ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

BÁO CÁO ĐỒ ÁN DỰ ĐOÁN BỆNH CHO CÂY TÁO

Môn học: Thị giác máy tính nâng cao

Giảng viên hướng dẫn: Mai Tiến Dũng

Nhóm sinh viên thực hiện: Lê Trung Hiếu - 18520738

Nguyễn Thanh Điền - 18520600

HỌC KỲ II NĂM HỌC 2020 - 2021

Lời cảm ơn

Nhóm xin chân thành gửi lời cảm ơn đến TS. Mai Tiến Dũng – Phó trưởng khoa Khoa học máy tính, Trường Đại học Công nghệ thông tin, Đại học Quốc gia Thành phố Hồ Chí Minh, đồng thời là giảng viên giảng dạy lớp Thị giác máy tính nâng cao - CS331.L22.KHCL, đã tận tình giảng dạy cho lớp học và góp ý trong quá trình nhóm thực hiện đồ án.

Mặc dù nhóm đã cố gắng hoàn thành đồ án một cách hoàn thiện nhất, song không thể tránh khỏi những sai sót không mong muốn trong quá trình thực hiện. Nhóm mong rằng sẽ nhận được những đóng góp và nhận xét chân thành từ quý Thầy/Cô, các bạn sinh viên và người đọc để đồ án ngày càng hoàn thiện hơn nữa. Mọi thắc mắc hoặc ý kiến đóng góp xin vui lòng gửi email về một trong các địa chỉ email sau:

18520738@gm.uit.edu.vn (Lê Trung Hiếu)

18520600@gm.uit.edu.vn (Nguyễn Thanh Điền)

Thành phố Hồ Chí Minh, tháng 7 năm 2021

Nhóm sinh viên thực hiện

Lê Trung Hiếu, Nguyễn Thanh Điền

Mục lục

1. Giới thiệu bài toán

1.1 Giới thiệu chung

Hiện nay, vườn táo đang bị đe dọa thường xuyên bởi một số lượng lớn mầm bệnh và côn trùng. Việc chẩn đoán không chính xác và chậm trễ có thể dẫn đến việc sử dụng quá nhiều hoặc không đủ hóa chất, phân báo, làm tăng chi phí sản xuất, ảnh hưởng đến môi trường và sức khỏe. Hiện nay, việc phát hiện bệnh và sâu bệnh trong các vườn táo thương mại phụ thuộc vào việc dò tìm thủ công của các nhà tư vấn cây trồng. Do vậy, rất nhiều khó khăn vẫn còn tồn tại như: tỉ lệ **chẩn đoán sai** cao, **tốn kém** chi phí, thời gian. Với sự phát triển của mấy tính trong thời đại 4.0, việc xây dựng 1 hệ thống thị giác máy tính để chẩn đoán thay cho con người giúp tăng hiệu quả và tốc độ chẩn đoán cũng như độ chính xác là hoàn toàn có thể.

Hình 1: Bệnh nấm (Apple scab - trái), bệnh gỉ sắt (Apple rush – phải) là những bệnh phổ biến trên vườn táo ảnh hưởng đến vẻ ngoài thẩm mỹ, chất lượng và sản lượng quả.

1.2 Giới thiệu cuộc thi Plant Pathology 2020 - FGVC7

Hình 2: Minh họa cho ảnh trong tập dữ liệu.

Cuộc thi được tổ chức trên kaggle vào năm 2020 thu hút 1300 đội tham gia nhằm khuyến khích giải quyết bài toán chẩn đoán loại bệnh cho cây táo dựa trên lá của nó.

Yêu cầu của cuộc thi: Cho ảnh lá của một cây táo , chẩn đoán các loại bệnh cho cây táo vào **4 loại** :

1. Healthy (khoẻ mạnh)
2. Rush (bệnh rỉ sắt)
3. Scab (bệnh nấm)
4. Multiple_diseases (nhiều loại bệnh)

Do nhóm nhận thấy cuộc thi thú vị và có nhiều khả năng ứng dụng trong thực tế nên nhóm quyết định sẽ giải quyết bài toán trong cuộc thi này.

2. Định nghĩa bài toán

Input: Ảnh chụp lá của một cây táo

Hình 3: Minh họa cho ảnh input.

Output: Loại bệnh được chẩn đoán của cây táo đó

3. Tập dữ liệu

Lấy từ cuộc thi Cuộc thi **Plant Pathology 2020 - FGVC7**
Gồm **3642** hình ảnh lá cây táo được chụp với nhiều góc độ, độ sáng khác khác nhau
Kích thước ảnh phần lớn là **(1365, 2048)**, còn lại là **(2048, 1365)**

Hình 4: Hình ảnh từ tập dữ liệu với nhiều góc độ, độ sáng khác nhau.

