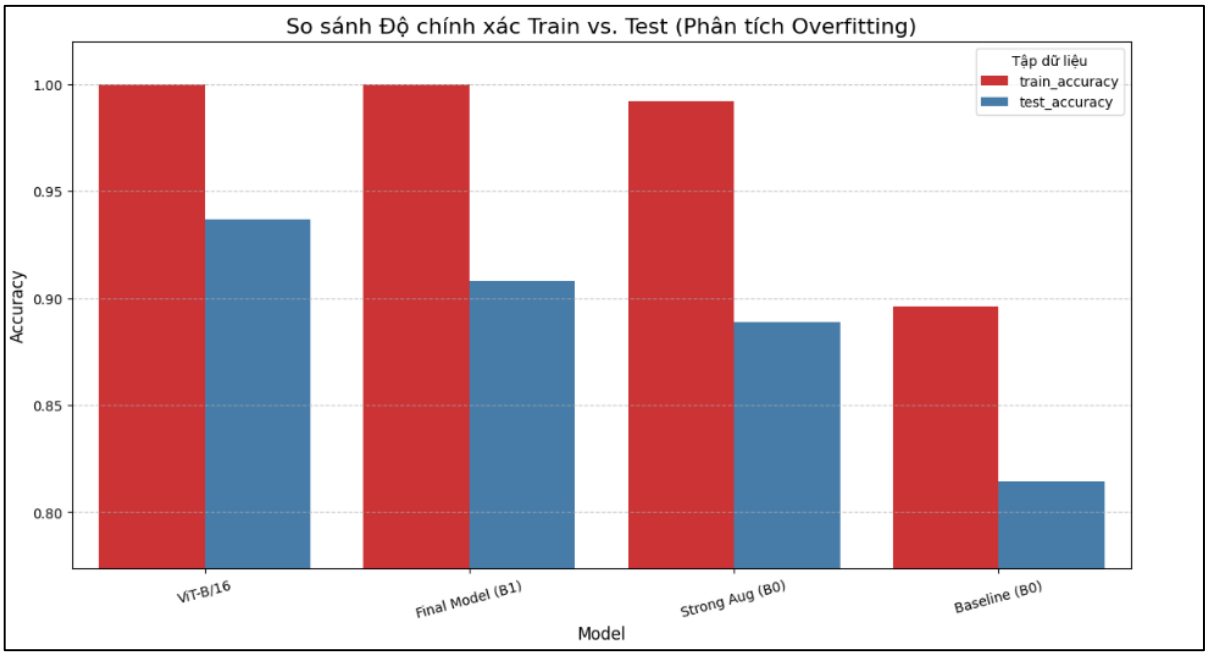
Lưu kết quả vào results_summary.

Tự động chọn mô hình có test_accuracy cao nhất.

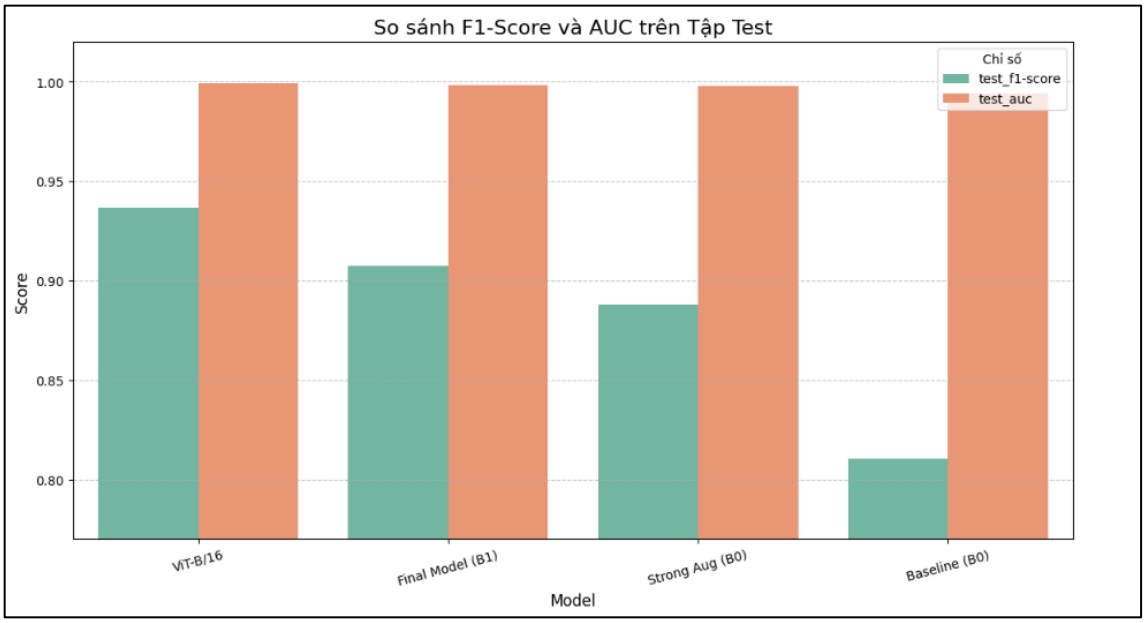
-Mô hình tốt nhất được chọn để phân tích chi tiết là: ViT-B/16 (Accuracy: 0.9367)

```
================================================================================
                    📊 BẢNG SO SÁNH HIỆU SUẤT TỔNG THỂ CÁC MÔ HÌNH
================================================================================
                 train_accuracy test_accuracy test_auc test_f1-score test_precision test_recall
ViT-B/16                100.00%        93.67%    0.999         0.937          0.941       0.937
Final Model (B1)        100.00%        90.81%    0.999         0.908          0.915       0.908
Strong Aug (B0)          99.22%        88.91%    0.998         0.888          0.899       0.889
Baseline (B0)            89.61%        81.43%    0.994         0.811          0.840       0.814
================================================================================
```

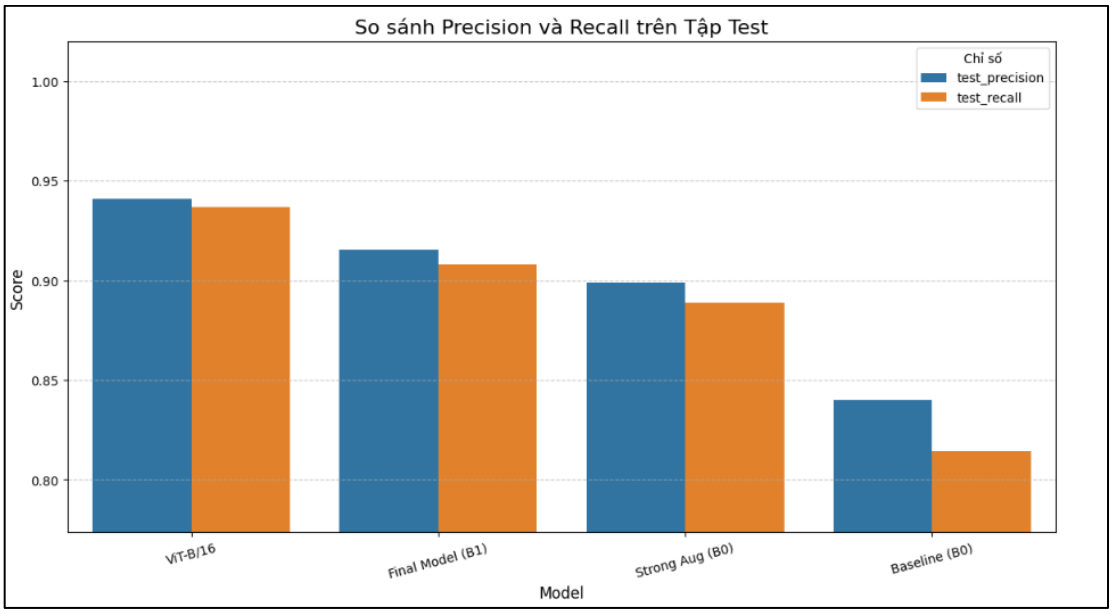
Hình 11: Bảng so sánh hiệu suất tổng thể các mô hình



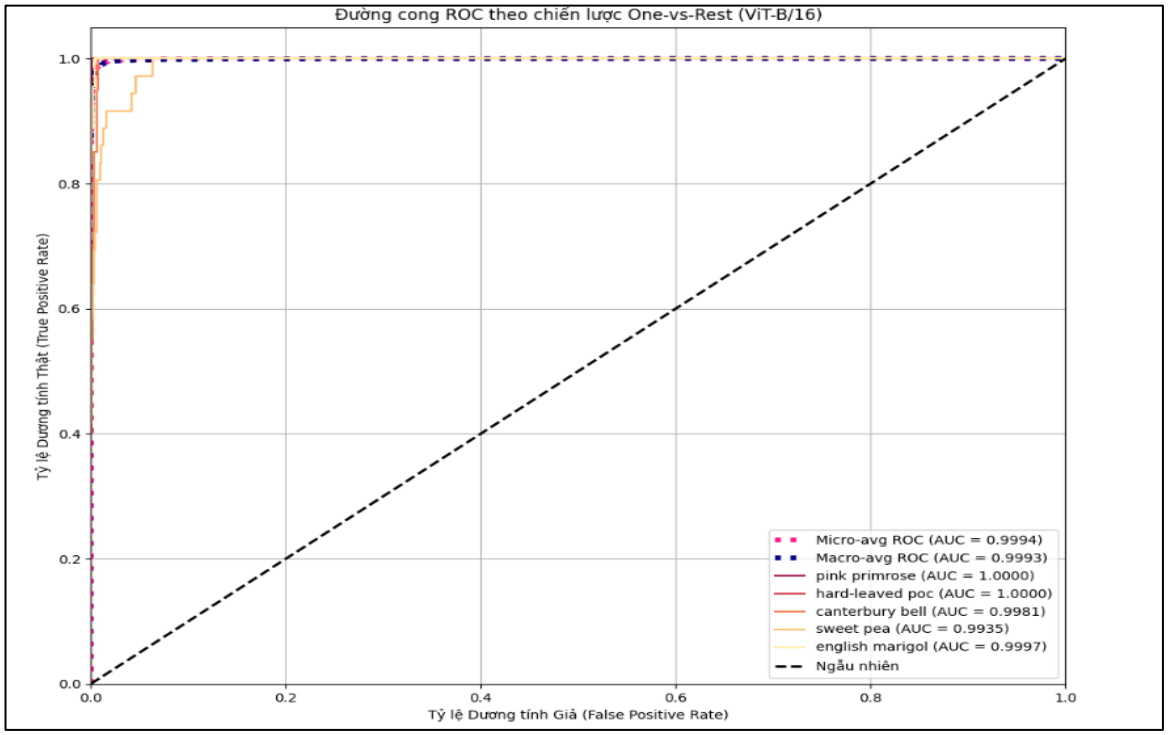
Hình 12: So sánh độ chính xác Train vs Test(Phân tích overfiting)



