Fall 2021

CSC 4301

Final Project

Shape Classifier Using Neural Network

Team Members

Full Name	ID
Othmane Atanane	76576
Yassir Benabdallah	76865

Supervised By:
Dr. Tajjeeddine Rachidi

Contents

Introduction

The aim of this paper is to show how we can implement a neural network that recognizes shapes (squares, circles and triangles). Throughout this paper, we will explain how we built our own data, and how we trained, so that it gives better results.

This project is an illustration of the supervised learning. In fact, we will train models to yield a desired output. This training will include inputs with 100% correct outputs, and eventually this model will be used to predict new shape.

In this project, we used two software, which are both JavaScript libraries that are meant for creative in terms of drawing shapes and designs. The first one is Processing, and the other is P5.

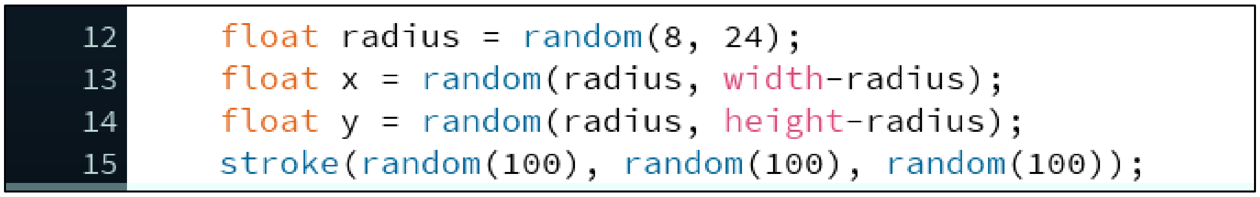
Figure 1.a: p5.js Logo

Figure 1.b: Processing Logo

First, we will explain how we generated our data from scratch. Then, we will see how built a model based on the data we generated before. Finally, we will test our model by hand-written drawings, and see the accuracy of the answer in terms of percentage.

I. Generating Data

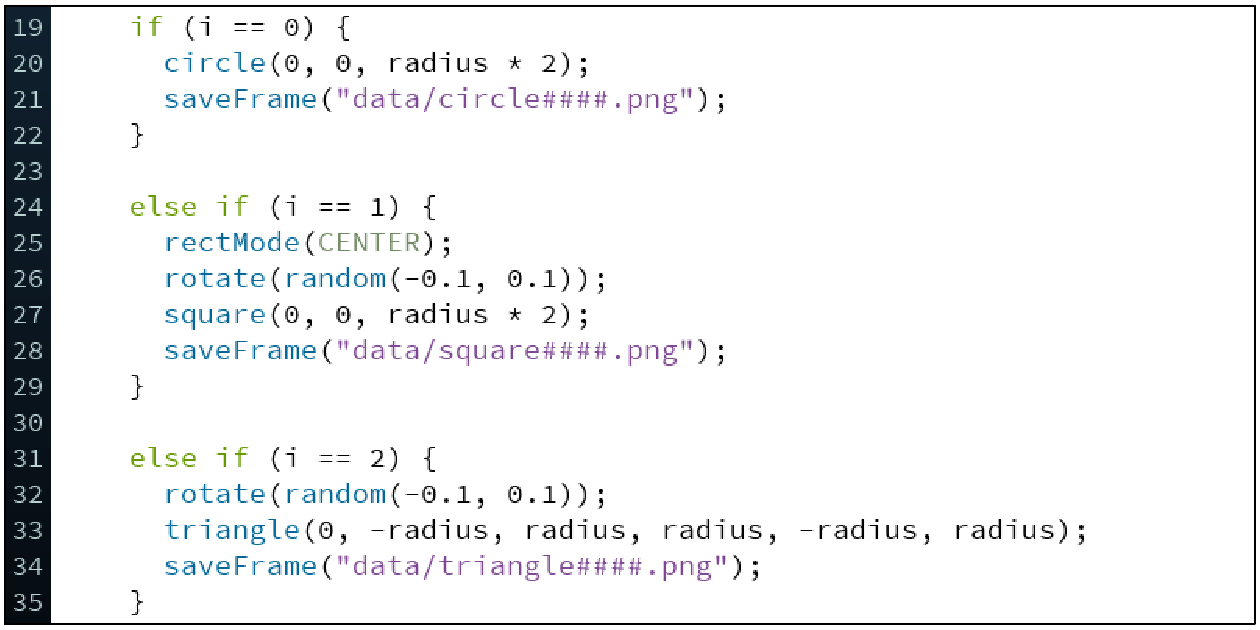
For this purpose, we will use processing. You can find the file under the name "Final_Project_Report.pde". However, we will focus here on important lines of code.



```
    float radius = random(8, 24);
    float x = random(radius, width-radius);
    float y = random(radius, height-radius);
    stroke(random(100), random(100), random(100));
```

Figure 2: Placing Shapes on the Pictures Randomly

These lines of codes would allow us to randomly place the shape in a certain picture. For example, a circle in one picture could be in the middle, while in another it could be on top. Also, each shape would have different size to ensure that our training data include the maximum possible cases.



```
    if (i == 0) {
      circle(0, 0, radius * 2);
      saveFrame("data/circle####.png");
    }

    else if (i == 1) {
      rectMode(CENTER);
      rotate(random(-0.1, 0.1));
      square(0, 0, radius * 2);
      saveFrame("data/square####.png");
    }

    else if (i == 2) {
      rotate(random(-0.1, 0.1));
      triangle(0, -radius, radius, radius, -radius, radius);
      saveFrame("data/triangle####.png");
    }
```

Figure 3: Saving Shapes

Now we can save our shapes on the same file this Processing is saved. The syntax `saveFrame("data/circle####.png")` takes care of this. We will generate 100 pictures for each shape. Therefore, the total result of all shapes would be 300 pictures.