Tập train: 1821 ảnh và 1 file .csv chứa nhãn của từng ảnh

Hình 5: Hình ảnh mẫu từ tập dữ liệu cho thấy các loại bệnh.

a) Bệnh rỉ sắt (rush)
b) Bệnh nấm (scab)
c) Nhiều bệnh trên một lá (multiple_diseases)

	A	B	C	D	E	F
1	image_id	healthy	multiple_	rust	scab	
2	Train_0	0	0	0	1	
3	Train_1	0	1	0	0	
4	Train_2	1	0	0	0	
5	Train_3	0	0	1	0	
6	Train_4	1	0	0	0	
7	Train_5	1	0	0	0	
8	Train_6	0	1	0	0	
9	Train_7	0	0	0	1	
10	Train_8	0	0	0	1	

Hình 6: Ảnh minh họa cho nhãn của từng ảnh trong tập train.

Số lượng ảnh của từng nhãn trong tập train như sau:

1. Healthy: 516 ảnh
2. Rust: 622 ảnh
3. Scab: 592 ảnh
4. Multiple diseases: 91

Ta có thể dễ dàng nhận thấy đây là tập dữ liệu không cân bằng, gây ảnh hưởng đến việc huấn luyện mô hình.

Tập test: 1821 ảnh không có nhãn

Hình 7: Ảnh minh họa cho tập test.

4. Phương pháp giải quyết

4.1 Cơ sở lý thuyết

Transfer learning

Transfer learning là việc sử dụng lại mô hình đã được đào tạo trước để giải quyết một vấn đề mới

VD: Đào tạo trước mô hình trên một tập dữ liệu khổng lồ như ImageNet và sau đó trích xuất các đặc trưng từ nó để sử dụng cho một mục đích khác

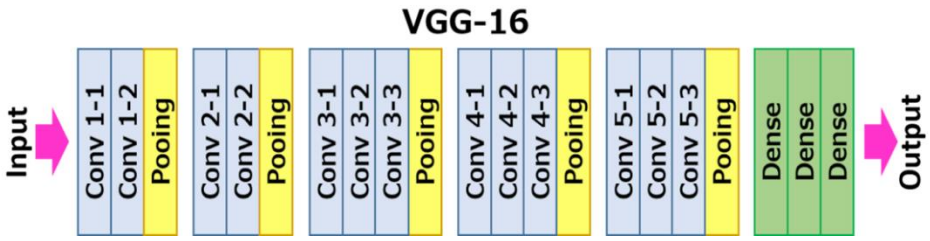
Lợi ích :

- Tiết kiệm thời gian
- Cải thiện hiệu suất học tập và giảm công sức thu thập dữ liệu

=> Vì vậy, nhóm quyết định sử dụng lại 2 mô hình VGG16 và VGG19 trong công đoạn trích xuất đặc trưng của bài toán.

4.2 Mạng VGG16

4.2.1 Tổng quan



Hình 8: Kiến trúc mạng VGG16.

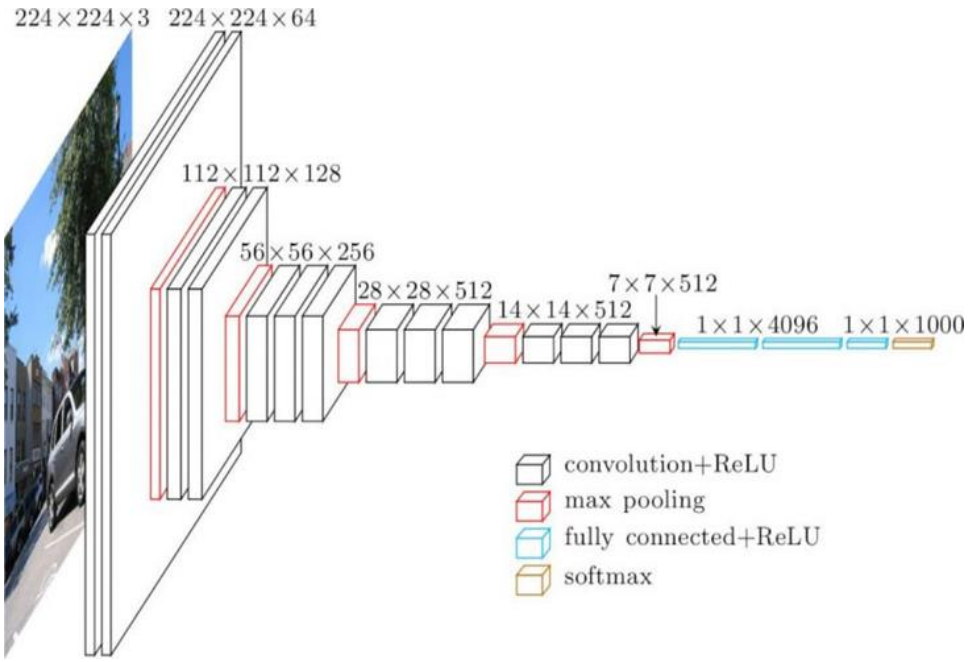
VGG16 là một mô hình mạng nơ-ron tích chập được đề xuất bởi Karen Simonyan và Andrew Zisserman từ Đại học Oxford trong bài báo “Very Deep Convolutional Networks for Large-Scale Image Recognition”, và ILSVRC-2014.

