**Trường Đại học Ngoại ngữ - Tin học TP.HCM**

**Khoa Công nghệ thông tin**



**TÌM HIỂU MẠNG NƠ RON HỌC SÂU**

**CONVOLUTIONAL NEURAL NETWORK**

**VÀ PHẦN MỀM TENSORFLOW.JS**

**ÁP DỤNG VÀO XÂY DỰNG CHƯƠNG TRÌNH**

**PHÂN LỚP HÌNH ẢNH DIGIT 0-9**

Môn học : CƠ SỞ TRÍ TUỆ NHÂN TẠO

**GVHD : PGS.TS.Dương Tuấn Anh**

Sinh viên thực hiện :

Đặng Lê Gia Vũ – 18DH110138

Trần Hy Khang – 18DH110035

Phan Tuấn Anh – 18DH110184

**TPHCM**,**Tháng 12, Năm 2020**

MỤC LỤC BÁO CÁO

[Phần 1 GIỚI THIỆU TENSORFLOW.JS 1](#_Toc58512424)

[Phần 2 : Mạng nơ ron học sâu Convolutional Neurel Network 2](#_Toc58512425)

I.1.Convolutional Layer………………………………………………………………………………………………………5

[I.1. Pooling layer 6](#_Toc58512426)

[I.3 Fully connected layer 8](#_Toc58512427)

[I.4 Visualise convolutional neural network 8](#_Toc58512428)

[Phần 3 MNIST DATASET 9](#_Toc58512429)

[Phần 4 ỨNG DỤNG TENSORFLOW.JS VÀ MNIST DATASET ĐỂ XÂY DỰNG ỨNG DỤNG PHẦN LỚP HÌNH ẢNH DIGIT 0-9 12](#_Toc58512430)

[I. Demo ứng dụng 12](#_Toc58512431)

[II. Dữ liệu đầu vào MNIST Dataset 13](#_Toc58512432)

[III. Xử lý Data đầu vào 14](#_Toc58512433)

[IV. Setup Sequential Network 15](#_Toc58512434)

[V. Quy trình training 18](#_Toc58512435)

[Phần 5 TÀI LIỆU THAM KHẢO 21](#_Toc58512436)

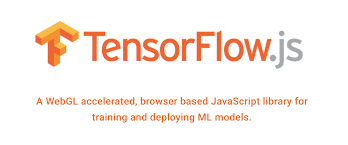
# GIỚI THIỆU TENSORFLOW.JS

[TensorFlow](https://www.tensorflow.org/" \t "_blank) là một thư viện mã nguồn mở (open-source) cho các ứng dụng machine learning. TensorFlow cũng có thể được sử dụng để implement các neural networks hay các thuật toán deep learning.Được phát hành tháng 11, 2015, ban đầu TensoFlow là một [thư viện Python](https://www.tensorflow.org/api_docs/python/). Nó được sử dụng khả năng tính toán của CPU và GPU để training và đánh giá các machine learing models. Và tất nhiên, thời điểm đó, nó được thiết để chạy trên các máy chủ có hiệu suất cực cao và đắt tiền.Gần đây, TensorFlow đã thay đổi rất nhiều, khi nó có thể chạy trên các thiết bị có hiệu suất thấp hơn Mobile hay trình duyệt web.

Các chức năng nổi bật của Tensorflow.js :

* 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 train model trên website.



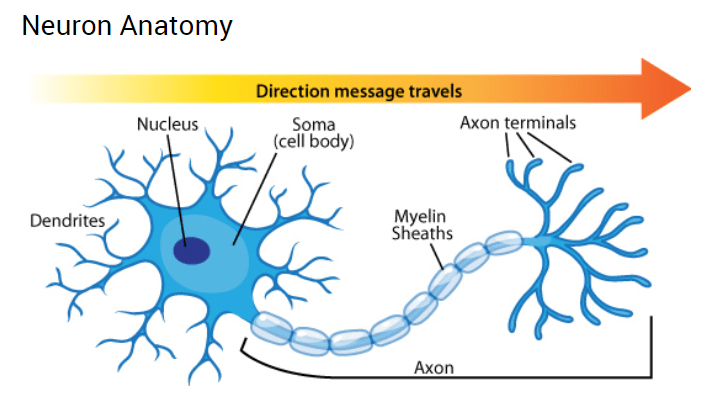
Hình 1.1 – Framework Tensorflow.js

# Phần 2 : Mạng nơ ron học sâu Convolutional Neurel Network

Neurel network là gì ?

Con người chúng ta có thể phân biệt được người thân trong gia đình và người lạ hay đứa trẻ có thể phân biệt được các con vật hay màu sắc. 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 bộ não với lượng lớn các nơ-ron thần kinh liên kết với nhau. Thế thì máy tính có nên mô phỏng lại mô hình ấy để giải các bài toán trên ?

Neural là tính từ của neuron (nơ-ron), network chỉ cấu trúc đồ thị nên neural network (**NN**) là một hệ thống tính toán lấy cảm hứng từ sự hoạt động của các nơ-ron trong hệ thần kinh.



Hình 2.1 - Nơ ron thần kinh con người

Nơ-ron là đơn vị cơ bản cấu tạo hệ thống thần kinh và là một phần quan trọng nhất của não. Não chúng ta gồm khoảng 10 triệu nơ-ron và mỗi nơ-ron liên kết với 10.000 nơ-ron khác.

