TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT THÀNH PHỐ HỒ CHÍ MINH

KHOA CƠ KHÍ CHẾ TẠO MÁY

BỘ MÔN CƠ ĐIỆN TỬ

BÁO CÁO CUỐI KÌ

TRÍ TUỆ NHÂN TẠO

NHẬN DẠNG BỆNH CỦA LÁ TRÊN CÂY CÀ CHUA

GVHD: PGS. TS Nguyễn Trường Thịnh

MHP: ARIN337629_22_2_09

Họ và tên: Lê Đăng Khoa

MSSV: 20146497

Năm học: Học kì 2 2022 - 2023

Thành phố Hồ Chí Minh, tháng 5 năm 2023

MỤC LỤC

CHƯƠNG 1. TỔNG QUAN

1.1. Giới thiệu về bệnh trên lá của cây cà chua

Hình 1.1. Các bệnh phổ biến trên lá cây cà chua

Lá cây cà chua thường bị nhiễm các bệnh gây hại như bệnh đốm lá, bệnh gỉ sắt, nấm và vi khuẩn. Các bệnh này gây ảnh hưởng tiêu cực đến sự phát triển và năng suất của cây trồng. Việc nhận dạng sớm và xử lý kịp thời các bệnh trên lá cà chua là rất quan trọng để ngăn chặn sự lây lan và giảm thiểu thiệt hại.

Những nguyên nhân phổ biến gây bệnh trên lá cây cà chua:

- Nấm và vi khuẩn: Một số bệnh nấm phổ biến trên cây cà chua bao gồm bệnh đốm lá, bệnh thối trái, và bệnh thối gốc. Những bệnh này thường do nấm và vi khuẩn gây ra thông qua việc xâm nhập và tấn công lá, quả và hệ thống rễ của cây.
- Côn trùng: Một số loại côn trùng như bọ cánh cứng, bọ cánh cam và rệp cánh nâu có thể gây ra các vết châm, tổn thương hoặc lây lan vi khuẩn lên lá cây cà chua. Điều này có thể gây ra các triệu chứng như vết rạn nứt, thối lá, và chết cây.
- Điều kiện thời tiết: Một số bệnh trên cây cà chua phát triển và lây lan dễ dàng trong điều kiện thời tiết ẩm ướt hoặc nhiệt độ cao, như bệnh nấm mốc trắng.

Theo số liệu của Tổ chức Nông nghiệp và Lương thực Liên Hiệp Quốc (FAO), mức độ thiệt hại do bệnh trên lá cây cà chua có thể lên tới 40-50% của tổng năng suất. Điều này đồng nghĩa với việc hàng triệu tấn cà chua bị mất đi hàng năm do bệnh tác động. Ngoài ra, bệnh trên lá cây cà chua cũng gây ra mất môi trường, tốn kém nguồn tài nguyên và tăng chi phí sản xuất trong nông nghiệp.

Nghiên cứu cũng chỉ ra rằng bệnh trên lá cây cà chua không chỉ gây thiệt hại về năng suất, mà còn ảnh hưởng đến chất lượng sản phẩm. Các bệnh nấm và vi khuẩn có thể làm giảm giá trị thương mại của cà chua bằng cách gây sự biến dạng, thối rữa và làm mất đi tính thẩm mỹ của quả.

1.2. Lý do chọn đề tài

Như đã đề cập ở 1.1 Bệnh trên lá cây cà chua có thể gây tác động kinh tế đáng kể đến ngành nông nghiệp. Thiệt hại năng suất và chất lượng sản phẩm do bệnh tác động có thể làm giảm lợi nhuận của nhà nông và gây thiệt hại cho nền kinh tế địa phương. Ngoài ra, việc sử dụng thuốc bảo vệ thực vật để kiểm soát bệnh có thể gây tác động tiêu cực đến môi trường. Vì vậy, việc phát triển một phương pháp nhận dạng bệnh tự động và chính xác có thể giúp giảm thiểu tác động kinh tế và môi trường.

Việc nhận dạng bệnh trên lá cây cà chua thông qua quan sát trực tiếp và kinh nghiệm của nhà nông có thể gặp khó khăn và chưa đảm bảo độ chính xác cao. Các triệu chứng của bệnh có thể tương đối tương đồng và khó phân biệt, đặc biệt khi quá trình mắc bệnh ở giai đoạn đầu. Việc sử dụng mô hình học máy và mạng nơ-ron tích chập (CNN) có thể cung cấp một phương pháp tự động và chính xác để nhận dạng bệnh trên lá cây cà chua.

Với những lợi ích và tiềm năng mà việc sử dụng mô hình CNN mang lại, em tin rằng với đề tài đề tài: "**Nhận dạng bệnh của lá trên cây cà chua"** này sẽ đóng góp vào việc cải thiện quy trình nhận dạng và kiểm soát bệnh trên lá cây cà chua, góp phần nâng cao năng suất, chất lượng và bền vững trong ngành trồng cây cà chua.