Mô hình này đạt độ chính xác top 5 là 92,7% trong ImageNet 14 triệu hình ảnh, 1000 lớp).

VGG16 đã được train trong nhiều tuần và đang sử dụng GPU NVIDIA Titan Black.

VGG16 được sử dụng nhiều trong trong cái bài toán phân lớp.

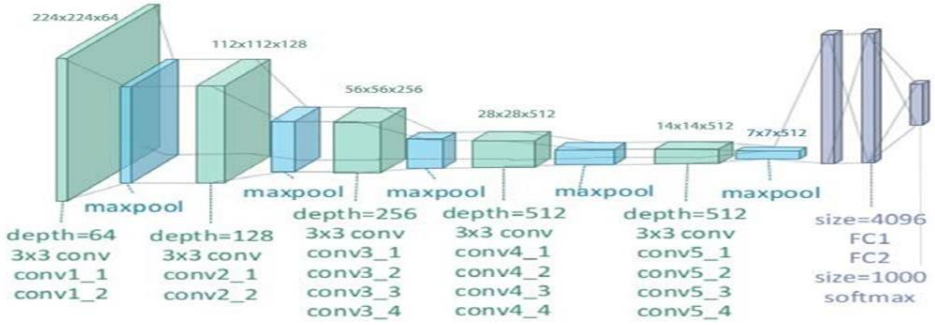
4.2.2 Chi tiết



Hình 9: Kiến trúc của mạng VGG16.

- Input: một bức ảnh RGB kích thước 224x224
- Outptut: một vector 1000 giá trị biểu diễn xác suất phân lớp

4.3 Mạng VGG19



Hình 10: Kiến trúc của mạng VGG19.

- VGG19 là một biến thể của mô hình VGG, ngắn gọn là bao gồm 19 lớp cũng được train trên tập dữ liệu ImageNet cùng năm với VGG16.
- VGG19 có cấu trúc tương tự như VGG16.

4.4 Xây dựng mô hình mạng dựa trên VGG16, VGG19

Nhóm đã thiết kế 2 mạng tích chập với phần base là VGG16, VGG19 dùng để trích xuất đặc trưng và 1 lớp fully-connected với hàm kích hoạt softmax dùng để phân lớp và ánh xạ kết quả phân lớp theo xác suất.

```
Model type: VGG16
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Functional)           (None, 7, 7, 512)         14714688
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
dense (Dense)                (None, 4)                 100356
=================================================================
Total params: 14,815,044
Trainable params: 100,356
Non-trainable params: 14,714,688
```

Hình 11: Mô hình mạng đã thiết kế với phần base VGG16.

```
Model type: VGG19
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg19 (Functional)           (None, 7, 7, 512)         20024384
_________________________________________________________________
flatten_1 (Flatten)          (None, 25088)             0
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 100356
=================================================================
Total params: 20,124,740
Trainable params: 100,356
Non-trainable params: 20,024,384
```

Hình 12: Mô hình mạng đã thiết kế với phần base VGG19.

4.5 Mô hình mạng CNN tham khảo

Nhóm đã tham khảo một mô hình CNN trong phần ví dụ của chương 5 - "Deep learning for computer vision", trong sách "Deep Learning with Python" phiên bản đầu tiên, của tác giả François Chollet - tác giả của thư viện Keras.

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(len(CLASS_NAMES), activation='softmax'))

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 222, 222, 32)      896
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 111, 111, 32)      0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 109, 109, 64)      18496
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 54, 54, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 52, 52, 64)        36928
_________________________________________________________________
flatten_1 (Flatten)          (None, 173056)            0
_________________________________________________________________
dense_2 (Dense)              (None, 64)                11075648
_________________________________________________________________
dense_3 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_4 (Dense)              (None, 4)                 260
=================================================================
Total params: 11,136,388
Trainable params: 11,136,388
Non-trainable params: 0
_________________________________________________________________
```

Hình 13: Kiến trúc mô hình CNN tham khảo từ sách.

Sau khi tiến hành việc huấn luyện mô hình trên, nhóm nhận thấy mô hình nhanh chóng đạt được kết quả khá cao qua mỗi epoch, hình dưới.

