**ĐOÀN TNCS HỒ CHÍ MINH**

**TÓM TẮT DỰ ÁN**

**THAM GIA HỘI THI TIN HỌC TRẺ TOÀN QUỐC LẦN THỨ XX VII, NĂM 2021**

**TÊN DỰ ÁN:**

**NỀN TẢNG THIẾT KẾ GAME TĂNG TỶ LỆ CÂY TRỒNG CHO MÔI TRƯỜNG.**

**TÁC GIẢ: NGUYỄN THẾ VINH**

**ĐƠN VỊ: TRƯỜNG THPT CHUYÊN ĐẠI HỌC VINH**

Nghệ An, ngày 07 tháng 08 năm 2021

**MỤC LỤC**

[**LỜI MỞ ĐẦU** 3](#_Toc79591665)

[**CHƯƠNG I:** 5](#_Toc79591666)

[**TỔNG QUAN VỀ MÔ HÌNH MERN** 5](#_Toc79591667)

[**1.** **ReactJS (Front End)** 6](#_Toc79591668)

[**2.** **NodeJS** 6](#_Toc79591669)

[**3.** **Express (Back End)** 7](#_Toc79591670)

[**4.** **MongoDB (Database)** 7](#_Toc79591671)

[**5.** **Tổng hợp** 8](#_Toc79591672)

[**CHƯƠNG II:** 9](#_Toc79591673)

[**TỔNG QUAN VỀ DEEP LEARNING** 9](#_Toc79591674)

[**1. Machine Learning** 9](#_Toc79591675)

[**2. Deep Learning** 9](#_Toc79591676)

[**a.** **Linear Regression** 9](#_Toc79591677)

[**b.** **Logistic Regression** 12](#_Toc79591678)

[**c.** **CNN (Convolutional neural network)** 13](#_Toc79591679)

[**d.** **Mạng VGG 16** 18](#_Toc79591680)

[**e.** **Mạng InceptionV3** 19](#_Toc79591681)

[**CHƯƠNG III:** 20](#_Toc79591682)

[**ENVGAME – NỀN TẢNG THIẾT KẾ GAME TĂNG TỶ LỆ CÂY TRỒNG CHO MÔI TRƯỜNG** 20](#_Toc79591683)

[**1. Xây dựng template** 20](#_Toc79591684)

[**a. Xây dựng giao diện cho laptop, PC** 20](#_Toc79591685)

[**b. Xây dựng giao diện cho điện thoại** 20](#_Toc79591686)

[**2. Kết nối mongoose và express.** 21](#_Toc79591687)

[**3. Giao thức giao tiếp giữa Front End và Back End** 23](#_Toc79591688)

[**4. Quản lý Router** 25](#_Toc79591689)

[**5. Đăng nhập và đăng ký** 25](#_Toc79591690)

[**6. Phát triển game** 26](#_Toc79591691)

[**7. Trang cá nhân.** 27](#_Toc79591692)

[**8. Quản lý game** 28](#_Toc79591693)

[**CHƯƠNG IV:** 30](#_Toc79591694)

[**DEEP LEARNING TRONG CHĂM SÓC CÂY TRỒNG ENVGAME** 30](#_Toc79591695)

[**1.** **Sơ lược** 30](#_Toc79591696)

[**2.** **Dataset** 30](#_Toc79591697)

[**3.** **Thử nghiệm model (Training)** 32](#_Toc79591698)

[**a.** **InceptionV3 Model (GoogLeNet)** 32](#_Toc79591699)

[**b.** **Resnet Inception** 33](#_Toc79591700)

[**c.** **Alexnet 2012** 35](#_Toc79591701)

[**4.** **TensorFolowJS** 37](#_Toc79591702)

[**TÀI LIỆU THAM KHẢO** 37](#_Toc79591703)

# **LỜI MỞ ĐẦU**

* **Tổng quan**

Hiện nay chặt phá rừng bừa bại đang xảy ra rất phổ biến ở mọi nơi trên thế giới gây biến đổi khí hậu của trái đất, ảnh hưởng đến chất lượng không khí, mất nơi trú ẩn của rất nhiều động vật… [1]

Ngoài ra Việt Nam có tỷ lệ người trưởng thành chơi game ở Việt Nam cao nhất thế giới. Nên việc tận dụng lợi ích có được từ game để bảo vệ môi trường là một ý tưởng rất khả thi.

* **Tại sao Envgame ra đời**

Trong bối cảnh các tựa Mobile Game đang rất phát triển, có những tựa Game rất lành mạnh và mang tính văn hóa, giáo dục. Nhưng cũng không ít các tựa Game không lành mạnh, bạo lực và có nội dung không phù hợp với trẻ con.

Song song với điều đó “lá phổi xanh” của trái đất đang bị tàn phá nặng nề. Tốc độ phá rừng là 1 giây 1000 ha rừng, trong đó [châu Á](https://vi.wikipedia.org/wiki/Ch%C3%A2u_%C3%81) có tỷ lệ mất rừng cao nhất (2 , 9%). Riêng đối với [Việt Nam](https://vi.wikipedia.org/wiki/Vi%E1%BB%87t_Nam), trong vòng nửa thế kỷ từ 1943 đến 1993 có khoảng 5 triệu ha [rừng tự nhiên](https://vi.wikipedia.org/w/index.php?title=R%E1%BB%ABng_t%E1%BB%B1_nhi%C3%AAn&action=edit&redlink=1) bị mất, nghĩa là tốc độ phá rừng hàng năm ở [Việt Nam](https://vi.wikipedia.org/wiki/Vi%E1%BB%87t_Nam) vào khoảng 100.000 ha. [2].

Thông qua tìm hiểu tính khả thi của việc kiếm doanh thu từ Game. Mình quyết định làm ra dự án mang tên Envgame, nền tảng thiết kế game tăng tỷ lệ cây trồng cho môi trường, giúp bạn bảo vệ môi trường bằng những giờ giải trí của chính bạn.

* **Tóm tắt dự án**

Envgame lấy các nguồn thu nhập từ Game. Sau đó nguồn thu nhập đó được trích khoảng 40% để trồng cây, chăm sóc cây, xử lý rác thải, khí thải và các hoạt động bảo vệ môi trường khác.

Các nguồn doanh thu chính của Envgame bao gồm:

* Mua thời gian chơi trong Game
* Quảng cáo bằng Google Ads
* Mua bán item trong Game
* Bán dữ liệu người chơi
* Tiền ủng hộ của những người từ thiện
* Nâng cấp để loại bỏ quảng cáo (Premium)

Envgame được xây dựng bằng MERN stack, là tổng hợp open source của các công nghệ đều liên quan đến Javascript là cũng mới nhất hiện nay: **MongoDB, ExpressJS, React/React Native, NodeJS**. Để build và phát triển game thì minh sử dụng các thư viện đồ họa như là **P5, Matter**.

Hệ thống trồng cây và chăm sóc cây của Envgame được áp dụng hệ thống **Deep Learning** giúp tăng hiệu quả trồng cây và giảm được chi phí chăm sóc cây trồng.

* **Thông tin liên hệ**

**Github:**<https://github.com/athevinha>

**Blog:** <https://thevinh-1002.medium.com/>

**Telegram & Zalo:** 0918519745

**Facebook:**<https://www.facebook.com/nguyen.thevinh.5496>

**Gmail:** [nguyenthevinh1002@gmail.com](file:///C:\Users\Code\Desktop\RMIT\Envgame_docx\nguyenthevinh1002@gmail.com)

**Website sản phẩm:** <https://envgame.online/>

# **CHƯƠNG I:**

# **TỔNG QUAN VỀ MÔ HÌNH MERN**

MERN là viết tắt của 4 open source MongoDB, Express, ReactJS, NodeJS. Trong chương này mình sẽ nói về từng cái một rồi combine lại để mọi người dễ hình dung hơn về mô hình này.

## **ReactJS (Front End)**

ReacJS thuộc về phía frontend dùng để xây dựng UI (giao diện người dùng). Tốc độ phản hồi tuyệt vời bằng phương pháp render real time.

Mỗi component của ReactJS đều chứa một kho lưu dữ liệu gọi là state và được render bằng JSX. Mỗi khi mình thay đổi (update) state thì hàm render sẽ được chạy lại và return tất cả các thẻ có trong component. Vòng lặp này tạo ra hệ thống real time của reactJS giúp tăng tốc độ phản hồi khi người dùng khi thao tác lên rất nhiều.



Hình 1.1 Cách hoạt động chung của ReactJS

JSX giúp mình có thể nhúng dữ liệu của Javascript vào HTML rất đơn giản, nên nó rất thích hợp để làm các ứng dụng có khối lượng lớn.

Để nói đến sự phố biến của ReactJS thì các công ty lớn trên thế giới cũng đã sử dụng công nghệ này có thể kể đến như là: Netflix, Airbnb, American Express, Facebook, WhatsApp, eBay và Instagram.

## **NodeJS**

Công bằng mà nói khi NodeJS chưa ra đời thì Javascript chỉ là một ngôn ngữ lập trình phía client chạy trên trình duyệt nhưng sau khi NodeJS ra đời thì việc chạy Javascript trên server được coi là phổ biến.

Một khái niệm cốt lõi của NodeJS đó là các function bất đồng bộ (asynchronous functions) [3]. Các function sẽ chạy để hoàn thành một tác vụ nào đó, trong khi phần còn lại của ứng dụng vẫn chạy đồng thời. Đây là một trong những đặc trưng tiêu biểu của NodeJS.

Tất nhiên tốc độ và khả năng mở rộng dự án của NodeJS là rất tốt, nó được rất nhiều startup lựa chọn để build web vì khả năng mở rộng dự án và tốc độ quá tuyệt vời của NodeJS. Heroku hỗ trợ rất tốt việc deploy ứng dụng chứa NodeJS, bạn có thể an tâm trong việc phát triển trang web trên đó. Nhưng giống như hầu hết các công nghệ mới việc triển khai Node.js trên host không phải là điều dễ dàng.

## **Express (Back End)**

Express là một framework được xây dựng trên nền tảng NodeJS mà mình mới giới thiệu ở trên, cũng chính là phần backend (server) của dự án. Nó còn được coi là khung máy chủ tiêu chuẩn trên thực tế cho NodeJS.

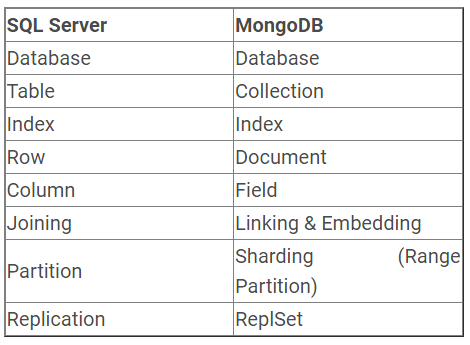
Tổng hợp một số chức năng chính của Express như sau:

* Thiết lập các lớp trung gian để trả về các HTTP request.
* Define router cho phép sử dụng với các hành động khác nhau dựa trên phương thức HTTP và URL.
* Cho phép trả về các trang HTML dựa vào các tham số.

Mọi chức năng chính của Express đều hướng tới sự trao đổi dữ liệu, sửa dữ liệu, xóa dữ liệu, thêm dữ liệu (CRUD) giữa database, server và client qua giao thức HTTP.

