

# Bài 30 - Xây dựng Web AI trên tensorflow js

28 Mar 2020 - phamdinhhkhanh

## Menu

- 1. Giới thiệu chung
- 2. Tensorflow js
- 3. Model
  - 3.1. Giới thiệu về model Mobilenet
    - 3.1.1. Mạng tích chập 2 chiều chuẩn (Standard 2D Convolution)
    - 3.1.2. Tích chập chiều sâu tách biệt (Depthwise Separable Convolution)
  - 3.2. Kiến trúc model MobileNet
  - 3.3. Convert model trên tensorflow js
- 4. Giao diện
  - 4.1. Tạo giao diện
  - 4.2. Tương tác giao diện
- 5. Server
- 6. Deploy lên heroku
- 7. Tổng kết:
- 8. Tài liệu tham khảo:

## 1. Giới thiệu chung

Ở bài trước (<https://phamdinhhkhanh.github.io/2020/03/23/FlaskRestAPI.html>) mình đã giới thiệu với các bạn cách nào để xây dựng được một API và vai trò của API đối với hoạt động của các ứng dụng.

Tiếp nối bài trước, ở bài này mình sẽ hướng dẫn mọi người làm thế nào để xây dựng và triển khai một ứng dụng web trên heroku.

Ý tưởng của ứng dụng đó là chúng ta sẽ dự đoán vật thể trong các bức ảnh. Như chúng ta đã biết, nguồn tài nguyên ảnh thì khá nhiều nhưng rất nhiều ảnh không được gán nhãn. Đây sẽ là một trong những ứng dụng giải quyết được vấn đề nhức nhối là sử dụng sức lao động con người ngồi phân loại và gán nhãn cho hàng nghìn bức ảnh. Hãy để những công việc chân tay cho AI làm. Còn bạn có thể ngồi thư giãn. Ứng dụng này có thể nhận biết được hơn 1000 vật thể khác nhau từ các bức ảnh, khá nhiều phải không nào?

Để giúp dễ hiểu hơn đối với bạn đọc không chuyên về lập trình ứng dụng, bài viết được mình giảm nhẹ các phần kỹ thuật nhất có thể. Nếu bạn chỉ quan tâm đến ứng dụng có thể đọc bắt đầu từ mục 3.3. Nếu bạn đọc quan tâm đến kiến trúc mô hình có thể đọc bắt đầu từ mục đầu tới mục 3.3.

Ngoài ra mình cũng sẽ cung cấp mã nguồn code miễn phí tới các bạn nhằm mục đích hỗ trợ cộng đồng tốt hơn.

Ứng dụng của mình được viết trên ngôn ngữ javascript vì đây là ngôn ngữ có nhiều ưu điểm như dễ học, triển khai ứng dụng nhanh và tương tác đồng thời được với frontend và backend.

Framework mà mình lựa chọn để xây dựng model classification đó là tensorflow js. Đây là một trong những hot trend framework của bác google ở thời điểm hiện tại.

Top

## 2. Tensorflow js

Được bổ sung vào hệ sinh thái tensorflow của google từ 2018, tensorflow js đã tạo nên một làn sóng chấn động trong cộng đồng AI. Tensorflow JS đã giúp cho việc triển khai các ứng dụng deep learning trên nền tảng website dễ dàng hơn. Ngôn ngữ mà google lựa chọn để build framework này không phải là C, C++, Java hay những ngôn ngữ mạnh mẽ nào khác mà lại là ngôn ngữ có tính ứng dụng rất thiết thực. Đó chính là javascript. Tại sao lại là javascript? Mình nghĩ lý do chính là javascript rất phổ biến trong cộng đồng web. Việc lựa chọn javascript sẽ thu hút được một cộng đồng các developer web đông đảo. Đồng thời javascript có những framework mạnh như nodejs hỗ trợ build app và web đường như là realtime. Bạn có thể deploy ứng dụng của mình trong chỉ vài giây, một điều tuyệt vời mà các ngôn ngữ compile như java, C, C++ không làm được. Đồng thời javascript là ngôn ngữ đa năng hoạt động được trên cả frontend và backend nên các developer chỉ cần học 1 ngôn ngữ mà làm được 2 nhiệm vụ. Không hiểu sao mình rất thích javascript vì có thể coi javascript như là một học sinh giỏi học 1 mà biết 2.