1.3. Mục tiêu nghiên cứu

- Xây dựng mô hình nhận dạng bằng các hình ảnh dữ liệu dựa trên phương pháp CNN
- Tạo một giao diện người dùng (GUI) đơn giản để mọi người sử dụng

1.4. Phương pháp nghiên cứu

- Tìm hiểu nguyên nhân gây ra bệnh trên lá cây cà chua, tiến hành lấy dữ liệu bằng cách tìm kiếm trên internet và một phần dữ liệu từ Kaglee
- Tìm hiểu các phương pháp hiện có thông qua Github, Youtube…
- Xây dựng "Model trainning" và tiến hành chạy thử để kiểm chứng

1.5. Nội dung báo cáo

Bài báo cáo tập trung vào việc nghiên cứu phương pháp tối ưu để tạo model training bằng CNN sao cho độ chính xác đạt cao nhất. Từ đó đưa ra chuẩn đoán lá cây có bị bệnh hay không. Model tạo ra sẽ được đưa vào giao diện người dùng được xây dựng bằng Python language trên Microsoft Visual Studio.

CHƯƠNG 2. CƠ SỞ LÝ THUYẾT

2.1. Thuật toán CNN - Convolutional Neural Network

2.1.1. Convolutional Neural Network là gì

CNN (Convolutional Neural Network) là một loại mạng nơ-ron nhân tạo được sử dụng rộng rãi trong lĩnh vực thị giác máy tính và xử lý ảnh. Được giới thiệu bởi Yann LeCun và đồng nghiệp vào những năm 1990, CNN đã trở thành một trong những thuật toán quan trọng nhất trong việc nhận dạng và phân loại hình ảnh.

Để tìm hiểu chi tiết về mô hình CNN, hãy đến với phần tiếp theo để có thể hiểu rõ hơn về mô hình.

2.1.2. Cấu trúc của mạng CNN

Mô hình CNN bao gồm một chuỗi các lớp, trong đó mỗi lớp có một nhiệm vụ cụ thể và thực hiện một phép biến đổi trên dữ liệu đầu vào. Dưới đây là một giới thiệu chi tiết về các lớp quan trọng trong mô hình CNN:

- Lớp đầu vào (input layer):
  - Lớp đầu tiên của mô hình, nhận dữ liệu đầu vào dưới dạng tensor (thường là hình ảnh).
  - Kích thước của lớp đầu vào phụ thuộc vào kích thước hình ảnh và số lượng kênh màu.
- Lớp tích chập (Convolutional Layer):
  - Lớp tích chập là trung tâm của mô hình CNN và thực hiện việc trích xuất đặc trưng từ dữ liệu đầu vào.
  - Mỗi lớp tích chập sử dụng một tập hợp các bộ lọc (filters) để thực hiện phép tích chập trên đầu vào.
  - Kết quả của phép tích chập là các bản đồ đặc trưng biểu diễn thông tin cấu trúc trong dữ liệu.
- Lớp kích hoạt (Activation Layer):
  - Sau mỗi lớp tích chập, một lớp kích hoạt thường được áp dụng để đưa ra một phản hồi phi tuyến tính cho mạng.
  - Hàm kích hoạt phổ biến nhất là ReLU (Rectified Linear Unit), nhưng cũng có thể sử dụng các hàm khác như Sigmoid hoặc Tanh.

- Lớp gộp (Pooling Layer):
    - Lớp gộp được sử dụng để giảm kích thước không gian của các bản đồ đặc trưng và giảm độ phức tạp tính toán.
    - Một kỹ thuật phổ biến trong lớp gộp là lớp gộp tối đa (max pooling), nơi giá trị lớn nhất trong một cửa sổ được chọn để trích xuất và làm đại diện cho khu vực tương ứng.
- Lớp đầy đủ kết nối (Fully Connected Layer):
    - Lớp đầy đủ kết nối sử dụng các nơ-ron để kết nối các bản đồ đặc trưng đã được trích xuất với lớp đầu ra.
    - Mỗi nơ-ron trong lớp đầy đủ kết nối kết nối với tất cả các nơ-ron trong lớp trước đó và truyền tải thông tin qua mạng.
- Lớp đầu ra (Output Layer):
    - Lớp đầu ra cuối cùng của mô hình chứa các nơ-ron đưa ra dự đoán cho các lớp đầu vào.
    - Số lượng nơ-ron trong lớp đầu ra phụ thuộc vào số lượng lớp phân loại hoặc số lượng giá trị đầu ra mong muốn.

Mô hình CNN có thể có nhiều lớp tích chập và lớp kích hoạt xen kẽ với nhau để trích xuất và học các đặc trưng cấu trúc phức tạp từ dữ liệu đầu vào. Các kiến trúc CNN nâng cao như VGGNet, ResNet và InceptionNet sử dụng các khối lặp và kết nối cắt ngang để nâng cao hiệu suất của mô hình. Ở bài báo cáo này em sẽ không sử dụng mô hình Pre-train sẵn mà sẽ tự huấn luyện mô hình dựa vào hình ảnh tự thu thập.