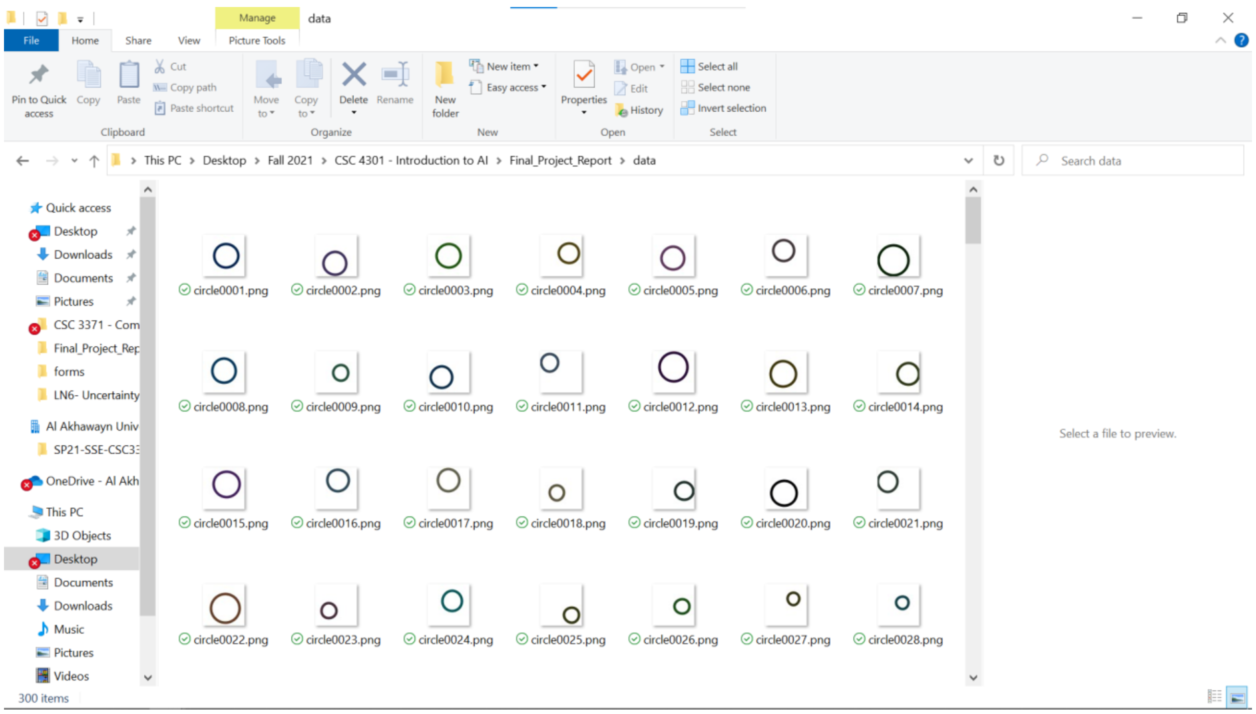


Figure 4: Data Generation

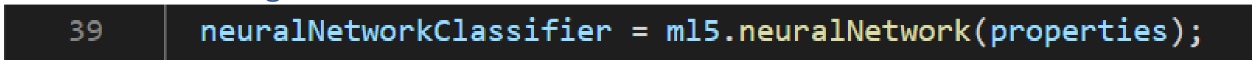
After running our code, we can find that all these pictures have been generated. They are indeed 300 items: 100 circles, 100 square and 100 triangles.

II. Creating a Neural Network

For this purpose, we will use JavaScript, especially the p5.js library. We do not need to work on p5.js web editor for the moment because our data (300 items) should be uploaded on the website, which does not look very optimal. Therefore, we are using Visual Studio Code as a text editor and attach the data to it locally since both of them are located in the same file.

In addition to the JavaScript file, we are using an html file, that would allow us to see the process of generating model on the browser after running the js file on a local host. Note that this index.html is very important because it holds the ml5.js library, which is a machine learning library that is supposed to generate our neural network model.

1. Declaring the Neural Network

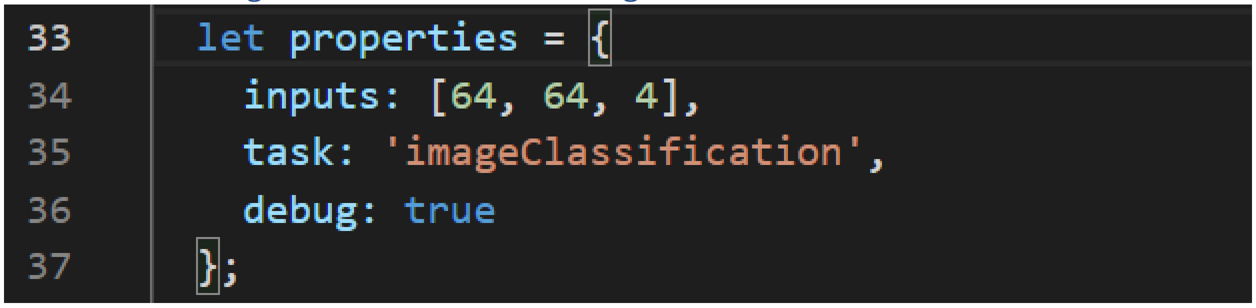


```
neuralNetworkClassifier = ml5.neuralNetwork(properties);
```

Figure 5: Declaring a Neural Network

In order to create a neural work, we just call the ml5.neuralNetwrok, which one parameter as input that is properties.