Quay lại vấn đề chính. Theo như quảng cáo của trang chủ tensorflow thì tensorflow js có các ưu điểm:

- Có thể convert được model từ nhiều format của các deep learning framework khác nhau.
- Có nhiều pretrain model.
- Có một cộng đồng sử dụng rộng.
- Có thể huấn luyện và retrain lại model tensorflow của bạn trên chính website.
- Đóng gói được model của bạn lên front end và truy cập trực tiếp nên tốc độ cao hơn so với call thông qua API.

Tóm lại tensorflow js là một lựa chọn khá hữu ích cho những dự án deploy model trên website.

## 3. Model

### 3.1. Giới thiệu về model Mobilenet

Vì chỉ là demo nên mô hình mà mình sử dụng để deploy ứng dụng này là MobileNet, một mô hình khá nhẹ nên tốc độ load và dự báo nhanh. Tên gọi MobileNet đã nói lên kích thước của mô hình rất nhẹ, phù hợp để triển khai trên các thiết bị di động và IoT.

Mô hình này được nhóm nghiên cứu của google publish năm 2017 (Andrew G. Howard, Menglong Zhu,...).

Điều tuyệt vời khiến mô hình này rất nhẹ là bởi sử dụng kiến trúc tích chập chiều sâu tách biệt (Depthwise Seperable Convolution) có tác dụng giảm thiểu số lượng tham số.

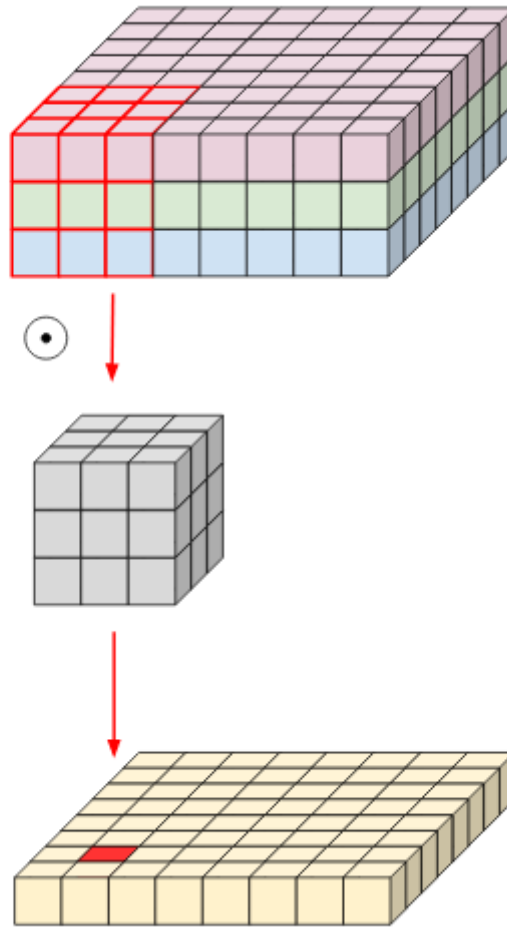
Chúng ta cùng so sánh tích chập chiều sâu tách biệt có gì khác so với tích chập 2 chiều chuẩn nhé.

#### 3.1.1. Mạng tích chập 2 chiều chuẩn (Standard 2D Convolution)

Như chúng ta đã biết, các layer tích chập 2 chiều viết tắt là Conv2D sẽ kết nối đến toàn bộ chiều sâu của layer trước đó. Như vậy kernel sẽ có dạng hình khối  $F \times F \times C_{input}$  trong đó  $F$  là kích thước bộ lọc và  $C_{input}$  là kích thước input channels.

Top

Giả sử chúng ta áp dụng một bộ lọc kích thước (width x height) =  $3 \times 3$  lên hình ảnh input có kích thước  $(8 \times 8 \times 3)$ . Khi đó áp dụng tích chập 2D chuẩn lên toàn bộ độ sâu của input layer thì bộ lọc phải có độ sâu là 3. Đồng thời, kích thước dạng khối của nó là (width x height x depths) =  $3 \times 3 \times 3$ , với channels bằng chính độ sâu của layer trước.



**Hình 1:** Kiến trúc mạng tích chập 2 chiều thông thường. Khối đầu tiên là input layer, khối ở giữa là bộ lọc của tích chập 2D chuẩn. Hình vuông cuối cùng là output của kết quả tích chập có kích thước  $8 \times 8 \times 1$ .

