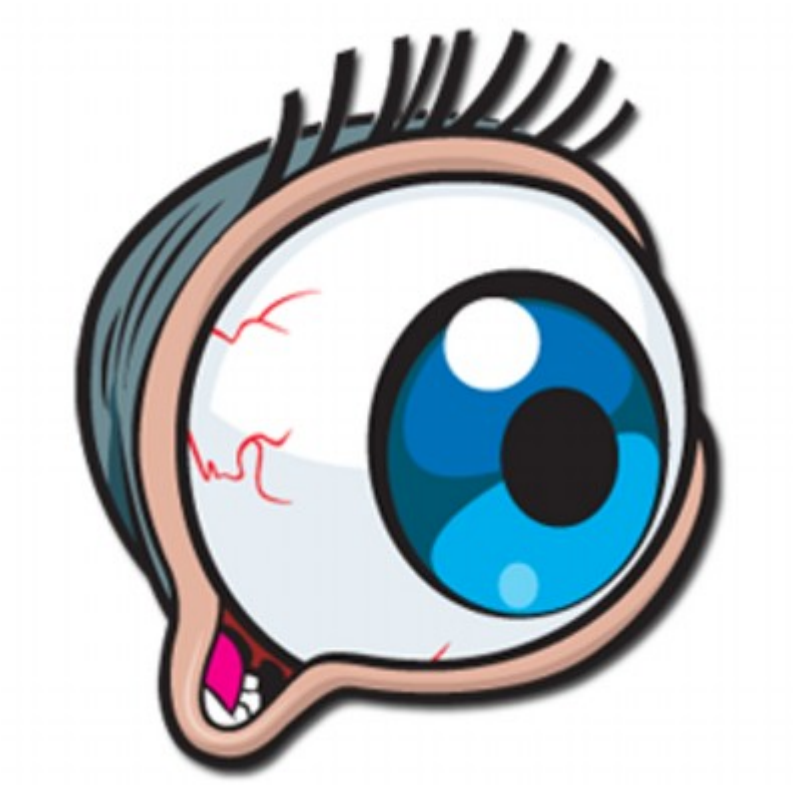
Say what you see
Facial Recognition Project



Joshua John-Baptiste

Cazoo

Content

Model Design

Teachable machine was the platform I used to complete this project. Teachable machine is web-based tool from Google to facilitate the creation of machine learning models. It has the capability to use your web camera, microphone, any uploaded audio or image files to train a machine learning model. Teachable machine makes it easier for everyone to have access and create machine models because you don't need to have programming knowledge to use it. The process of creating a new model is straightforward:

- Gather data
- Train the model
- Test the model
- Fine-tune the model

I choose the following characteristics that have the following classifications:

- Age: young and old
- Gender: Male and female
- Glasses: Wearing glasses/ sunglassess and not wearing glasses/ sunglassess

I choose two classification for each characteristic to keep things simple, improving the accuracy of the model. Furthermore, in the social climate of today, a lot of our characteristics can be subjective. A great example of this is gender. By limiting the classifications, we don't need to think too much about it. In additional, implementing subjective classifications could overcomplicated our model compromising its accuracy.

I used 15 datapoints for each classification to validate the model's accuracy and under 90 datapoints to train and test the model. This give me more time and flexibility to experiment with the hyper-parameters of the model. For the gender characteristic, I used 88 datapoints split evenly between the male and female classifications. This number felt right because the dataset was diverse and the classifications were simple. I felt the performance of the model would still be good without adding more datasets. This was also the case for the glasses and age characteristic too, but instead of 88 I used 90 datapoints to train and test this model. More so than gender, age is a tricky thing to classify because its so diverse, subjective and tricky to distinguish between someone old and young.
All the data I used to train, test and validate my model came from Kaggle, where the data was used to train similar or identical models. However, I personally selected every image in the each dataset for quality assurance, avoiding to protect the model's accuracy. I also made sure they every image in the dataset was a jpeg file, so they're numerically viewed identically. I made sure every classification was balanced and scanned for any duplicates, which could cause a bias to one of the characteristics. I didn't personally split my dataset into training and testing buckets because Teachable machine does that for me. The small sample size I worked with can be seen as an issue but I believe the dynamic dataset I used and the validation will offset this.