2. The Arguments Passed with Calling the Neural Network



```
let properties = {
  inputs: [64, 64, 4],
  task: 'imageClassification',
  debug: true
};
```

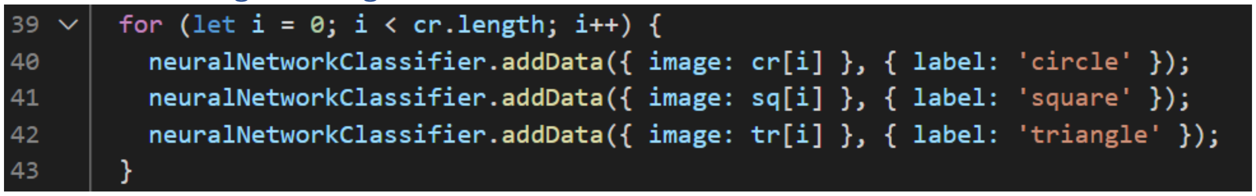
Figure 6: Declaring the Properties Passed to the Neural Network

These properties are a set of three options: inputs, task and debug.

- **Inputs:** this is the most important property because it indicates the shape of the input data. The word shape refers to the dimensions of the image in terms of width and height. For our case, our dataset is 64x64 images, as indicated in figure 6. The last digit, 4, is called the channel of the image or the digital image. This means that each pixel is made of a combination of three colors (green, blue and red) in addition to an alpha number that stores the transparency information. The sum, therefore, is 4.
- **Task:** this property refers to the task the neural network will perform. This task could be simple classification, regression or image classification.
- **Debug:** this one determines whether we want to show a training visualization when running our code. Since it is set to true, then we will see it.

Note that there more properties that can be associated with the properties of any neural network. However, for the sake of simplicity, those are enough to build our project.

3. Adding Training Data

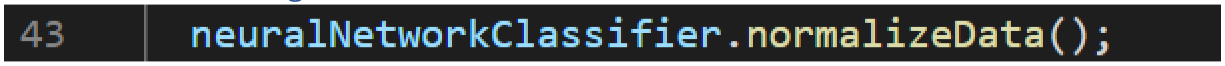


```
for (let i = 0; i < cr.length; i++) {
  neuralNetworkClassifier.addData({ image: cr[i] }, { label: 'circle' });
  neuralNetworkClassifier.addData({ image: sq[i] }, { label: 'square' });
  neuralNetworkClassifier.addData({ image: tr[i] }, { label: 'triangle' });
}
```

Figure 7: Adding Training Data

Again, the ml5 library takes care of adding the data to the neural network thanks to the function `addData()`. The training dataset should include the input associated with its target, and then each image is associated with its label.

4. Normalizing the Data



```
neuralNetworkClassifier.normalizeData();
```

Figure 8: Data Normalization

This step is very important in building our neural network. Normalization is the process of analyzing data in a way that unifies the ranges of each pixel to be (0,1). For example, if the range of a certain pixel range from 0 to 1, and the values of another one range from 100 to 1,000, normalization would create new values that maintain a general distribution in the input data, which is usually the range (0,1).

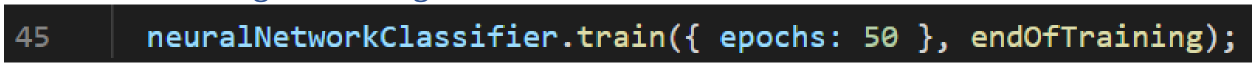
There are many techniques to normalize data, such as MinMax. The values in each pixel are normalized in the range 0-1 according to the following formula:

$$\boldsymbol{z = \frac{x - min\,(x)}{[max(x) - min\,(x)]}}$$

where min(x) is the minimum value of the range in which x falls (same thing goes for max(x)).

Luckily, there is a pre-built function *`normalizeData()`* in p5 that normalizes data, so do not have to worry about its implementation.

5. Training and Saving the Model

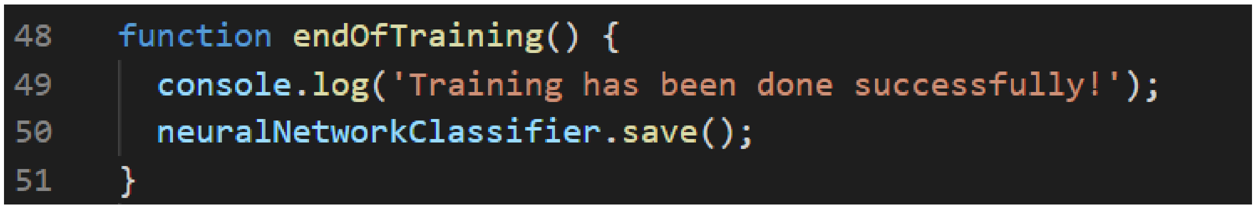


```
neuralNetworkClassifier.train({ epochs: 50 }, endOfTraining);
```

Figure 9: Model Training

The function `train()` takes care of training the model with the data loaded with the function `addData()`. This function takes two arguments: epochs and a call back that prints that the training is done.

- **Epochs**: refers to how much we train the neural network with the training data for one cycle. In our case, we have epochs = 50. In other words, we will send all the data in the neural network 50 times.
- **Call back Function:** besides notifying us that the training is done, it also saves the model as shown in figure 10.