Như vậy theo kiến trúc tích chập này, bộ lọc được mở rộng theo số lượng channels nên chúng ta sẽ phải đầu tư cho mạng số lượng tham số là  $3 \times 3 \times 3 = 27$ . Nếu áp dụng rất nhiều bộ lọc thì số lượng tham số sẽ lớn hơn rất nhiều.

Chẳng hạn áp dụng 3 bộ lọc, số lượng sẽ là:  $(3 \times 3 \times 3) \times 3 = 81$ .

### 3.1.2. Tích chập chiều sâu tách biệt (Depthwise Separable Convolution)

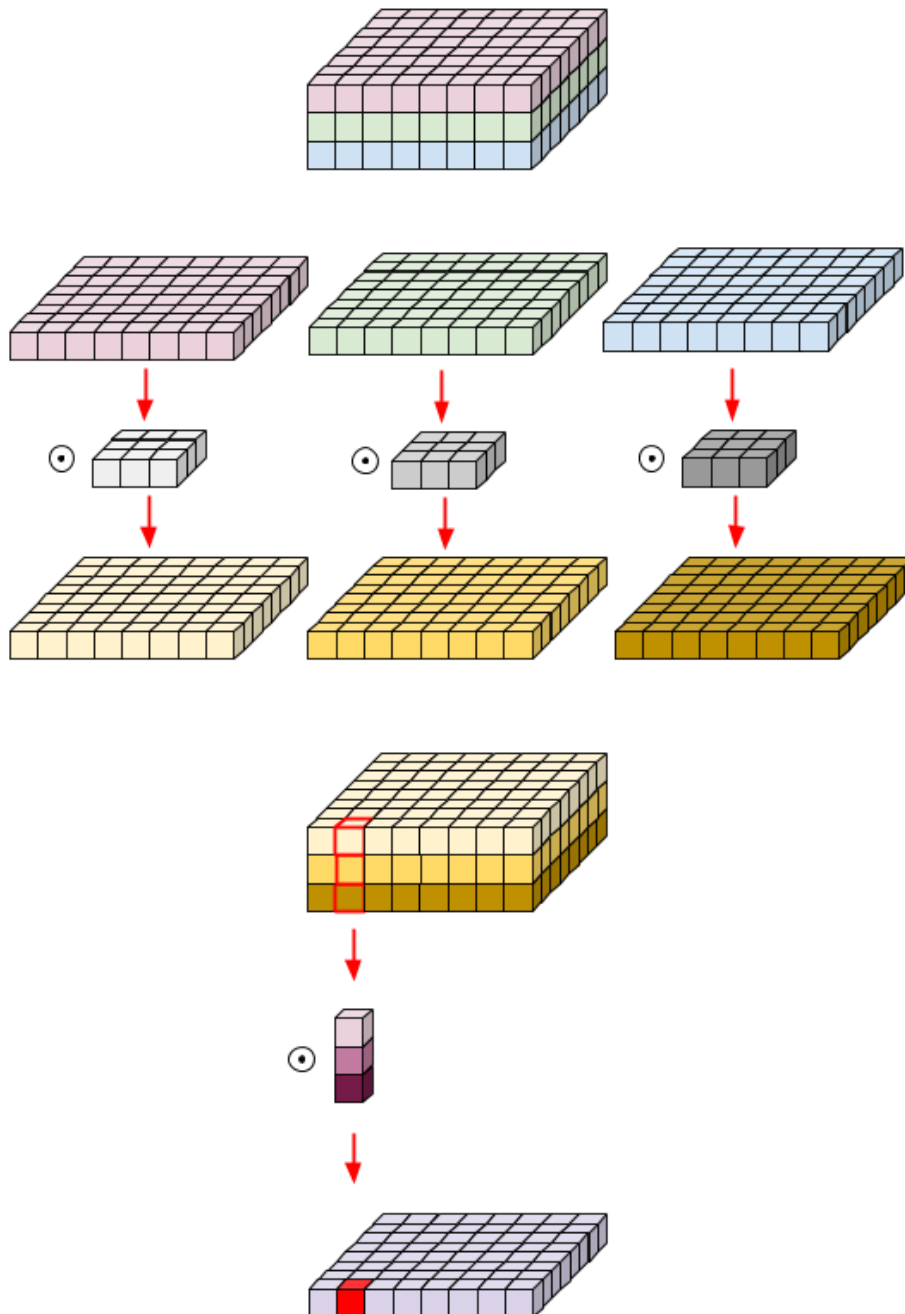
Một cải tiến được áp dụng trong MobileNet giúp giảm thiểu số lượng tham số đó là Depthwise Separable Convolution.