## **MongoDB (Database)**

MongoDB giống như là nơi để mình chứa các dữ liệu của trang web, nhưng MongoDB lưu dữ liệu dưới dạng document JSON, nó có một thuật ngữ riêng khác với các database bình thường, dưới đây là bảng thuật ngữ của MongoDB.



Hình 1.2 Bảng thuật ngữ của MongoDB

[**MongoDB**](https://jobs.hybrid-technologies.vn/blog/mongodb-la-gi/) là một cơ sở dữ liệu NoSQL hướng đối tượng, đơn giản, linh động và có thể mở rộng. NoSQL sử dụng nhiều mô hình dữ liệu để truy cập và quản lý dữ liệu. Các loại cơ sở dữ liệu này được tối ưu hóa dành riêng cho các ứng dụng yêu cầu mô hình dữ liệu linh hoạt có lượng dữ liệu lớn và độ trễ thấp, có thể đạt được bằng cách giảm bớt một số hạn chế về tính nhất quán của dữ liệu của các cơ sở dữ liệu khác.

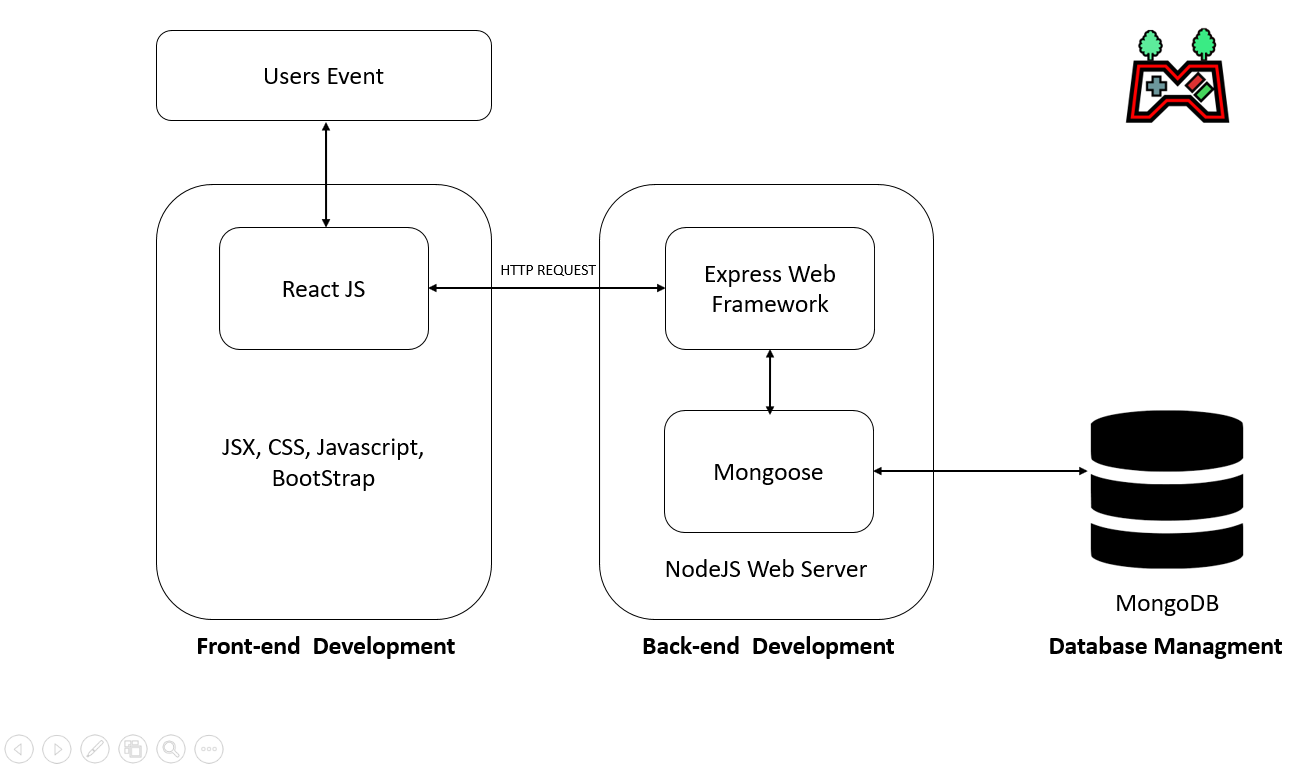
**Ưu điểm của MongoDB**

* Biểu diễn dữ liệu trong JSON hoặc BSON
* Tốc độ truy vấn và cập nhật dữ liệu nhanh, linh hoạt
* Khả năng mở rộng cao
* Dữ liệu không đồng nhất
* Ngôn ngữ truy vấn dựa trên tài liệu mạnh mẽ như SQL

**Nhược điểm của MongoDB**

* Giao dịch phức tạp
* Không có chức năng hoặc thủ tục lưu trữ tồn tại nơi bạn có thể liên kết logic

## **Tổng hợp**



Hình 1.3 Cơ cấu hoạt động chung của MERN

# **CHƯƠNG II:**

# **TỔNG QUAN VỀ DEEP LEARNING**

## **1. Machine Learning**

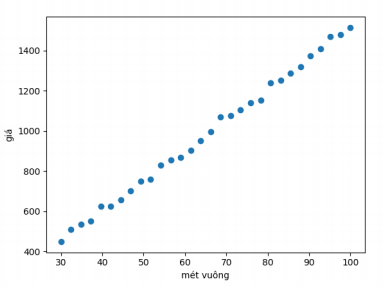
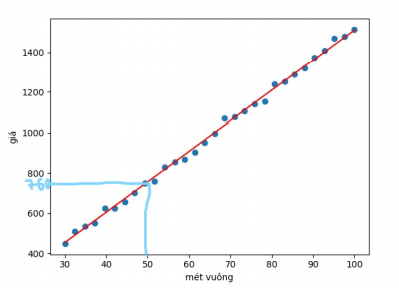
Những năm gần đây, AI - Artificial Intelligence (Trí Tuệ Nhân Tạo) v cụ thể hơn là Machine Learning (Máy Học) nổi lên như một minh chứng của cuộc cách mạng công nghiệp lần thứ tư (1 - động cơ hơi nước, 2 - năng lượng điện, 3 - công nghệ thông tin). AI hiện diện trong mọi lĩnh vực của đời sống con người, từ kinh tế, giáo dục, y khoa cho đến những công việc nhà, giải trí hay thậm chí là trong quân sự. Những ứng dụng nổi bật trong việc phát triển AI đến từ nhiều lĩnh vực để giải quyết nhiều vấn đề khác nhau.

## **2. Deep Learning**

Nhưng những đột phá phần nhiều đến từ Deep Learning (học sâu) - một mảng nhỏ đang mở rộng dần đến từng loại công việc, từ đơn giản đến phức tạp. Deep Learning đã giúp máy tính thực thi những việc tưởng chừng như không thể vào 15 năm trước: phân loại cả ngàn vật thể khác nhau trong các bức ảnh, tự tạo chú thích cho ảnh, bắt chước giọng nói và chữ viết của con người, giao tiếp với con người, hay thậm chí cả sáng tác văn, phim, ảnh, âm nhạc.

Con chó có thể phân biệt được người thân trong gia đình và người lạ hay đứa trẻ có thể phân biệt được các con vật. Những việc tưởng chừng như rất đơn giản nhưng lại cực kì khó để thực hiện bằng máy tính. Vậy sự khác biệt nằm ở đâu? Câu trả lời nằm ở cấu trúc bộ não với lượng lớn các nơ-ron thần kinh liên kết với nhau. Deep Learning giúp mình mô phỏng lại một phần cấu trúc bộ não trên máy tính *(thường gọi là Neural Network****)*** *[4]* để giải các bài toán trên.

### **Linear Regression**

Thuật toán linear regression giải quyết các bài toán có đầu ra là giá trị thực, ví dụ: dự đoán giá nhà, giá cổ phiếu, tuổi...

VD:

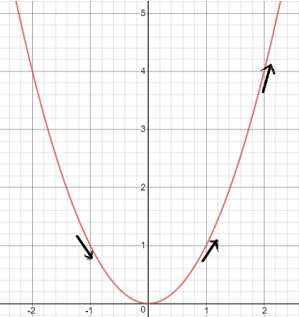
Hình 2.1 Đồ thị biểu hiện dữ liệu

Hình 2.2 Đường thẳng được vẽ bằng linear regression

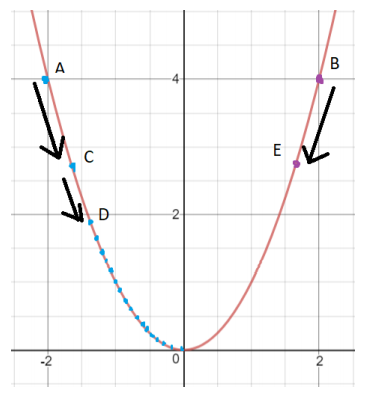
Giả sử mình gọi phương trình đường thẳng tối ưu nhất cho các điểm dữ liệu ở trên là y = ax + b (phương trình đường thẳng). Mình ta đã biết x và y, nhiệm vụ là phải tìm ra phương trình tổng quát (tức là tìm a, b). Để tìm mình sẽ đưa ra 1 giá trị a, b bất kì rồi thay bộ x vào phương trình này để tìm ra y\_test (người ta gọi là y mũ), tiếp theo mình sẽ lấy | y\_test – y |. Kết quả của cái này sẽ được gọi là loss (error). Nếu tham số loss này càng nhỏ thì giá trị a, b thử của mình lại càng đúng. Vậy bài toán bây giờ quy về việc tối ưu loss function. Bây giờ mình sẽ giới thiệu mọi người thuật toán được dùng để tối ưu hàm loss.

Gradient descent là thuật toán tìm giá trị nhỏ nhất của hàm số f(x) dựa trên đạo hàm.

Thuật toán:

* 1. Khởi tạo giá trị x = x0 tùy ý
  2. Gán x = x - learning\_rate \* f’(x) (learning\_rate là hằng số dương ví dụ learning\_rate = 0.001)
  3. Tính lại f(x): Nếu f(x) đủ nhỏ thì dừng lại, ngược lại tiếp tục bước 2

Hình 2.3 Đồ thị parabol thể hiện đạo hàm ở từng điểm

Thuật toán sẽ lặp lại bước 2 một số lần đủ lớn (100 hoặc 1000 lần tùy vào bài toán và hệ số learning\_rate) cho đến khi f(x) đạt giá trị đủ nhỏ. Ví dụ cần tìm giá trị nhỏ nhất hàm y = x 2, hàm này ai cũng biết là giá giá trị nhỏ nhất là 0 tại x = 0 nhưng để cho mọi người dễ hình dung hơn về thuật toán Gradient descent nên mình lấy ví dụ đơn giản

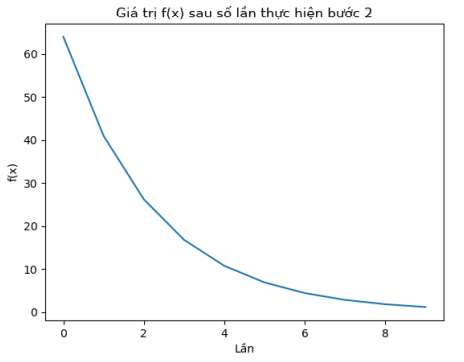
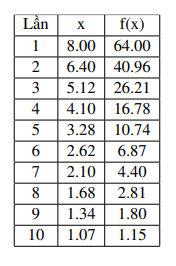
Hình 2.4 Đồ thị parabol thể hiện đạo hàm ở từng điểm

**Bước 1**: Khởi tạo giá trị ngẫu nhiên x = -2 (điểm A).

**Bước 2:** Do ở A đồ thị giảm nên f’(x=-2) = 2\*(-2) = -4 < 0 => Khi gán x = x - learning\_rate \* f’(x) nên x tăng nên đồ thị bước tiếp theo ở điểm C. Tiếp tục thực hiện bước 2, gán x = x - learning\_rate \* f’(x) thì đồ thị ở điểm D... => hàm số giảm dần dần tiến tới giá trị nhỏ nhất.