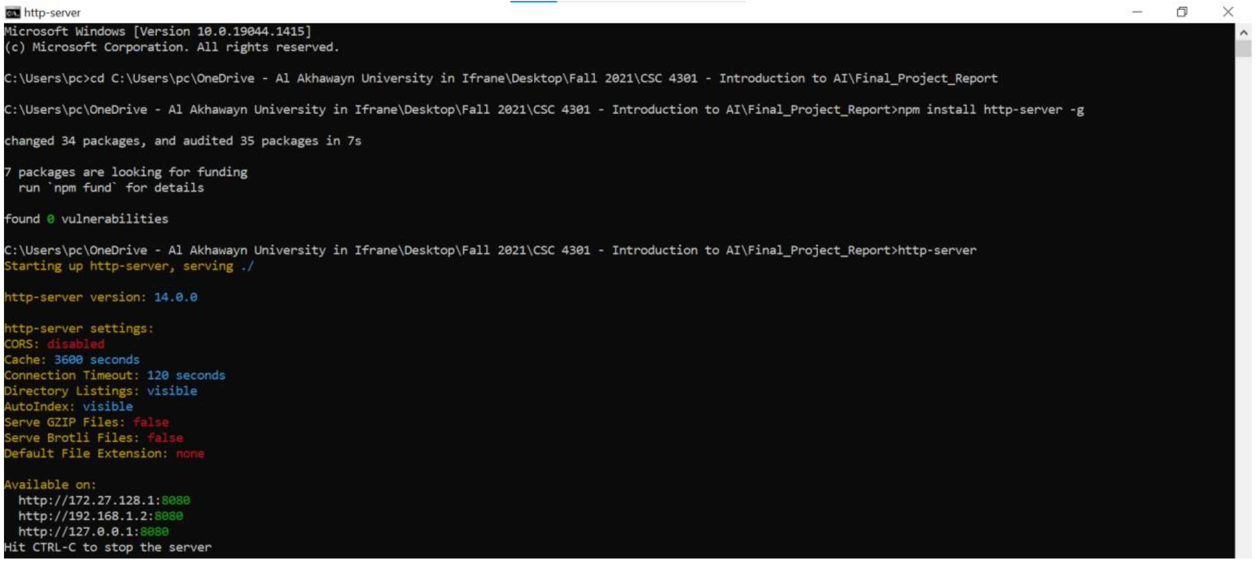


```javascript
function endOfTraining() {
  console.log('Training has been done successfully!');
  neuralNetworkClassifier.save();
}
```

Figure 10: Saving the Model

III. Running the Neural Network

1. Creating a Local Server

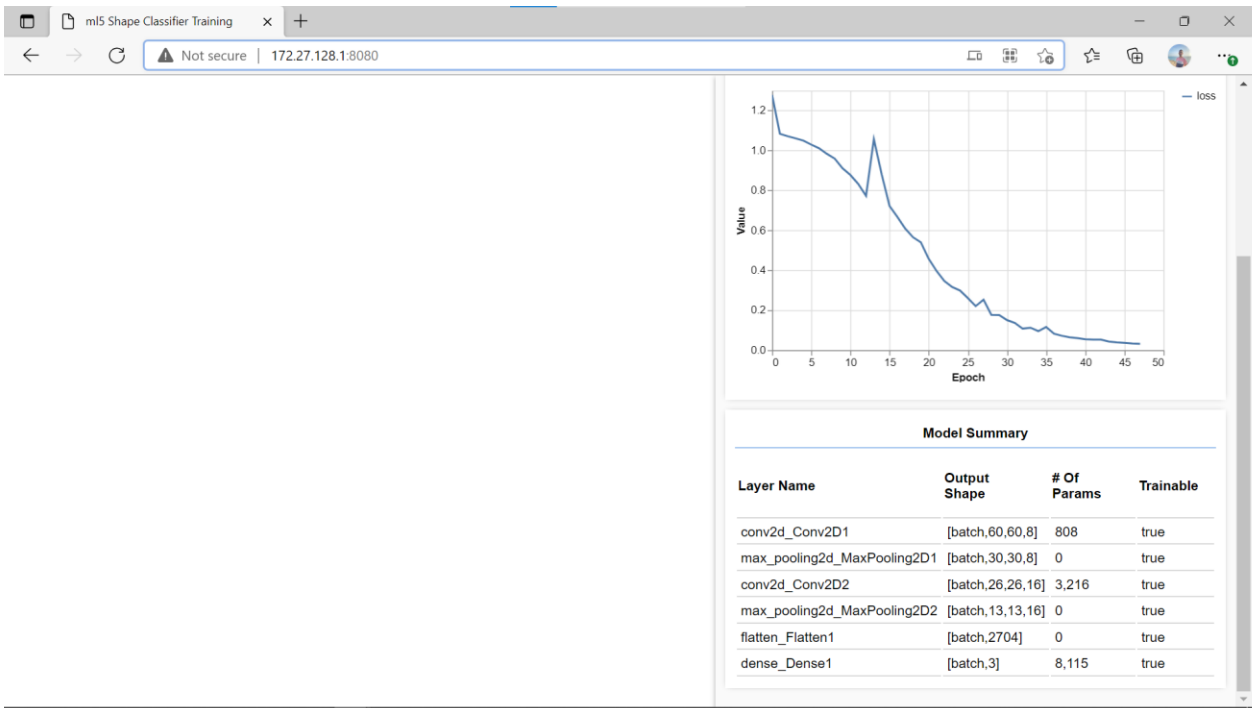


```
http-server
Microsoft Windows [Version 10.0.19044.1415]
(c) Microsoft Corporation. All rights reserved.

C:\Users\pc>cd C:\Users\pc\OneDrive - Al Akhawayn University in Ifrane\Desktop\Fall 2021\CSC 4301 - Introduction to AI\Final_Project_Report

C:\Users\pc\OneDrive - Al Akhawayn University in Ifrane\Desktop\Fall 2021\CSC 4301 - Introduction to AI\Final_Project_Report>npm install http-server -g

changed 34 packages, and audited 35 packages in 7s

7 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

C:\Users\pc\OneDrive - Al Akhawayn University in Ifrane\Desktop\Fall 2021\CSC 4301 - Introduction to AI\Final_Project_Report>http-server
Starting up http-server, serving ./

http-server version: 14.0.0

http-server settings:
CORS: disabled
Cache: 3600 seconds
Connection Timeout: 120 seconds
Directory Listings: visible
AutoIndex: visible
Serve GZIP Files: false
Serve Brotli Files: false
Default File Extension: none

Available on:
  http://172.27.128.1:8080
  http://192.168.1.2:8080
  http://127.0.0.1:8080
Hit CTRL-C to stop the server
```