Model Performance

Glasses

A confusion matrix is a table detailing all the possible outcomes of a model to evaluate its performance. Where a model has two classifications, one classification is set as positive and the other is set as a negative. The four possible combinations based on if the prediction was true or false is then documented. The confusion matrix shown in figure 1.2 is the confusion matrix provided by Google's teachable machine. As mentioned previously, teachable machine sorts out the split between the training and testing data, so I cant speak on what test data was used in this confusion matrix.

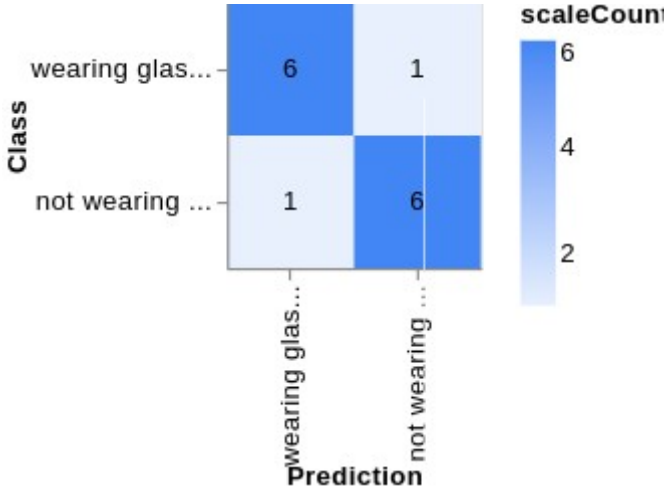


Figure 1.1: Confusion matrix provided by the teachable machine

Recall or sensitivity is the measure of your true positive results over the count of the actual positive data. The formula for this measurement can be expressed as TP/(TP + FN). For the confusion matrix in figure 1.1, the calculation would be 6/7, which equals 0.857. This means the model got a true prediction correct over 80% of the time.

Precision is the measurement of true positives against the number of positives predicted by the model. This calculation will identify the rate correct positive predictions are made by the model. The formula for this calculation is TP/(TP+FP). In this situation, the confusion matrix is 6/ 7, which again equals 0.857. This means my model makes correct predictions 85.7% of the time.

Specificity is the performance metric used to evaluate the models ability to correctly identify the true negative instances. This is calculated as TN/(TN +FP). In this situation, the calculation is 6/7, which, again equals 0.857.

This sample size is small, so I also replicated the calculations on my manual validations confusion matrix, show in figure 1.2. The breakdown of those results follow:

- Recall- 13/15 = 0.86 or 86%
- Precision- 13/ 14 = 0.93 or 93%
- Specificity- 14/15 = 0.93 or 93%

In hindsight, the precision and specificity are too accurate which is concerning when considering the sample size. I also attached some additional stats, shown in figure 1.3.

Validation			
TARGET / OUTPUT	wearing glasses	Not wearing glasses or sunglasses	SUM
wearing glasses	13 43.33%	1 3.33%	14 92.86% 7.14%
ng glasses or su	2 6.67%	14 46.67%	16 87.50% 12.50%
SUM	15 86.67% 13.33%	15 93.33% 6.67%	27 / 30 90.00% 10.00%

Figure 1.2: Confusion matrix for my personal manual validation test

Class Name	Precision	1-Precision	Recall	1-Recall	f1-score
wearing glasses	0.929	0.071	0.867	0.133	0.897
Not wearing glasses or sunglasses	0.875	0.125	0.933	0.067	0.903
Accuracy	0.900				
Misclassification Rate	0.100				
Macro-F1	0.900				
Weighted-F1	0.900				

Figure 1.3: The confusion matrix validations stats for the glasses characteristic

Finally, I combined the test and validation confusion matrixs too, which produced the outlook shown in figure 1.4 and 1.5.