Moị người có để ý là trị tuyệt đối của đạo hàm tại A lớn hơn tại C và tại C lớn hơn tại D không? Đến khi đến gần điểm đạt giá trị nhỏ nhất x = 0, thì đạo hàm xấp xỉ 0 đến khi hàm đạt giá trị nhỏ nhất tại x = 0, thì đạo hàm bằng 0, nên tại điểm gần giá trị nhỏ nhất thì bước 2 gán x = x - learning\_rate \* f’(x) là không đáng kể và gần như là giữ nguyên giá trị của x.

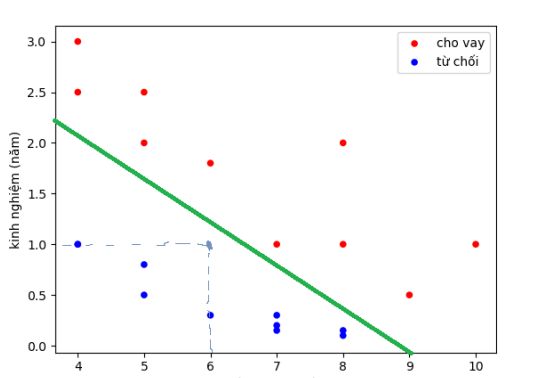
Tương tự nếu giá trị khởi tạo tại x = 2 (tại B) thì đạo hàm tại B dương nên do x = x - learning\_rate \* f’(x) giảm -> đồ thị ở điểm E -> rồi tiếp tục gán x=x -learning\_rate \* f’(x) thì hàm f(x) cũng sẽ giảm dần dần đến giá trị nhỏ nhất.

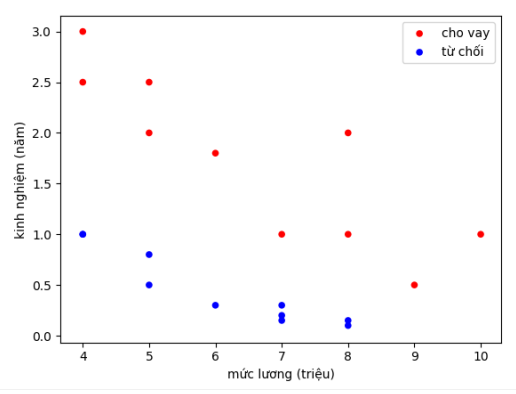
Ví dụ: chọn x = 10, learning\_rate = 0.1, bước 2 sẽ cập nhất x = x - learning\_rate\*f’(x) = x - learning\_rate \*2\*x, giá trị f(x) = x 2 sẽ thay đổi qua các lần thực hiện bước 2 như sau:

Hình 2.6 Giá trị của x và f (x)

Hình 2.5 Đồ thị giá trị f (x) sau số lần thực hiện bước 2

### **Logistic Regression**

Linear regression với đầu ra là giá trị thực thì logistic regression có đầu ra là giá trị nhị phân (0 hoặc 1), ví dụ: email gửi đến hòm thư của bạn có phải spam hay không; u là u lành tính hay ác tính...



Hình 2.8 Đường thẳng được vẽ bằng logistic regression

Hình 2.7 Đồ thị biểu hiện dữ liệu

### **CNN (Convolutional neural network)**

CNN sẽ được dùng trong deep learning khi input của Neural Network là ảnh.

Ví dụ: nhận diện được 9 số từ 1 đến 9 được viết bằng tay thì chúng ta cần có 1 dataset gồm rất nhiều những chữ viết tay từ 1 đến 9. Sau đó cho máy nhận biết rằng ảnh như thế này là số 1, ảnh như thế này là số 2… Sau khi train xong hệ thống sẽ cho ra kết quả gọi là model và hiệu suất của model.

#### **Xử lý ảnh:**

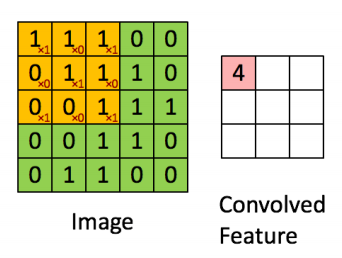
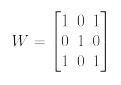
Ảnh trong máy tính RGB viết tắt của Red (đỏ), Green (xanh lục), Blue (xanh lam), là ba màu chính của ánh sáng khi tách ra từ lăng kính. Khi trộn ba màu trên theo tỉ lệ nhất định có thể tạo thành các màu khác nhau.

Theo wiki, pixel (hay điểm ảnh) là một khối màu rất nhỏ và là đơn vị cơ bản nhất để tạo nên một bức ảnh kỹ thuật số. Vậy bức ảnh trên kích thước 800 pixel \* 600 pixel, có thể biểu diễn dưới dạng một ma trận kích thước 600 \* 800 (vì định nghĩa ma trận là số hàng nhân số cột)

Khi dữ liệu biểu diễn dạng 1 chiều, người ta gọi là vector, mặc định khi viết vector sẽ viết dưới dạng cột. Khi dữ liệu dạng 2 chiều, người ta gọi là ma trận, kích thước là số hàng \* số cột. Nếu dữ liệu nhiều hơn 2 nhiều thì sẽ được gọi là tensor *VD: Ví dụ biểu diễn ảnh màu kích thước 28\*28, biểu diễn dưới dạng tensor 28\*28\*3, ảnh xám thì sẽ là 28\*28 (vì chỉ có 1 khung màu)*

Khi chuyển từ ảnh màu sang ảnh xám ta có thể dùng công thức: x = r \* 0.299 + g \* 0.587 + b \* 0.114.

#### **Phép tính Convolution**

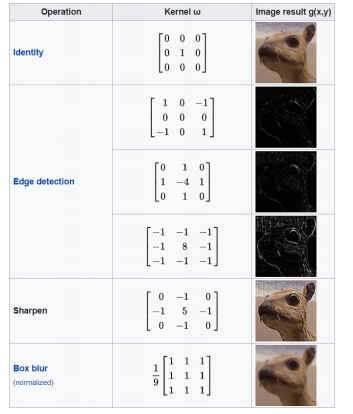
Để cho dễ hình dung mình sẽ lấy ví dụ trên ảnh xám, tức là ảnh được biểu diễn dưới dạng ma trận A kích thước m\*n. Ta định nghĩa kernel là một ma trận vuông kích thước k\*k trong đó k là số lẻ. k có thể bằng 1, 3, 5, 7, 9... Ví dụ kernel kích thước 3\*3

Hình 2.10 Mô tả phép tính convolution

Hình 2.9 Giá trị của kernel

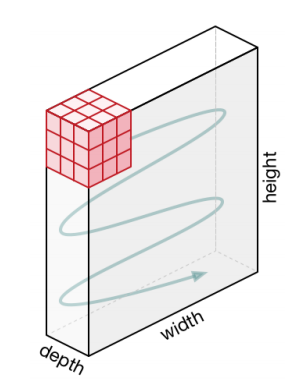
Kí hiệu phép tính convolution (⊗), kí hiệu Y = X ⊗W Với mỗi phần tử xi j trong ma trận X lấy ra một ma trận có kích thước bằng kích thước của kernel W có phần tử xi j làm trung tâm (đây là vì sao kích thước của kernel thường lẻ) gọi là ma trận A. Sau đó tính tổng các phần tử của phép tính element-wise của ma trận A và ma trận W, rồi viết vào ma trận kết quả Y.

Mục đích của phép tính Convolution trên ảnh là làm mờ, làm nét ảnh; xác định các đường... Mỗi kernel khác nhau thì sẽ phép tính convolution sẽ có ý nghĩa khác nhau. Ví dụ:



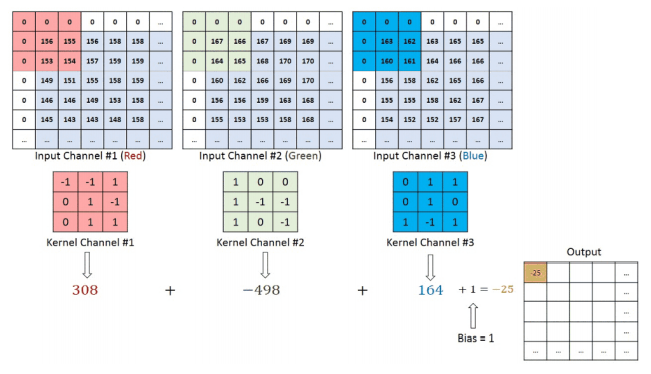
Hình 2.11 một số ứng dụng của phép tính Convolution

#### **Convolutional layer**

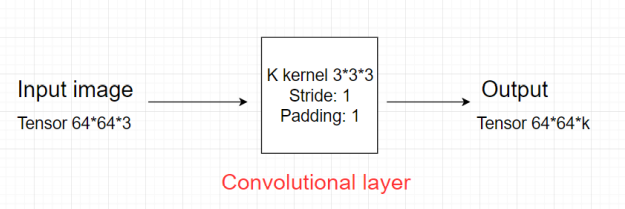
Phép tính convolution thực hiện trên ảnh màu với biểu diễn ảnh dạng ma trận. Vì ảnh màu có tới 3 channels red, green, blue nên khi biểu diễn ảnh dưới dạng tensor 3 chiều. Nên mình cũng sẽ định nghĩa kernel là 1 tensor 3 chiều kích thước k\*k\*3.

Hình 2.12 phép tính convolution trong ảnh màu (1)

Ta định nghĩa kernel có cùng độ sâu (depth) với biểu diễn ảnh, rồi sau đó thực hiện di chuyển khối kernel tương tự như khi thực hiện trên ảnh xám.

Với mỗi kernel khác nhau ta sẽ học được những đặc trưng khác nhau của ảnh, nên trong mỗi convolutional layer ta sẽ dùng nhiều kernel để học được nhiều thuộc tính của ảnh. Vì mỗi kernel cho ra output là 1 matrix nên k kernel sẽ cho ra k output matrix. Ta kết hợp k output matrix này lại thành 1 tensor 3 chiều có chiều sâu k.

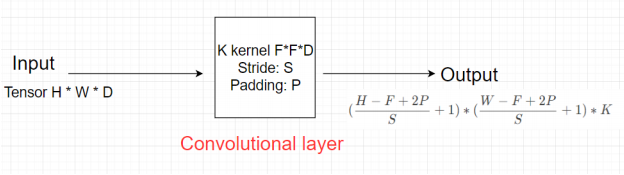
Hình 2.13 phép tính convolution trong ảnh màu (2)

Với lớp convolutional đầu tiên thì Output của nó sẽ thành input của convolutional layer tiếp theo.