Kiến trúc này sẽ không áp dụng bộ lọc lên toàn bộ độ sâu của layer trước đó. Thay vào đó, 1 bộ lọc chỉ được áp dụng trên 1 channel đơn lẻ.

Khi đó số lượng các tham số sẽ được giảm đi đáng kể.

Để dễ hình dung các bạn theo dõi hình bên dưới:

Top



**Hình 2:** Kiến trúc Depthwise Seperable Convolution. Mục tiêu của chúng ta là cùng áp dụng tích chập để tạo ra output shape  $8 \times 8 \times 1$  với cùng input như ví dụ 1. Ở hàng đầu tiên là khối input với kích thước  $8 \times 8 \times 3$ . Ở hàng thứ 2 chúng ta tách khối thành 3 channels input độc lập. Hàng thứ 4 là kết quả sau khi áp dụng tích chập giữa bộ lọc và channel riêng biệt. Tiếp theo hàng thứ 5 ta stack các kết quả lại thành 1 khối kích thước là  $8 \times 8 \times 3$ . Để thu được output, một bộ lọc tích chập theo chiều sâu kích thước  $1 \times 1 \times 3$  được áp dụng. Output shape là một khối có kích thước  $8 \times 8 \times 1$ .

Như vậy với cùng một output shape, Depthwise Separable Convolution chỉ sử dụng tổng cộng là  $(3 \times 3 \times 3)$  tham số ở bộ lọc hàng thứ 3 và cộng thêm 3 tham số ở bộ lọc hàng thứ 6. Kết quả ta sử dụng chỉ 30 tham số so với 81 tham số.

## 3.2. Kiến trúc model MobileNet

Top

Về cụ thể kiến trúc của từng layers trong Mobile Net các bạn có thể xem tại bài báo gốc  
 MobileNet: Efficient Convolutional Neural Networks for Mobile Vision Applications  
 (<https://arxiv.org/pdf/1704.04861.pdf>).

Mình có thể liệt kê một số layers như bên dưới:

1	Model: "mobilenet_1.00_224"		
2			
3	Layer (type)	Output Shape	Param #
4	=====		
5	input_1 (InputLayer)	[(None, 224, 224, 3)]	0
6			
7	conv1_pad (ZeroPadding2D)	(None, 225, 225, 3)	0
8			
9	conv1 (Conv2D)	(None, 112, 112, 32)	864
10			
11	conv1_bn (BatchNormalization)	(None, 112, 112, 32)	128
12			
13	conv1_relu (ReLU)	(None, 112, 112, 32)	0
14			
15	conv_dw_1 (DepthwiseConv2D)	(None, 112, 112, 32)	288
16			
17	conv_dw_1_bn (BatchNormaliza	(None, 112, 112, 32)	128
18			
19	conv_dw_1_relu (ReLU)	(None, 112, 112, 32)	0
20			
21	conv_pw_1 (Conv2D)	(None, 112, 112, 64)	2048
22			
23	conv_pw_1_bn (BatchNormaliza	(None, 112, 112, 64)	256
24			
25	conv_pw_1_relu (ReLU)	(None, 112, 112, 64)	0
26			
27	conv_pad_2 (ZeroPadding2D)	(None, 113, 113, 64)	0
28			
29	...		
30			
31	conv_pw_13_bn (BatchNormaliz	(None, 7, 7, 1024)	4096
32			
33	conv_pw_13_relu (ReLU)	(None, 7, 7, 1024)	0
34			
35	global_average_pooling2d (Gl	(None, 1024)	0
36			
37	reshape_1 (Reshape)	(None, 1, 1, 1024)	0
38			
39	dropout (Dropout)	(None, 1, 1, 1024)	0
40			
41	conv_preds (Conv2D)	(None, 1, 1, 1000)	1025000
42			
43	reshape_2 (Reshape)	(None, 1000)	0
44			
45	predictions (Activation)	(None, 1000)	0
46	=====		
47	Total params: 4,253,864		
48	Trainable params: 4,231,976		
49	Non-trainable params: 21,888		

Top

Để không quá rối mắt, mình đã truncate một số layers trung gian. Bạn có thể simulate lại kết quả trên tensorflow:

```
1 from tensorflow.keras.applications import MobileNet
2
3 model = MobileNet(weights=None)
4 model.summary()
```

Những dòng (DepthwiseConv2D) là áp dụng tích chập chiều sâu tách biệt. Cách tính số lượng tham số sẽ khác so với tích chập 2 chiều chuẩn. Bạn đọc quan tâm có thể tính thử.