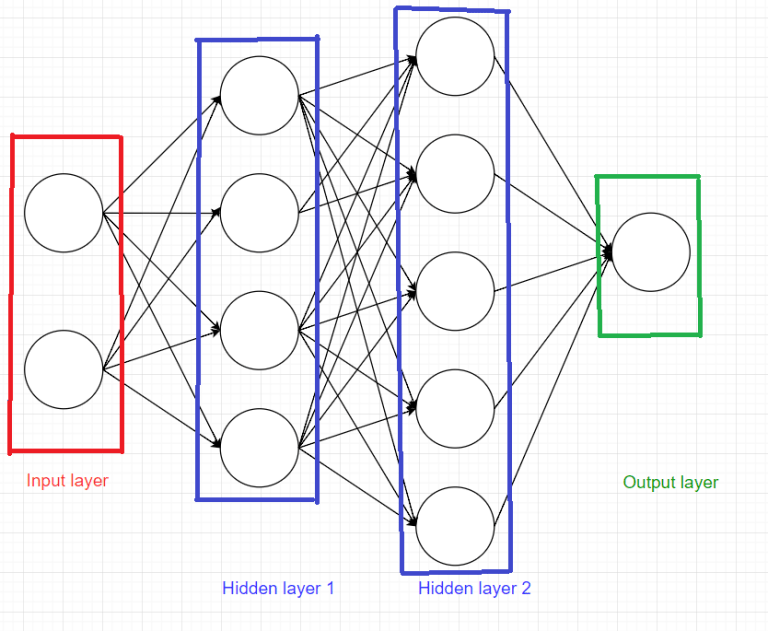
Ở mỗi nơ-ron có phần thân (soma) chứa nhân, các tín hiệu đầu vào qua sợi nhánh (dendrites) và các tín hiệu đầu ra qua sợi trục (axon) kết nối với các nơ-ron khác. Hiểu đơn giản mỗi nơ-ron nhận dữ liệu đầu vào qua sợi nhánh và truyền dữ liệu đầu ra qua sợi trục, đến các sợi nhánh của các nơ-ron khác.

Mỗi nơ-ron nhận xung điện từ các nơ-ron khác qua sợi nhánh. Nếu các xung điện này đủ lớn để kích hoạt nơ-ron, thì tín hiệu này đi qua sợi trục đến các sợi nhánh của các nơ-ron khác. => Ở mỗi nơ-ron cần quyết định có kích hoạt nơ-ron đấy hay không.

Tuy nhiên NN chỉ là lấy cảm hứng từ não bộ và cách nó hoạt động, chứ không phải bắt chước toàn bộ các chức năng của nó. Việc chính của chúng ta là dùng mô hình đấy đi giải quyết các bài toán chúng ta cần.

Mô hình tổng quát của neural network :

Mỗi hidden layer được gọi là **fully connected layer**, tên gọi theo đúng ý nghĩa, mỗi node trong hidden layer được kết nối với tất cả các node trong layer trước. Cả mô hình được gọi là **fully connected neural network (FCN)**.



Hình 2.2 - **fully connected neural network (FCN)**.

Layer đầu tiên là input layer, các layer ở giữa được gọi là hidden layer, layer cuối cùng được gọi là output layer. Các hình tròn được gọi là node.

Mỗi mô hình luôn có 1 input layer, 1 output layer, có thể có hoặc không các hidden layer. Tổng số layer trong mô hình được quy ước là số layer – 1 (Không tính input layer).

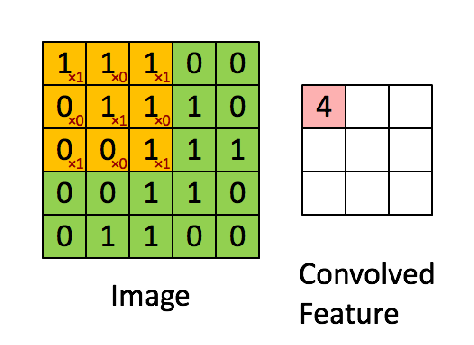
Ví dụ như ở hình trên có 1 input layer, 2 hidden layer và 1 output layer. Số lượng layer của mô hình là 3 layer.

Mỗi node trong hidden layer và output layer :

* Liên kết với tất cả các node ở layer trước đó với các hệ số w riêng.
* Mỗi node có 1 hệ số bias b riêng.
* Diễn ra 2 bước: tính tổng linear và áp dụng activation function.

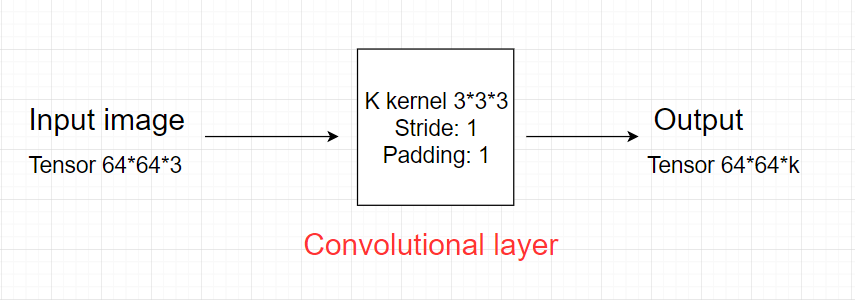
Convolutional Layer

Bài toán convolution thực hiện trên ảnh xám với biểu diễn ảnh dạng ma trận. Tuy nhiên ảnh màu có tới 3 channels red, green, blue nên khi biểu diễn ảnh dưới dạng tensor 3 chiều. Nên ta cũng sẽ định nghĩa kernel là 1 tensor 3 chiều kích thước k\*k\*3.Do ảnh màu RGB có 3 channels.



Hình 2.3 – Cách mà kernel trượt trên image

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 2.4 – Mô hình quá convolutional layer

Convolutional layer đầu tiên

Output của convolutional layer đầu tiên sẽ thành input của convolutional layer tiếp theo.

#### **I.1 Convolutional layer tổng quát**

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.



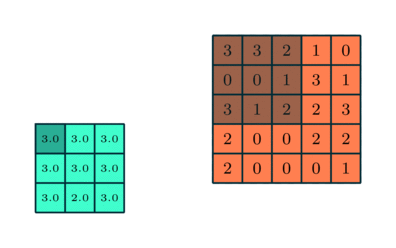
Hình 2.5 – Convolutional layer tổng quát

* Output của convolutional layer sẽ qua hàm **activation function** trước khi trở thành input của convolutional layer tiếp theo.
* Tổng số parameter của layer: Mỗi kernel có kích thước F\*F\*D và có 1 hệ số bias, nên tổng parameter của 1 kernel là F\*F\*D + 1. Mà convolutional layer áp dụng K kernel => Tổng số parameter trong layer này là K \* (F\*F\*D + 1).