```
Epoch 1/10
56/56 [==============================] - 22s 357ms/step - loss: 1.2804 - auc_2: 0.6343
Epoch 2/10
56/56 [==============================] - 20s 350ms/step - loss: 1.2282 - auc_2: 0.6547
Epoch 3/10
56/56 [==============================] - 20s 352ms/step - loss: 1.2344 - auc_2: 0.6631
Epoch 4/10
56/56 [==============================] - 20s 349ms/step - loss: 1.2288 - auc_2: 0.6712
Epoch 5/10
56/56 [==============================] - 20s 348ms/step - loss: 1.2094 - auc_2: 0.6773
Epoch 6/10
56/56 [==============================] - 20s 349ms/step - loss: 1.2196 - auc_2: 0.6812
Epoch 7/10
56/56 [==============================] - 20s 348ms/step - loss: 1.2024 - auc_2: 0.6865
Epoch 8/10
56/56 [==============================] - 19s 345ms/step - loss: 1.1869 - auc_2: 0.6900
Epoch 9/10
56/56 [==============================] - 19s 347ms/step - loss: 1.2073 - auc_2: 0.6931
Epoch 10/10
56/56 [==============================] - 20s 348ms/step - loss: 1.1775 - auc_2: 0.6968
57/57 [==============================] - 1s 17ms/step
```

Hình 14: Kết quả khi train với mô hình tham khảo.

Nhưng khi đăng tải kết quả lên cuộc thi trong Kaggle thì kết quả lại khá tệ, tập test private: 0.47772, tập test public: 0.53274. Kết quả khá tệ này có thể là do mô hình có khá nhiều lớp, trong khi số lượng dữ liệu của tập train lại khá ít, vì thế mô hình học đã rất tốt trên tập train và khi test thì lại cho kết quả thấp hơn khoảng 20%. Vì thế nên nhóm quyết định sẽ tinh chỉnh kiến trúc của mô hình trên, bằng cách loại bỏ lớp convolution thứ 3 và một lớp fully connected, hình dưới, việc giảm bớt số layer cho kiến trúc mạng nhằm kiểm tra xem với ít layer hơn thì có tốt cho việc huấn luyện hay không.

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(len(CLASS_NAMES), activation='softmax'))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 222, 222, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 111, 111, 32)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 109, 109, 64)      18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 54, 54, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 186624)            0
_________________________________________________________________
dense (Dense)                (None, 64)                11944000
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 260
=================================================================
Total params: 11,963,652
Trainable params: 11,963,652
Non-trainable params: 0
_________________________________________________________________
```

Hình 15: Kiến trúc mô hình tham khảo sau khi thay đổi.

Tuy là đã được thay đổi kiến trúc mạng, nhưng kết quả test cũng vẫn không cải thiện được, cụ thể tập test private: 0.48455 và tập test public: 0.53161. Do đó, nhóm nghĩ rằng nguyên nhân phần lớn là do kiến trúc mạng và các hyper parameter chưa được tối ưu nên dẫn tới mô hình bị overfitting. Tập dữ liệu khá ít không phải là nguyên nhân model bị overfitting, vì thực tế là các nhóm tham gia cuộc thi được kết quả rất cao, thậm chí nhóm dẫn đầu còn được kết quả ở tập test private là 0.98445. Nhưng do không có đủ kinh nghiệm, trình độ để tinh chỉnh kiến trúc của model, các hyper parameter sao cho tối ưu và không muốn thử nghiệm ngẫu nhiên trong khi không hiểu rõ các thay đổi sẽ ảnh hưởng ra sao tới mô hình, nên nhóm quyết định không thay đổi thêm gì nữa.

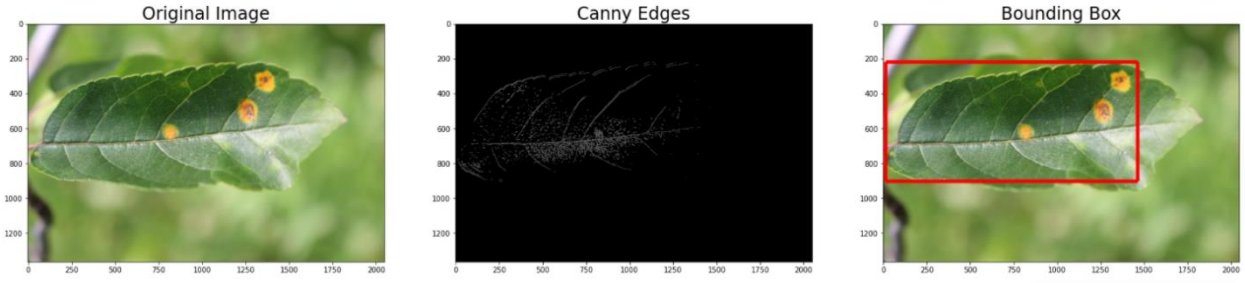
```
Epoch 1/10
56/56 [==============================] - 22s 369ms/step - loss: 1.2735 - auc_1: 0.6330
Epoch 2/10
56/56 [==============================] - 21s 370ms/step - loss: 1.2230 - auc_1: 0.6502
Epoch 3/10
56/56 [==============================] - 20s 365ms/step - loss: 1.2259 - auc_1: 0.6625
Epoch 4/10
56/56 [==============================] - 21s 369ms/step - loss: 1.2289 - auc_1: 0.6713
Epoch 5/10
56/56 [==============================] - 21s 367ms/step - loss: 1.2254 - auc_1: 0.6768
Epoch 6/10
56/56 [==============================] - 21s 367ms/step - loss: 1.2059 - auc_1: 0.6830
Epoch 7/10
56/56 [==============================] - 21s 367ms/step - loss: 1.2092 - auc_1: 0.6870
Epoch 8/10
56/56 [==============================] - 21s 375ms/step - loss: 1.1895 - auc_1: 0.6916
Epoch 9/10
56/56 [==============================] - 20s 363ms/step - loss: 1.1885 - auc_1: 0.6948
Epoch 10/10
56/56 [==============================] - 20s 364ms/step - loss: 1.1903 - auc_1: 0.6977
57/57 [==============================] - 1s 17ms/step
```