### 3.3. Convert model trên tensorflow js

Mô hình MobileNet của tensorflowjs sẽ hoạt động trực tiếp lên frontend. Do đó tốc độ load và train, predict nhanh hơn so với việc call API. Đồng thời các bạn sẽ thấy được rằng javascript đã chia nhỏ model lớn thành các nodes có kích thước bằng nhau giúp tăng tốc độ xử lý và đồng thời kích thước file cũng nhẹ hơn rất nhiều so với mô hình trên tensorflow. Kích thước model MobileNet gốc của mình khoảng 16 MB nhưng convert sang tensorflow js chỉ còn 5.5 MB.

Trước tiên chúng ta sẽ cần convert model tensorflow để sao cho nó có thể hoạt động được trên javascript. Bạn sẽ cần sử dụng package tensorflow js như sau:

- Cài đặt package tensorflowjs

```
1 !pip install tensorflowjs
```

- Load và save pretrain model MobileNet

```
1 from tensorflow.keras.applications import MobileNet
2
3 model = MobileNet(weights='imagenet')
4 model.save('mobilenet.h5')
```

- Convert file mobilenet.h5 về một folder của tensorflow js






```
1 import tensorflowjs as tfjs
2 import os
3
4 os.mkdir('mobilenet_js')
5 tfjs.converters.save_keras_model(model, 'mobilenet_js')
```

Hoặc bạn cũng có thể gõ trên commandline theo cú pháp:

```
tensorflowjs_converter --input_format keras \ mobilenet.h5 \ mobilenet_js
```

Trong đó --input\_format là argument khai báo định dạng model của keras. Hai file mobilenet.h5 là file model gốc cần convert và mobilenet\_js là folder đích sau khi đã convert. Chắc bạn thắc mắc tại sao lại là folder? Như mình đã nói ở trên. Đó là bởi tensorflow javascript phân mô hình thành nhiều nodes có kích thước bằng nhau và đặt chung trong 1 folder. Cơ chế này giúp cho việc load, train, predict nhanh hơn.

Top

Name	Date modified	Type	Size
 group1-shard1of4.bin	3/27/2020 7:40 PM	BIN File	4,096 KB
 group1-shard2of4.bin	3/27/2020 7:40 PM	BIN File	4,096 KB
 group1-shard3of4.bin	3/27/2020 7:40 PM	BIN File	4,096 KB
 group1-shard4of4.bin	3/27/2020 7:40 PM	BIN File	1,537 KB
 model	3/27/2020 7:40 PM	JSON File	100 KB

**Hình 3:** Folder của model tensorflow javascript. File model.json là file chứa kiến trúc của mô hình. Các file group1-shard1of4.bin là những node của model.

## 4. Giao diện

### 4.1. Tạo giao diện

Sau khi đã có model rồi thì chúng ta sẽ cần tạo một giao diện cho app của chúng ta.

Việc thiết kế giao diện khá tốn thời gian đối với ai chưa làm quen với frontend. Bạn phải vừa biết về html, vừa biết về UI/UX và material design. Mình thì không chuyên về cả 3 thứ này nên làm một giao diện thật đơn giản nhưng đầy đủ chức năng là được.

Đầu tiên hãy hình dung mình app của mình có chức năng gì nhé:

- Load model đã được pretrain từ tensorflowjs.
- Upload hình ảnh và hiển thị nó.
- Dự báo hình ảnh.

Như vậy là chúng ta sẽ phải có 3 cái nút lần lượt với 3 tác vụ trên là Upload model, Upload Image và Predict.