### 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. Kích thước dữ liệu giảm giúp giảm việc tính toán trong model.

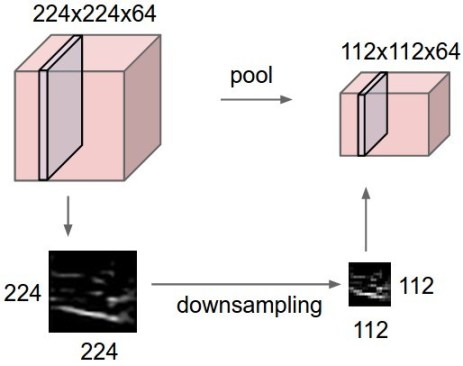
Gọi pooling size kích thước K\*K. Input của pooling layer có kích thước H\*W\*D, ta tách ra làm D ma trận kích thước H\*W. Với mỗi ma trận, trên vùng kích thước K\*K trên ma trận ta tìm maximum hoặc average của dữ liệu rồi viết vào ma trận kết quả. Quy tắc về stride và padding áp dụng như phép tính convolution trên ảnh.



Hình 2.6 – Pooling layer

max pooling layer với size=(3,3), stride=1, padding=0

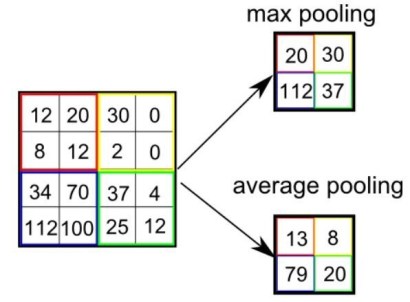
Nhưng **hầu hế**t khi dùng pooling layer thì sẽ dùng size=(2,2), stride=2, padding=0. Khi đó output width và height của dữ liệu giảm đi một nửa, depth thì được giữ nguyên .



Hình 2.7 – Pooling layer

Sau pooling layer (2\*2).  
Source: http://cs231n.github.io/convolutional-networks/

Có 2 loại pooling layer phổ biến là: max pooling và average pooling.

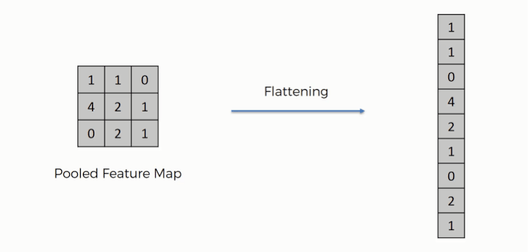


Hình 2.8 – Max pooling và average pooling layer

**Trong một số model người ta dùng convolutional layer với stride > 1 để giảm kích thước dữ liệu thay cho pooling layer.**

### I.3 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)



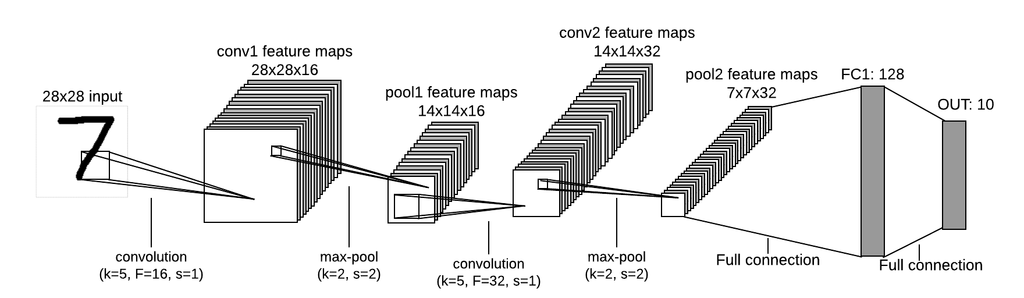
Hình 2.9 – Fully connected layer

Sau đó ta dùng các fully connected layer để kết hợp các đặc điểm của ảnh để ra được output của model.

### I.4 Visualise convolutional neural network

Mô hình convolutional neural network:

Input image -> Convolutional layer (Conv) + Pooling layer (Pool) -> Fully connected layer (FC) -> Output.

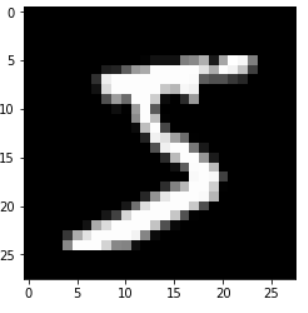


Hình 2.10

Ví dụ mô hình convolutional neural network.

# Phần 3 MNIST DATASET

Bạn có một ảnh xám kích thước 28 \* 28 của chữ số từ 1 đến 9 (digits) và muốn dự đoán số từ ảnh cho vào là số mấy . Ví dụ :



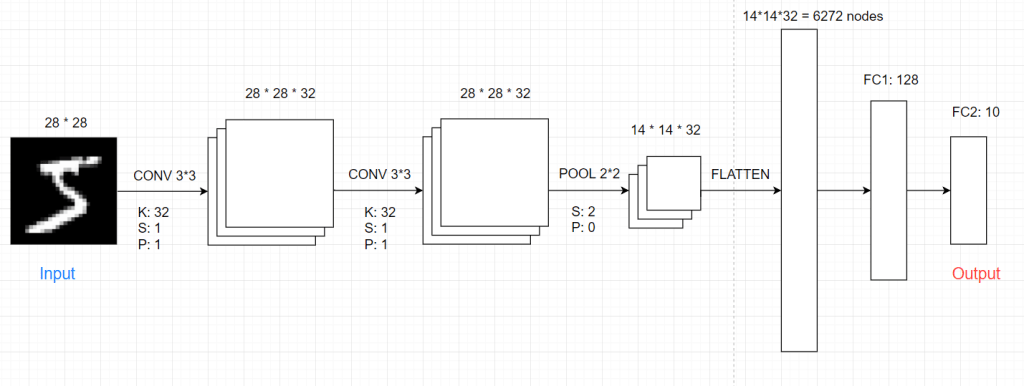
Hình 3.1 – Input dataset

[MNIST](http://yann.lecun.com/exdb/mnist/) là bộ cơ sở dữ liệu về chữ số viết tay, bao gồm 2 tập con: training set gồm 60.000 ảnh các chữ số viết tay và test set gồm 10.000 ảnh các chữ số.