Hình 2.14 Lớp convolutional đầu tiền

Giả sử input của 1 convolutional layer tổng quát là tensor kích thước H \* W \* D. Kernel có kích thước F \* F \* D (kernel luôn có depth bằng depth của input và F là số lẻ), stride: S, padding: P.

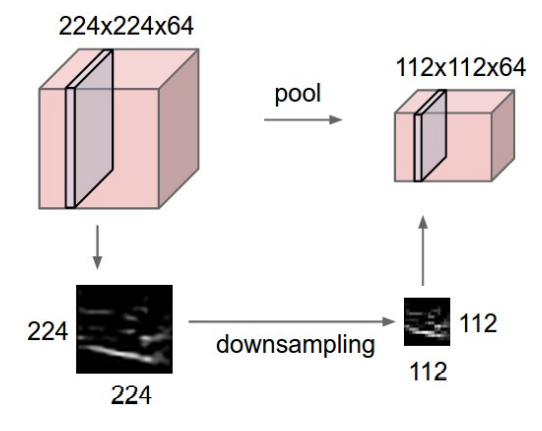
Convolutional layer áp dụng K kernel.



Hình 2.15 Lớp convolutional khác

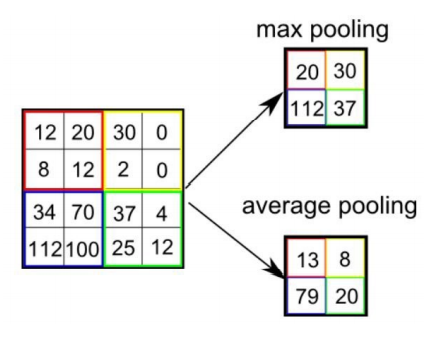
#### **Pooling layer**

Pooling layer thường được dùng giữa các convolutional layer, để giảm kích thước dữ liệu nhưng vẫn giữ được các thuộc tính quan trọng. Việc giảm kích thước dữ liệu giúp giảm các phép tính toán trong model.

Bên cạnh đó, với phép pooling kích thước ảnh giảm, do đó lớp convolution học được các vùng có kích thước lớn hơn. Ví dụ như ảnh kích thước 224\*224 qua pooling về 112\*112 thì vùng 3\*3 ở ảnh 112\*112 tương ứng với vùng 6\*6 ở ảnh ban đầu. Vì vậy qua các pooling thì kích thước ảnh nhỏ đi và convolutional layer 67sẽ học được các thuộc tính lớn hơn.

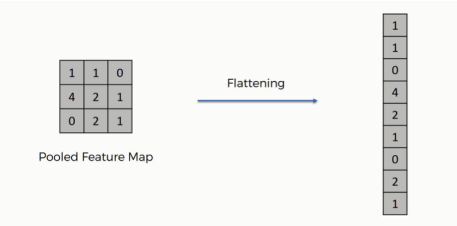
Hình 2.16 Pooling layer

Có 2 loại pooling layer phổ biến là: max pooling và average pooling Trong một số model người ta dùng convolutional layer với stride > 1 để giảm kích thước dữ liệu thay cho pooling layer.



Hình 2.17 Max pooling và average pooling

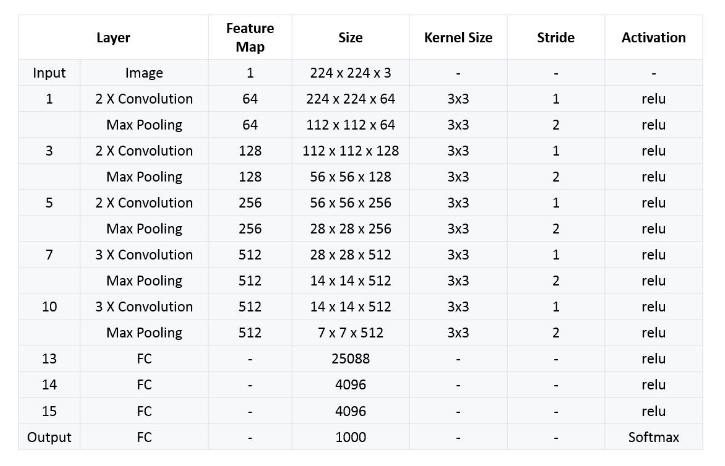
#### **Fully connected layer**

Sau khi ảnh được truyền qua nhiều convolutional layer và pooling layer thì model đã học được tương đối các đặc điểm của ảnh (ví dụ mắt, mũi, khung mặt...) thì tensor của output của layer cuối cùng, kích thước H\*W\*D, sẽ được chuyển về 1 vector kích thước (H\*W\*D, 1)

Hình 2.18 Flatten layer

### **Mạng VGG 16**

VGG16 là mạng convolutional neural network được đề xuất bởi K. Simonyan and A. Zisserman, University of Oxford [5]. Model sau khi train bởi mạng VGG16 đạt độ chính xác 92.7% top-5 test trong dữ liệu ImageNet gồm 14 triệu hình ảnh thuộc 1000 lớp khác nhau. Giờ áp dụng kiến thức ở trên để phân tích mạng VGG 16.

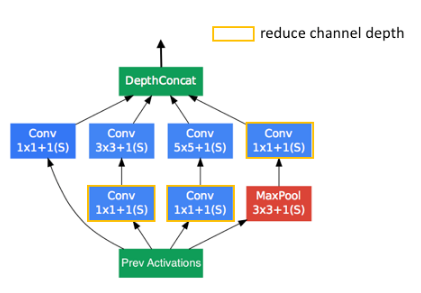


Hình 2.19 Cấu trúc của mạng VGG16

### **Mạng InceptionV3**

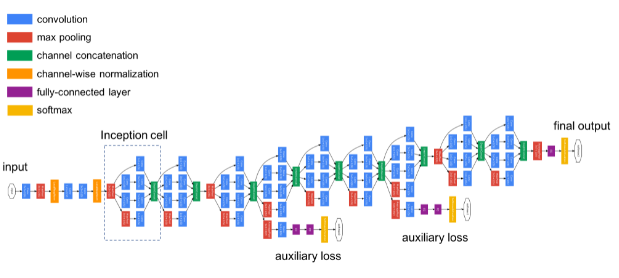
Vào năm 2014, các nhà nghiên cứu của google đã đưa ra mạng Inception -> tham dự cuộc thi ImageNet 2014 [6].

Mô hình này khá đặc biệt, không hoàn toàn là các tầng layer nối tiếp gối đầu lên nhau như các mạng trên. Ý tưởng của họ là tạo ra một mạng có tên là Inception làm cho mạng sâu hơn và nhanh hơn Alexnet rất nhiều vì mạng Inception chỉ có 5 triệu tham số trong khi Alexnet có gần 60 triệu tham số.

Cách thông thường để tăng độ phức tạp của model là tăng số layers và parameters thì mạng Inception tăng độ deeper của mỗi layers (Tránh được một phần overfitting).

Mỗi layer trong mạng CNN sẽ cho ra các thông tin khác nhau, kernel 3x3 của conv sẽ extract thông tin khác với kernel 5x5. Vậy để có thêm thông tin mình sẽ train nhiều conv với mỗi kernel khác nhau và concatenate nó lại với nhau, đây cũng chính là cách mà mạng Inception hoạt động.