2.2. Thư viện Tensorflow

TensorFlow là một thư viện mã nguồn mở được phát triển bởi Google, đặc biệt dành cho việc xây dựng và triển khai các mô hình học máy và học sâu. Với TensorFlow, người dùng có thể tạo ra các mô hình học máy mạnh mẽ để giải quyết các vấn đề phức tạp từ nhận dạng hình ảnh, xử lý ngôn ngữ tự nhiên, đến dự đoán và phân loại dữ liệu.

Một trong những điểm mạnh của TensorFlow là kiến trúc đồ thị tính toán, cho phép người dùng mô hình hóa và tối ưu hóa các quá trình tính toán phức tạp bằng cách sử dụng các nút đại diện cho các phép toán và các cạnh đại diện cho dữ liệu. Điều này giúp TensorFlow tối ưu hóa hiệu suất tính toán và cung cấp khả năng phân phối tính toán trên nhiều máy tính hoặc thiết bị khác nhau.

TensorFlow hỗ trợ nhiều ngôn ngữ lập trình như Python, C++, Java và Go, giúp người dùng có thể sử dụng thư viện trong nhiều dự án khác nhau và trên nhiều nền tảng khác nhau. Nó cũng tích hợp tốt với các công cụ và framework phổ biến khác như Keras, scikit-learn và OpenCV.

Thư viện TensorFlow cung cấp nhiều lớp và hàm tiện ích để xây dựng các mạng neural và mô hình học sâu. Nó cung cấp một cách tiếp cận linh hoạt để xây dựng, đào tạo và triển khai các mô hình học máy và học sâu trên các tập dữ liệu lớn.

2.3. Thư viện Keras

Keras là một thư viện mã nguồn mở và cao cấp cho việc xây dựng và đào tạo mạng neural. Ban đầu được phát triển bởi François Chollet, Keras trở thành một phần của TensorFlow từ phiên bản 2.0. Thư viện này nhằm mục đích giúp người dùng dễ dàng tạo ra các mô hình học sâu một cách nhanh chóng và hiệu quả. Keras dựa trên cấu trúc tối thiểu, cung cấp một cách dễ dàng và dễ dàng để tạo các mô hình học sâu dựa trên TensorFlow hoặc Theano. Keras được thiết kế để xác định nhanh các mô hình học sâu.

Một trong những ưu điểm quan trọng của Keras là cú pháp đơn giản và dễ sử dụng. Với một số dòng mã ngắn, người dùng có thể tạo, đào tạo và đánh giá mô hình. Keras cung cấp một API dễ hiểu và trực quan, cho phép người dùng mô hình hóa các quá trình học máy một cách tự nhiên và dễ dàng. Đồng thời, Keras cũng tích hợp tốt với TensorFlow, cho phép người dùng sử dụng các tính năng mạnh mẽ của TensorFlow mà không cần lo lắng về việc cài đặt và quản lý chi tiết.

Keras có thể chạy trên nhiều nền tảng khác nhau mà tiêu biểu là Python. Thư viện này cũng có sẵn một số công cụ tiện ích bổ sung như tạo đồ thị, tăng tốc GPU và tạo mô hình song song, giúp tăng tốc quá trình đào tạo và triển khai mô hình trên các tài nguyên phần cứng mạnh mẽ.

2.4. Thư viện Scikit-learn (sklearn)

Scikit-learn (hay còn gọi là sklearn) là một thư viện Python mã nguồn mở phổ biến cho machine learning và data mining. Nó được xây dựng trên nền tảng của NumPy, SciPy và matplotlib, và cung cấp các công cụ đơn giản và hiệu quả để xây dựng và đào tạo các mô hình học máy.

Một trong những ưu điểm của scikit-learn là cú pháp đơn giản và dễ sử dụng. Các mô hình và thuật toán được triển khai theo một giao diện chung, cho phép người dùng dễ dàng xây dựng, huấn luyện và đánh giá các mô hình. Ngoài ra, scikit-learn cũng cung cấp các công cụ để tiền xử lý dữ liệu (data preprocessing) như chuẩn hóa dữ liệu, xử lý giá trị thiếu, và mã hóa biến rời rạc.

Scikit-learn cũng có tích hợp tốt với các công cụ và thư viện phổ biến khác trong hệ sinh thái Python như Pandas và NumPy, giúp người dùng dễ dàng làm việc với dữ liệu và tích hợp vào quy trình làm việc tổng thể.

Mặc dù được viết cho Python nhưng thực ra các thư viện nền tảng của scikit-learn lại được viết dưới các thư viện của C để tăng hiệu suất làm việc. Ví dụ như: Numpy (Tính toán ma trận), LAPACK, LibSVM và Cython.

Thư viện tập trung vào việc mô hình hóa dữ liệu. Nó không tập trung vào việc truyền tải dữ liệu, biến đổi hay tổng hợp dữ liệu. Những công việc này dành cho thư viện Numpy và Pandas.