Hình 16: Kết quả khi huấn luyện mô hình tham khảo đã được sửa đổi.

4.6 Tiền xử lý

4.6.1 Edge dectection: Canny

Vì ảnh trong tập dữ liệu đều gồm những chiếc lá được gán nhãn và background, do đó, nhóm nghĩ rằng các mô hình sẽ học tốt hơn khi mà ảnh trong tập dữ liệu chỉ gồm phần lớn là lá được gán nhãn. Nhóm quyết định áp dụng thuật toán Edge detection là Canny để có thể cắt ra được phần lá được gán nhãn của bức ảnh. Nhóm quyết định chọn 2 threshold cho thuật toán Canny là 100 và 200 vì thấy rằng đây là con số thường được dùng. Với đầu ra của thuật toán Canny là một ảnh binary (đen trắng) chỉ gồm các cạnh, việc còn lại chỉ là cắt ra một bounding box bao phủ các pixel edge.



Hình 17: Áp dụng Canny để cắt ảnh.

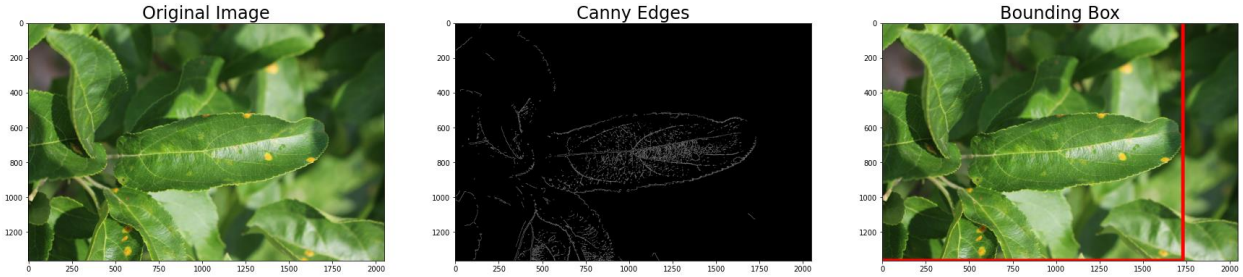
4.6.2 Resize + Padding

Vì đầu vào của 2 mạng VGG16 và VGG19 đều là (224, 224), tỉ lệ 1:1. Do ảnh sau khi cắt khó có thể có tỉ lệ 1:1, nếu resize ảnh về tỉ lệ 1:1, các vật thể trong ảnh sẽ bị biến dạng, nên nhóm quyết định sẽ resize ảnh và pad (thêm) các pixel màu đen để ảnh có thể có kích thước (224, 224) mà không bị biến dạng các vật thể trong ảnh.

Hình 18: Ảnh đã được cắt, sau đó được reszie + pad để có kích thước (224, 224).

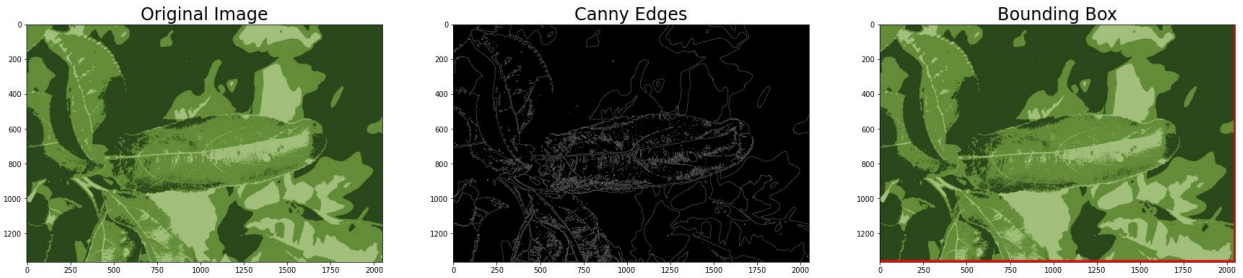
4.6.3 Áp dụng Segmentation: K-Means

Vì có nhiều ảnh sau khi áp dụng Canny thì có lẫn rất nhiều edge của các vật thể không được gán nhãn và background, hình dưới.