Hình 2.20 Ý tưởng của mạng Inception



Hình 2.21 Cấu trúc của mạng Inception V3

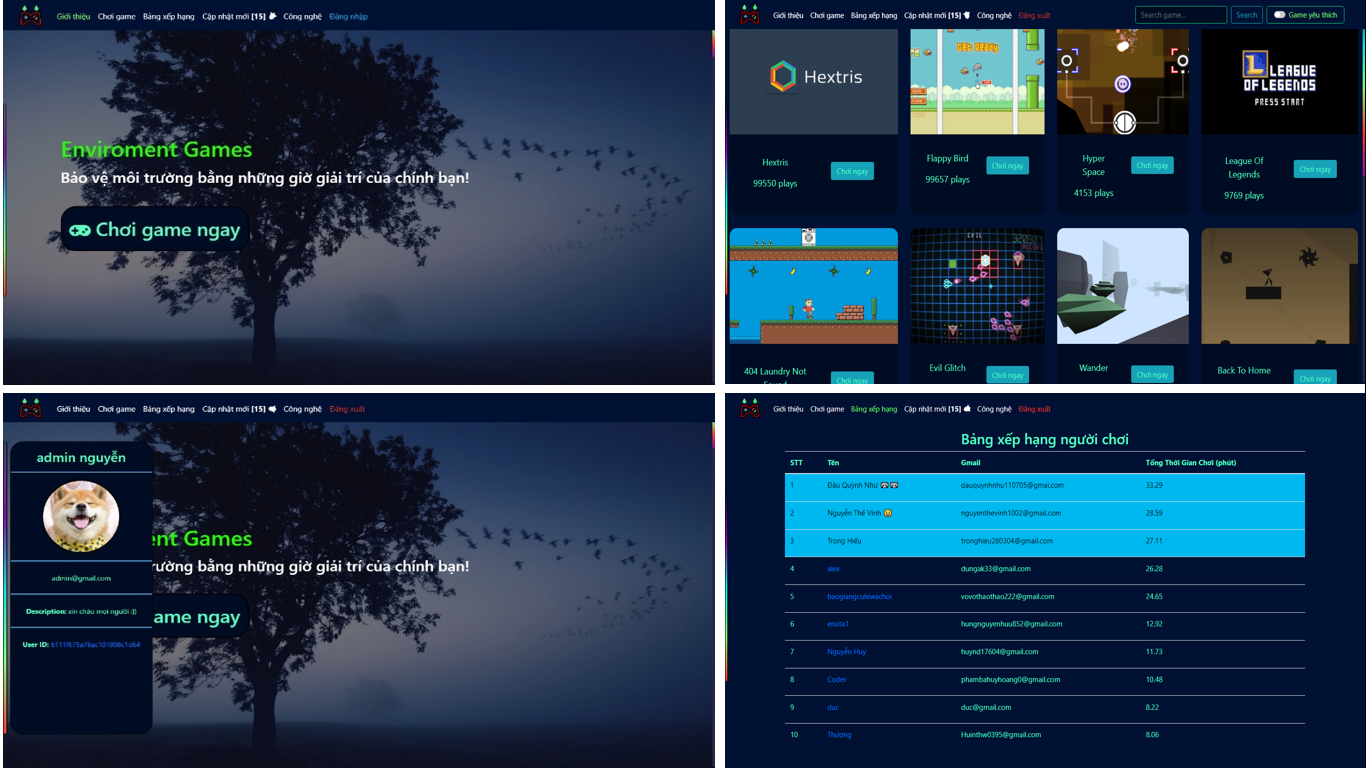
# **CHƯƠNG III:**

# **ENVGAME – NỀN TẢNG THIẾT KẾ GAME TĂNG TỶ LỆ CÂY TRỒNG CHO MÔI TRƯỜNG**

## **1. Xây dựng template**

### **a. Xây dựng giao diện cho laptop, PC**

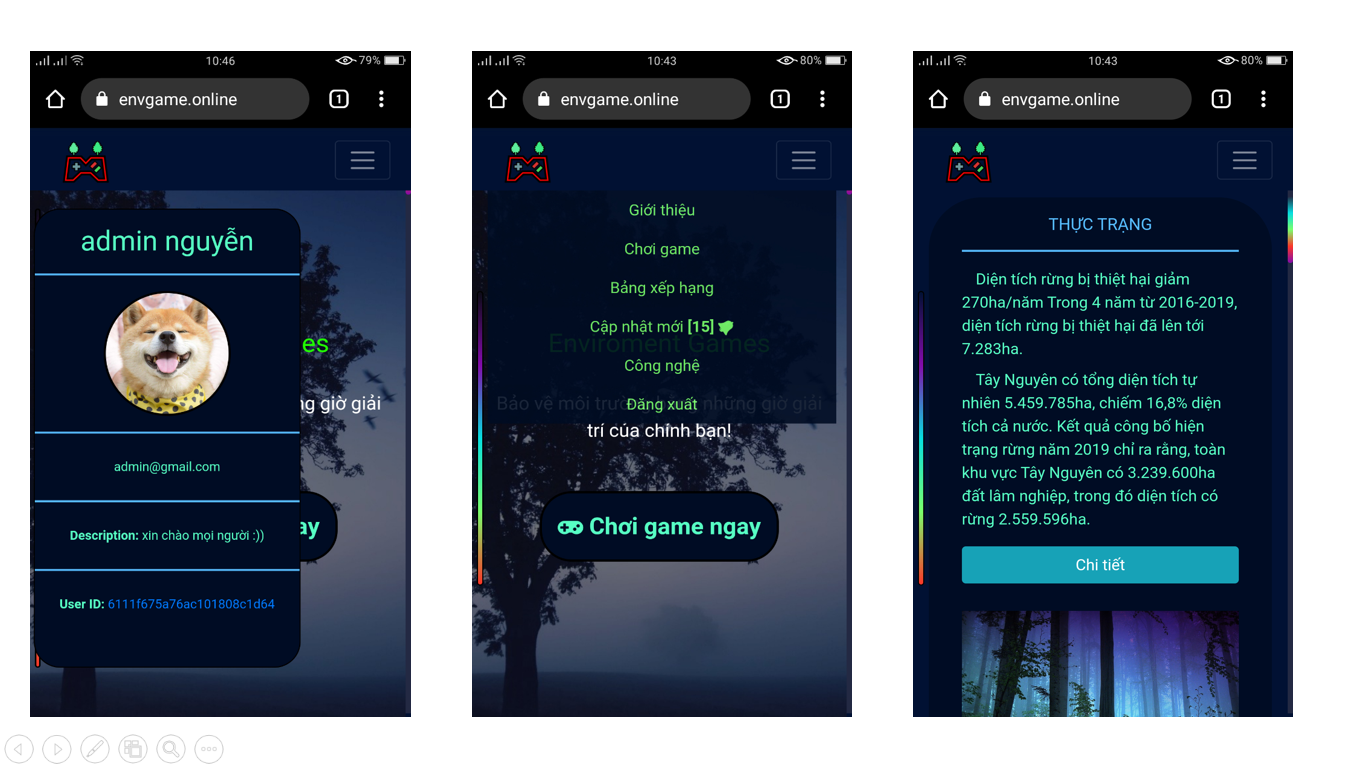
Giống như việc mọi người vẽ một bức tranh vậy, mọi người cần phác thảo qua xem bức tranh trông như thế nào. Giao diện này hoàn toàn là mình tự build bằng ReactJS, CSS và một số thư viện như là bootstrap, toast\_reactJS.

Dưới đây là giao diện mình đã tự viết cho envgame. Vì mình là một người theo chủ nghĩa dark form nên mình rất thích phối theo kiểu tối màu thế nên mình quyết định chọn chủ tông là màu xanh đậm

Hình 3.1 Giao diện cho laptop, pc của envgame

### **b. Xây dựng giao diện cho điện thoại**

Cũng như giao diện dành cho laptop, PC mình cũng viết riêng giao diện dành cho điện thoại. Dưới đây là giao diện dành cho điện thoại.



Hình 3.2 Giao diện cho điện thoại của envgame

## **2. Kết nối mongoose và express.**

Đầu tiên mình sẽ tạo một database bằng MongoDBCompass, đây là một phần mềm hiển thị dữ liệu của database và cũng có thể thực hiện các thao tác xử lý dữ liệu ở đây. Sau khi tạo xong thì chúng ta cần cài mongoose, đây là một thư viện của mongoDB giúp mọi người kết nối từ Express, NodeJS đến cơ sở dữ liệu. Mình cài thư viện này bằng NodeJS trong CMD là “npm install mongoose”.

Ngoài mongoose ra thì chúng ta cần cài thêm một số thứ như sau:

* **Axios:** giúp đồng bộ hóa quá trình chạy code, ví dụ là chúng ta lấy dữ liệu từ database nhưng phải mất khoảng 1, 2 giây thì hệ thống mongoDB mới gửi dữ liệu về Back End thì làm sao để máy tính hiểu được điều đó và chờ khi nào dữ liệu được trả về rồi mới chạy đó chính là nhờ Axios.
* **Body-parser:** Hỗ trợ thêm cho các giao thức HTTP request.
* **Cors:** Nếu mọi người code mà không có cors thì sẽ gặp những lỗi như thế này: *“has been blocked by CORS policy: “No'Access-Control-Allow-Origin' header is present on the requested resource”.* Thì đúng như tên của nó thì cors nghĩa là chia sẻ tài nguyên chéo nhau, hơi khó hiểu một tí nhưng dễ hiểu hơn thì nó là việc mọi người chia sẻ tài nguyên của các domain khác nhau cho nhau, hay đơn giản hơn là khi việc client gọi 1 api từ 1 nguồn khác có domain khác với trang hiện tại (nguồn gốc) chính là cors.
* **Express:** (*Mình đã định nghĩa ở chương I)*

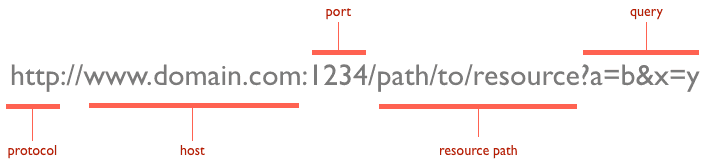
Tiếp theo mọi người cần tạo các thuộc tính quan trọng cho các đối tượng bằng Schema. Schema trong mongoose là một function giúp mọi người hình dung ra khung dữ liệu và kiểu dữ liệu từng thuộc tính của đối tượng. Ở đây mình có 2 đối tượng chính là users và games.

* User (người dùng)
* const mongoose = require("mongoose");
* const exerciseSchema = mongoose.Schema({
* username: String, // #
* gmail: String, // #
* password: String, // #
* description: String, // #
* type: Number,
* avatar: String, // #
* earned\_money: Array, // #
* played\_games: Array, // #
* time\_gaming: Array, // #
* tooken: String, // #
* });
* const users = mongoose.model("user", exerciseSchema);
* module.exports = users;
* Games
* const mongoose = require("mongoose");
* const exerciseSchema = mongoose.Schema({
* title: String,
* description: String,
* url: String,
* iframe: String,
* love\_game: Boolean,
* how2play: String,
* mobile\_game: Boolean,
* rank: Array,
* });
* const games = mongoose.model("game", exerciseSchema);
* module.exports = games;

## **3. Giao thức giao tiếp giữa Front End và Back End**

Giao thức HTTP cho chúng ta gửi dữ liệu (event) từ client lên server của nodeJS có dạng là một URL, rồi từ server của nodeJS phía back end sẽ nhận được thông tin và xử lý dữ liệu sau đó sẽ response về nodeJS server rồi trả lại kết quả về client và phản hồi lại người dùng.

Một URL có cấu trúc như dưới đây.



Hình 3.3 Ví dụ URL server của nodeJS

**Protocol**: chính là Giao thức sử dụng, thường là HTTP hoặc HTTPs. Trong trường hợp truyền nhận file thì là FTP.

Host: cái này chính là một tên miền hoặc một địa chỉ IP của một máy tính nào đó nơi sẽ xử lý của yêu cầu từ Client.

**Port**: Số cổng mà server đang chạy để lắng nghe việc xử lý các Request HTTP.

**Resource Path**: Đây chính là đường đến tài nguyên, đây là phần cực kỳ quan trọng, xác định tài nguyên nào sẽ được yêu cầu.

**Query**: Đây là tập các đối số (Param) gửi kèm theo để cung cấp thêm thông tin.

Ví dụ bạn muốn xóa game thông qua ID của game thì có thể thực hiện Request sau.

app.delete("/api/games/delete/:id", async (req, res) => {

  let \_id = req.params.id;

  games.findByIdAndDelete(\_id, (err, txt) => {

    try {

      res.send("Xóa thành công \_ID: " + \_id);

    } catch (err) {

      resp.send(err);

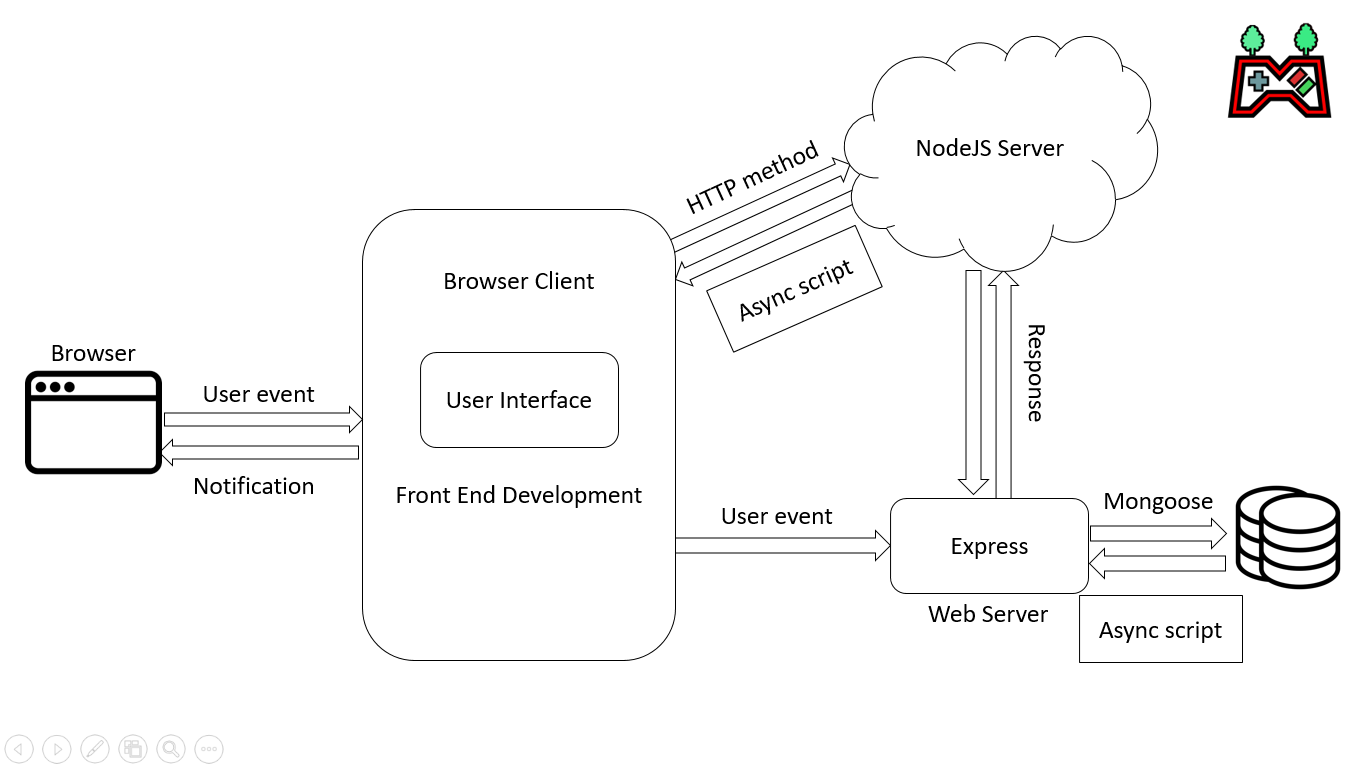
    }

  });

});

Mình đã ẩn đi host và port và chỉ để lại **Resource Path và Query. Vì việc lộ địa chỉ host và port sẽ tạo điều kiện cho kẻ xấu truy cập và database của mọi người.**

Phương thức (method) có thể là: GET, POST, HEAD, PUT và DELETE. Hai phương thức phổ biến nhất là GET và POST, Đường dẫn (path) dùng để định danh nguồn tài nguyên mà client yêu cầu, bắt buộc phải có ít nhất là dấu “/”.