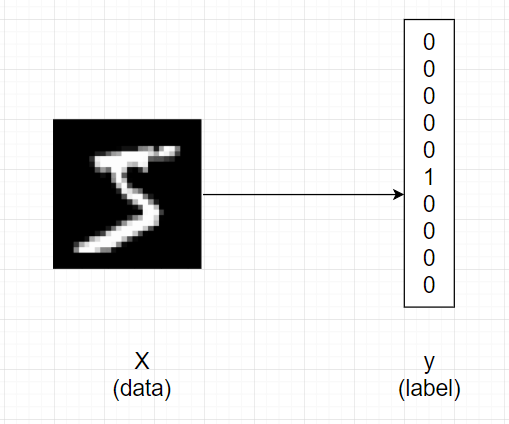
Như vậy MNIST dataset có 60.000 dữ liệu ở training set ở trong MNIST, ta sẽ chia ra 50.000 dữ liệu cho training set và 10.000 dữ liệu cho validation set. Vẫn giữ nguyên 10.000 dữ liệu của test set.

Xây dựng model cho bài toán

Input của model là ảnh xám kích thước 28\*28.

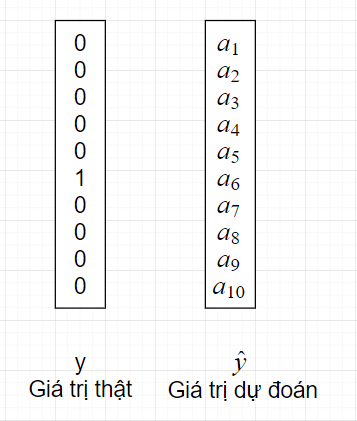
Mô hình chung bài toán CNN: Input image -> Convolutional layer (Conv) + Pooling layer (Pool) -> Fully connected layer (FC) -> Output.

Hình 3.2 – Model bài toán phân loại digits

Để định nghĩa loss function, trước hết ta dùng one-hot encoding chuyển đổi label của ảnh từ giá trị số sang vector cùng kích thước với output của model. Ví dụ:Hình 3.3 – Trích xuất label

Để ý là label của data là số i là vector v kích thước 10\*1 với v\_{i+1}= 1*vi*+1​=1 và các giá trị khác bằng 0. So với quy ước về phần trăm ở trên thì one-hot encoding có ý nghĩa là ta chắc chắn 100% ảnh này là số 5.

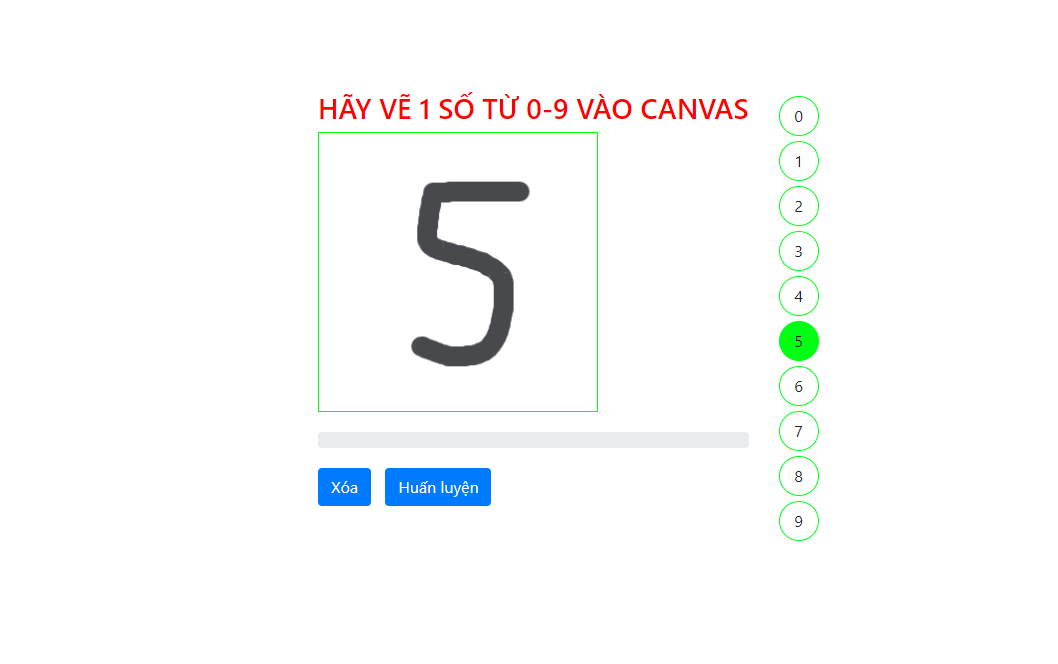
Giờ ta có giá trị thật (label) dạng one-hot encoding giá trị dự đoán ở output layer sau hàm softmax function cùng kích thước 10\*1. Ta cần định nghĩa hàm loss function để đánh giá độ tốt của model.