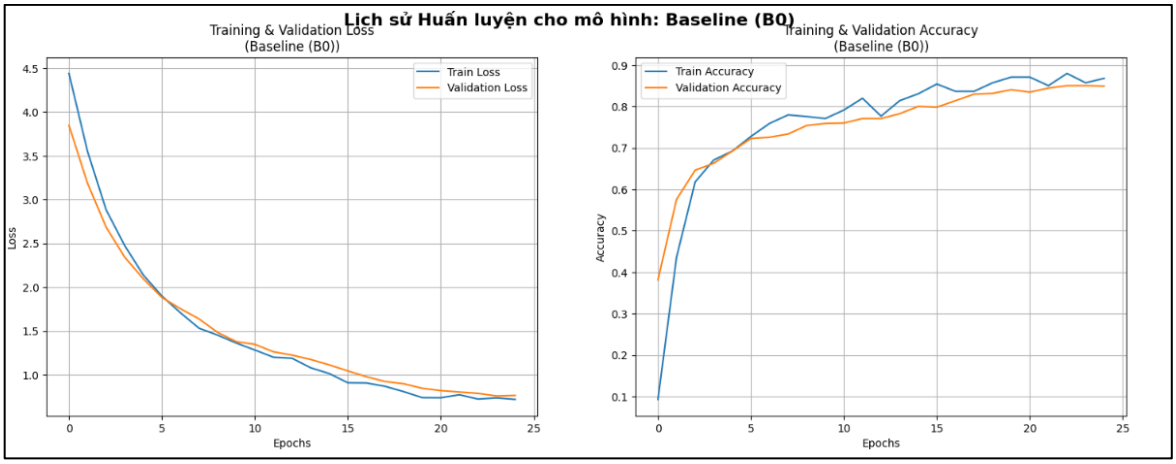
Hình 13: So sánh F1-Score và AUC trên tập Test



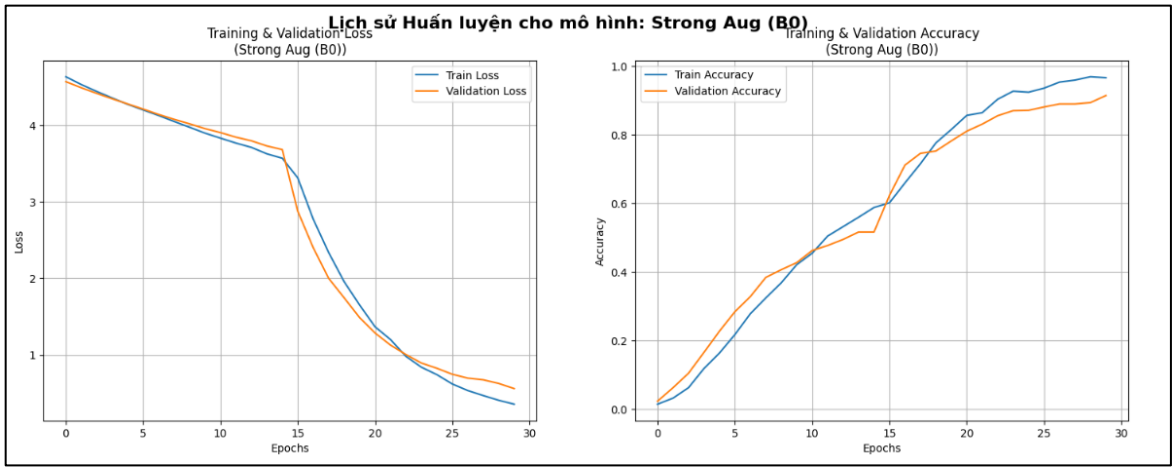
Hình 14: So sánh Precision và Recall trên tập Test



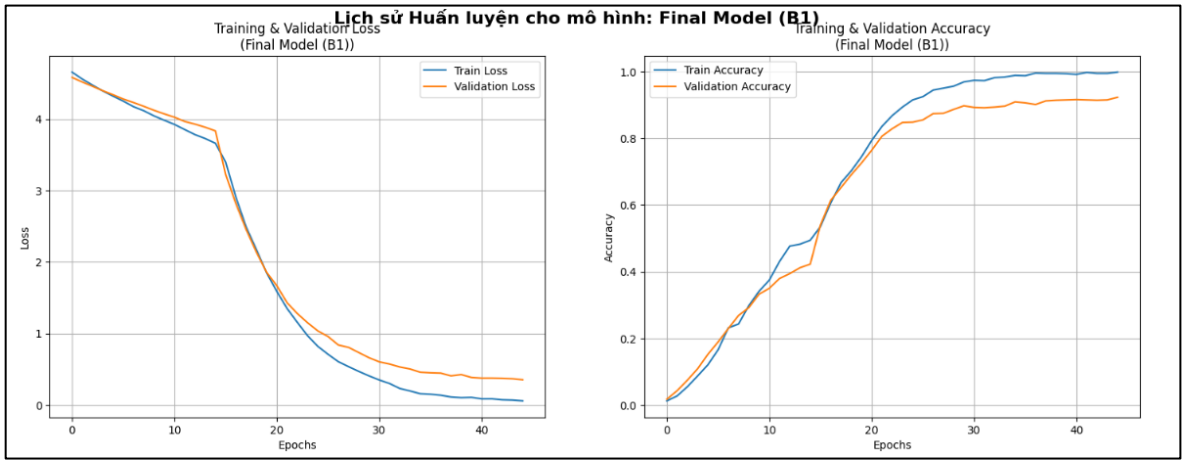
Hình 15: Trực quan hóa đường cong roc (vit-b/16)



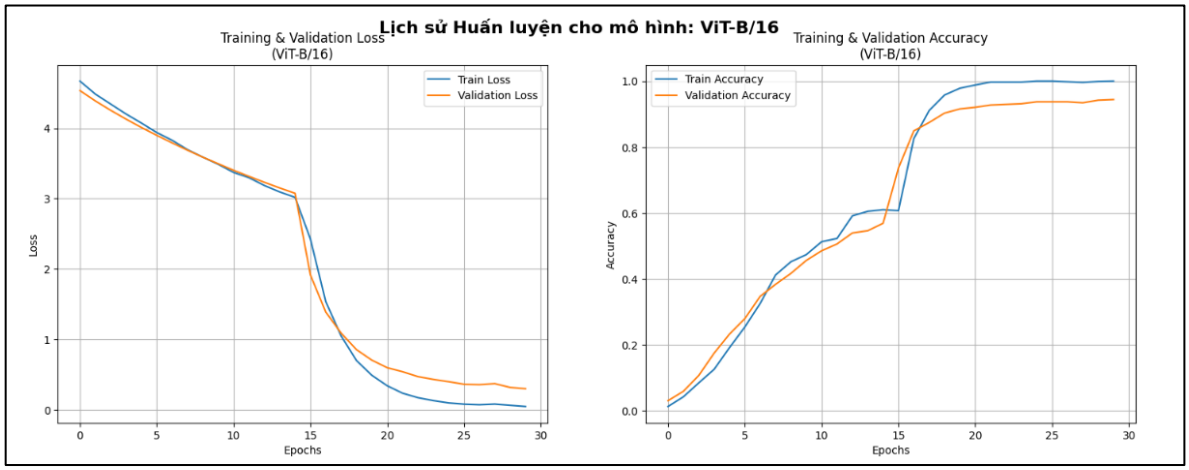
Hình 16: Lịch sử Huấn luyện cho mô hình BASELINE B0



Hình 17: Lịch sử Huấn luyện cho mô hình STRONG AUG B0



Hình 18: Lịch sử Huấn luyện cho mô hình Final Model (B1)



Hình 19: Lịch sử Huấn luyện cho mô hình ViT-B/16

3.6 Diễn giải mô hình và phân tích chi tiết

3.6.1 Ma trận nhầm lẫn tổng quát của 102 loài hoa

```
# DIỄN GIẢI MÔ HÌNH VÀ PHÂN TÍCH CHI TIẾT
def analyze_best_model():
    if 'best_model_labels' in globals() and best_model_labels is not None:
        ...
        if best_model_name == 'ViT-B/16':
            BEST_MODEL_PATH = 'models/exp_vit.pth'
            model_to_visualize = build_model_vit().to(DEVICE)
        ...
        model_to_visualize.load_state_dict(torch.load(BEST_MODEL_PATH,
map_location=DEVICE))
        model_to_visualize.eval()
        ...
        print(classification_report(best_model_labels, best_model_preds,
target_names=class_names))
```

Hàm dùng để phân tích chi tiết mô hình tốt nhất sau quá trình huấn luyện.

Tự động tải lại mô hình có hiệu suất cao nhất từ tệp lưu (.pth).

Chuyển mô hình sang chế độ đánh giá (eval()) để đảm bảo tính ổn định khi phân tích.

Sinh báo cáo phân loại chi tiết **(precision, recall, F1-score)** cho từng lớp hoa.

Chuẩn bị mô hình cho các bước diễn giải trực quan như **Grad-CAM, t-SNE, Occlusion.**

Giúp đánh giá độ chính xác và khả năng tổng quát của mô hình trên toàn bộ tập dữ liệu.

Hỗ trợ so sánh giữa các mô hình khác nhau trong quá trình nghiên cứu.

Là bước quan trọng nhất sau huấn luyện, giúp hiểu được cách mô hình đưa ra dự đoán.

model_to_visualize.load_state_dict(...): tải trọng số mô hình tốt nhất.

model_to_visualize.eval(): đặt mô hình vào chế độ đánh giá để tránh dropout/batchnorm ảnh hưởng.

classification_report(...): tạo báo cáo thống kê chính xác về từng lớp → là kết quả cốt lõi để đánh giá mô hình.

3.6.2 Trực quan hóa từng lớp hoa

```
# TRỰC QUAN HÓA ĐỘ CHÍNH XÁC TỪNG LỚP ---
def visualize_class_accuracy():
    if 'best_model_labels' in globals() and best_model_labels is not None:
        cm_best = confusion_matrix(best_model_labels, best_model_preds)
        per_class_accuracy_best = cm_best.diagonal() / cm_best.sum(axis=1)
        ...
        sns.barplot(data=chart_df, y='Loài hoa', x='Độ chính xác (%)', palette='coolwarm', ...)
        ...
```

Hàm này dùng để trực quan hóa độ chính xác **(Recall)** của từng lớp hoa trong mô hình tốt nhất.

Tính ma trận nhầm lẫn **(Confusion Matrix)** để xác định số dự đoán đúng/sai cho từng lớp.

Tính độ chính xác riêng của từng lớp bằng cách lấy tỉ lệ chéo chính trên tổng dòng.

Sắp xếp và chia nhỏ các lớp thành nhiều phần biểu đồ (nếu có hơn 10 lớp) để dễ quan sát.

Vẽ biểu đồ cột thể hiện độ chính xác của từng loài hoa với màu sắc theo mức hiệu suất.

Hiển thị giá trị phần trăm cụ thể ngay trên mỗi cột để tăng tính trực quan.