Dưới đây là cách giao thức HTTP hoạt động với MERN.

Hình 3.4 Sơ đồ hoạt động cơ bản của MERN

Thực chất mình có thể tạo ra các function để tiện cho việc phát triển sau này và giúp code dễ đọc và gọn hơn. Các function này thường được gọi là functional API.

import http from "../http-common";

class GameService {

  getAll() {

    return http.get("/games/read");

  }

  create(data) {

    return http.post("/games/create", data);

  }

  update(id, data) {

    return http.put(`/games/update/${id}`, data);

  }

  delete(id) {

    return http.delete(`/games/delete/${id}`);

  }

}

export default new GameService();

## **4. Quản lý Router**

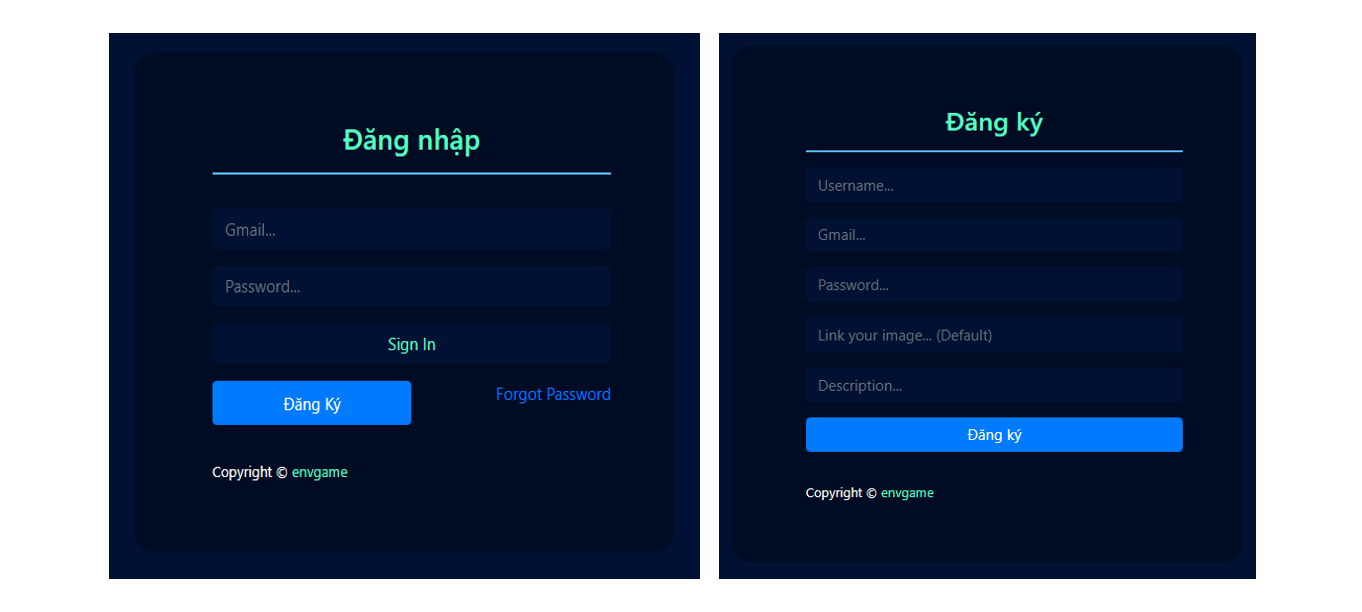
Đúng như tên gọi của nó Router giúp mình có thể chuyển qua lại giứa các components trong ReactJS. Ví dụ mình đặt link “/home” là giao diện của component Home, còn “/games” là giao diện của component Game. Cài đặt router như thế nào là tùy vào logic là sở thích của bạn.

Trong ReactJS chúng ta có một thư viện hỗ trợ về router đó chính là react-router. React-router là một thư viện định tuyến (routing) tiêu chuẩn trong ReactJS. Nó giữ cho giao diện của ứng dụng đồng bộ với URL trên trình duyệt. React-router cho phép mọi người định tuyến "luồng dữ liệu" (data flow) trong ứng dụng của mọi người một cách rõ ràng. Nó tương đương với sự khẳng định, nếu mọi người có URL này, nó sẽ tương đương với Router này, và giao diện tương ứng.

Sẽ tùy lúc chúng ta sẽ không cho người dùng truy cập vào Router ví dụ là user thường không thể vào page quản lý của admin được, trong trường hợp này chúng ta cần chuyển router của người dùng về một router khác có thể như là home, games…

Ngoài ra mọi người cũng có thể làm router lỗi khi trang web gặp lỗi hoặc là link không tồn tại.

## **5. Đăng nhập và đăng ký**

Đầu tiên như việc làm trang web mình phải build một template cho đăng nhập và đăng ký để tương tác với người dùng.

Hình 3.5 Form đăng nhập và đăng ký của envgame

Sau khi đã có form để tương tác và lấy thông tin của người dùng thì ta cần gửi những yêu cầu của người dùng tới server để xử lý và trả về kết quả.

* **Đăng ký:** Đăng ký thực chất là thêm một đối tượng user vào database (POST), dữ liệu của user được lấy từ thông tin mà người dùng nhập vào. Ở mục trước mình đã nói về functional Api, giừ là lúc mình sử dụng nó để data được nhập vào về server. Mình sẽ sử dụng các async script để chờ response từ database trả về khi data user đã được thêm vào server. Response sẽ được gửi ngược lại client và hiển thị với người dùng là “Bạn đã đăng ký thành công”. Ngoài ra mọi người cũng nên có 1 data về các họ tên chủ yếu ở Việt Nam (hoặc nước ngoài). Data này sẽ giúp mình lọc được đâu là tên thật và đâu là tên giả *tránh* được 1 số nick clone (nick giả).
* app.post("/api/users/create", async (req, res) => {
* let usera = new users(req.body);
* try {
* await usera.save();
* res.send("Create user successfully !");
* } catch (e) {console.log(e)};
* });
* **Đăng nhập:** Giống như việc lấy toàn bộ dữ liệu từ database (GET). Khi đã có toàn bộ dữ liệu users thì mình sẽ kiểm tra xem cái nào khớp với input của user, khớp thì mình sẽ lưu lại access-tooken trong local storage để ghi nhớ người dùng nếu sai thì mình sẽ trả về client một thông báo.

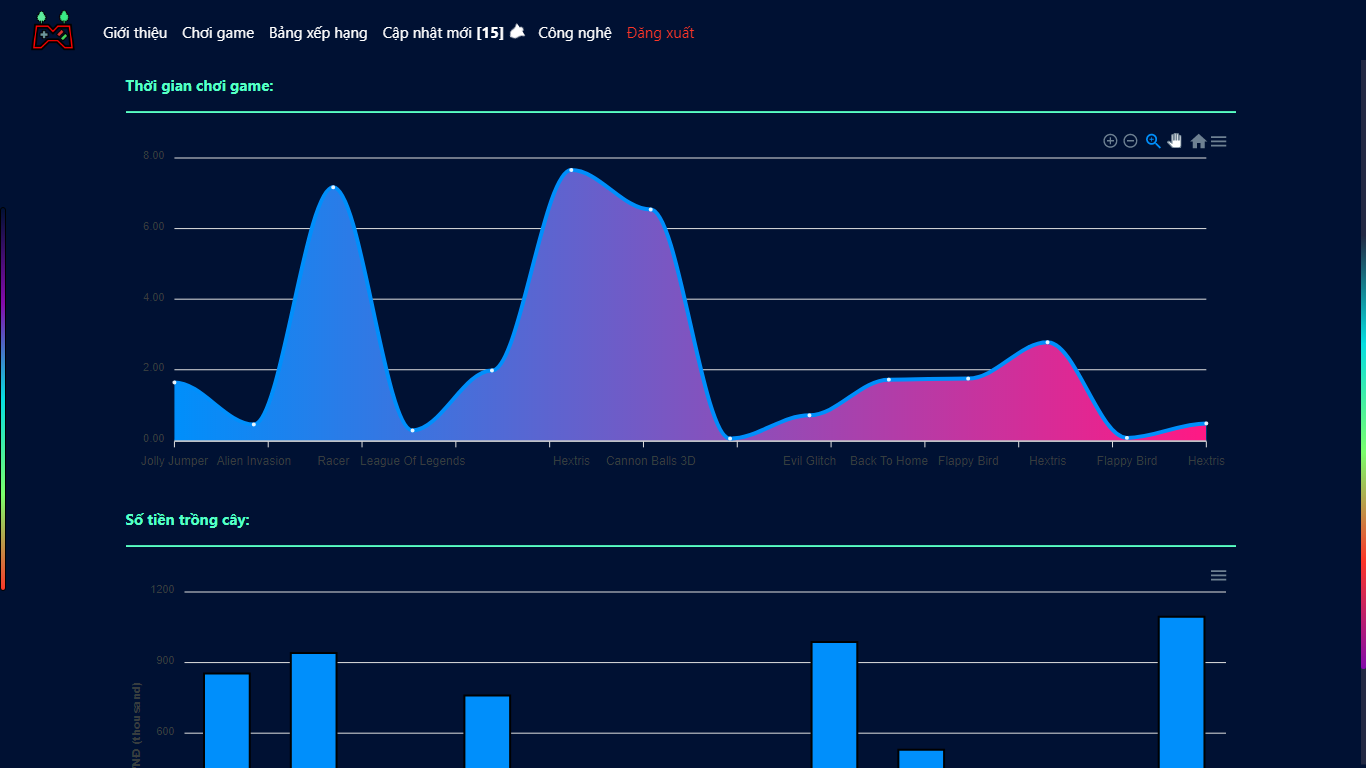
## **6. Phát triển game**

Việc phát triển game trực tiếp trên ReactJS rất là khó thực hiện và phát triển vì những thư viện liên quan đến đồ họa, hiệu ứng đang còn hạn chê. Thế nên mình quyết định chia ra một project riêng để viết game bằng javascript, JS có những thư viện đồ họa hỗ trợ làm game rất tốt như là [P5](https://p5js.org/), [FabricJS](http://fabricjs.com/), [Pixijs](http://www.pixijs.com/), [CreateJS](http://createjs.com/), [Phaser](http://phaser.io/), [Kiwijs](http://www.kiwijs.org/), [Quintus](http://www.html5quintus.com/), craftyjs…

Sau khi đã hoàn thiện game bằng JS thì mình sẽ deploy game lên github page sau đó dùng Iframe để nhúng game từ github page về envgame. Ngoài ra nếu mình sử dụng cách nhúng game vào trang web bằng Iframe như thế này thì mình có thể nhúng một số web game không bản quyền khác, tạo ra sự đa dạng về game dành cho envgame.