Figure 11: Creating a Local Host

In order to run the js file on the browser, we need to run a local web server. Since we are implementing our code in JavaScript, we will use NodeJS. As we can see in the bottom of figure 11, we can access our local website through http://172.27.128.1:8080

2. Local Website



Model Summary

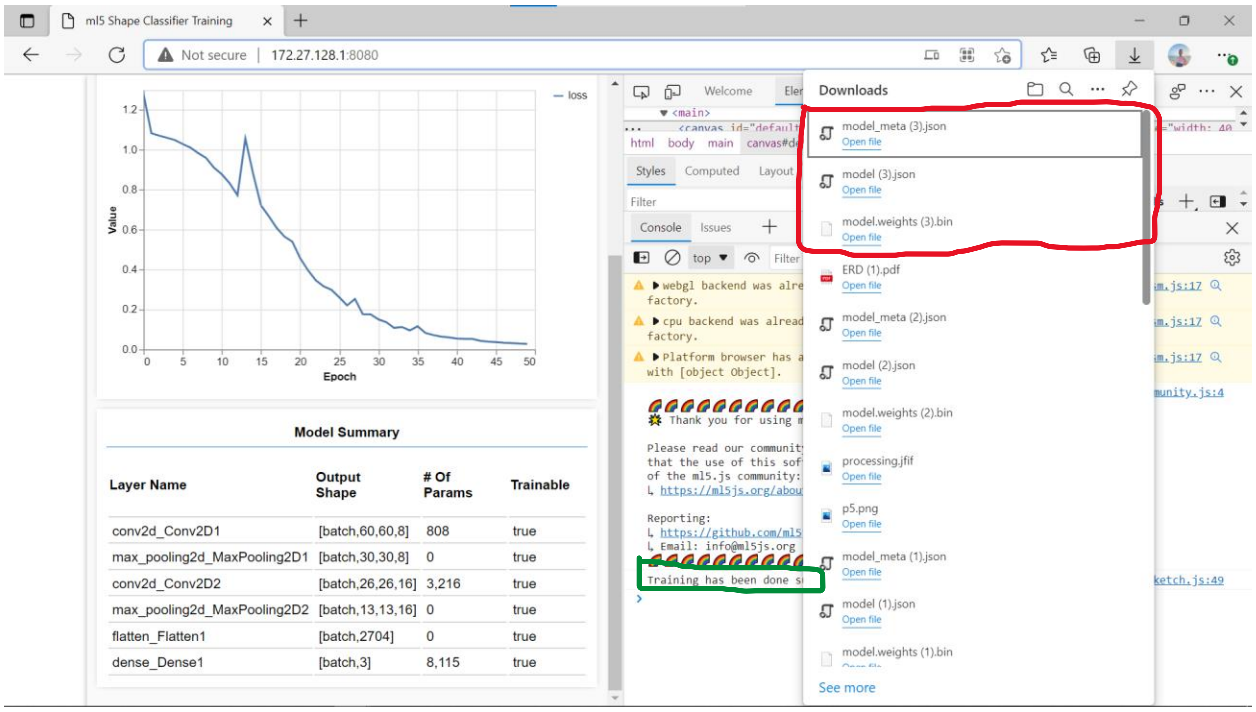
Layer Name	Output Shape	# Of Params	Trainable
conv2d_Conv2D1	[batch,60,60,8]	808	true
max_pooling2d_MaxPooling2D1	[batch,30,30,8]	0	true
conv2d_Conv2D2	[batch,26,26,16]	3,216	true
max_pooling2d_MaxPooling2D2	[batch,13,13,16]	0	true
flatten_Flatten1	[batch,2704]	0	true
dense_Dense1	[batch,3]	8,115	true

Figure 12: Local Website

The local website would a table that summarized the configuration of our neural network, while the graph how 50 epochs are getting a loss. In other words, a measure of the error during the process is being performed, meaning how close is the model is matching the training data. Since the graph is converging to 0, then our model is pretty well trained.

Actually, we can run our code many times so that we can get better results. We run the code four times, so we will keep the fourth model.

The red hand-written rectangle indicates the model, while the green one indicates that the training is done. Both of those features are provided by `endOfTraining()` method in our code.