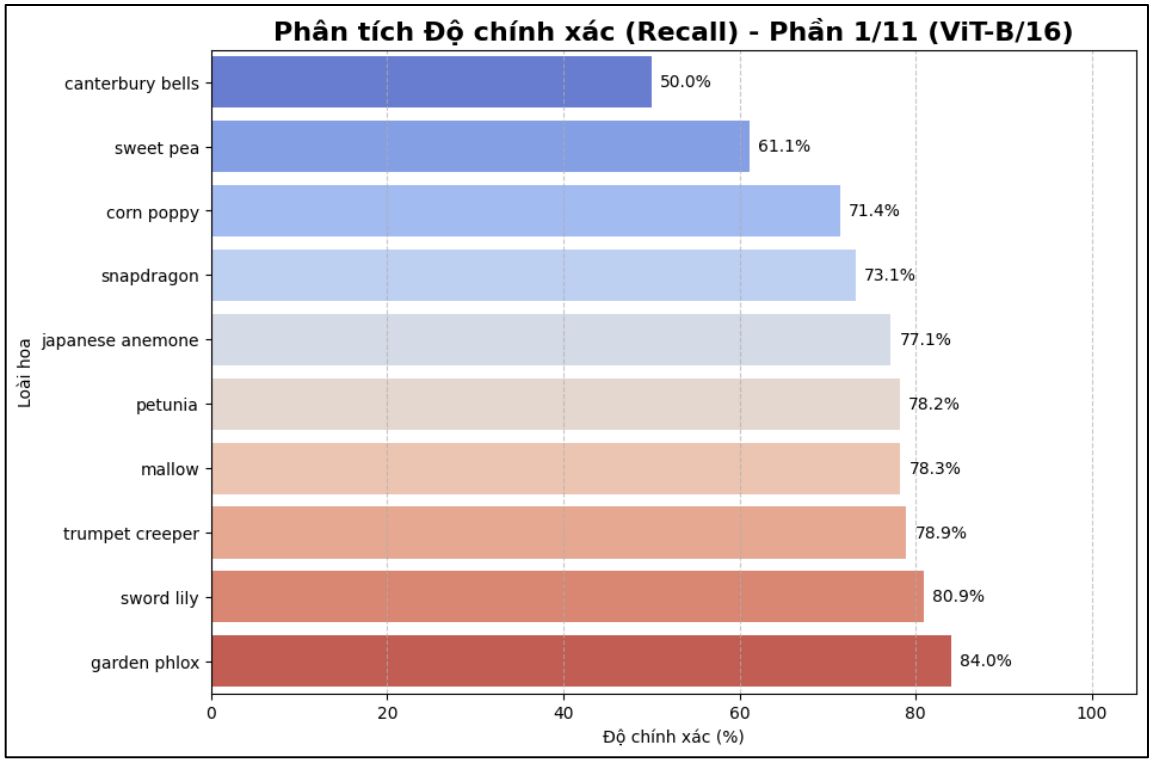
Giúp nhận biết lớp nào mô hình hoạt động tốt hoặc kém, hỗ trợ điều chỉnh dữ liệu/huấn luyện.

Là bước đánh giá định lượng và trực quan rất quan trọng trong phân tích mô hình học sâu.

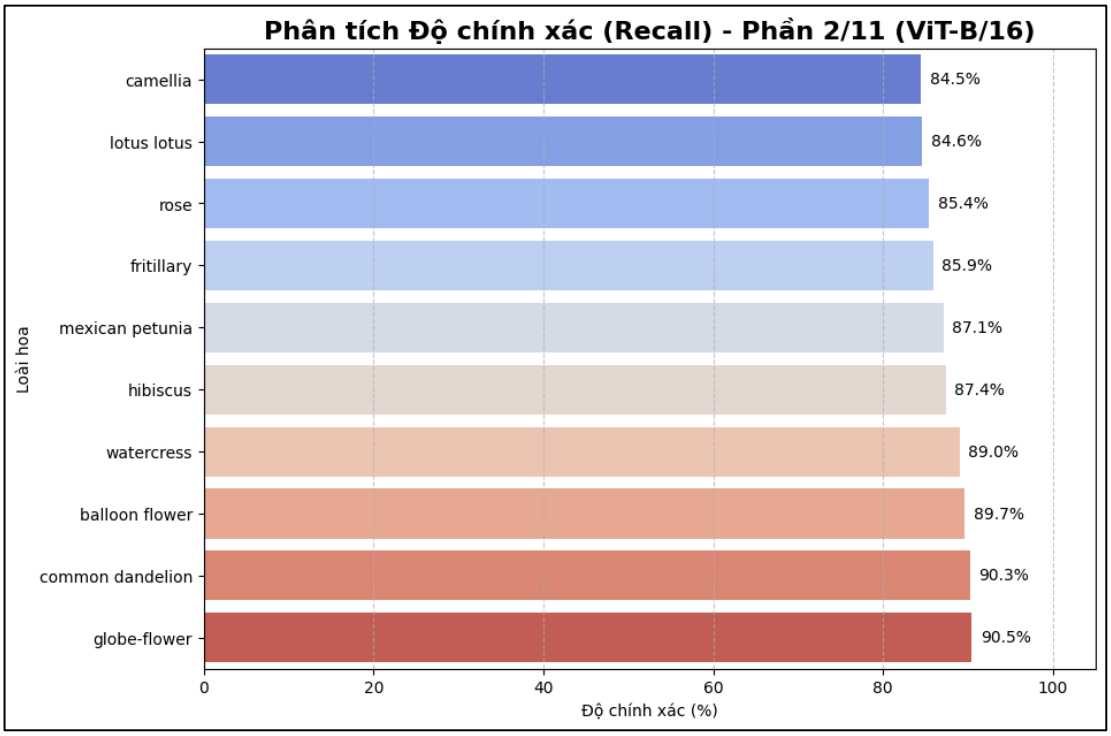
confusion_matrix(...): xác định số lượng dự đoán đúng/sai.

per_class_accuracy_best = cm_best.diagonal() / cm_best.sum(axis=1): tính độ chính xác từng lớp.

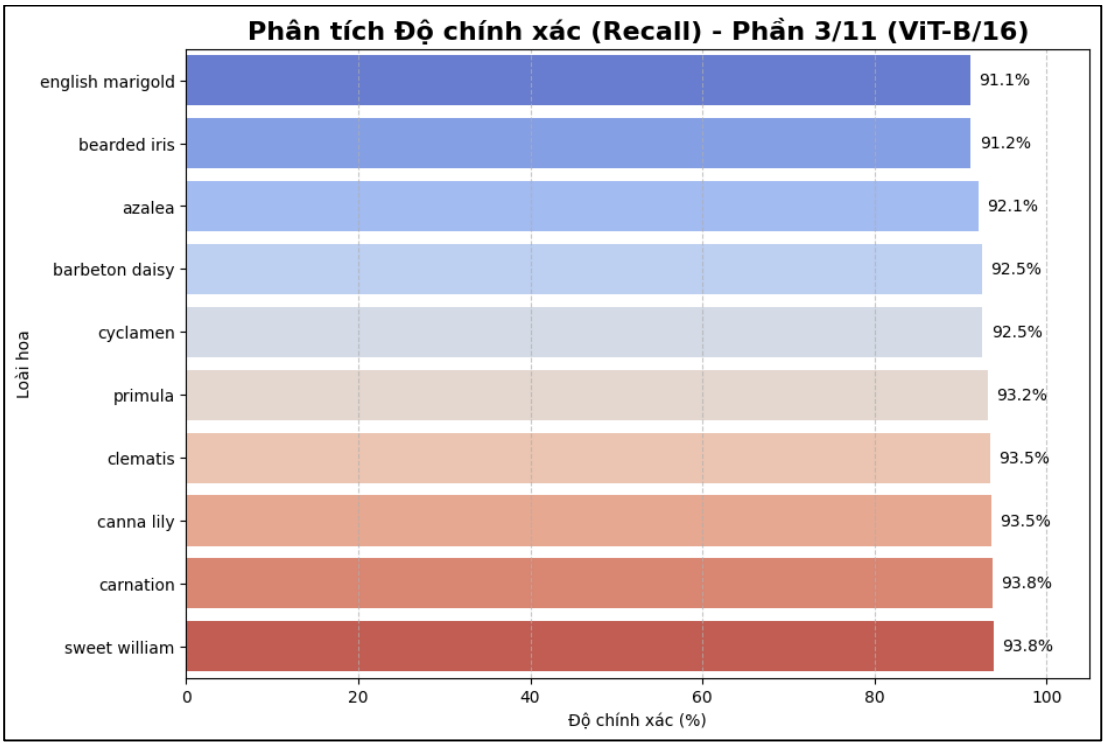
sns.barplot(...): trực quan hóa kết quả giúp phát hiện nhanh các lớp yếu hoặc lệch dữ liệu.



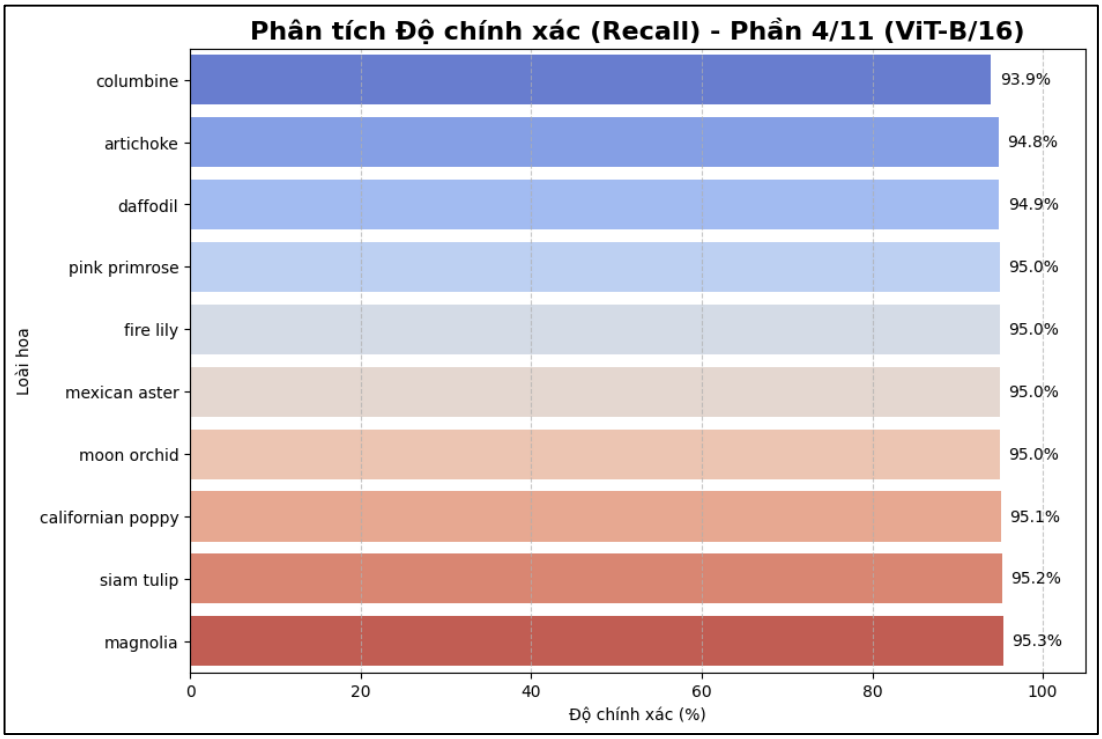
Hình 20: Độ chính xác trên ViT - B/16 phần 1/11