## **7. Trang cá nhân.**

Hình 3.6 Trang cá nhận của người dùng envgame (1)

Với các thông tin cơ bản dạng text thì mình có thể hiện thị khác dễ dàng nhưng đối với các dữ liệu mang tính thống kê nhiều thì mình sẽ vẽ đồ thị giúp người dùng có cái nhìn tổng quát hơn. Ở đây mình sử dụng react-chart để vẽ đồ thị thôi, react-chart có rất nhiều demo để mọi người có thể tìm hiểu ở [đây](https://apexcharts.com/react-chart-demos/)

Hình 3.7 Trang cá nhận của người dùng envgame (2)

## **8. Quản lý game**

Một trang web hay một tổ chức muốn phát triển mạnh mẽ và bền vững thì đều cũng cần người đứng đầu (ở đây là Admin). Điều mình muốn nói đến ở đây chính là hệ thống phân quyền, phân quyền giúp chúng ta nhận biết được đâu là admin và đâu là user bình thường. Để làm được điều đó mỗi user đều có một thuộc tính là “type”, mọi người có thể đặt các giá trị khác nhau tùy ý để phân quyền ở đây mình mặc định giá trị là “1” thì sẽ là user bình thường còn bằng “0” sẽ là admin.

Sau khi đã phân biệt được đâu là admin và đâu là user mình sẽ bắt đầu làm các chức năng dành cho admin bao gồm:

* Thêm game (POST)
* Hiển thị danh sách game (GET)
* Chỉnh sửa các thuộc tính của game (PUT)
* Xóa game (DELETE)

Các chức năng này được gọi chung là CRUD (Create – Read – Update – Delete). Phần này giống như việc mọi người đăng ký tài khoản vậy, chỉ khác là nó các chức năng khác như là xóa, chỉnh sửa…

**9. Bảng xếp hạng**

Mình quyết định xây dựng bảng xếp hạng cho một số tựa game tính điểm là vì game trong envgame đều chủ yếu là game tương tác với máy. Mình muốn có một cái gì đó để mọi người có thể ganh đua với nhau thế nên mình nghĩ đến bảng xếp hạng.

Như mình đã giới thiệu trước rồi, mình viết ra một project game riêng bằng Javascript rồi deploy lên github sau đó dùng iframe để nhúng vào trang web. Vậy làm thế nào để mình nhận biết được khi nào thì game kết thúc và điểm của người dùng ở một trang web bên project game. Câu trả lời là mình gửi dữ liệu từ Iframe sang bên phía envgame bằng “postMessage” sau đó lắng nghe event bên envgame bằng “addEventListener”

window.parent.postMessage( // đây là dữ liệu được gửi từ phía Iframe

              {score: score},"\*"

            );

// =========================================================================

// =========================================================================

// dưới đây là câu lệnh lắng nghe event ở envgame

window.addEventListener("message", *this*.receiveMessage, false);

Sau khi đã có được điểm của người chơi thì mình chỉ cần sort dữ liệu theo số điểm rồi hiển thị lên với số lượng giới hạn là bao nhiêu đấy tùy theo ý thích của mọi người.

Ngoài ra mình có làm thêm bảng xếp hạng cho người chơi có tổng thời gian chơi nhiều nhất. Mình chỉ cần lấy dữ liệu từ database (GET) rồi tính tổng thời gian chởi của user sau đó sort theo thứ tự sao đó hiển thị lên thôi.

# **CHƯƠNG IV:**

# **DEEP LEARNING TRONG CHĂM SÓC CÂY TRỒNG ENVGAME**

## **Sơ lược**

Vì thời gian và điều kiện còn khá ít *(Đội ngũ phát triển dự án chỉ có chắc mình)* nên việc tự tìm dữ liệu cho việc traning rất là khó. Thế nên mình quyết định sử dụng các dataset có sẵn của [Kaggle](https://www.kaggle.com/).

Đối với những người còn sử dụng một số laptop đời cũ như mình thì việc train trên laptop cũng rất là khó khăn vì quá trình training đòi hỏi sức mạnh của GPU rất nhiều. Thế nên Google Colab được ra đời, [Google Colab](https://colab.research.google.com/notebooks/intro.ipynb?utm_source=scs-index) [7] dựa trên Jupyter Notebook, người dùng có thể viết và thực thi đoạn mã python thông qua trình duyệt và đặc biệt rất phù hợp với data analysis, machine learning và giáo dục.

## **Dataset**

Vì mọi thao tác training của mình đều thực hiện trên Google Colab thế nên mình cần kết nối đến Kaggle API [8] sau đó tải dataset cần thiết lên google colab. Cụ thể thì mình sẽ chia làm 3 bước:

* B1: mọi người sẽ đăng nhập vào hệ thống, khi đăng nhập xong thì mọi người sẽ có một file JSON API.
* B2: sau khi mọi người đã có file JSON thì mình sẽ kết nối đến API và tải dữ liệu về bằng một số câu lệnh mà google colab đã hỗ trợ.

from google.colab import files

files.upload() # đẩy file lên google colab, ở đây mình đẩy file API json lúc nãy

# !pip install -q kaggle

!mkdir -p ~/.kaggle

!cp kaggle.json ~/.kaggle/

!chmod 600 ~/.kaggle/kaggle.json

#Cd đến thư mục có file JSON và kết nối đến hệ thống dữ liệu

!kaggle datasets list # Danh sách dữ liệu

!kaggle datasets download -d abdallahalidev/plantvillage-dataset

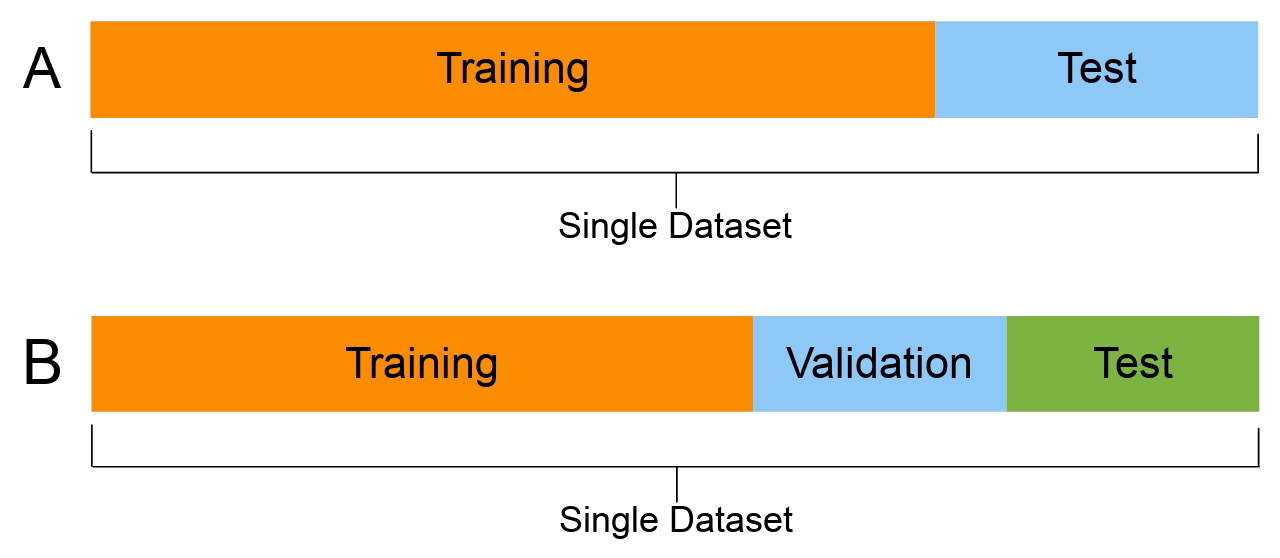
# Tải dữ liệu

* B3: Dữ liệu chủ yếu ở dạng nén, thế nên mọi người cần phải giải nén ra nhé.

!unzip /content/plantvillage-dataset.zip

# Giải nén dữ liệu

Dữ liệu của mình có dung lượng sấp xỉ 4GB [9] và bao gồm 38 classes khác nhau gồm các trạng thái của các loại lá khác nhau.

Mỗi dataset thường có được chia thành 2 - 3 cụm. Giống như việc mọi người luyện đề để đi thi vậy ví dụ mọi người có 10 đề để ôn luyện thì mọi người sẽ ôn 6 đề, 2 đề nựa là để xem bản thân có làm tốt đề, và 2 đề con lại là trong trường hợp thực tế giống như mọi người đi thi vậy. Người ta gọi tập 6 đề của mọi người là tập Traning, tập 2 đề để kiểm tra xem bản thân có làm tốt không là tập Validation và tập đề thực tế là tập test. Thì tập val (validation) và tập train (training) thường có 2 tham số chính là loss (mình đã giới thiệu ở chương II) và accuracy (xác suất).

Hình 4.1 Các cách chia dữ liệu phổ biến

Overfitting là một thuật ngữ để nói về việc model fit quá nhiều so với bộ dữ liệu. Vì model học quá nhiều thuộc tính và đặc tính của ảnh nên sẽ không tạo được sự đồng nhất cho model. Thế nên accuracy ở tập train thì rất cao nhưng accuracy ở tập val thì lại rất thấp và có xu hướng giảm xuống.

Để tránh được một phần trường hợp model bị overfitting thì mọi người cần làm phức tạp dữ liệu lên (data\_augmentation) bằng cách xoay ảnh, phóng to, thu nhỏ … dưới đây là một ví dụ về data\_augmentation.

data\_augmentation = tf.keras.Sequential([  
 tf.keras.layers.experimental.preprocessing.RandomFlip('horizontal'),  
 tf.keras.layers.experimental.preprocessing.RandomRotation(0.2),  
])

## **Thử nghiệm model (Training)**

### **InceptionV3 Model (GoogLeNet)**

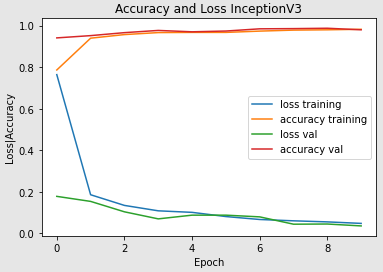
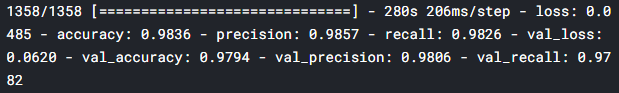
So với mạng VGG (VGG16, VGG19) thì mạng InceptionV3 có loss (error value) nhỏ hơn khoảng 0.6% trong đó (VGG: 7.3%, Inception: 6.7%) [10]. Ý tưởng của họ là tạo ra một mạng có tên là Inception làm cho mạng sâu hơn và nhanh hơn Alexnet rất nhiều vì mạng Inception chỉ có 5 triệu tham số trong khi Alexnet có gần 60 triệu tham số.