Hình 19: Thuật toán Canny lấy dư cả edge của các chiếc lá không được gán nhãn.

Nên nhóm có thử áp dụng segmentation, cụ thể là K-means (với k = 3) để segment ra được các màu chủ đạo của ảnh, nhóm nghĩ rằng việc có ít màu hơn sẽ làm cho ảnh có ít edge hơn và Canny sẽ có thể cho ra được ảnh binary tốt hơn khi mà số lượng edge đã được giảm. Nhưng thực tế lại ngược lại, ảnh sau khi được áp dụng K-means thì lại có nhiều edge hơn trước, nên Canny không những không lấy được ít edge hơn mà còn lấy edge của toàn bức ảnh, hình dưới.



Hình 20: Kết quả tệ sau khi áp dụng K-means, boundinh box đã bao phủ hết cả bức ảnh.

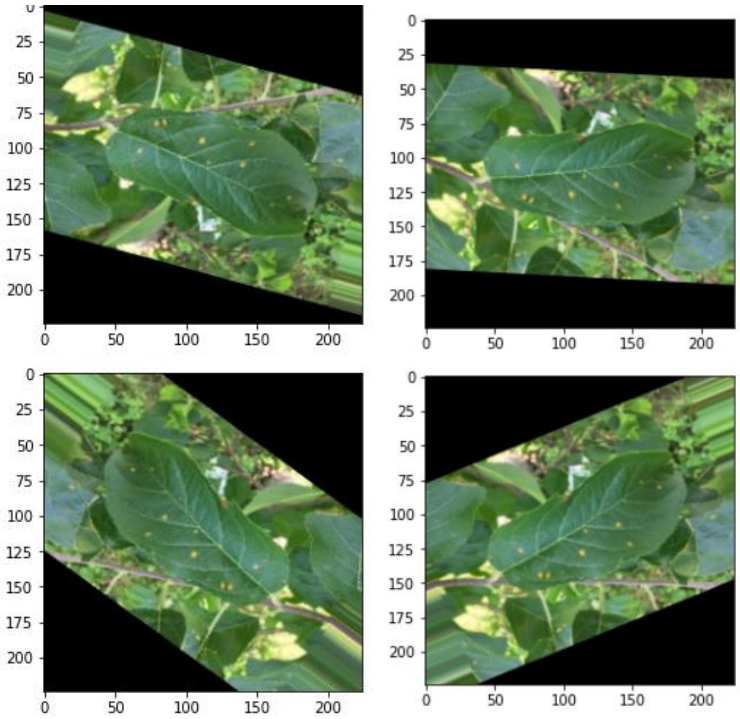
Khi k càng lớn thì ảnh sẽ càng giống ảnh gốc. Chỉ với k = 3 mà ảnh đã có quá nhiều edge, vậy thì càng tăng số k thì ảnh cũng chỉ càng giống như ban đầu, nghĩa là cũng giống như khi không áp dụng K-means. Vì thế, nhóm quyết định không áp dụng K-means (vì kết quả quá tệ) và các thuật toán segmentation khác (vì thời gian có hạn).

4.7 Data augmented

Vì số lượng ảnh để train là khá ít nên nhóm cũng có áp dụng data augmented khi huấn luyện để tăng độ đa dạng của ảnh từ những ảnh trong tập train. Nhóm đã sử dụng:

- Rotate: 40 độ
- Horizontal_flip
- Vertical_flip

Hình 21: Ảnh gốc 224x224.



Hình 22: Các ảnh đã được áp dụng các phép biến đổi.

5. Cài đặt thử nghiệm

5.1 Cài đặt chương trình

- Ngôn ngữ: Python 3
- Môi trường cài đặt: Google Colab
- Phần cứng: GPU miễn phí của Google Colab

Python 3: nhóm chọn vì ngôn ngữ này có rất nhiều thư viện hỗ trợ trong lĩnh vực Deep Learning, Computer Vision và khá dễ sử dụng.

Google Colab: vì các thành viên đều đã từng gặp khá nhiều khó khan khi cài đặt các thư viện hỗ trợ Deep Learning, Computer Vision, nên đã dùng Google Colab để không phải tốn thời gian cài đặt thư viện, lo lắng về độ tương thích của các thư viện.