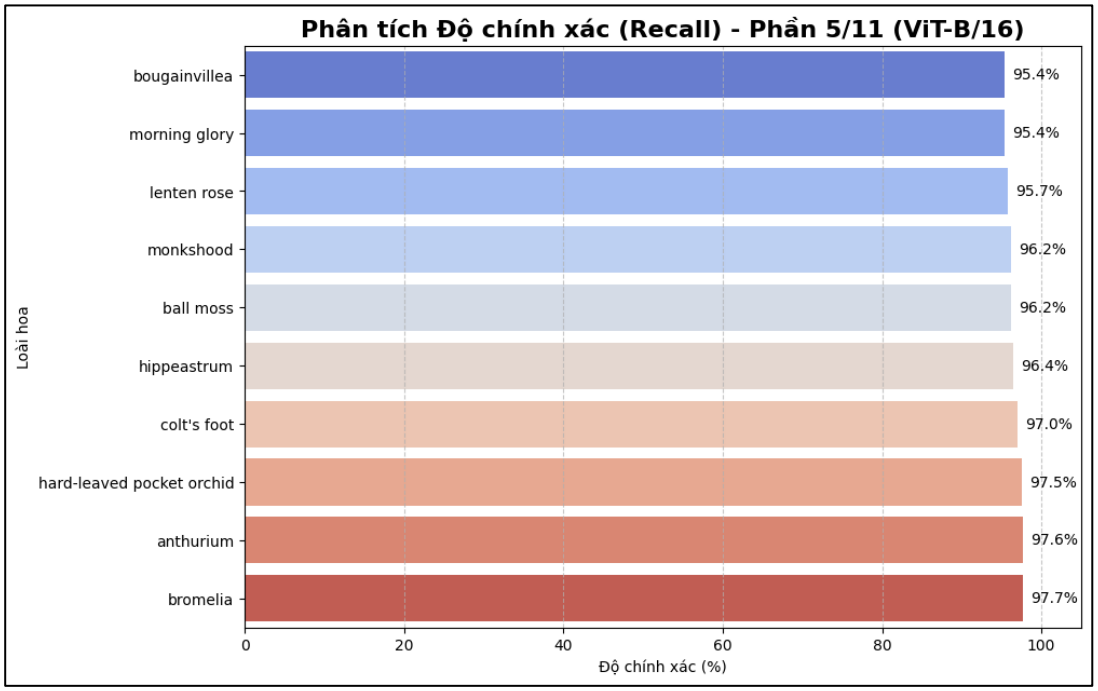
Hình 21: Độ chính xác trên ViT - B/16 phần 2/11



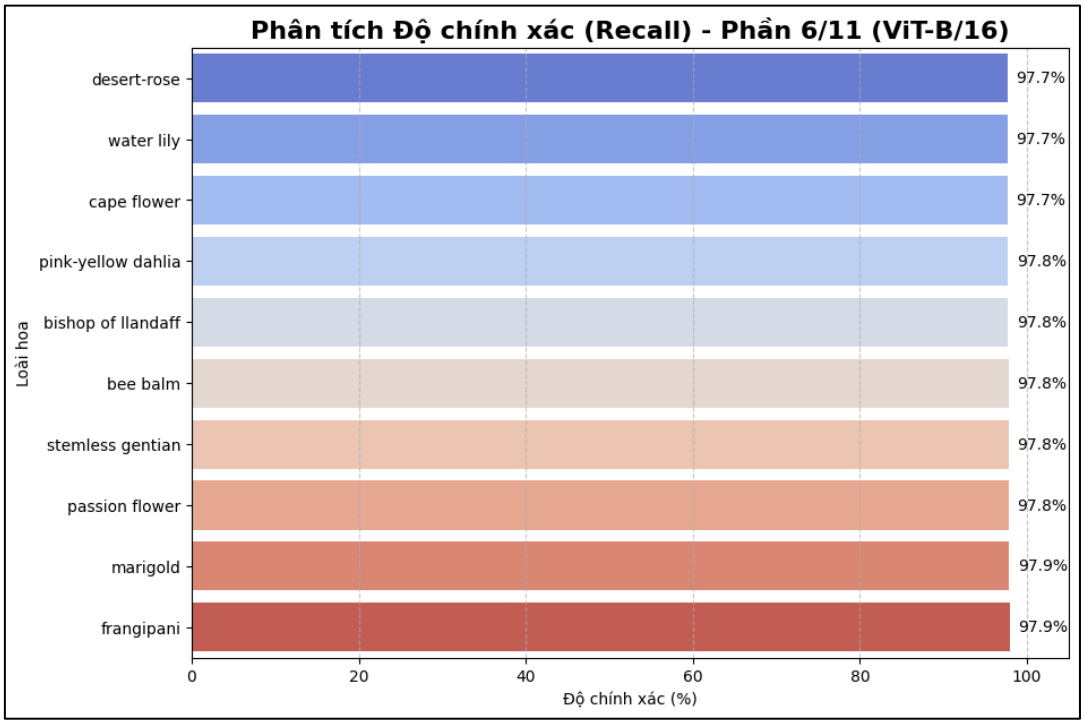
Hình 22: Độ chính xác trên ViT - B/16 phần 3/11



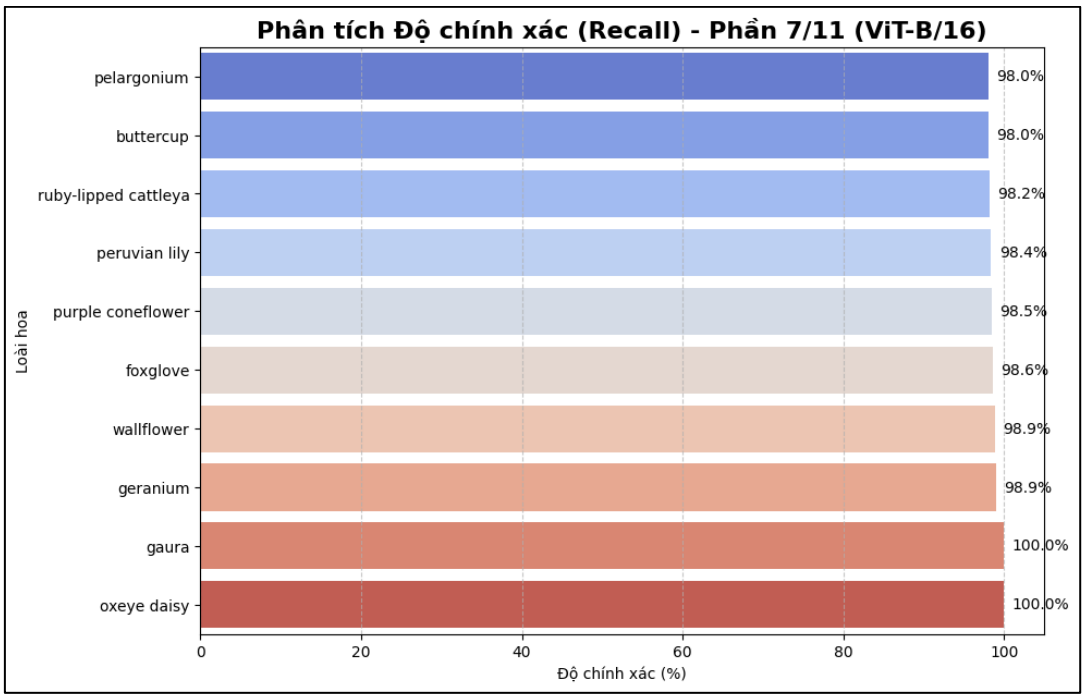
Hình 23: Độ chính xác trên ViT - B/16 phần 4/11



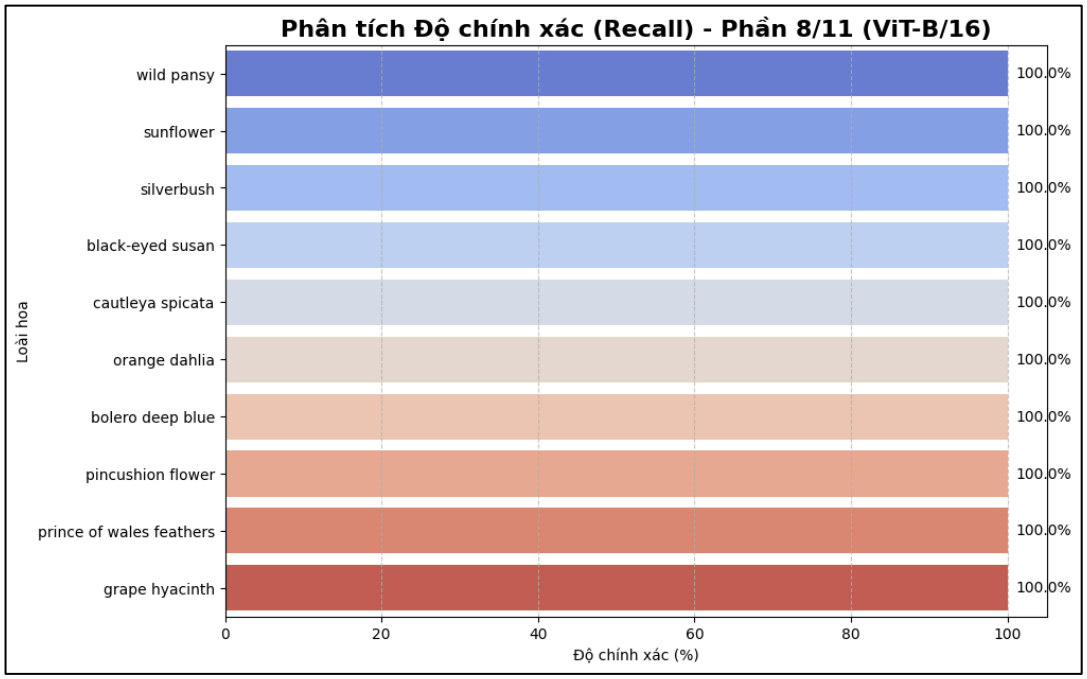
Hình 24: Độ chính xác trên ViT - B/16 phần 5/11



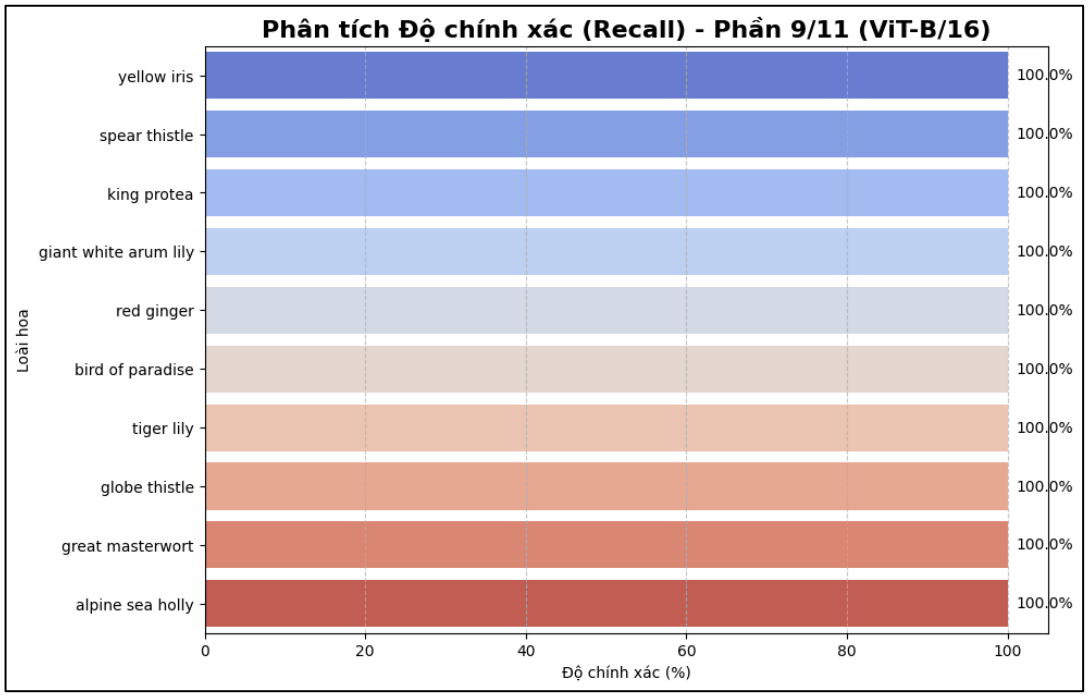
Hình 25: Độ chính xác trên ViT - B/16 phần 6/11