Mình sẽ show ra đây cho các bạn để hình dung.

```

<html>
<head>
  <title>Keras MobileNet + TensorFlow.js Demo</title>
  <link href="https://fonts.googleapis.com/css?family=Nunito" rel="stylesheet">
  <link rel="stylesheet" type="text/css" href="css/app.css">
  <script src="https://cdn.jsdelivr.net/npm/tensorflow/tfjs@latest"></script>
  <script type="text/javascript" src="https://code.jquery.com/jquery-2.1.1.min.js"></script>
  <script type="text/javascript" src="js/mobile-net.js"></script>
  <script type="text/javascript" src="js/imagenet_classes.js"></script>
</head>
<body>
  <div class="container">
    <div class="mobilenet-demo-container">|
      <div class="demo-wrapper">
        <div>
          <button id="load-button" class="input-button" onclick="loadModel()">Load Model</button>
          <label for="select-file-image" class="input-button">
            Upload Image
          </label>
          <input type="file" id="select-file-image" onchange="loadImageLocal()" style="display: none;" />
          <div id="progress-box" style="display: none; width: 100%; height: 10px; background-color: #ccc; position: relative;">
            <p id="progress-box-content" style="color: white; position: absolute; top: 0; left: 0; width: 100%; height: 100%; text-align: center; font-size: 10px;">Loading mobilenet model..</p>
          </div>
        </div>
        <div>
          <button id="predict-button" class="input-button predict-button" onclick="predictImage()">Predict</button>
        </div>
        <div class="demo-output">
          <div class="out-box" id="select-file-box" style="display: none;">
            <img id="test-image" />
          </div>
          <div class="out-box" id="predict-box" style="display: none;">
            <p id="prediction"></p>
            <br>
            <p><b style="color: #c2c2c2; font-style: italic; font-weight: 100; font-size: 11px;">Top-5 Predictions</b></p>
            <ul id="predict-list">
            </ul>
          </div>
        </div>
      </div>
    </div>
  </body>
</html>

```

load CDN packages

Button load model

Button Upload Image

Button Predict

**Hình 4:** Nội dung file index.html.

Trong file index.html chúng ta chú ý đến 2 phần chính đó là các đoạn mã nằm trong thẻ script có tác dụng load CDN packages từ các nguồn open source để sử dụng chúng trên frontend. Ngoài ra bạn cũng cần các button Upload model, Upload image, predict Image.

Để tương tác với các button ta cần phải đặt tên cho mỗi button bằng mã id, phần này bạn nào làm lập trình thì quá quen thuộc rồi.

Về chức năng của từng thẻ bạn không biết có thể tra trên w3school (<https://www.w3schools.com/html/default.asp>) nhé.

## 4.2. Tương tác giao diện

Để tương tác với giao diện ta cần code các hàm trên javascript. Các hàm tương tác giao diện được đặt trong file `js/mobile-net`. Phần này đòi hỏi bạn phải hiểu về ngôn ngữ javascript một chút nên bạn xem qua bài w3school Javascript (<https://www.w3schools.com/js/>) nhé.

Mình sẽ không đi sâu vào giải thích code ở đây mà chỉ nói các chức năng chính của từng hàm:

- `loadModel()` : Có tác dụng load model thông qua hàm `tf.loadLayersModel()` tương đương với tensorflow javascript.
- `loadImageLocal()` : Upload hình ảnh từ local file.
- `predictImage()` : Dự báo hình ảnh từ model tensorflow javascript.
- `preprocessImage()` : Để dự báo hình ảnh, chúng ta cần resize image về đúng kích thước phù hợp với input của model MobileNet là `224x224x3`.

Mình sẽ show ra đây cho bạn nào muốn tìm hiểu.



```

let model;
let IMAGE_WIDTH = 300;

async function loadModel() {
    console.log("model loading mobilenet model kdfah ...");
    loader = document.getElementById("progress-box");
    load_button = document.getElementById("load-button");
    loader.style.display = "block";
    modelName = "mobilenet";
    model = undefined;
    model = await tf.loadLayersModel('models/mobilenet/model.json');
    if (typeof model !== "undefined") {
        loader.style.display = "none";
        load_button.disabled = true;
        load_button.innerHTML = "Loaded Model";
        console.log("model loaded..");
    }
};

function loadImageLocal() {
    console.log("Click into selected file image");
    document.getElementById("select-file-box").style.display = "table-cell";
    document.getElementById("predict-box").style.display = "table-cell";
    document.getElementById("prediction").innerHTML = "Click predict to f
    renderImage(this.files);
};

function renderImage(file) {
    var reader = new FileReader();
    reader.onload = function(event) {
        let output = document.getElementById('test-image');
        output.src = reader.result;
        output.width = IMAGE_WIDTH;
    }
    if(event.target.files[0]){
        reader.readAsDataURL(event.target.files[0]);
    }
}

async function predictImage(){
    console.log("Click predict button");
    if (model == undefined) {
        alert("Please load the model first..")
    }
    if (document.getElementById("predict-box").style.display == "none") {
        alert("Please load an image using 'Upload Image' button..")
    }
    let image = document.getElementById("test-image");
    let tensor = preprocessImage(image, modelName);
    let predictions = await model.predict(tensor).data();
    let results = Array.from(predictions)
        .map(function (p, i) {
            return {
                probability: p,
                className: IMAGENET_CLASSES[i]
            };
        })
        .sort(function (a, b) {