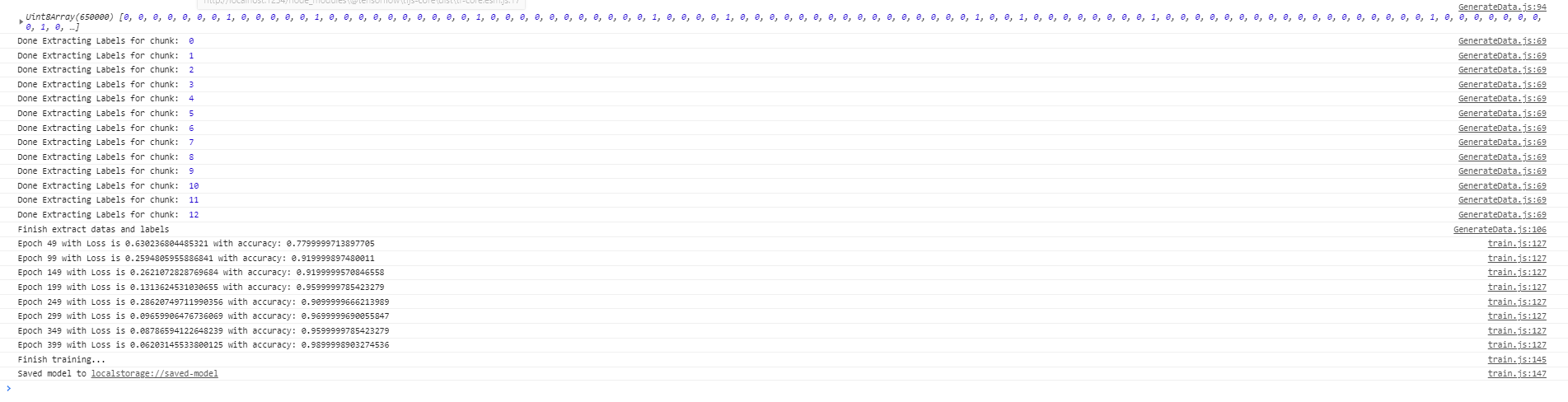
Hình 3.4 Giá trị thật và dự đoạn sau hàm Softmax