Model Summary

Layer Name	Output Shape	# Of Params	Trainable
conv2d_Conv2D1	[batch,60,60,8]	808	true
max_pooling2d_MaxPooling2D1	[batch,30,30,8]	0	true
conv2d_Conv2D2	[batch,26,26,16]	3,216	true
max_pooling2d_MaxPooling2D2	[batch,13,13,16]	0	true
flatten_Flatten1	[batch,2704]	0	true
dense_Dense1	[batch,3]	8,115	true

```
▶webgl backend was alre
factory.
▶cpu backend was alread
factory.
▶Platform browser has a
with [object Object].

 Thank you for using m

Please read our communit
that the use of this sof
of the ml5.js community:
↳ https://ml5js.org/abou

Reporting:
↳ https://github.com/ml5
↳ Email: info@ml5js.org

Training has been done s
```

model_meta (3).json — Open file
model (3).json — Open file
model.weights (3).bin — Open file
ERD (1).pdf — Open file
model_meta (2).json — Open file
model (2).json — Open file
model.weights (2).bin — Open file
processing.jfif — Open file
p5.png — Open file
model_meta (1).json — Open file
model (1).json — Open file
model.weights (1).bin

Figure 13: Post-Training Results

IV. Loading the Model

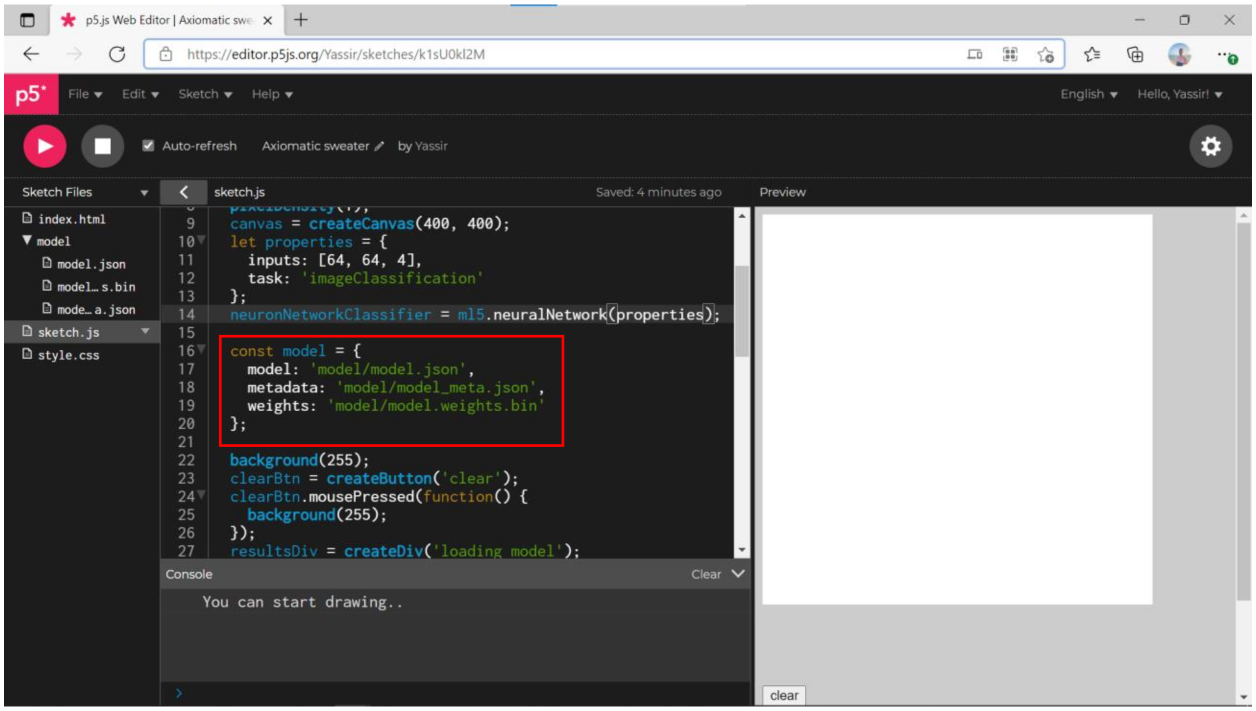
In this phase, we are using a p5.js web editor. This is the last step where we will code a board in which we will draw shapes, and the system will predict the shape. You can find its code under

A pre-built function `load()` will take care of loading the model we trained. We need to upload the pre-trained model into the web editor as a folder, as shown in figure 14.

We then add some lines of code that would allow us to draw on the board at the right of the screen.

Figures 15 show the results of drawing shapes and the prediction of the shape along with a percentage that indicates to what degree it is similar to the original shape.

Note that the percentage is not very accurate. The more we generate models, the higher the accuracy of this percentage.