```

Top

```

        return b.probability - a.probability;
    }).slice(0, 5);
    document.getElementById("predict-box").style.display = "block";
    document.getElementById("prediction").innerHTML = "MobileNet predicti
" + results[0].className + " ";

    var ul = document.getElementById("predict-list");
    ul.innerHTML = "";
    results.forEach(function (p) {
        console.log(p.className + " " + p.probability.toFixed(6));
        var li = document.createElement("LI");
        li.innerHTML = p.className + " " + p.probability.toFixed(6);
        ul.appendChild(li);
    })

    if (typeof predictions !== "undefined"){
        document.getElementById("progress-box").style.display = "none"
    }
}

```

## 5. Server

Tương tự như Bài 29 - Xây dựng flask API

(<https://phamdinhhkhanh.github.io/2020/03/23/FlaskRestAPI.html>) mà mình đã giới thiệu. Để một ứng dụng hoạt động được thì cần có một server bên dưới. Server tương tác với frontend thông qua API.

Để khởi tạo server mình sử dụng nodejs thay vì python vì nodejs tốc độ cao hơn, đồng thời render kết quả lên frontend mà không cần load và redirect lại trang.

App engine mà mình sử dụng là express, một app engine middleware trung gian giữa server và frontend có tác dụng điều chỉnh các response trước khi gửi cho client. Dành cho bạn nào muốn tìm hiểu thêm về express (<https://viblo.asia/p/nodejs-tutorial-phan-6-middleware-trong-expressjs-924IJXzWKPM>).

Code khởi tạo app của mình như sau:

```

1    const express = require('express');
2    let app = express();
3
4    app.get("/", cors(corsOptions), (req, res, next) => {
5        res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With,
6            res.sendFile(path.join(__dirname+"/index.html"));
7            next();
8    });

```

app đã khởi tạo một API GET khai báo tại vị trí root trả về trang chủ là index.html. Tại đây chúng ta sẽ tương tác với giao diện và thực hiện các chức năng upload, dự báo ảnh.

Để server hoạt động thì phần backend chúng ta phải import các packages của nodejs. Các bạn phải cài đặt chúng thông qua npm (node package manager), một công cụ tương tự như pip trên python. Tại root directory bạn gõ:

```
npm install
```

Top

Lệnh này sẽ lookup các thư viện được khai báo trong `package.json` (đã được mình định nghĩa sẵn) và cài đặt chúng.

Lưu ý trước đó phải cài npm (<https://www.npmjs.com/get-npm>).

Sau khi thiết kế xong server thì bạn đã có thể start nó lên bằng lệnh

```
node server.js
```

Quá trình này hoàn thành sẽ trả ra câu lệnh trên command line:

```
Website is Up on PORT 3000 and HOSTNAME development !
```

Bạn truy cập vào link local host `https://127.0.0.1:3000` và test các chức năng.