Cách thông thường để tăng độ phức tạp của model là tăng số layers và parameters thì mạng Inception tăng độ deeper của mỗi layers (Tránh được một phần overfitting).

Mỗi layer trong mạng CNN sẽ cho ra các thông tin khác nhau, kernel 3x3 của conv sẽ extract thông tin khác với kernel 5x5. Vậy để có thêm thông tin ta sẽ train nhiều conv với mỗi kernel khác nhau và concatenate nó lại với nhau, đây cũng chính là cách mà mạng Inception hoạt động.

Model: "model"  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Layer (type) Output Shape Param #   
=================================================================  
input\_2 (InputLayer) [(None, 255, 255, 3)] 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
sequential (Sequential) (None, 255, 255, 3) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
tf.math.truediv (TFOpLambda) (None, 255, 255, 3) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
tf.math.subtract (TFOpLambda (None, 255, 255, 3) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
inception\_v3 (Functional) (None, 6, 6, 2048) 21802784   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
global\_average\_pooling2d (Gl (None, 2048) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dropout (Dropout) (None, 2048) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense (Dense) (None, 38) 77862   
=================================================================  
Total params: 21,880,646  
Trainable params: 21,846,214  
Non-trainable params: 34,432

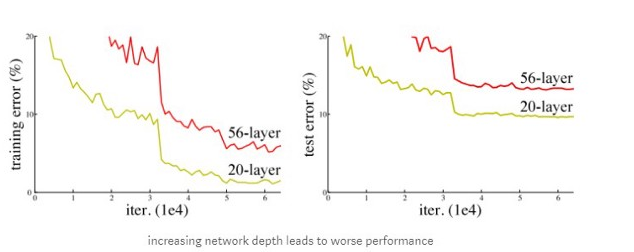
Sau khi train xong thì vẽ biểu đồ ra để đánh giá xem model có bị overfitting hay underfitting không.

Ở đây cả train\_loss và val\_loss giảm đều và rất thấp nên model fit vừa đủ. Hiệu suất lên tới 98% lúc train và ở tập val thì hiệu suất là 97%.

Hình 4.2 Đồ thị thể hiện các giá trị của model Inception

### **Resnet Inception**

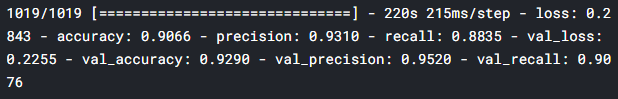
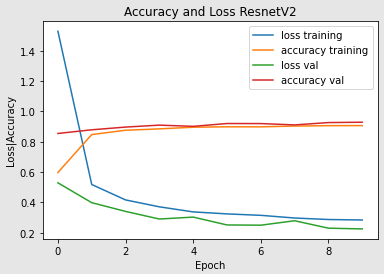
Đã có những thực nghiệm, nghiên cứu nếu train theo kiểu model có càng nhiều layer thì accuracy của mọi người luôn bị giảm. Sau một thời gian thì Resnet ra đời để giải quyết vấn đề đó bằng cách gộp 2 – 3 layer lại thành 1 khối rồi chạy từng khối một. Với cách này thì nó giúp model của bạn càng nhiều layer thì Accuracy giữ nguyên chứ không giảm như cách thông thường



Hình 4.3 Đồ thị thể hiện các giá trị của model Inception

Keras cũng đã hỗ trợ pre\_train model nên mình sử dụng luôn để thuận tiện hơn trong việc xây dựng và thử nghiệm model. Dưới đây là cấu trúc của một mạng inception resnet.

Model: "model"  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Layer (type) Output Shape Param #   
=================================================================  
input\_2 (InputLayer) [(None, 255, 255, 3)] 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
sequential (Sequential) (None, 255, 255, 3) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
tf.math.truediv (TFOpLambda) (None, 255, 255, 3) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
tf.math.subtract (TFOpLambda (None, 255, 255, 3) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
inception\_resnet\_v2 (Functio (None, 6, 6, 1536) 54336736   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
global\_average\_pooling2d (Gl (None, 1536) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dropout (Dropout) (None, 1536) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense (Dense) (None, 38) 58406   
=================================================================  
Total params: 54,395,142  
Trainable params: 58,406  
Non-trainable params: 54,336,736  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Đây là biểu đồ của mạng Inception Resnet vì loss function ở cả train và loss cùng giảm đều nên model cũng này fit vừa đủ với tập dữ liệu của mình

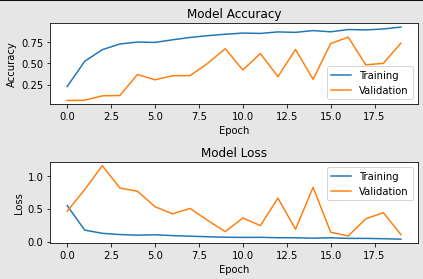
Hình 4.4 Đồ thị thể hiện các giá trị của model Resnet Inception

### **Alexnet 2012**

Kiến trúc AlexNet lớn hơn nhiều so với các kiến trúc CNNs được sử dụng trong thị giác máy tính trước kia (trước năm 2010), VD kiến trúc LeNet của Yann LeCun năm 1998. Nó có 60 triệu tham số và 650000 neural và tốn khoảng từ năm đến sáu ngày huấn luyện trên hai GPU GTX 580 3GB [11]. Ngày nay, với sự tiến bộ vượt bật của GPU, chúng ta có nhiều kiến trúc CNN có cấu trúc phức tạp hơn, và hoạt động rất hiệu quả trên những tập dữ liệu phức tạp. Nhưng tại thời điểm năm 2012 thì việc huấn luyện mô hình với lượng tham số và neural lớn như vậy là một vấn đề cực kỳ khó khăn.

Dưới đây là cấu trúc của mạng Alexnet để mọi người có thể tham khảo. Mạng Alexnet này có số lượng tham số khá lớn (~ 60 triệu).

Model: "alex\_net"  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Layer (type) Output Shape Param #   
=================================================================  
conv2d (Conv2D) (None, 55, 55, 96) 34944   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
max\_pooling2d (MaxPooling2D) (None, 27, 27, 96) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
conv2d\_1 (Conv2D) (None, 27, 27, 256) 614656   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
max\_pooling2d\_1 (MaxPooling2 (None, 13, 13, 256) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
conv2d\_2 (Conv2D) (None, 13, 13, 384) 885120   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
conv2d\_3 (Conv2D) (None, 13, 13, 384) 1327488   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
conv2d\_4 (Conv2D) (None, 13, 13, 256) 884992   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
max\_pooling2d\_2 (MaxPooling2 (None, 6, 6, 256) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
flatten (Flatten) (None, 9216) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense (Dense) (None, 4096) 37752832   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense\_1 (Dense) (None, 4096) 16781312   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense\_2 (Dense) (None, 1000) 4097000   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense\_3 (Dense) (None, 2) 2002   
=================================================================  
Total params: 62,380,346  
Trainable params: 62,380,346  
Non-trainable params: 0

Nếu để ý mọi người có thể thấy giá trị loss ở tập train giảm xuống còn rất thấp nhưng giá trị loss ở tập val thì lúc lên cao lúc lại xuống thấp (bất ổn định), vì vậy mình có thể kết luận rằng model Alexnet này khá overfitting cho với tập dữ liệu của mình.

Hình 4.5 Đồ thị thể hiện các giá trị của model Alexnet

Kết quả có thể cho thấy khi chúng ta dùng tập test để thử xác xuất của mô hình khi ứng dụng vào thực tế thì accuracy chỉ vào khoảng 73% trong khi đó accuracy ở tập train là gần 98%. Đây cũng chính là tác hại của việc model quá fit với tập dữ liệu đã cho nó tạo sự chênh lệch lớn giữa tập validation và tập training.

[INFO] Calculating model accuracy  
28/28 [==============================] - 24s 837ms/step - loss: 0.1069 - accuracy: 0.7359  
Test Accuracy: 73.59356880187988

## **TensorFolowJS**

TensorFlowJS [12] là một thư viện JavaScript để đào tạo (training) và triển khai (deploying) các mô hình (model) học máy trong trình duyệt (browser) và trong Node.

Để làm một hệ thống dùng thử model trực tiếp trên web của envgame thì mình cần chuyển từ model của keras sang model của tensorflowJS rồi predict trực tiếp ở trên web.

Vì model của mình khá là lớn nó có dung lượng khoảng 600MB quá nhiều so với dung lượng chứa được của host, thế nên mình quyết định cắt giảm với dữ liệu và train lại với một model có kích thước nhỏ hơn nhiều nhưng bù lại số lớp và độ chính xác bị giảm đi khá nhiều.

Dù hệ thống dùng thử trên web không có được xác suất và độ chính xác cao nhưng các model ban đầu lúc chưa bị cắt giảm layer vẫn có thể áp dụng vào thực tế với xác suất gần 98.5%.

# **TÀI LIỆU THAM KHẢO**

[1][*https://onhiemmoitruong.vn/hau-qua-cua-viec-pha-rung/*](https://onhiemmoitruong.vn/hau-qua-cua-viec-pha-rung/)*https://paperswithcode.com/method/inception-v3*

[2][*https://vi.wikipedia.org/wiki/Ph%C3%A1\_r%E1%BB%ABng*](https://vi.wikipedia.org/wiki/Ph%C3%A1_r%E1%BB%ABng)

[3[] *https://developer.mozilla.org/enUS/docs/Web/JavaScript/Reference/Statements/async\_function*](file:///C:\Users\Code\Desktop\RMIT\Envgame_docx\%5d%20https:\developer.mozilla.org\enUS\docs\Web\JavaScript\Reference\Statements\async_function)

[4] [*Blog | Deep Learning cơ bản (nttuan8.com)*](https://nttuan8.com/)

[5][*https://neurohive.io/en/popular-networks/vgg16/*](https://neurohive.io/en/popular-networks/vgg16/)

[6][*https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202*](https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202)

[7] *https://colab.research.google.com/notebooks/intro.ipynb*

[8][*https://github.com/Kaggle/kaggle-api*](https://github.com/Kaggle/kaggle-api)

[9][*https://www.kaggle.com/abdallahalidev/plantvillage-dataset*](https://www.kaggle.com/abdallahalidev/plantvillage-dataset)

[10][*https://paperswithcode.com/method/inception-v3*](https://paperswithcode.com/method/inception-v3)

[11] [*https://www.phamduytung.com/blog/2018-06-15-understanding-alexnet/*](https://www.phamduytung.com/blog/2018-06-15-understanding-alexnet/)

[12][*https://www.tensorflow.org/js*](https://www.tensorflow.org/js)