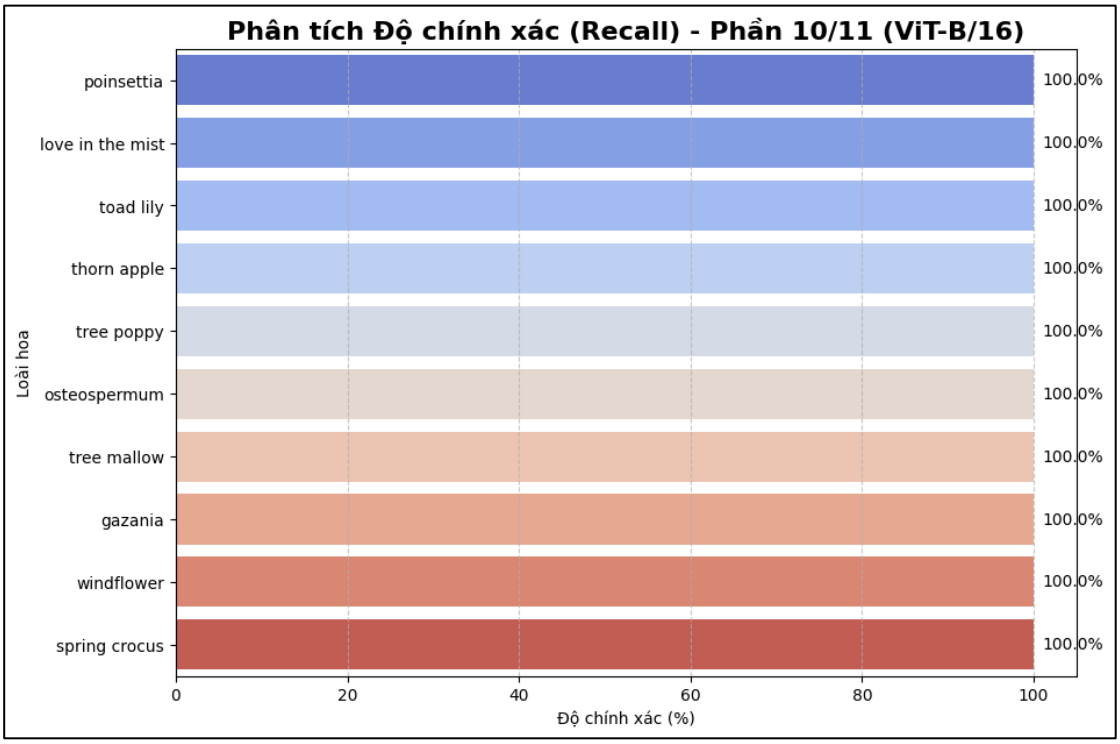
Hình 26: Độ chính xác trên ViT - B/16 phần 7/11



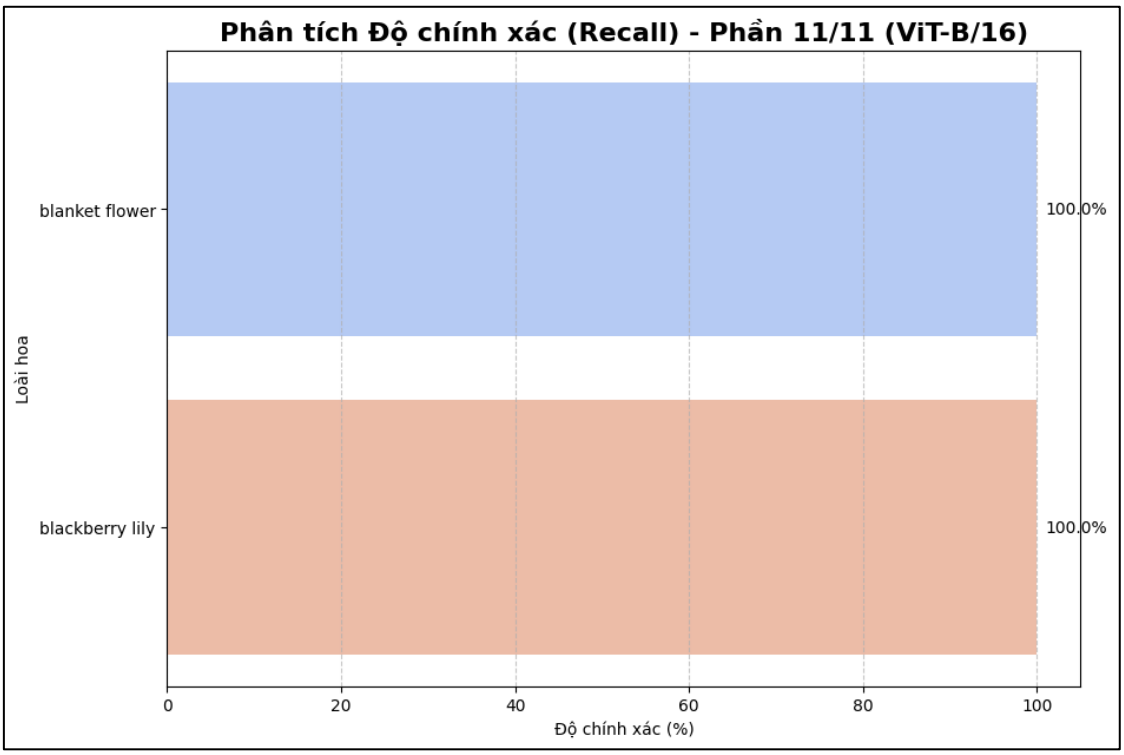
Hình 27: Độ chính xác trên ViT - B/16 phần 8/11



Hình 28: Độ chính xác trên ViT - B/16 phần 9/11



Hình 29: Độ chính xác trên ViT - B/16 phần 10/11



Hình 30: Độ chính xác trên ViT - B/16 phần 11/10

3.6.3 Diễn giải mô hình bằng Occlusion Sensitivity

```
# PHÂN TÍCH OCCLUSION SENSITIVITY ---
def occlusion_sensitivity_analysis():
    def occlusion_sensitivity(...): ...
    def visualize_occlusion_overlay(...): ...
    test_df_occl = df[df['split'] == 'test']
    random_samples = test_df_occl.sample(n=3)
    for i, (idx, row) in enumerate(random_samples.iterrows()):
        ...
```

Hàm này dùng để phân tích **Occlusion Sensitivity** – một kỹ thuật diễn giải mô hình thị giác (model interpretability).

Mục tiêu là xác định vùng ảnh ảnh hưởng mạnh nhất đến quyết định của mô hình khi dự đoán một loài hoa.

Hàm occlusion_sensitivity() lần lượt che các vùng nhỏ (patch) của ảnh và quan sát độ thay đổi xác suất dự đoán.

Nếu che vùng nào làm xác suất giảm mạnh → vùng đó quan trọng với mô hình.

Hàm **visualize_occlusion_overlay()** vẽ bản đồ nhiệt (heatmap) chồng lên ảnh gốc để trực quan vùng ảnh có ảnh hưởng lớn.

Hệ thống chọn ngẫu nhiên 3 ảnh test, chạy mô hình dự đoán, rồi hiển thị vùng tập trung của mô hình.

Giúp hiểu rõ mô hình học được gì từ ảnh – có thật sự tập trung vào bông hoa hay bị nhiễu bởi nền.

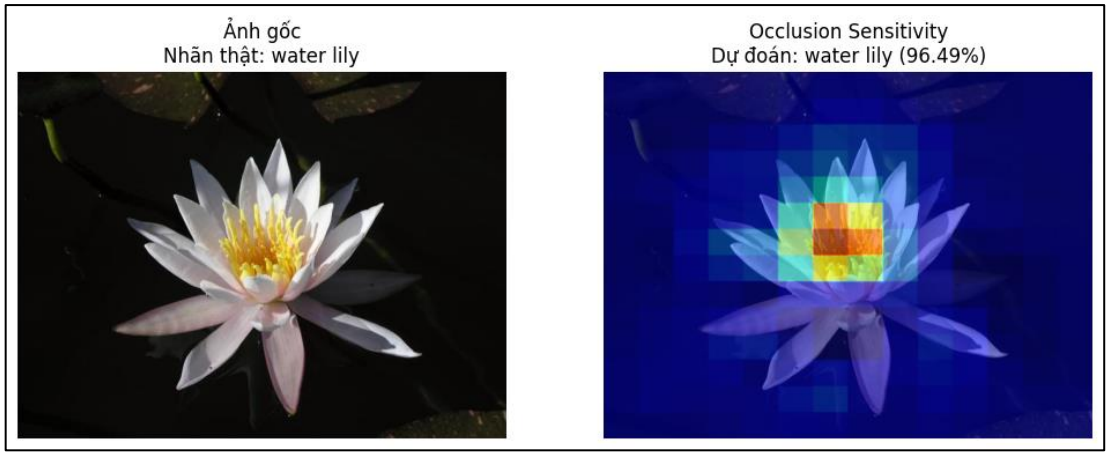
Là bước quan trọng trong phân tích diễn giải mô hình học sâu và kiểm chứng tính tin cậy của mô hình.

occluded_input[:, :, h:h+patch_size, w:w+patch_size] = 0: che vùng ảnh.

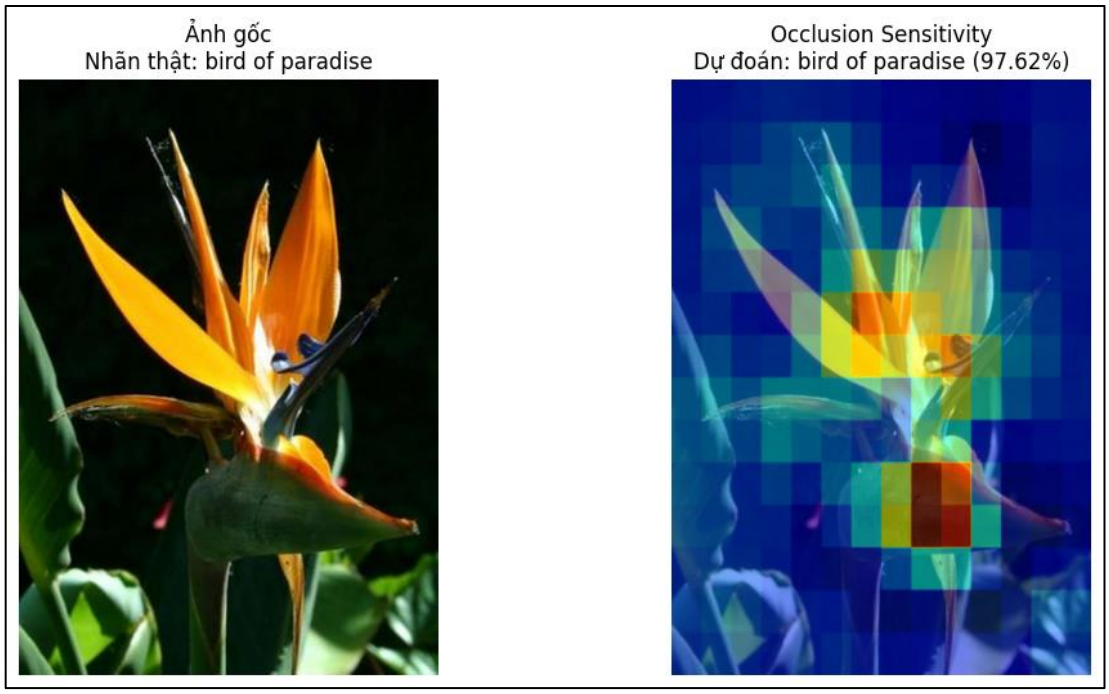
original_prob - occluded_prob: tính mức giảm xác suất khi che vùng.

cv2.addWeighted(...): tạo hình ảnh chồng giữa heatmap và ảnh gốc để trực quan hóa vùng quan trọng.