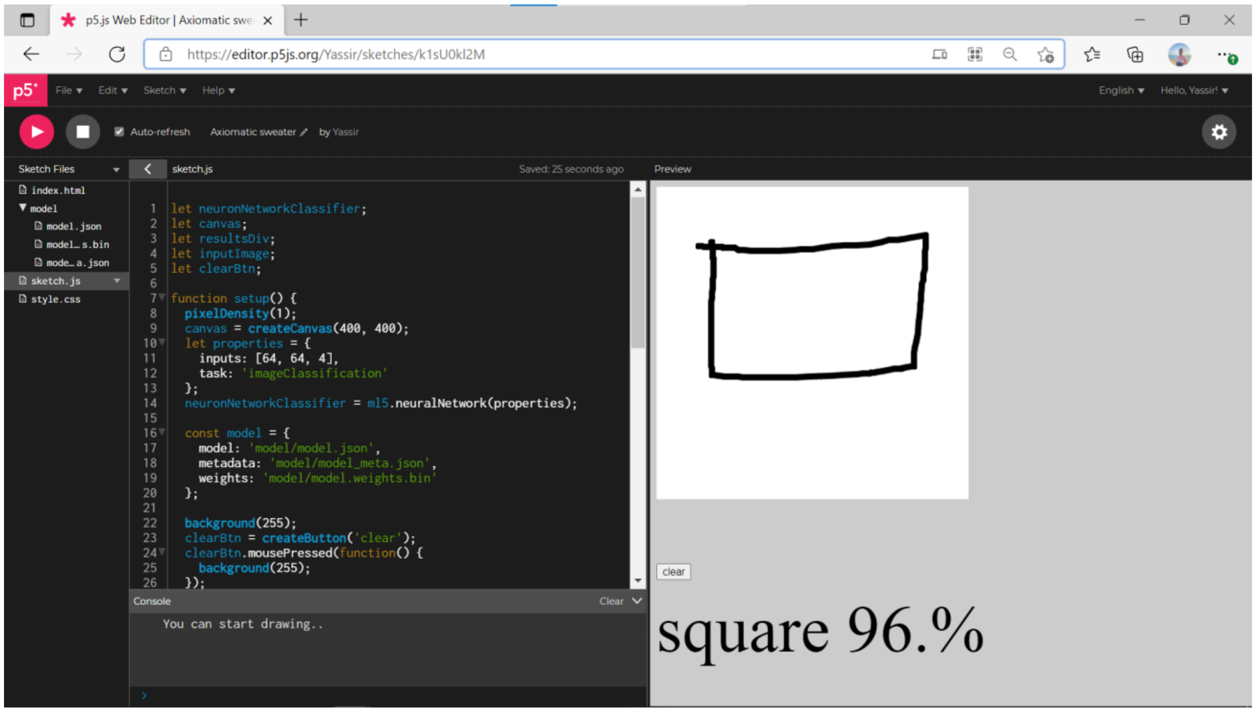


```
  canvas = createCanvas(400, 400);
  let properties = {
    inputs: [64, 64, 4],
    task: 'imageClassification'
  };
  neuronNetworkClassifier = ml5.neuralNetwork(properties);

  const model = {
    model: 'model/model.json',
    metadata: 'model/model_meta.json',
    weights: 'model/model.weights.bin'
  };

  background(255);
  clearBtn = createButton('clear');
  clearBtn.mousePressed(function() {
    background(255);
  });
  resultsDiv = createDiv('loading model');
```

Console

```
You can start drawing..
```

Figure 14: Loading the Pre-Trained Model on the Web Editor



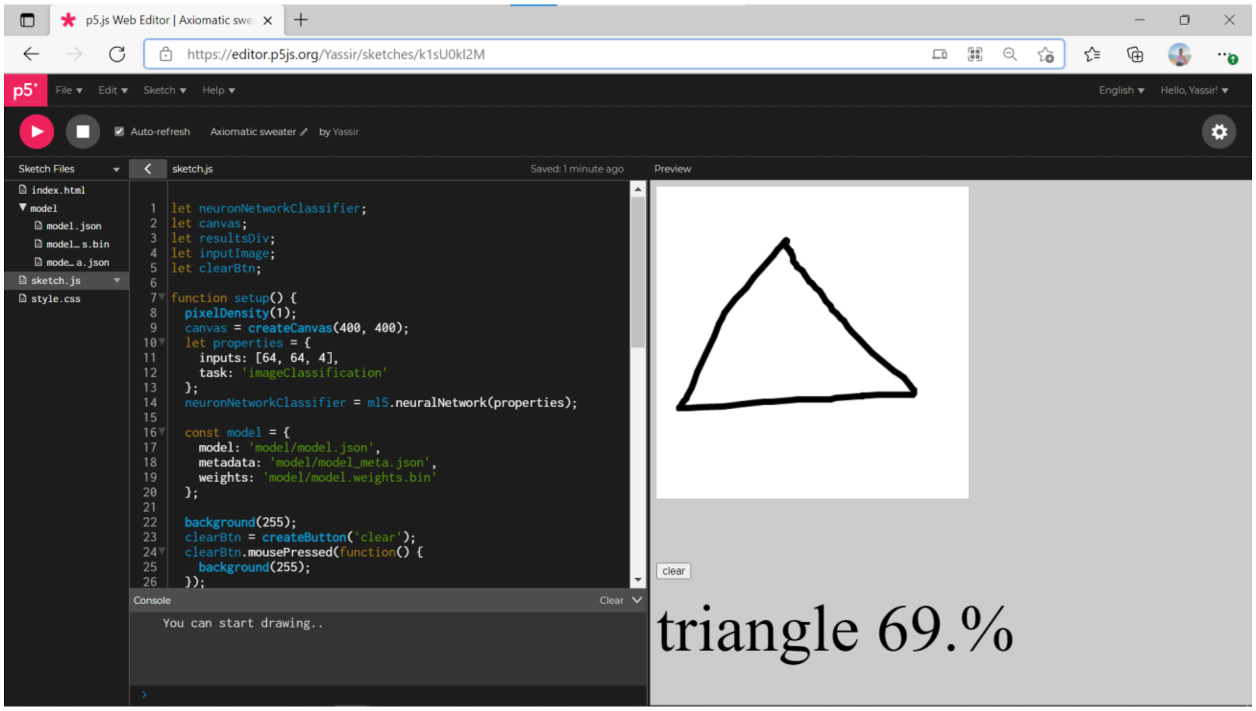
```
let neuronNetworkClassifier;
let canvas;
let resultsDiv;
let inputImage;
let clearBtn;

function setup() {
  pixelDensity(1);
  canvas = createCanvas(400, 400);
  let properties = {
    inputs: [64, 64, 4],
    task: 'imageClassification'
  };
  neuronNetworkClassifier = ml5.neuralNetwork(properties);

  const model = {
    model: 'model/model.json',
    metadata: 'model/model_meta.json',
    weights: 'model/model.weights.bin'
  };

  background(255);
  clearBtn = createButton('clear');
  clearBtn.mousePressed(function() {
    background(255);
  });
```