2.5. GUI (GRAPHICAL USER INTERFACE)

Giao diện GUI được xây dựng bằng ngôn ngữ python trên Microsoft Visual Studio bằng cách thiết lập một giao diện Tkinter đọc ảnh, xử lý dữ liệu đã được training để từ đó so sánh giữa dữ liệu training và dữ liệu testing và hiển thị kết quả nhận dạng lên trên giao diện.

Giao diện được thiết kế đơn giản với mục đích mang lại sự tiện lợi để bất kì người dùng nào cũng có thể sử dụng.

CHƯƠNG 3. MÔ HÌNH CNN

3.1. Xây dựng mô hình – Trainning model

Trong báo cáo lần này, em sẽ sử dụng môi trường Google Colab để tiến hành xây dựng Model.

3.1.1. Datasets

Trong dự án này, em sử dụng dữ liệu về hình ảnh với tổng cộng 1150 hình ảnh tự thu thập được trên internet và cả Kagle. Dữ liệu này được chia ra thành 3 thư mục nhỏ: train (dùng để trainning model), test (dùng để test model), validation (dùng để kiểm định model).

Bảng 3.1. Datasets

	Loại ảnh	Số lượng ảnh	Tổng
train	HEALTHY	65	650
	DISEASE	585	
test	HEALTHY	30	300
	DISEASE	270	
val	HEALTHY	30	300
	DISEASE	270	

3.1.2. Xây dựng mô hình

Bước 1: Liên kết Google Drive

```
from google.colab import drive
drive.mount('/content/drive')
```

Bước 2: Khai báo các thư viện cần thiết

```
import os
import numpy as np
from os import listdir
import tensorflow as tf
from tensorflow import keras
from numpy import asarray,save
import matplotlib.pyplot as plt
from keras.optimizers import Adam
from keras.layers import LeakyReLU
from keras.models import Sequential
from keras.utils import load_img, img_to_array
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from keras.layers import Dense, Dropout, Conv2D, MaxPooling2D,Flatten
```

Bước 3: Lấy dữ liệu và mô tả dữ liệu

```
test_path="/content/drive/MyDrive/AI_Final_Report/Tomato_Leaf/TestTomato"
train_path="/content/drive/MyDrive/AI_Final_Report/Tomato_Leaf/TrainTomato"
val_path="/content/drive/MyDrive/AI_Final_Report/Tomato_Leaf/ValTomato"
```

Như mô tả ở bảng 3.1, dữ liệu em dùng để tạo mô hình có 3 thư mục bao gồm train_path, test_path và val_path, các dữ liệu bao gồm hình ảnh của lá cây cà chua khỏe mạnh và bị bệnh. Tất cả hình ảnh được tải lên Google Drive để sử dụng cho việc trainning.

Bước 3: Tăng cường dữ liệu (data augmentation)

Augmentation là kĩ thuật tạo ra dữ liệu training từ dữ liệu mà ta đang có. Việc này giúp chúng ta tạo ra nhiều hình ảnh hơn từ hình ảnh gốc.

```
train_datagen = ImageDataGenerator( rescale=1./255,
                rotation_range=20,
                width_shift_range=0.2,
                height_shift_range=0.2,
                shear_range=0.2,
                zoom_range=0.2,
                horizontal_flip=True,
                fill_mode='nearest')
test_datagen = ImageDataGenerator(rescale=1./255)
```

Trong phạm vi dữ liệu em có, em tiến hành tăng cường dữ liệu:

- **zoom_range**: Thực hiện phép thu/phóng ảnh ngẫu nhiên trong khoảng từ 0.8 đến 1.2. Giá trị dưới 1 cho biết phép thu nhỏ, và giá trị trên 1 cho biết phép phóng to.
- **rescale**: Thực hiện phép chia giá trị pixel của ảnh cho 255 để đưa về khoảng từ 0 đến 1. Điều này giúp chuẩn hóa dữ liệu.
- **rotation_range**: Xoay ảnh ngẫu nhiên trong khoảng góc từ -20 đến 20 độ.
- **width_shift_range** và **height_shift_range**: Dịch chuyển ngẫu nhiên ảnh theo chiều ngang và chiều dọc. Giá trị 0.2 ở đây cho biết rằng ảnh có thể dịch chuyển lên tới 20% của kích thước ban đầu.
- **shear_range**: Biến đổi cắt biến ảnh ngẫu nhiên với mức độ cắt từ -0.2 đến 0.2.
- **horizontal_flip**: Lật ảnh theo chiều ngang ngẫu nhiên.