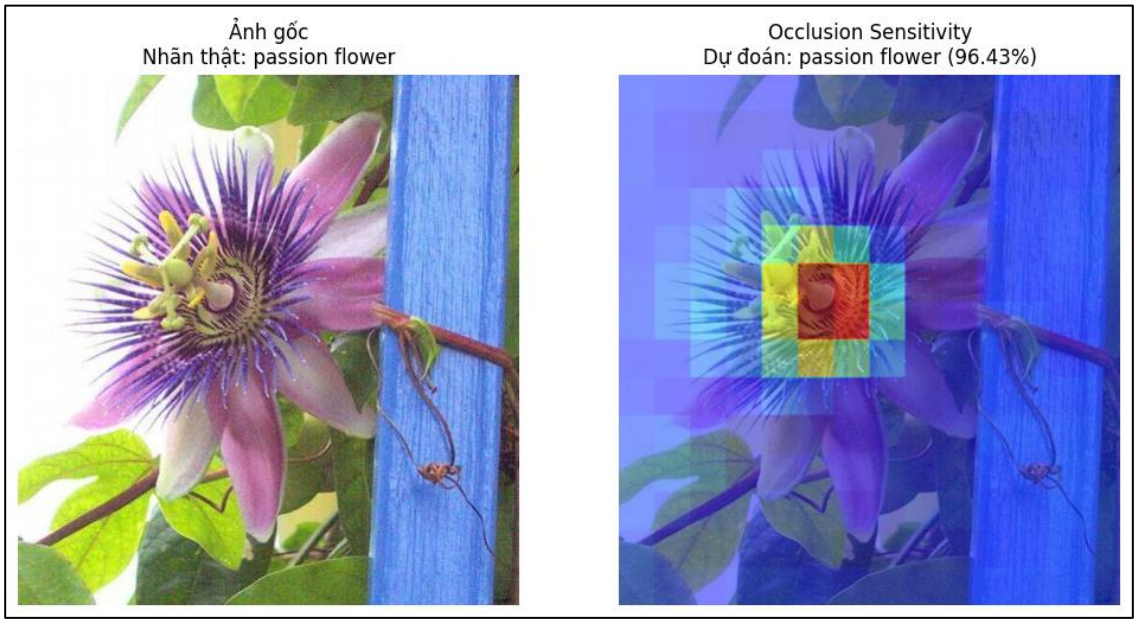
Kết quả:



Hình 31: Phân tích Occlusion Sensitivity trên water lily



Hình 32: Phân tích Occlusion Sensitivity trên bird of paradise



Hình 33: Phân tích Occlusion Sensitivity trên passion flower

3.6.4 T-SNE VISUALIZATION

```
# T-SNE VISUALIZATION (ĐÃ SỬA LỖI BẰNG KỸ THUẬT HOOK) ---
def tsne_visualization(...):
    ...
```

Hàm này thực hiện trích xuất đặc trưng (**embedding**) từ mô hình học sâu tốt nhất và biểu diễn chúng bằng **t-SNE** trong không gian 2 chiều.

Sử dụng hook để lấy dữ liệu từ lớp trung gian (feature layer) trong mô hình mà không cần sửa kiến trúc gốc.

Các đặc trưng được trích xuất thể hiện cách mô hình hiểu và nhóm các loài hoa khác nhau.

Sau đó dùng **t-SNE (t-Distributed Stochastic Neighbor Embedding)** để giảm chiều từ hàng trăm/thousands về 2D để trực quan.

Kết quả là biểu đồ scatter cho thấy mức độ phân tách giữa các lớp hoa — các cụm rõ ràng chứng tỏ mô hình học tốt.

Hữu ích trong phân tích không gian đặc trưng **(feature space**) và đánh giá khả năng phân biệt lớp của mô hình.

Hook giúp linh hoạt với cả **ViT (Vision Transformer)** và **EfficientNet (CNN)** mà không phải chỉnh sửa code gốc.

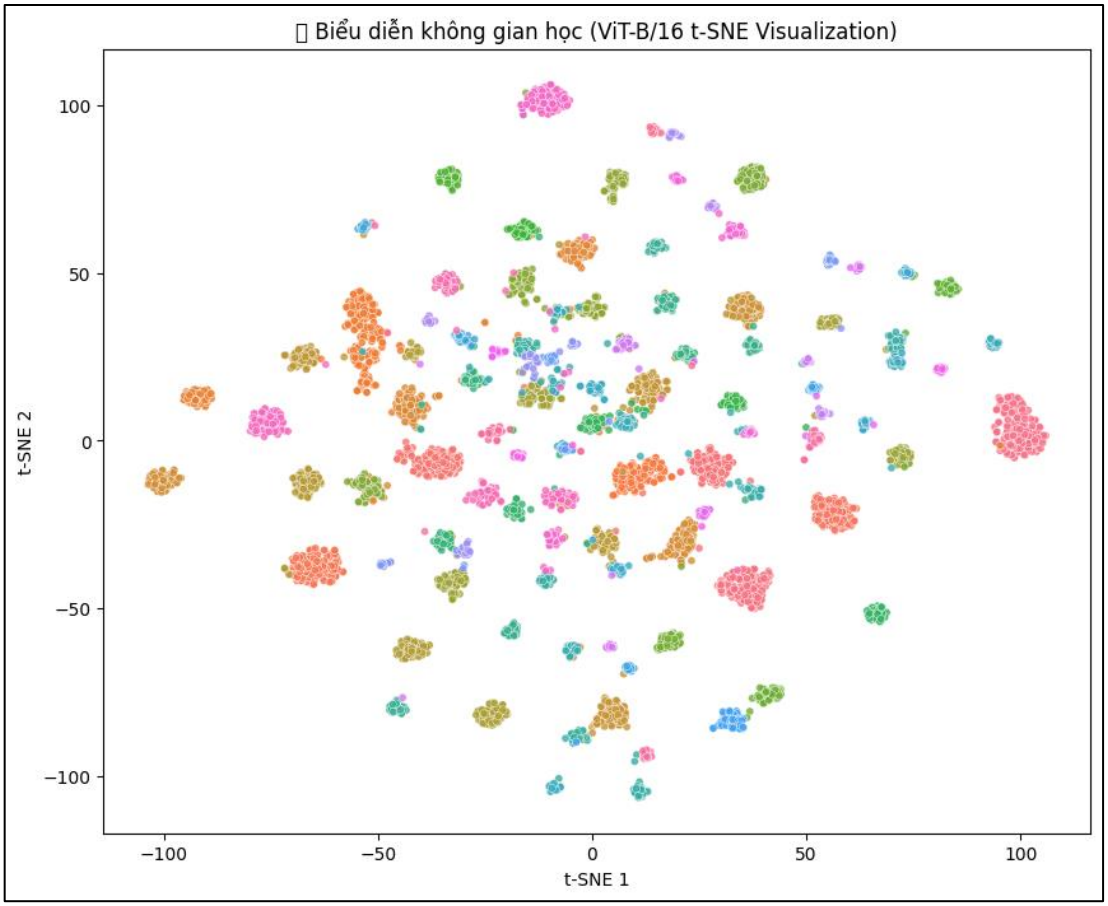
Là bước quan trọng để kiểm chứng mức độ "hiểu" hình ảnh của mô hình thị giác sau huấn luyện.

hook_handle = target_layer.register_forward_hook(hook_fn) → đăng ký hook để lấy output của lớp đặc trưng.

feature_map['features'] = output[:, 0, :] → lấy embedding từ token [CLS] (ViT).

tsne.fit_transform(embeddings) → giảm chiều embedding xuống 2D để trực quan.

sns.scatterplot(...) → vẽ biểu đồ phân bố không gian học của các loài hoa.



Hình 34: Biểu diễn phân loại hoa trong không gian học t-SNE

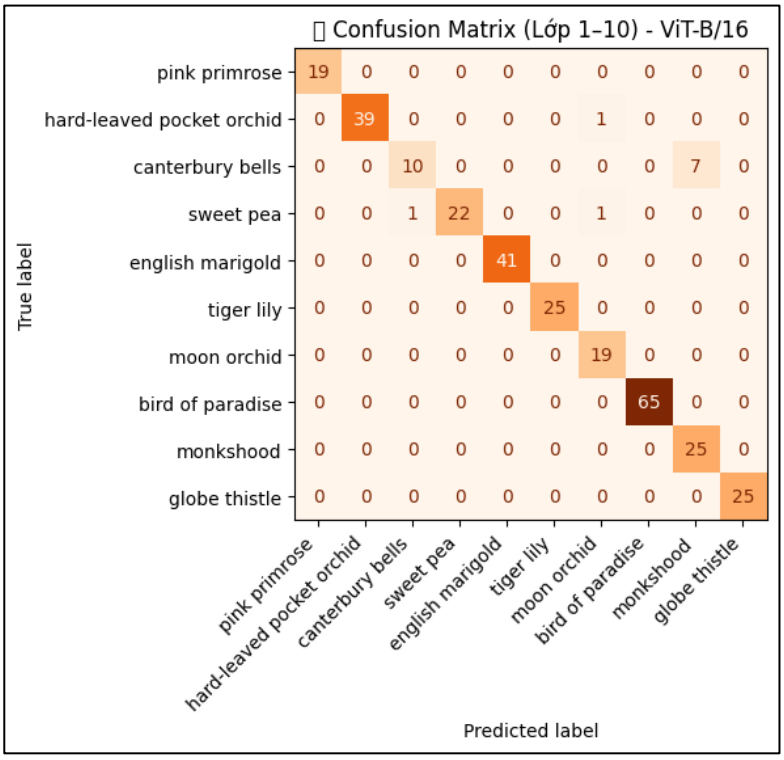
3.6.5 Ma trận nhầm lẫn cho từng lớp hoa

```
# --- 6. CONFUSION MATRIX CHIA NHỎ ---
if 'best_model_labels' in globals() and best_model_labels is not None:
  …..
   for start in range(0, num_classes, group_size):
       end = min(start + group_size, num_classes)
       subset_classes = class_names[start:end]; cm_subset = cm[start:end, start:end]
       fig, ax = plt.subplots(figsize=(6, 6))
….
```

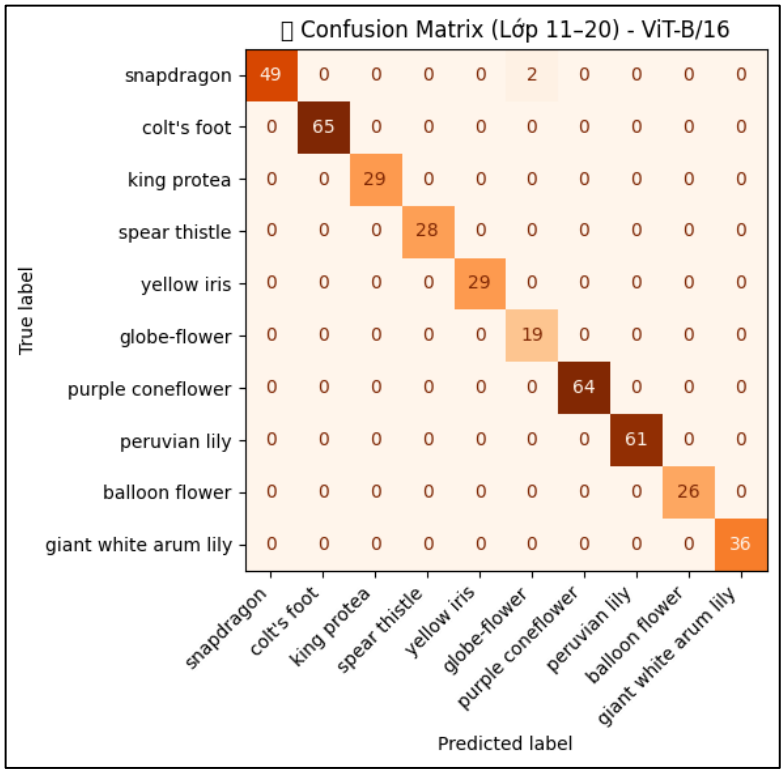
Hàm này trực quan hóa Confusion Matrix theo từng nhóm lớp (mỗi nhóm ~10 lớp) để dễ quan sát hơn khi số lượng lớp lớn (102 lớp).