Console

```
You can start drawing..
```

```
square 96.%
```

Figure 15.a: Square Shape

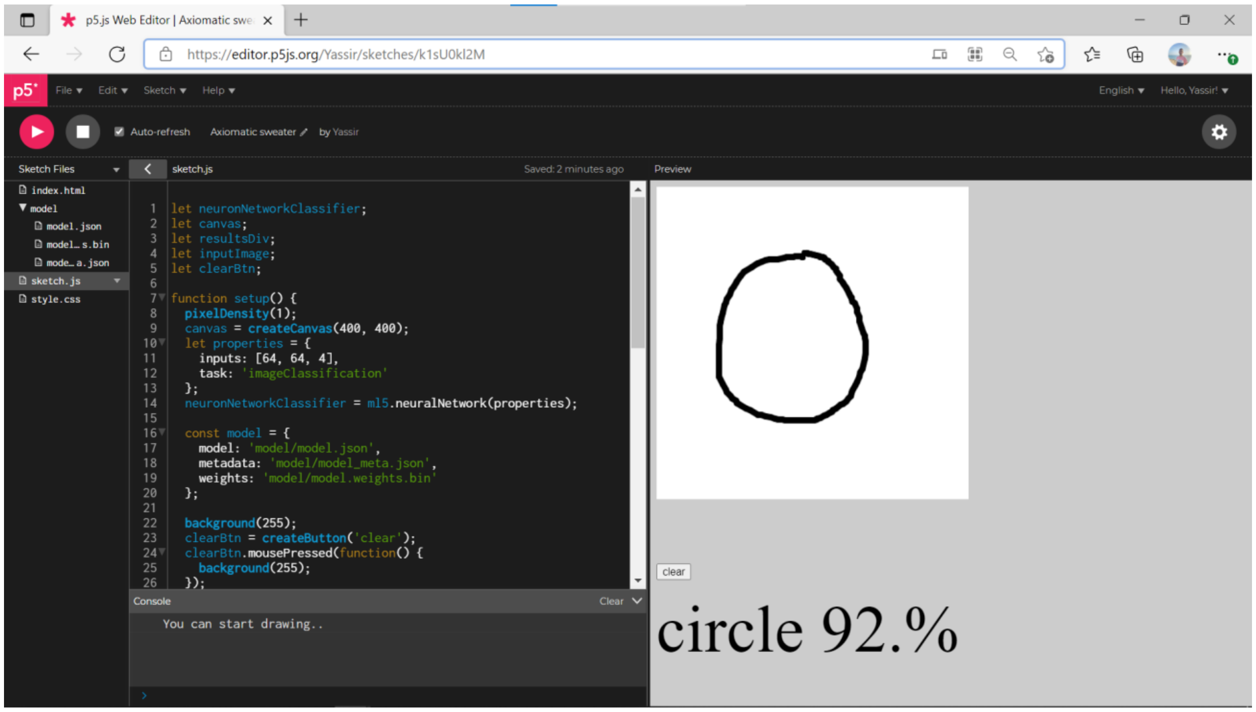


```
let neuronNetworkClassifier;
let canvas;
let resultsDiv;
let inputImage;
let clearBtn;

function setup() {
  pixelDensity(1);
  canvas = createCanvas(400, 400);
  let properties = {
    inputs: [64, 64, 4],
    task: 'imageClassification'
  };
  neuronNetworkClassifier = ml5.neuralNetwork(properties);

  const model = {
    model: 'model/model.json',
    metadata: 'model/model_meta.json',
    weights: 'model/model.weights.bin'
  };

  background(255);
  clearBtn = createButton('clear');
  clearBtn.mousePressed(function() {
    background(255);
  });
```

```
You can start drawing..
```

triangle 69.%

Figure 15.a: Triangle Shape



```
let neuronNetworkClassifier;
let canvas;
let resultsDiv;
let inputImage;
let clearBtn;

function setup() {
  pixelDensity(1);
  canvas = createCanvas(400, 400);
  let properties = {
    inputs: [64, 64, 4],
    task: 'imageClassification'
  };
  neuronNetworkClassifier = ml5.neuralNetwork(properties);

  const model = {
    model: 'model/model.json',
    metadata: 'model/model_meta.json',
    weights: 'model/model.weights.bin'
  };

  background(255);
  clearBtn = createButton('clear');
  clearBtn.mousePressed(function() {
    background(255);
  });
```

```
You can start drawing..
```

circle 92.%

Figure 15.a: Circle Shape

References

- Helpers ✨ - NeuralNetwork - 《ml5.js - Machine Learning for Web》 - 书栈网 · BookStack
- What is Supervised Learning? | IBM
- website/CodingChallenges/CC_158_Shape_Classifier at main · CodingTrain/website (github.com)
- Channel (digital image) - Wikipedia