# Phần 4 ỨNG DỤNG TENSORFLOW.JS VÀ MNIST DATASET ĐỂ XÂY DỰNG ỨNG DỤNG PHẦN LỚP HÌNH ẢNH DIGIT 0-9

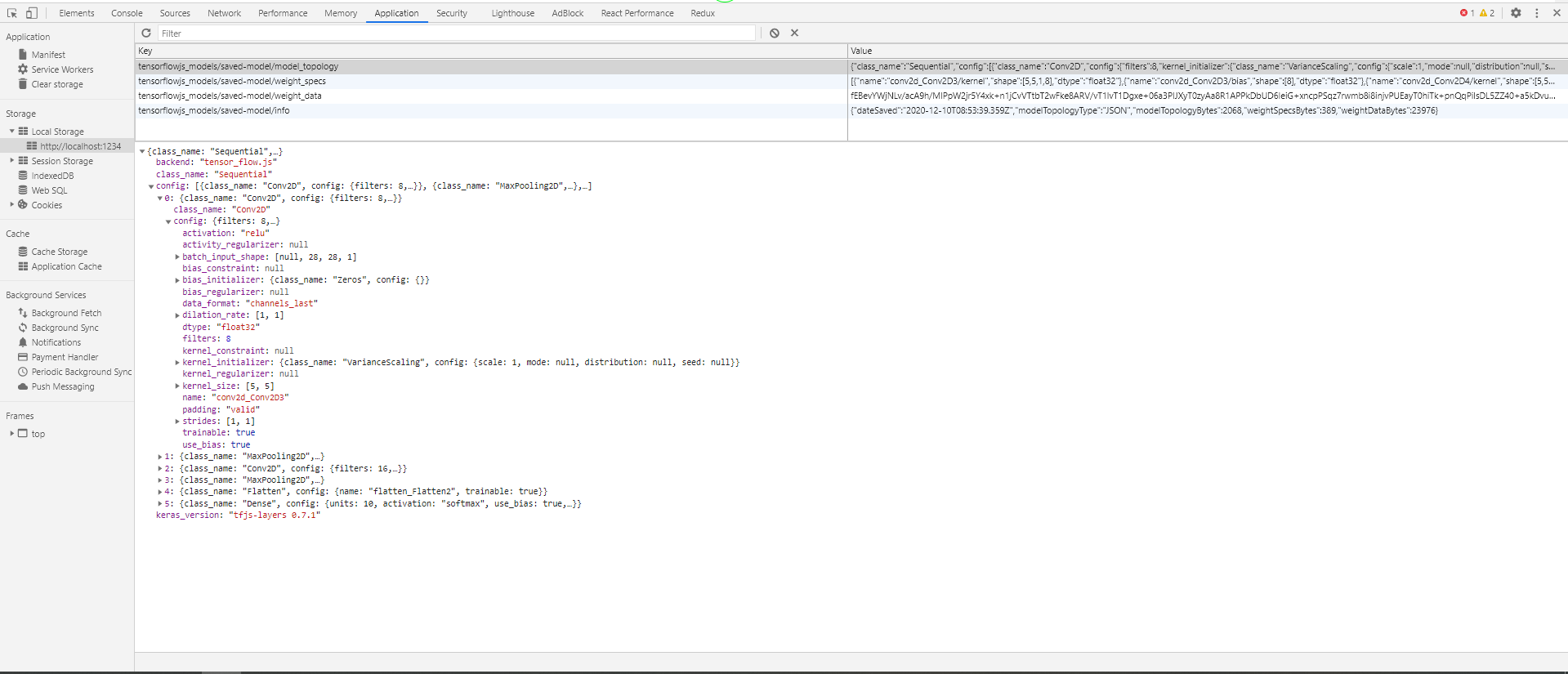
## Demo ứng dụng



Hình 4.1 – Giao diện chính của ứng dụng

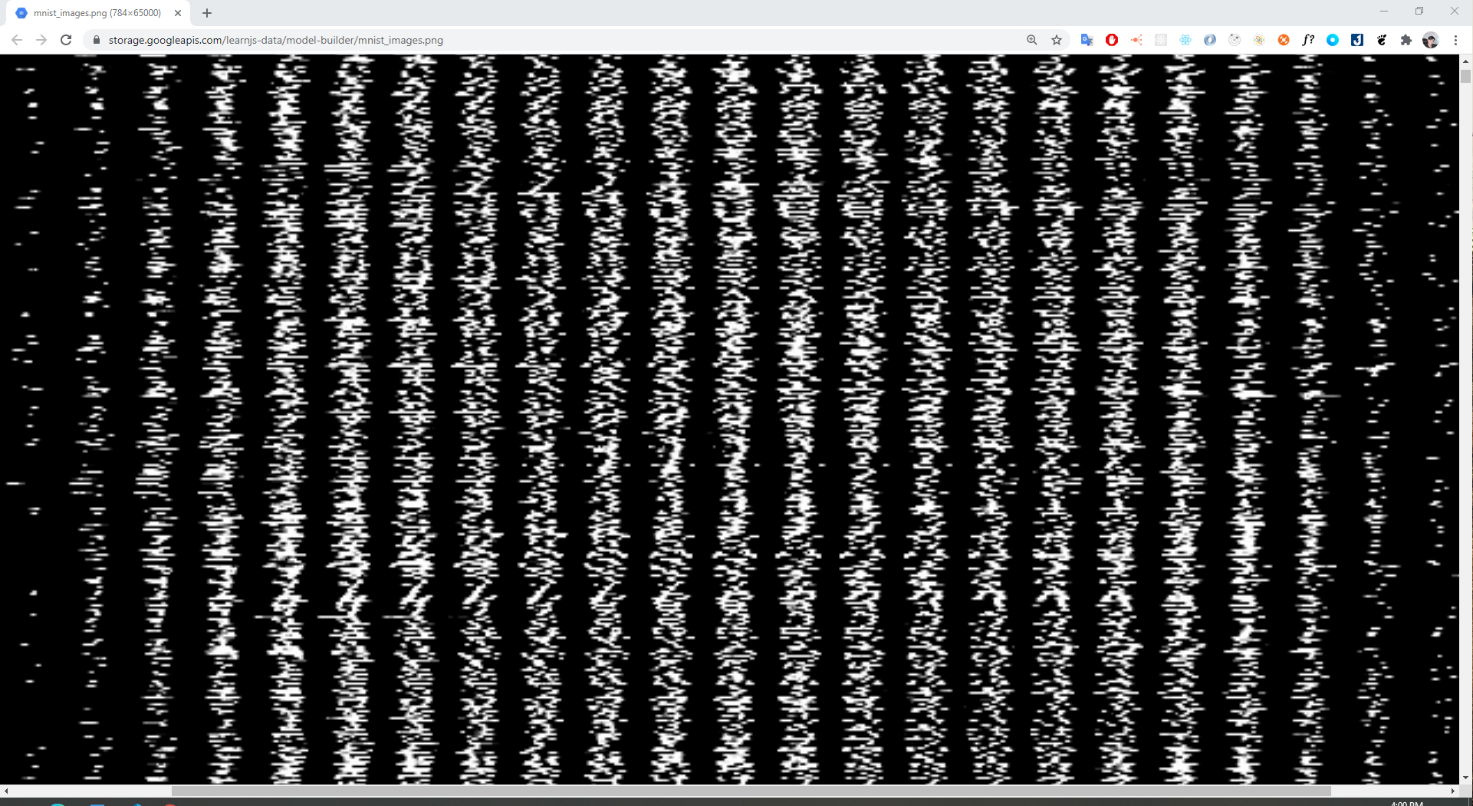


Hình 4.2 – Quá trình train qua các epochs



Hình 4.3 – Nơi lưu models - localstorage

## Dữ liệu đầu vào MNIST Dataset



Hình 4.4 – Tấm ảnh lớn từ MNIST dataset

Matrix 6500 \* 784 = 6500 \* 28 \* 28

65000 ảnh về chữ số viết tay

Mỗi ảnh có kích thước 28 x 28

Vẽ ra chiều ngang 784 sau đó chồng lên nhau.

Ứng với nó cũng có bộ labels của MNIST

## Xử lý Data đầu vào

    for(let i=0; i < this.TOTAL\_DATASET\_NUM/this.CHUNK\_SIZE; i+=1) { // 0 -> 12

                    // Float32Array(3920000)

                    // Tạo dataview để hold những pixel

                    // Set chunk = 5000 ->

                    // 65000 / 5000 = 13 dataview

                    const datasetBytesView = **new** Float32Array(datasetBuff,

                        i \* this.CHUNK\_SIZE \* this.IMG\_SIZE \* 4

                        , this.IMG\_SIZE \* this.CHUNK\_SIZE);

Load data vào trình duyệt sau đó chia data thành 13 phần (CHUNK\_SIZE).Sau đó vẽ lên Canvas. Sau đó extract pixel từ canvas ra ( mỗi pixel là 1 node trong mạng nơ ron )

  ctx.drawImage(img, 0, i \* this.CHUNK\_SIZE, img.width, this.CHUNK\_SIZE, 0, 0, img.width, this.CHUNK\_SIZE);

                    const imgData = ctx.getImageData(0, 0, img.width, this.CHUNK\_SIZE);

                    const imgDateLength = imgData.data.length

Tương tự ta cũng extract labels ra :

  extractLabels() {

        // Xử lý bất đồng bộ với Promise

        return **new** Promise((resolve, reject) => {

            fetch(this.LABELS\_LINK).then(res => {

                res.arrayBuffer().then(buff => {

                    const labels = **new** Uint8Array(buff);

                    this.labels = labels;

                    console.log(labels);

                    resolve();

                }).catch(err => reject(err))

            }).catch(err => reject(err))

        })

    }

Để tránh việc mạng CNN của chúng ta học thuộc data.Ta phải luôn có hàm test hoặc validation data.Và phần data này mạng CNN sẽ không được học mà chỉ dung để so khớp kiểm tra tình trạng học qua những lần epochs.Ở đây ta phân mini batch để chạy nhanh hơn và sắp hộp index.Sau khi xử lý xong ta extracts ra Tensor 2D.

 nextBatch(type, batchSize) {

        let images;

        let labels;

        const batchImgs = **new** Float32Array(this.IMG\_SIZE \* batchSize);

        const batchLabels = **new** Uint8Array(this.CLASSES\_NUM \* batchSize);

        let idx;

        if(type === "train") {

            [ images, labels ] = [ this.trainImgs, this.trainLabels ];

            const newTrainIndex = this.currentTrainIndex + batchSize;

            idx = this.trainIndicies.slice(this.currentTrainIndex, newTrainIndex);

            this.currentTrainIndex = newTrainIndex;

        } else if (type === "test") {

            [ images, labels ] = [ this.testImgs, this.testLabels ];

            const newTestIndex = this.currentTestIndex + batchSize;

            idx = this.trainIndicies.slice(this.currentTestIndex, newTestIndex);

            this.currentTestIndex = newTestIndex;