Giúp phân tích chi tiết lỗi phân loại của mô hình tốt nhất, xác định các nhóm hoa dễ bị nhầm lẫn và độ chính xác trong từng cụm lớp nhỏ.

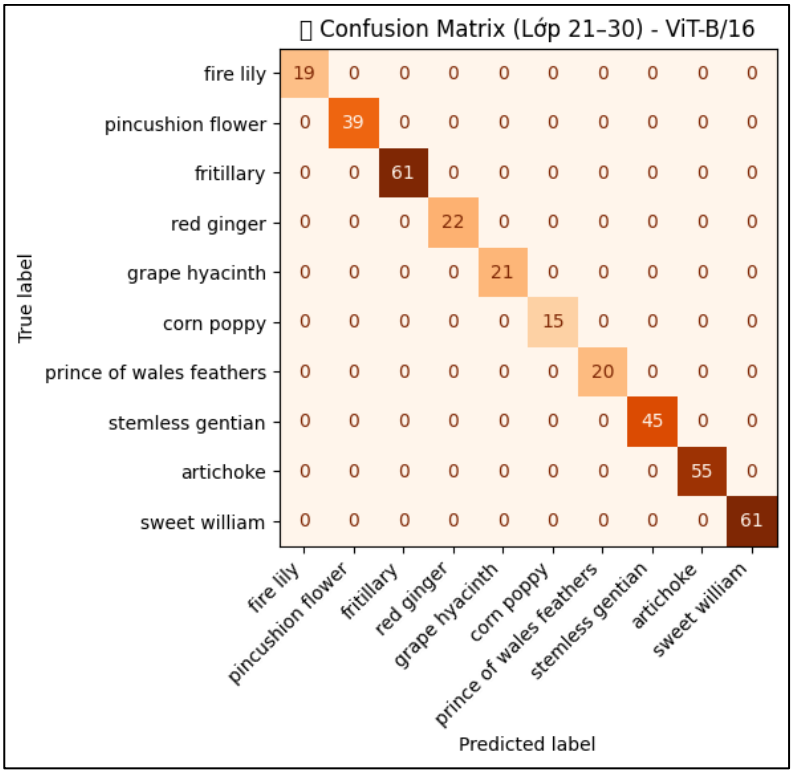
Kết quả:



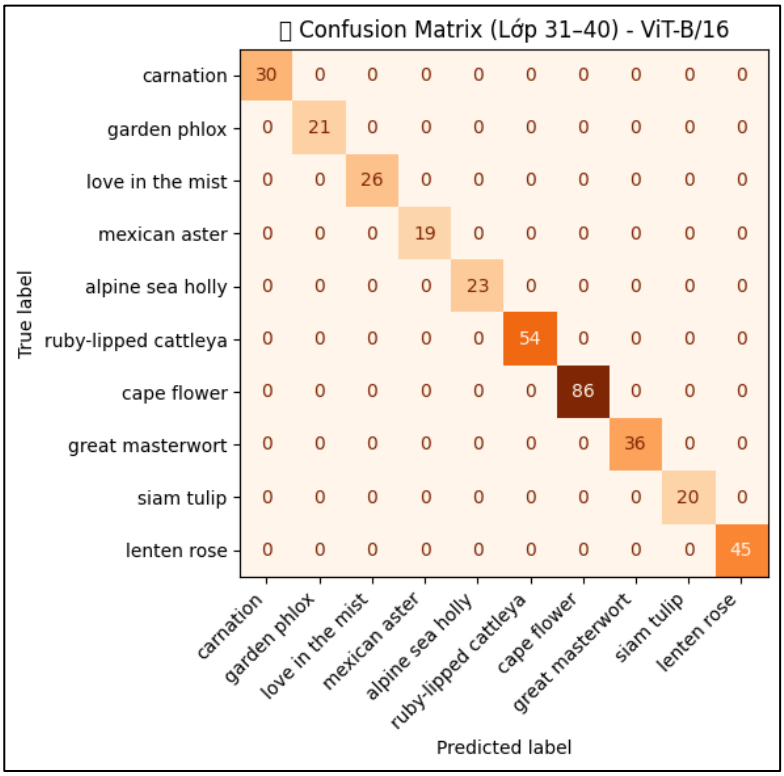
Hình 35: Biểu diễn ma trận nhầm lẫn lớp 1 đến 10



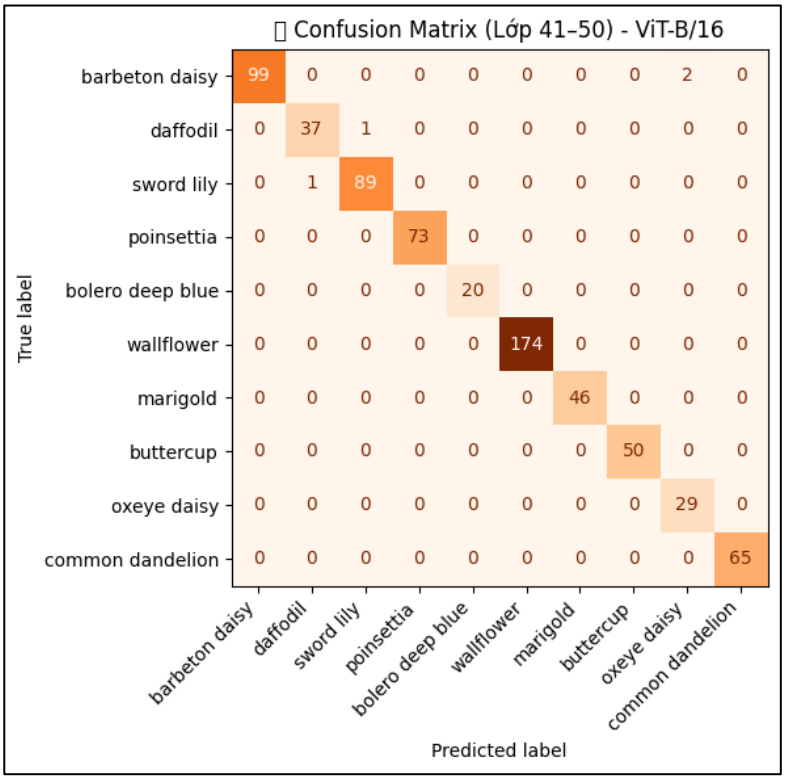
Hình 36: Biểu diễn ma trận nhầm lẫn lớp 10 đến 20



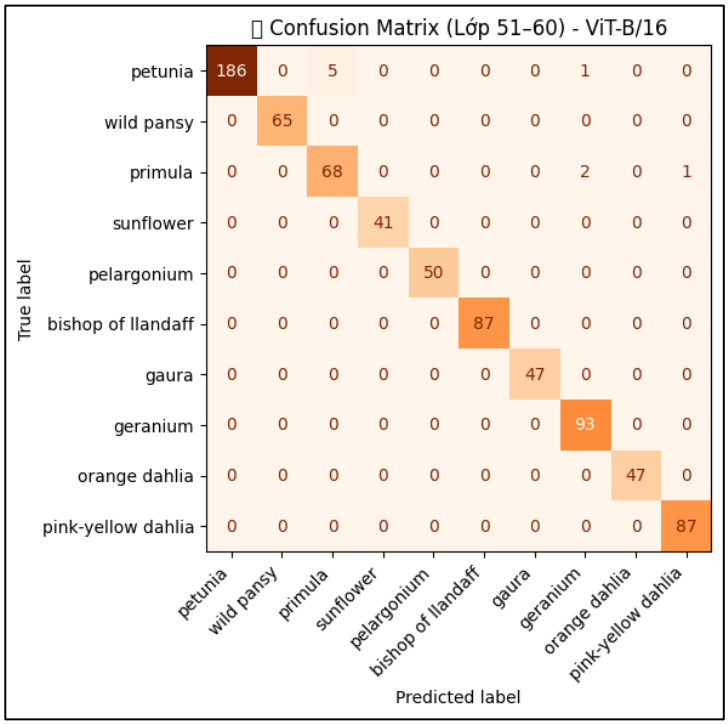
Hình 37: Biểu diễn ma trận nhầm lẫn lớp 21 đến 30



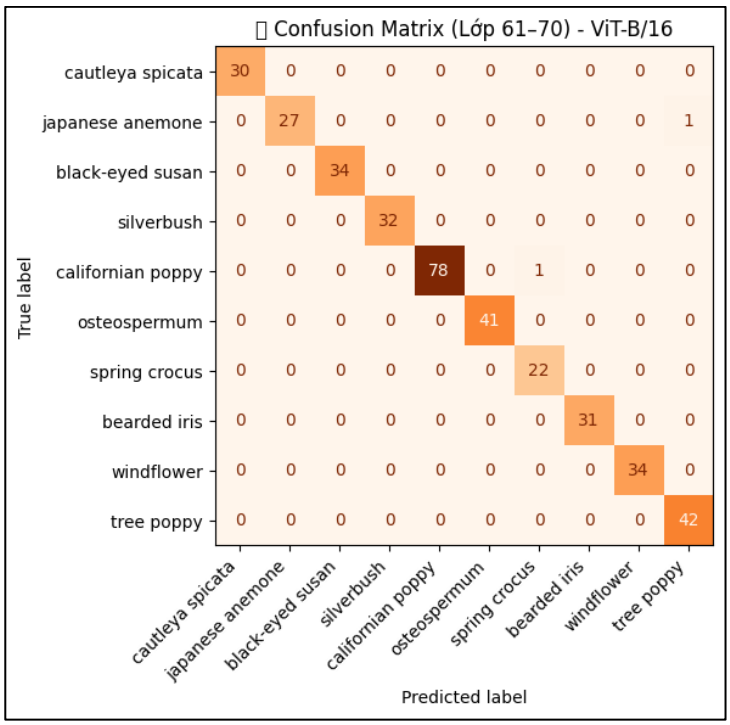
Hình 38: Biểu diễn ma trận nhầm lẫn lớp 31 đến 40