GPU miễn phí của Google Colab: Vì một thành viên – Hiếu, đã huấn luyện thử một mô hình trong lúc làm đồ án và nhận thấy khi huấn luyện bằng CPU thì độ đo Accuracy thấp hơn 20% so với khi huấn luyện bằng GPU. Ngoài ra, cộng động Deep Learning cũng chỉ huấn luyện các mô hình bằng GPU. Nhóm chọn GPU miễn phí là vì vấn đề tài chính của cả nhóm.

5.2 Phương thức đánh giá

Phương thức dùng để đánh giá là độ đo ROC AUC. Độ đo này được chọn là vì cuộc thi trên Kaggle cũng dùng độ đo này để đánh giá mô hình, nên nhóm cần phải dùng cùng độ đo này thì mới có thể biết được hiệu quả hiện tại của mô hình.

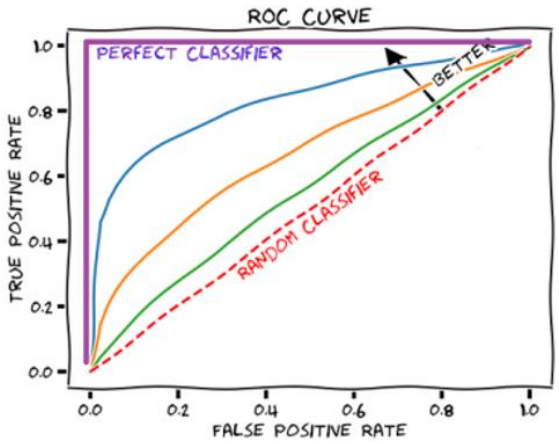
ROC (Receiver operating characteristic curve): là một đồ thị biểu diễn sự hiệu quả (performance) của mô hình phân lớp theo giá trị ngưỡng phân lớp. Đồ thị này được vẽ bằng cách tính TPR (True Positive Rate) và FPR (False Positive Rate) ở các mức threshold (giá trị để xác định xem bao nhiêu % là đủ để xem nhãn của mô hình dự đoán cho dữ liệu là nhãn của dữ liệu).

$$TPR = \frac{TP}{TP + FN}$$

- TP: True Positive
- FN: False Positive

$$FPR = \frac{FP}{FP + TN}$$

- FP: False Positive
- TN: True Negative



Hình 23: Biểu đồ ROC, Từng điểm trong biểu đồ là ứng với mỗi threshold cho mô hình.

AUC (Area under the ROC Curve): đo diện tích của phần bên dưới đường cong ROC. Vì mỗi trục của biểu độ ROC có chiều dài là 1 nên phần diện tích $0 \leq AUC \leq 1$.

Khi $AUC = 1$, thì mô hình phân loại có thể phân biệt hoàn hảo giữa tất cả các điểm của lớp Positive và lớp Negative một cách chính xác. Tuy nhiên, nếu $AUC = 0$, thì mô hình phân loại sẽ dự đoán tất cả dữ liệu Negative là Positive và tất cả dữ liệu Posivetive là Negative.

Khi $0.5 < AUC < 1$, có nhiều khả năng mô hình phân loại sẽ có thể phân biệt các dữ liệu lớp Positive với các dữ liệu lớp Negative. Bởi vì mô hình phân loại có thể phát hiện số lượng được nhiều True positives và True negatives hơn là False negatives và False positives.

Khi $AUC = 0.5$, thì mô hình phân loại không thể phân biệt giữa điểm các dữ liệu Positive và Negative. Có nghĩa là mô hình phân loại dự đoán lớp ngẫu nhiên cho các dữ liệu hoặc dự đoán duy nhất một lớp cho tất cả các điểm dữ liệu.

Vì vậy, AUC càng cao thì mô hình phân lớp sẽ càng tốt trong việc phân lớp giữa lớp Positive và lớp Negative.

Độ đo ROC AUC là dùng cho phân lớp nhị phân, nhưng bài toán mà nhóm đang giải quyết lại là phân loại nhiều lớp. Nên nhóm đã sử dụng phương pháp One-vs-all, nghĩa là xem một lớp là lớp Positive, các lớp còn lại sẽ thuộc về lớp Negative. Do đó, khi tính AUC cho “Healthy” xong thì sẽ tính AUC cho lần lượt các lớp khác, cuối cùng là tính trung bình cộng bằng 4 giá trị AUC tương ứng với mỗi lớp, đúng với cách mà cuộc thi trên Kaggle đã sử dụng.

5.3 Kết quả thực nghiệm

Vì có 4 mô hình nên sẽ có 4 kết quả thực nghiệm. Các kết quả đều là được đánh giá theo độ đo ROC AUC.