## 6. Deploy lên heroku

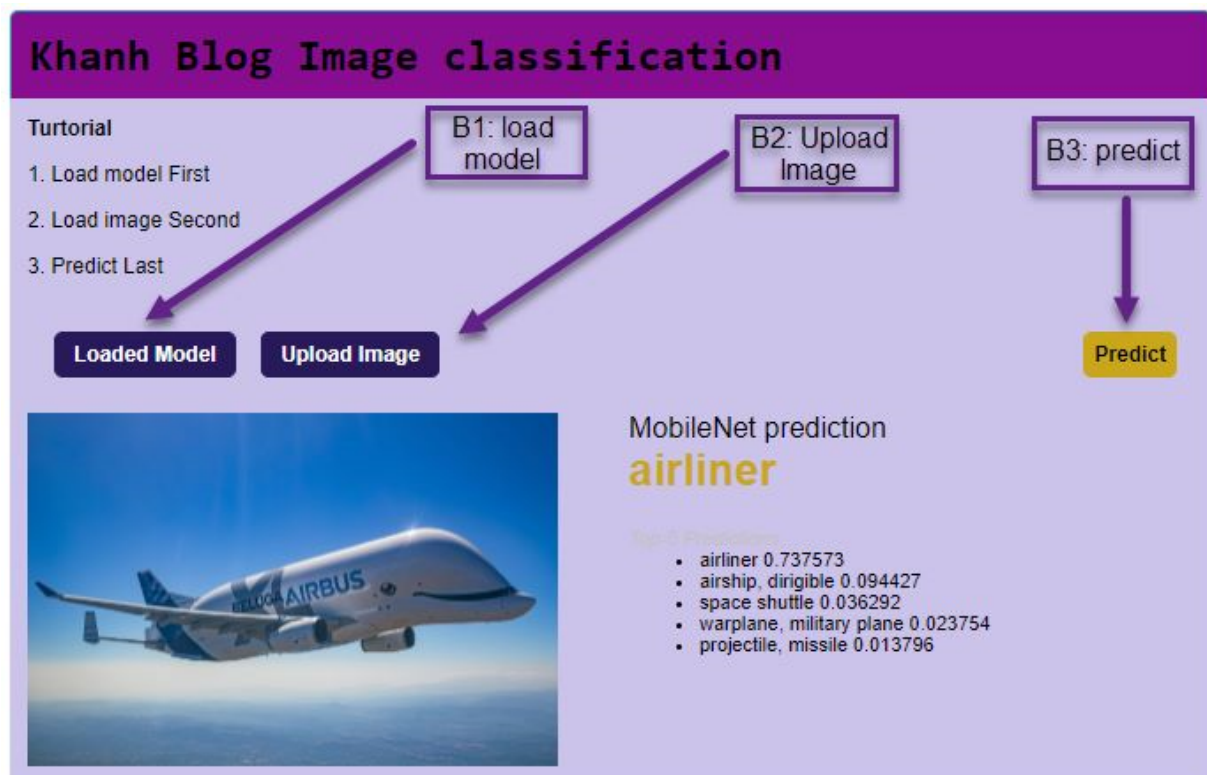
Sau khi đã xây dựng thành công model chúng ta sẽ cần deploy website lên một domain web service. Có khá nhiều domain free khác nhau như heroku, github.io,....

Ở đây mình lựa chọn heroku vì tên miền này hỗ trợ triển khai cho cả website và app. Ngoài ra bạn được miễn phí 500MB cơ sở dữ liệu lưu trữ sandbox.

Vì phần hướng dẫn của bài viết đã khá dài và đêm cũng đã khá muộn. Mặc dù rất muốn viết cho bạn đọc thêm nữa nhưng mình đã khá mệt.

Và đây sẽ là lúc các bạn được rèn luyện tiếng anh của mình bằng cách làm theo hướng dẫn deploy heroku website (<https://devcenter.heroku.com/categories/deployment>).

Kết quả cuối cùng bạn thu được là một ứng dụng AI tuyệt vời, có tác dụng phân loại và gắn nhãn thay cho sức lao động của hàng trăm con người:



**Hình 5:** Giao diện của ứng dụng web AI.

Bạn có thể trải nghiệm thêm sản phẩm tại: Alcode heroku app (<https://aicode.herokuapp.com/>)

Top

## 7. Tổng kết:

Như vậy qua bài viết này mình đã hướng dẫn các bạn triển khai xây dựng một sản phẩm web AI. Các bạn sinh viên có thể sử dụng nó để đưa vào CV của mình khi xin việc. Ngoài ra bạn cũng có thể làm các ứng dụng web AI nho nhỏ để phục vụ cộng đồng rồi đó.

Github code của dự án được để tại [khanhBlogTutorial](https://github.com/phamdinhhkhanh/khanhBlogTutorial) (<https://github.com/phamdinhhkhanh/khanhBlogTutorial>).

Hãy cùng join fanpage của mình để nhận được các open source hay: AICode (<https://www.facebook.com/groups/3235479620010379/>) và hỗ trợ các lỗi code phát sinh nếu có.

Và subscribe KhanhBlog (<https://www.facebook.com/TowardDataScience/>) để theo dõi các bài viết về AI của mình nhé.

## 8. Tài liệu tham khảo:

1. Tensorflow js (<https://www.tensorflow.org/js>)
2. w3schools (<https://www.w3schools.com/>)
3. Aicode heroku app (<https://aicode.herokuapp.com/>)
4. heroku app (<https://devcenter.heroku.com/>)

Top