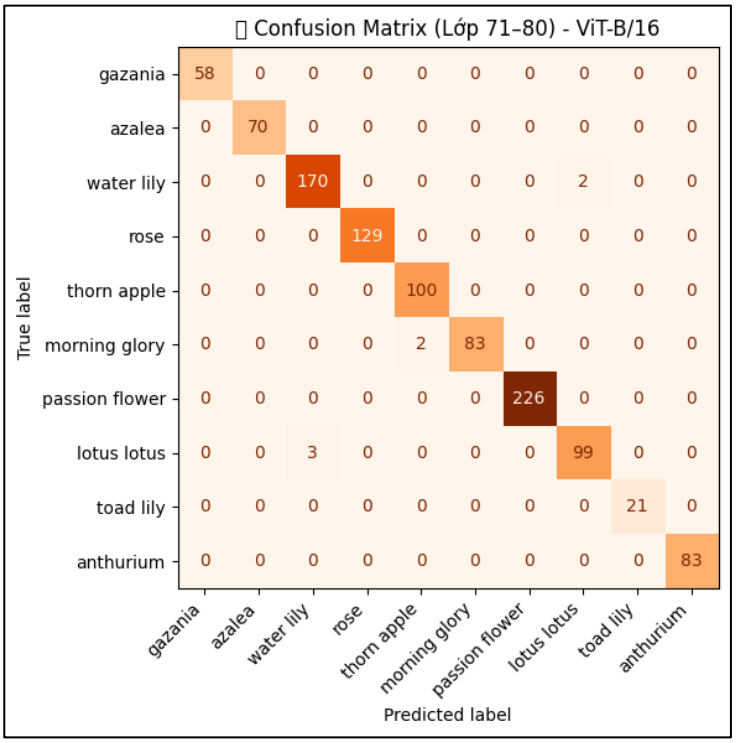
Hình 39: Biểu diễn ma trận nhầm lẫn lớp 41 đến 50



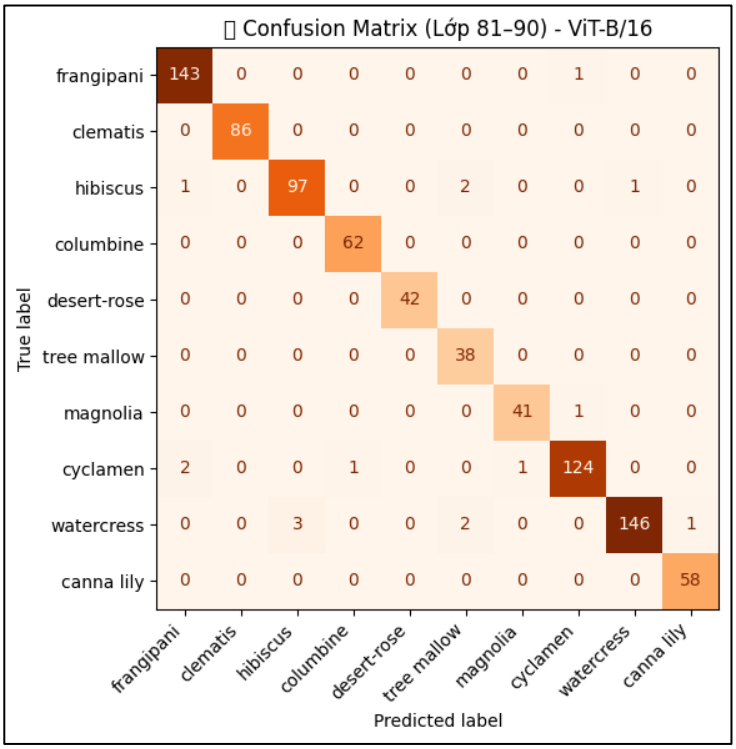
Hình 40: Biểu diễn ma trận nhầm lẫn lớp 51 đến 60



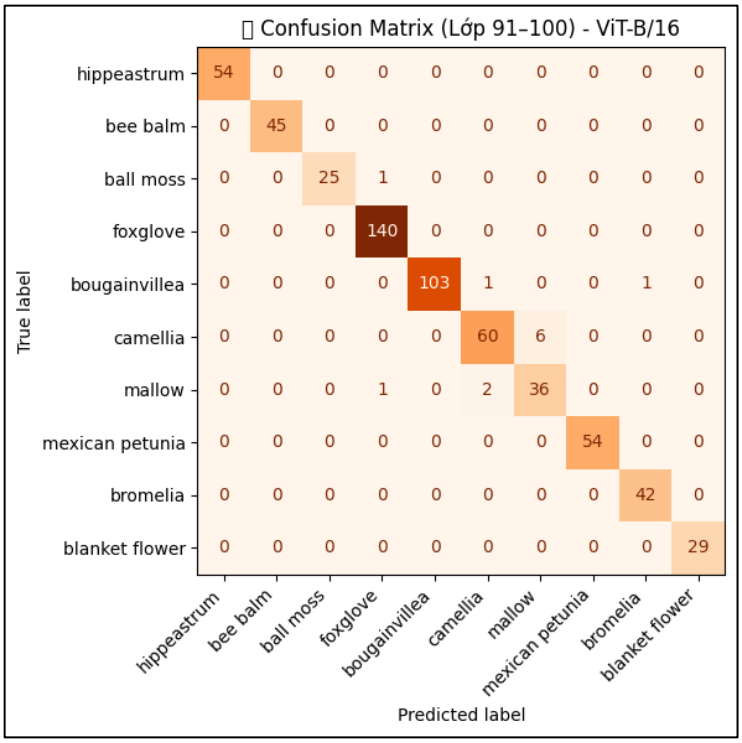
Hình 41: Biểu diễn ma trận nhầm lẫn lớp 61 đến 70



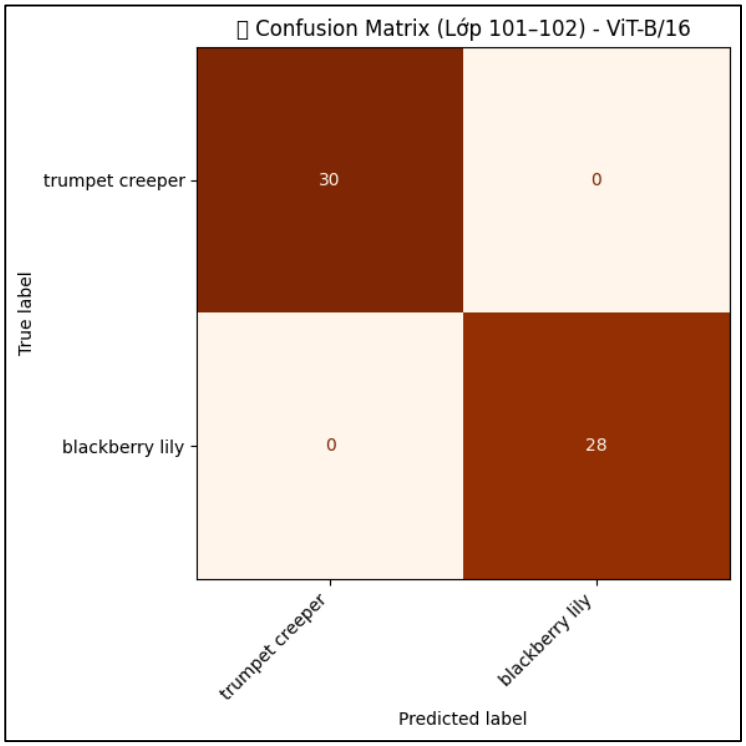
Hình 42: Biểu diễn ma trận nhầm lẫn lớp 71 đến 80



Hình 43: Biểu diễn ma trận nhầm lẫn lớp 81 đến 90

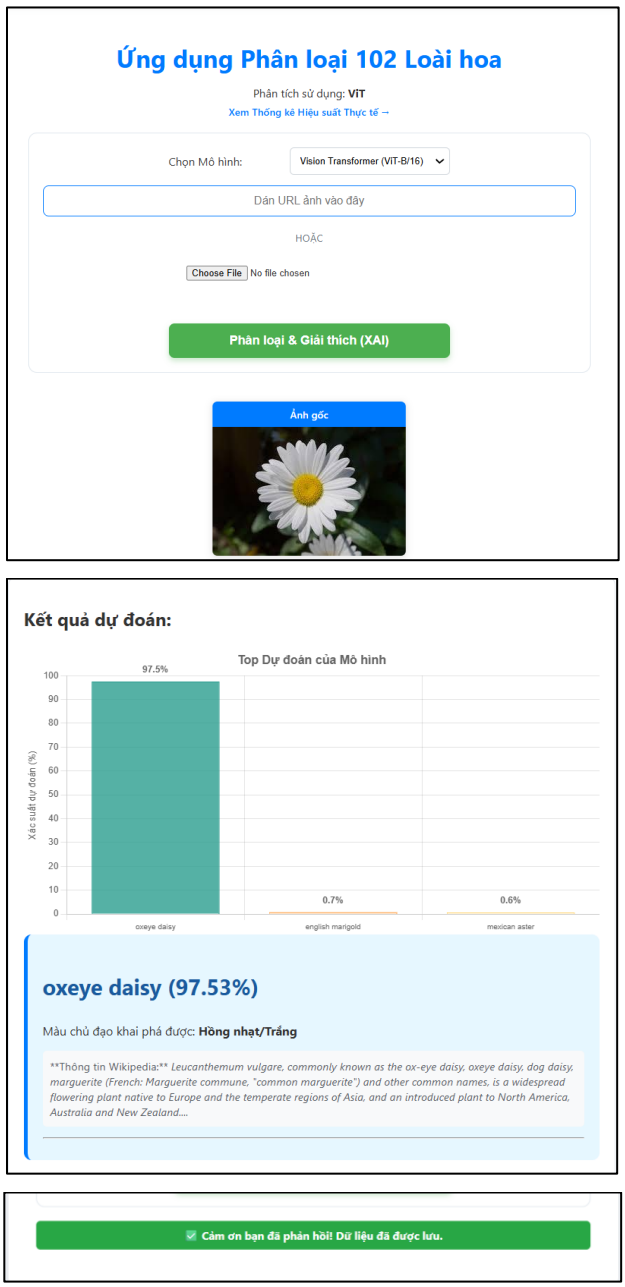


Hình 44: Biểu diễn ma trận nhầm lẫn lớp 91 đến 100



Hình 45: Biểu diễn ma trận nhầm lẫn lớp 101 đến 102

3.7 Kết quả của web-app với mô hình tốt nhất



Hình 46: Dự đoán phân loại ảnh trên web-app

Ứng dụng sử dụng mô hình học sâu Vision Transformer (ViT-B/16) để phân tích hình ảnh hoa. Khi người dùng tải lên ảnh hoặc nhập URL, mô hình sẽ trích xuất đặc trưng từ hình ảnh và so sánh với dữ liệu huấn luyện để xác định loài hoa. Sau đó, nó hiển thị kết quả dự đoán kèm theo mức độ tin cậy và mô tả ngắn về loài hoa đó. Tính năng XAI (Giải thích AI) giúp người dùng hiểu lý do vì sao mô hình đưa ra độ

chính xác, ví dụ như dựa vào màu sắc, hình dạng cánh hoa, hoặc các đặc điểm nổi bật khác. Và cũng có thể click mô hình EfficientNetB1 để dự đoán hoa.

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

Đề tài "Phân loại hoa – Flower Classification" đã ứng dụng thành công các mô hình học sâu như EfficientNet-B0, EfficientNet-B1 và Vision Transformer (ViT-B/16) trên bộ dữ liệu Oxford 102 Flower Dataset.

Kết quả cho thấy ViT-B/16 đạt độ chính xác cao nhất, khẳng định hiệu quả của kiến trúc Transformer trong bài toán phân loại hình ảnh.

Đề tài giúp củng cố kiến thức về khai phá dữ liệu, transfer learning, và quy trình huấn luyện mô hình học sâu, đồng thời mở ra khả năng ứng dụng trong nhận dạng hoa thực tế.

Trong tương lai, nhóm mong muốn:

- Mở rộng tập dữ liệu để tăng khả năng tổng quát của mô hình.
- Thử nghiệm với các mô hình mới như Swin Transformer hoặc EfficientNet-B4.
- Phát triển ứng dụng web hoặc di động thực tế để người dùng có thể tải ảnh và nhận diện loài hoa trực tiếp.