Conv base VGG16 + 1 FC:
• Train: 0.7394
• Test:
  ○ Private: 0.48835
  ○ Public: 0.51689
	Mô hình mạng tham khảo ban đầu:
	• Train: 0.6968
	• Test:
	  ○ Private: 0.49568
	  ○ Public: 0.53622

Conv base VGG19 + 1 FC:
• Train: 0.7397
• Test:
  ○ Private: 0.46924
  ○ Public: 0.5015
	Mô hình mạng tham khảo đã thay đổi:
	• Train: 0.6892
	• Test:
	  ○ Private: 0.48455
	  ○ Public: 0.53161

Bảng 1: Kết quả tổng kết của 4 mô hình.

Có thể thấy rõ là cả 4 mô hình đều bị overfitting, khi huấn luyện thì AUC ở mức ≈ 70%, nhưng khi test thì lại kém khoảng 20%. Nguyên nhân phần lớn là do các kiến trúc mạng và các hyper parameter chưa được tối ưu cho bài toán và tập dữ liệu hiện có.

6. Kết luận

Sau khi thực hiện xong đồ án thì nhóm có rút ra vài kết luận như sau:

- Cả 3 mô hình đã thử nghiệm chưa thể sử dụng trong thực tế
- Cần sử dụng thêm nhiều phương pháp xử lý ảnh và tinh chỉnh các của mô hình
- Tập dữ liệu khá ít và không cân bằng giữa các lớp khiến việc huấn luyện mô hình gặp nhiều khó khăn

Cả 3 mô hình đều bị overfitting, nếu áp dụng trong thực tế thì sẽ gây ra các mất mát không đáng có và ảnh hưởng nhiều tới vấn đề kinh tế, môi trường cho những người trồng táo, nên vẫn chưa thể sử dụng trong thực tế.

Do đó, cần phải xử lý ảnh tốt hơn ở bước tiền xử lý để loại bớt nhiễu. Tinh chỉnh các hyper parameter phù hợp hơn với bài toán và tập dữ liệu hiện có. Từ đó nâng cao được độ đo AUC cho mô hình.

Tập dữ liệu khá ít và không cân bằng giữa các lớp, 2 điều này góp phần ảnh hưởng không nhỏ đến việc huấn luyện mô hình. Khi thực hiện đồ án thì nhóm nhận thấy rằng vì 2 lý do trên, nên khó có thể tách ra được tập dữ liệu validation để mà có thể tinh chỉnh các hyper parameter. Vì không thể thấy được hiệu suất của mô hình qua các epoch nên nhóm đã khó có thể chọn ra được số epoch cho phù hợp, cùng với đó là chỉ số learning rate, hàm optimizer, số layer, số node (filter) cho các layer,…

7. Tham khảo

alkasm, n.d. *stackoverflow.* [Online]
Available at: https://stackoverflow.com/questions/44720580/resize-image-canvas-to-maintain-square-aspect-ratio-in-python-opencv
[Accessed 23 Jun 2017].

Anon., n.d. *Keras.* [Online]
Available at: https://keras.io/api/metrics/classification_metrics/#auc-class

Anon., n.d. *Machine Learning Crash Course.* [Online]
Available at: https://developers.google.com/machine-learning/crash-course/classification/check-your-understanding-roc-and-auc

Anon., n.d. *Quora.* [Online]
Available at: https://www.quora.com/What-is-the-difference-between-VGG16-and-VGG19-neural-network

Anon., n.d. *stackoverflow.* [Online]
Available at: https://stackoverflow.com/questions/9374747/optimal-approach-for-detecting-leaf-like-shapes-in-opencv
[Accessed 21 February 2012].

Anon., n.d. *YouTube.* [Online]
Available at: https://www.youtube.com/watch?v=A_ZKMsZ3f3o
[Accessed 6 March 2020].

Bhandari, A., n.d. *Analytics Vidhya.* [Online]
Available at: https://www.analyticsvidhya.com/blog/2020/06/auc-roc-curve-machine-learning/
[Accessed 16 June 2020].

Chollet, F., 2017. *Deep Learning with Python.* First Edition ed. s.l.:Manning.

doxygen, n.d. *OpenCV.* [Online]
Available at: https://docs.opencv.org/3.4/da/d22/tutorial_py_canny.html
[Accessed 1 July 2021].

Hassan, M. u., n.d. *Neurohive.* [Online]
Available at: https://neurohive.io/en/popular-networks/vgg16/
[Accessed 20 November 2018].

pawangfg, n.d. *Geeksforgeek.* [Online]
Available at: https://www.geeksforgeeks.org/vgg-16-cnn-model/
[Accessed 27 February 2020].

pknowledge, n.d. [Online]
Available at: https://gist.github.com/pknowledge/05d68179cdb364e4fa4db608517f8d17

Thapa, R., Snavely, N., Belongie, S. & Khan, A., 2020. *The Plant Pathology 2020 challenge dataset to classify foliar disease of apples.* s.l., ArXiv.