Sau đó, tiến hành tạo các biến train_set, test_set, val_set để dữ liệu được tăng cường

```
train_set = train_datagen.flow_from_directory(train_path,
                          target_size=(240, 240),
                          batch_size=32,
                          class_mode='categorical')
test_set = test_datagen.flow_from_directory(test_path,
                          target_size=(240, 240),
                          batch_size=32,
                          class_mode='categorical')
val_set = train_datagen.flow_from_directory(val_path,
                          target_size=(240, 240),
                          batch_size=32,
                          shuffle=True,class_mode='categorical')
```

```
Found 650 images belonging to 10 classes.
Found 300 images belonging to 10 classes.
Found 300 images belonging to 10 classes.
```

Kết quả ta được train_set có 650 hình ảnh thuộc 10 lớp, test_set và val_set có mỗi loại 300 hình ảnh thuộc 10 lớp.

Bước 4: Xây dựn mô hình CNN

```
model = Sequential()
#Convolutional
model.add(Conv2D(32, kernel_size= (3,3), activation= 'relu',
input_shape= (240,240, 3), padding= 'same'))
model.add(MaxPooling2D((2,2), padding= 'same'))

model.add(Conv2D(64, kernel_size= (3,3), activation= 'relu', padding=
'same'))
```

```
model.add(MaxPooling2D((2,2), padding= 'same'))

model.add(Conv2D(128, (3,3), activation= 'relu', padding= 'same'))
model.add(MaxPooling2D((2,2), padding= 'same'))

model.add(Conv2D(256, (3,3), activation= 'relu', padding= 'same'))
model.add(MaxPooling2D((2,2), padding= 'same'))



#ANN
model.add(Flatten())
model.add(Dense(512, activation= 'relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation= 'softmax'))
model.summary()
```

Cấu trúc mô hình CNN được sử dụng trong đề tài cụ thể như sau:

- Lớp đầu tiên là lớp Convolutional với với 32 bộ lọc (filters) có kích thước 3x3, hàm kích hoạt relu và kích thước đầu vào là (240,240,3). Phép chập này sẽ tạo ra các đặc trưng từ ảnh đầu vào.
- Thêm một lớp MaxPooling với kích thước cửa sổ 2x2 và padding='same'. Lớp này sẽ giảm kích thước không gian của đặc trưng và tăng cường tính đại diện của chúng.
- Tiếp theo, các lớp Conv2D và MaxPooling2D tương tự được thêm vào mô hình với số lượng bộ lọc và kích thước khác nhau. Số lượng bộ lọc tăng dần theo từng lớp (64, 128, 256), trong khi kích thước của các đặc trưng được giữ nguyên (padding='same') và kích thước không gian của chúng giảm đi nửa (2x2) sau mỗi lớp MaxPooling2D.
- Thêm một lớp Flatten để biến đổi các đặc trưng 2D thành một vector 1D. Lớp này được sử dụng để chuyển từ một mô hình CNN sang một mô hình ANN truyền thống.
- Thêm một lớp fully-connected (dense) với 512 nơ-ron và hàm kích hoạt relu. Lớp này sẽ học các mối quan hệ tuyến tính giữa các đặc trưng đã được biến đổi.
- Thêm một lớp Dropout để ngẫu nhiên bỏ đi 50% các nơ-ron trong quá trình đào tạo. Lớp này giúp giảm hiện tượng overfitting.
- Thêm lớp fully-connected cuối cùng với 10 nơ-ron và hàm kích hoạt softmax. Lớp này được sử dụng để phân loại ảnh vào 10 lớp khác nhau.

- Cuối cùng là in ra thông tin tóm tắt về kiến trúc của mô hình, bao gồm tên các lớp, kích thước đầu vào/đầu ra của mỗi lớp và số lượng tham số trong mô hình.

Các thông số Input, Ouput cụ thể được thể hiện trong bảng thông số mô hình dưới dây:

Model: "sequential_2"

Layer (type)	Output Shape	Param #
conv2d_8 (Conv2D)	(None, 240, 240, 32)	896
max_pooling2d_8 (MaxPooling 2D)	(None, 120, 120, 32)	0
conv2d_9 (Conv2D)	(None, 120, 120, 64)	18496
max_pooling2d_9 (MaxPooling 2D)	(None, 60, 60, 64)	0
conv2d_10 (Conv2D)	(None, 60, 60, 128)	73856
max_pooling2d_10 (MaxPoolin g2D)	(None, 30, 30, 128)	0
conv2d_11 (Conv2D)	(None, 30, 30, 256)	295168
max_pooling2d_11 (MaxPoolin g2D)	(None, 15, 15, 256)	0
flatten_2 (Flatten)	(None, 57600)	0
dense_4 (Dense)	(None, 512)	29491712
dropout_2 (Dropout)	(None, 512)	0
dense_5 (Dense)	(None, 10)	5130

Total params: 29,885,258

Trainable params: 29,885,258

Non-trainable params: 0

Bước 5: Huấn luyện mô hình

Tiến hành Compile mô hình và huấn luyện mô hình

```
# Compile model
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
```