        }

        for(let i =0; i < batchSize; i += 1) {

            const index = idx[i];

            const image = images.slice(index \* this.IMG\_SIZE, (index+1) \* this.IMG\_SIZE)

            const label = labels.slice(index \* this.CLASSES\_NUM, (index + 1) \* this.CLASSES\_NUM)

            batchImgs.set(image, i \* this.IMG\_SIZE);

            batchLabels.set(label, i \* this.CLASSES\_NUM);

        }

        return {

            images: tf.tensor2d(batchImgs, [ batchSize, this.IMG\_SIZE ]),

            labels: tf.tensor2d(batchLabels, [ batchSize, this.CLASSES\_NUM ])

        }

    }

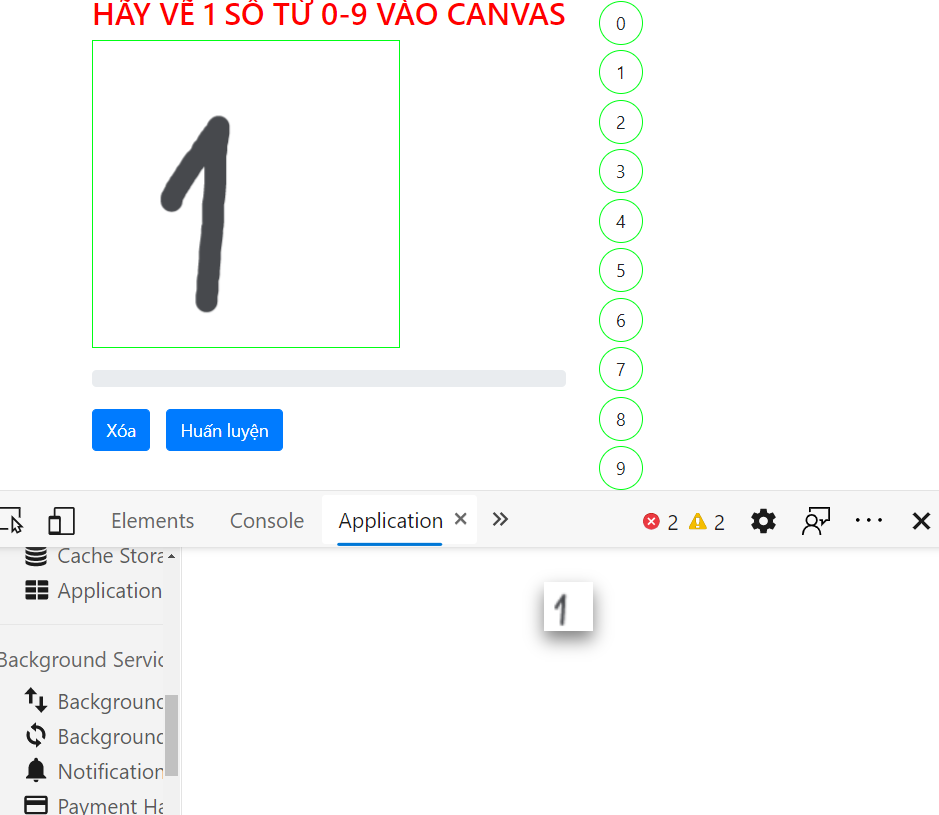
## Setup Sequential Network

Sau khi xử lý Data xong, ta sử dụng mạng Sequential (API của Tensorflow.js)

 // Implement Sequential Neural Network

    const model = tf.sequential()

**Thiết kế lớp đè lên lớp Convolutional với input là 1 ảnh : 28\*28**

****

  // Output ((n + 2p -f) / s + 1) \* ((n + 2p -f) / s + 1) \* numsFilters

    // = ((28 + 2 \* 0 - 5) / 1 + 1) \* ((28 + 2 \* 0 - 5) / 1 + 1) \* 8

    // = (24 \* 24 \* 8)

    // Lớp convolutional 1

    const conv2d\_1 = tf.layers.conv2d({

        inputShape: [28, 28, 1],

        kernelSize: 5,

        filters: 8,

        strides: 1,

        activation: 'relu',

        kernelInitializer: 'varianceScaling'

    })

Hàm activation ta dùng hàm Relu : chuyển từ tuyến tính sang phi tuyến

 // Add convolution layer 1

    model.add(conv2d\_1)

    // Output ((n + 2p -f) / s + 1) \* ((n + 2p -f) / s + 1) \* numsFilters

    // = ((24 + 2 \* 0 - 2) / 2 + 1) \* ((24 + 2 \* 0 - 2) / 2 + 1) \* 8

    // = (12 \* 12 \* 8)

    // Lớp max pooling 1

    const maxPooling2d\_1 = tf.layers.maxPooling2d({

        poolSize: [2, 2],

        strides: [2, 2]

    })

    // Add max pooling layer  1

    model.add(maxPooling2d\_1)

Lớp maxpooling khác với lớp convolutional là : khi filter trượt qua (hay kernel) nó nhân vào sau đó action vation function sẽ chuyển từ tuyến tính => phi tuyến nhưng max pooling chỉ lấy giá trị lớn nhất mà thôi.Ví dụ trượt qua ma trận 2x2 ta chỉ lấy giá trị lớn nhất bên trong ma trận đó.

Và đâu ra cuối cùng do ta cần xem xác suất của 10 lớp ( 0 => 9 ) nên sẽ dung softmax để scale tỉ lệ.