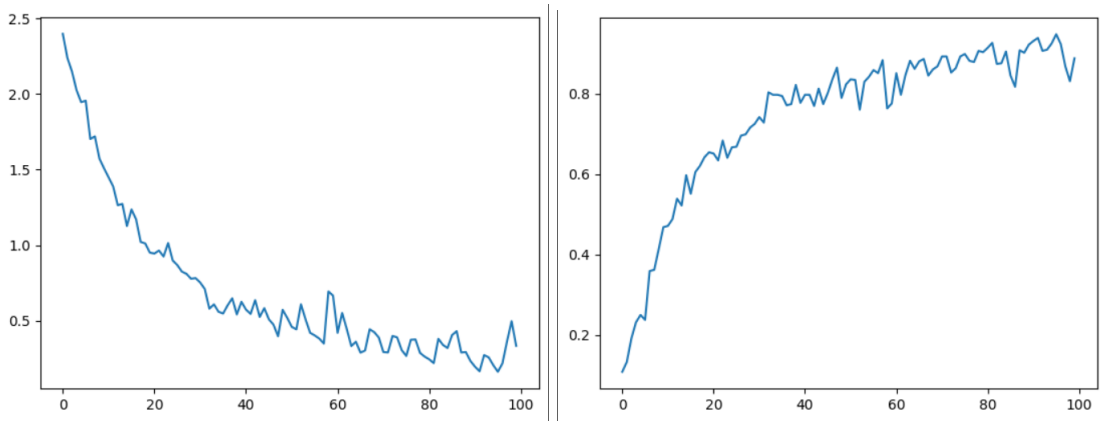
```
# Train model
history=model.fit_generator(train_set, epochs=100, verbose=1,
                                    validation_data=test_set)
```

Mô hình được huấn luyện với số lần lọc 100, ta được kết quả như sau:

```
Epoch 1/100

21/21 [==============================] - 24s 990ms/step - loss: 2.3985 -
accuracy: 0.1077 - val_loss: 2.2794 - val_accuracy: 0.1000

Epoch 2/100

21/21 [==============================] - 12s 588ms/step - loss: 2.2390 -
accuracy: 0.1323 - val_loss: 2.1488 - val_accuracy: 0.2267

Epoch 3/100

21/21 [==============================] - 12s 583ms/step - loss: 2.1494 -
accuracy: 0.1908 - val_loss: 2.1161 - val_accuracy: 0.1867

Epoch 4/100

21/21 [==============================] - 13s 605ms/step - loss: 2.0257 -
accuracy: 0.2308 - val_loss: 1.9044 - val_accuracy: 0.2367

Epoch 5/100

21/21 [==============================] - 12s 564ms/step - loss: 1.9464 -
accuracy: 0.2492 - val_loss: 1.9802 - val_accuracy: 0.2433

Epoch 6/100

21/21 [==============================] - 12s 593ms/step - loss: 1.9561 -
accuracy: 0.2369 - val_loss: 2.1008 - val_accuracy: 0.2400

Epoch 7/100

21/21 [==============================] - 12s 583ms/step - loss: 1.7028 -
accuracy: 0.3585 - val_loss: 1.7405 - val_accuracy: 0.3867

Epoch 8/100

21/21 [==============================] - 12s 588ms/step - loss: 1.7199 -
accuracy: 0.3615 - val_loss: 1.8431 - val_accuracy: 0.3467

Epoch 9/100

21/21 [==============================] - 16s 769ms/step - loss: 1.5726 -
accuracy: 0.4138 - val_loss: 1.7304 - val_accuracy: 0.4100

.....

.....


Epoch 95/100

21/21 [==============================] - 12s 573ms/step - loss: 0.2059 -
accuracy: 0.9246 - val_loss: 1.5046 - val_accuracy: 0.7367

Epoch 96/100
```

```
21/21 [==============================] - 12s 559ms/step - loss: 0.1622 -
accuracy: 0.9477 - val_loss: 2.3024 - val_accuracy: 0.6667

Epoch 97/100

21/21 [==============================] - 12s 585ms/step - loss: 0.2181 -
accuracy: 0.9231 - val_loss: 1.6339 - val_accuracy: 0.6833

Epoch 98/100

21/21 [==============================] - 12s 564ms/step - loss: 0.3636 -
accuracy: 0.8677 - val_loss: 3.6970 - val_accuracy: 0.5300

Epoch 99/100

21/21 [==============================] - 12s 575ms/step - loss: 0.4970 -
accuracy: 0.8308 - val_loss: 0.8802 - val_accuracy: 0.7800

Epoch 100/100

21/21 [==============================] - 11s 536ms/step - loss: 0.3341 -
accuracy: 0.8877 - val_loss: 1.3090 - val_accuracy: 0.7600
```

Từ kết quả mô hình huấn luyện, ta có được đồ thị độ chính xác theo số lần học là 100:



Loss **Train**

Hình 3.1. Đồ thị độ chính xác của mô hình và giá trị mất mát theo số lần học

Để tăng độ chính xác của mô hình, ta tiến hành giảm số lần học từ 100 thành 50, ta được kết quả như sau:

Epoch 1/50

21/21 [==============================] - 12s 554ms/step - loss: 0.3113 - accuracy: 0.8892 - val_loss: 0.8932 - val_accuracy: 0.7567

Epoch 2/50

21/21 [==============================] - 12s 565ms/step - loss: 0.2252 - accuracy: 0.9169 - val_loss: 1.3970 - val_accuracy: 0.7567

Epoch 3/50

21/21 [==============================] - 12s 574ms/step - loss: 0.2072 - accuracy: 0.9169 - val_loss: 1.4530 - val_accuracy: 0.7267

Epoch 4/50

21/21 [==============================] - 13s 613ms/step - loss: 0.1921 - accuracy: 0.9338 - val_loss: 0.7735 - val_accuracy: 0.8167

Epoch 5/50

21/21 [==============================] - 12s 579ms/step - loss: 0.2165 - accuracy: 0.9185 - val_loss: 1.4642 - val_accuracy: 0.7200

.....

.....

Epoch 45/50

21/21 [==============================] - 12s 558ms/step - loss: 0.3178 - accuracy: 0.9185 - val_loss: 0.8502 - val_accuracy: 0.8300

Epoch 46/50

21/21 [==============================] - 12s 562ms/step - loss: 0.1778 - accuracy: 0.9431 - val_loss: 1.7694 - val_accuracy: 0.7167

Epoch 47/50

21/21 [==============================] - 13s 600ms/step - loss: 0.1788 - accuracy: 0.9323 - val_loss: 2.4897 - val_accuracy: 0.6633

Epoch 48/50

21/21 [==============================] - 11s 519ms/step - loss: 0.1816 - accuracy: 0.9415 - val_loss: 3.0522 - val_accuracy: 0.6467

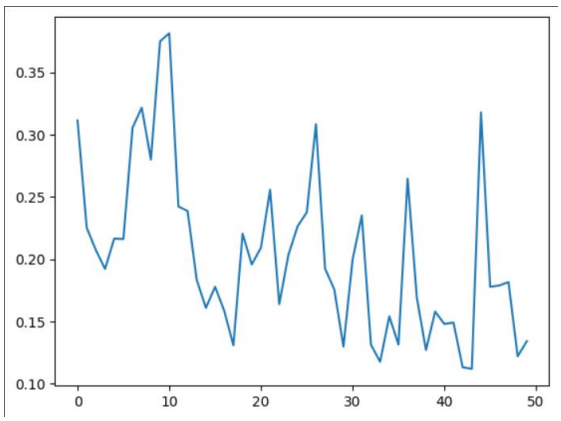
Epoch 49/50

21/21 [==============================] - 11s 526ms/step - loss: 0.1220 - accuracy: 0.9615 - val_loss: 0.5423 - val_accuracy: 0.8533

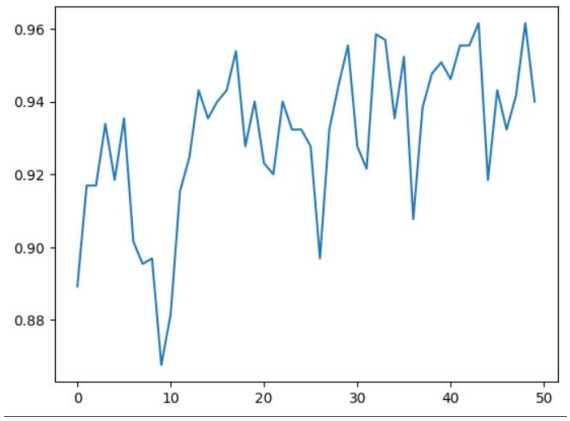
Epoch 50/50

21/21 [==============================] - 12s 592ms/step - loss: 0.1341 - accuracy: 0.9400 - val_loss: 2.0920 - val_accuracy: 0.7333

Từ kết quả mô hình huấn luyện, ta có được đồ thị độ chính xác theo số lần học:



Loss



Train

3.2. Xây dựng giao diện người dùng với Tkinter

Tkinter là một thư viện Python được sử dụng để phát triển giao diện đồ họa (GUI - Graphical User Interface). Tên "Tkinter" được tạo ra từ việc kết hợp của "Tk" - một toolkit giao diện người dùng đa nền tảng, và "inter" - viết tắt của "interface" (giao diện).

Tkinter cung cấp các đối tượng và phương thức cho phép bạn xây dựng và quản lý các thành phần GUI như cửa sổ, nút bấm, hộp văn bản, menu, và nhiều hơn nữa. Nó cung cấp các công cụ để xử lý sự kiện, tương tác người dùng và hiển thị thông tin trên giao diện người dùng.

Trong đề tài này, mô hình sau khi kết thúc quá trình training sẽ được lưu dưới dạng file .h5, model sẽ được load lên Microsoft Visual Studio tiến hành xử lí hình ảnh theo các thông số đầu vào như đã trainning ở trên và đưa ra dự đoán, giao diện được thiết kế đơn giản và dễ dùng, phù hợp với mọi lứa tuổi.