 // Flattern output to 1D vector to pass through fully-connected network

    // Làm phẳng thông qua fully-connected

    model.add(tf.layers.flatten())

    // Dense layer with softmax function with 10 class {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

    const dense = tf.layers.dense({

        units: 10,

        kernelInitializer: 'varianceScaling',

        activation: 'softmax'

    })

    // Add dense layer

    model.add(dense)

Cuối cùng dung hàm crosstenroy để xử lý sai lệch

  const optimizer = tf.train.sgd(LEARNING\_RATE);

    // Generate Model

    model.compile({

        optimizer,

        loss: 'categoricalCrossentropy',

        metrics: ['accuracy'],

    })

Fit model và tiến hành train.Check lost và accuracy để kiểm tra mạng CNN chúng ta chạy có đúng hay không .

  const history = await model.fit(

            trainBatch.images, trainBatch.labels,

            {

                batchSize: BATCH\_SIZE, validationData, epochs: 1,

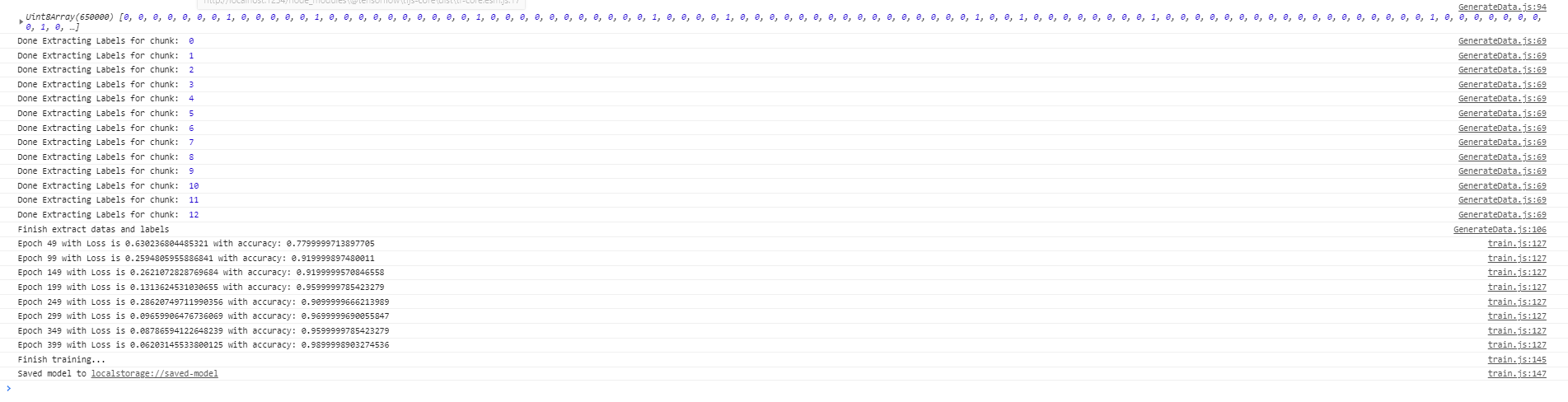
            }

        )

const loss = history.history.loss[0]

        const acc = history.history.acc[0]

## Quy trình training



1. SAVED MODEL

Training trên trình duyệt rất tốn kém, ta phải tìm chỗ để lưu models . Ở đây tụi em chọn LocalStorage của trình duyệt

 console.log('Finish training...');

    const saveResult = await model.save(LINK\_SAVE\_MODEL);

    console.log(`Saved model to ${LINK\_SAVE\_MODEL}`)

Và model học được lưu ở dạng base 64.Sau khi có models dung hàm predict của tensorflow.js

 // Load model

    const loadedModel = await tf.loadModel('localstorage://saved-model');  // Nếu chưa có models phải train

    // Load model là core api từ tensorflow js

    if (loadedModel){

        const output = loadedModel.predict(imgToInference);

        const axis = 1;

        const predictions = Array.from(output.argMax(axis).dataSync());

        const labels = document.getElementsByClassName("number");

        for(let i=0; i< labels.length; i +=1 ) {

            labels[i].style.backgroundColor = "#fff";

        }

        const label = document.getElementById(`${predictions[0]}`);

        label.style.backgroundColor = "#00FF15";

    } else {

        alert('Không tìm thấy models bên trong LocalStorage, vui lòng train models trước khi predict');

    }

UI/UX

Ở phần giao diện chúng em sử dụng canvas để vẽ dữ liệu vào

// Set up canvas

const cv = **new** fabric.Canvas('cv')

cv.isDrawingMode = true;

cv.freeDrawingBrush.width = 20;

cv.freeDrawingBrush.color = "#47494d";

cv.backgroundColor = "#ffffff";

cv.renderAll();

cv.on("mouse:up", () => {

    // Get canvas contents as url

    const scale = (1.) / 10.;

    const url = cv.toDataURL({

        format: 'png',

        multiplier: scale,

    });

    console.log('url', url);

    // FIXME

    const cv28 = document.createElement('canvas');

    cv28.width = 28;

    cv28.height = 28;

    const ctx = cv28.getContext('2d');

    const img = **new** Image;

    img.src = url;

    img.onload = () => {

        ctx.drawImage(img, 0, 0)

        const imgData = ctx.getImageData(0, 0, 28, 28);

        console.log('data', imgData);

        const imgDateLength = imgData.data.length

        // Loop through each pixel in chunk

        const imgArrayData = **new** Float32Array(28 \* 28);

        for(let j = 0; j < imgDateLength / 4; j +=1) {

            // All channel has same value -> only need to read red channel

            const red\_index = j \* 4;

            // Nomarlize pixel value to [0, 1]

            imgArrayData[j] = imgData.data[red\_index] / 255;

        }

        inference(imgArrayData, imgData);

    }

})

Dữ liệu sẽ được đưa vào hàm inferences để tiến hang predict.

# Phần 5 TÀI LIỆU THAM KHẢO

[1] PGS.TS Dương Tuấn Anh, Bài giảng phần Học Máy

[2] Internet : <https://www.easy-tensorflow.com/tf-tutorials/convolutional-neural-nets-cnns>

[3] Deep learning cơ bản : [Bài 6: Convolutional neural network | Deep Learning cơ bản (nttuan8.com)](https://nttuan8.com/bai-6-convolutional-neural-network/)

[4] [[External] Machine learning on Browser and TensorFlow for JavaScript - Google Trang trình bày](https://docs.google.com/presentation/d/1ZVO6Ripu0JL6d-aFTsUuDdlfJVKbBZC3xVupGrl_P3k/edit#slide=id.p1)

[5][Bài 7: Giới thiệu keras và bài toán phân loại ảnh. | Deep Learning cơ bản (nttuan8.com)](https://nttuan8.com/bai-7-gioi-thieu-keras-va-bai-toan-phan-loai-anh/)