CHƯƠNG 4. KẾT LUẬN

4.1. Kết quả đạt được

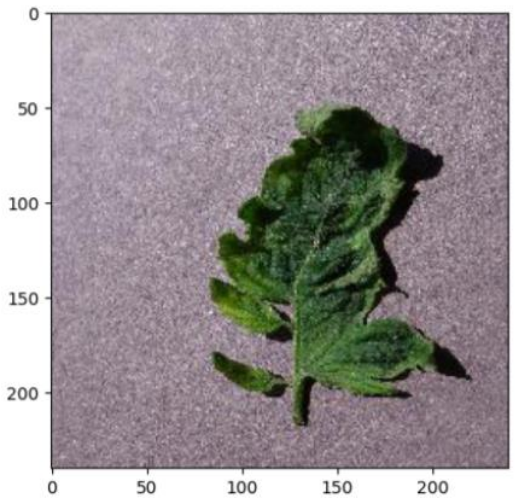
4.1.1. Mô hình chuẩn đoán

Mô hình CNN nhận dạng bệnh trên lá cây cà chua đạt được độ chính xác là 88% sau lần train đầu tiên với 100 lần học, và đạt 94% ở lần train thứ hai sau khi đã giảm số lần học xuống còn 50 lần.

Sử dụng google colab để chạy dự đoán, ta thu được kết quả sau:

```
Name_Disease_Leaf = ['Tomato_Bacterial_Spot_Disease',
                     'Tomato_EarlyBlight_Disease',
                     'Tomato_Healthy',
                     'Tomato_LateBlight_Disease',
                     'Tomato_LeafMold_Disease',
                     'Tomato_MosaicVirus_Disease ',
                     'Tomato_Septoria_LeafSpot_Disease ',
                     'Tomato_SpiderMites Two-spotted_SpiderMite_Disease ',
                     'Tomato_TargetSpot_Disease',
                     'Tomato_Tomato_YellowLeaf_CurlVirus_Disease']
filePath = '/content/drive/MyDrive/AI_Final_Report/Tomato_Leaf/TestTomato/MosaicVirus/Tomato___Tomato_mosaic_virus (13).JPG'
testImage = load_img(filePath, color_mode = 'rgb', target_size=(240,240))
arrayImage = img_to_array(testImage).reshape(1,240,240,3)/255
pred = int(np.argmax(model.predict(arrayImage)))
print("Predicted: ", Name_Disease_Leaf[pred])
plt.imshow(testImage)
```

```
1/1 [==============================] - 0s 80ms/step
Predicted:  Tomato_MosaicVirus_Disease
<matplotlib.image.AxesImage at 0x7f5f22cf4910>
```



Hình 4.1. Hình chuẩn đoán lá bị bệnh

4.1.2. GUI

Hình 4.2. Giao diện Tkinter

Khi bấm Start sẽ chuyển đến Frame tiếp theo để tiến hành chọn ảnh

Hình 4.3. Giao diện chuẩn đoán

Ở đây:

- nút “ Choose Test File “ sẽ dùng để lấy hình ảnh trong máy tính để tiến hành dự đoán
- nút “ Back ” để trở về giao diện trước đó
- nút “ Exit “ để thoát khỏi giao diện

Dưới đây là một số dự đoán từ kết quả của mô hình chạy trên giao diện:

Hình 4.5. Chuẩn đoán lá cây khỏe

Hình 4.6. Chuẩn đoán lá cây bị bênhư

4.2. Nhược điểm

Mô hình được huấn luyện với độ chính xác khá cao 94% nhưng vẫn nhận dạng sai ở một số hình ảnh, lí do ở đây là các loại bệnh có đặc trưng nhận dạng bên ngoài tương đối giống nhau.

Mô hình chưa kết hợp với nhận dạng trực tiếp trên thời gian thực nên tính ứng dugj vào thực tế còn chưa cao.

4.3. Hướng phát triển

Trong tương lai, mô hình sẽ được huấn luyện nhằm tang cường độ chính xác lên cao hơn, phát triển giao diện thông minh, tinh gọn và dễ sử dụng, kết hợp chạy thời gian thực để có thể sử dụng hình ảnh trực tiếp nhận được từ camera để tiến hành nhận dạng mà không cần phải có hình sẵn trong thiết bị

TÀI LIỆU THAM KHẢO

[1]. Tomato Detection using Python

Link truy cập: (141) Plant Leaf Disease Prediction using Deep learning - YouTube

[2]. 16 loại bệnh thường gặp trên cây cà chua

Link truy cập: 16 loại bệnh thường gặp trên cây cà chua và cách trị tận gốc (lisado.vn)

[3]. TOPDev, Thuật toán CNN – Convolutional Neural Network

Link truy cập: https://topdev.vn/blog/thuat-toan-cnn-convolutional-neural-network/

PHỤ LỤC

1. Link Github: khoabom02 (Le Dang Khoa) (github.com)
2. Link Youtube: (141) Final Report for AI (2022-2023) - YouTube