Validation			
TARGET / OUTPUT	wearing glasses	Not wearing glasses or sunglasses	SUM
wearing glasses	19 43%	2 5%	21 90% 10%
ng glasses or su	3 7%	20 45%	23 87% 13%
SUM	22 86% 14%	22 91% 9%	39 / 44 89% 11%

Figure 1.4: Confusion matrix for the test and validation dataset

Class Name	Precision	1-Precision	Recall	1-Recall	f1-score
wearing glasses	0.905	0.095	0.864	0.136	0.884
Not wearing glasses or sunglasses	0.870	0.130	0.909	0.091	0.889
Accuracy	0.886				
Misclassification Rate	0.114				
Macro-F1	0.886				
Weighted-F1	0.886				

Figure 1.5: Stats for the confusion matrix shown in figure 1.4

Although my model is really accurate, I did find edge cases and outliers that can exploit the model, shown in figure 1.6. The woman in the image is not wearing glasses or sunglasses but the model is classifying her as wearing glasses or sunglasses. The cause of this outliers is likely the artistic marking around her eye. When comparing someone wearing glasses/ sunglasses and not wearing glasses or sunglass, the model may be looking for anything around eye area, rather than glasses specifically.

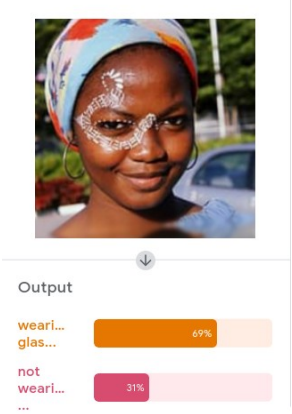


Figure 1.6: The model detecting someone with painting on their face as wearing glasses {FP)

Gender:

Figure 1.7 is the confusion matrix for the gender model I created provided by Teachable machine and figure 1.8 are the stats for that confusion matrix.

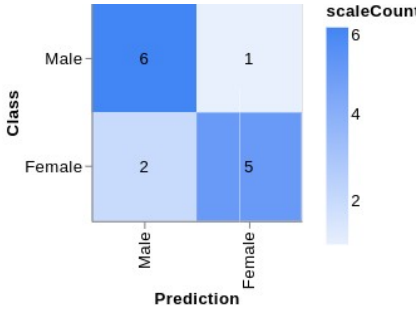


Figure 1.7: The confusion matrix for the gender model

Class Name	Precision	1-Precision	Recall	1-Recall	f1-score
Male	0.857	0.143	0.750	0.250	0.800
Female	0.714	0.286	0.833	0.167	0.769
Accuracy	0.786				
Misclassification Rate	0.214				
Macro-F1	0.785				
Weighted-F1	0.787				

Figure 1.8: Calculations and stats for the Google teachable machine

I also attached my confusion matrix from the validation tests and its stats.

Validation			
TARGET / OUTPUT	Male	Female	SUM
Male	11 37%	4 13%	15 73% 27%
Female	4 13%	11 37%	15 73% 27%
SUM	15 73% 27%	15 73% 27%	22 / 30 73% 27%

Figure 1.9: Confusion matrix from my validation test

Class Name	Precision	1-Precision	Recall	1-Recall	f1-score
Male	0.733	0.267	0.733	0.267	0.733
Female	0.733	0.267	0.733	0.267	0.733
Accuracy	0.733				
Misclassification Rate	0.267				
Macro-F1	0.733				
Weighted-F1	0.733				

Figure 1.10: The stats from my validation confusion matrix (figure 1.9)

Compared to the glasses performance, this models more balanced with give me confidence in its reliability. The model's precision and accuracy are at a threshold that I am comfortable accepting.

There were some outliers and edge cases I found that could impact the results. If the models doesn't receive an image of the entire head and receive a image of the face, it can easily be mistaken for the opposite gender, as seen in figure 1.11. In this case, the facial structure of the man in the image must be similar to what the model believes to be a woman. On the other hand, in society today, gender is subjective and can come down to our perception. Furthermore, this could be a bias that could be exploited with transgender people.

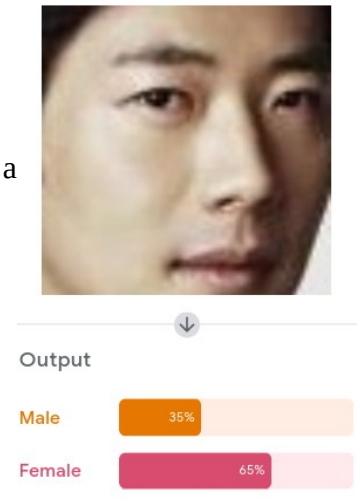


Figure 1.11: gender model mistaking a man as a woman (FN)

Age:

Figure 1.12 is the confusion matrix for the age model and figure 1.13 are some stats for that confusion matrix.

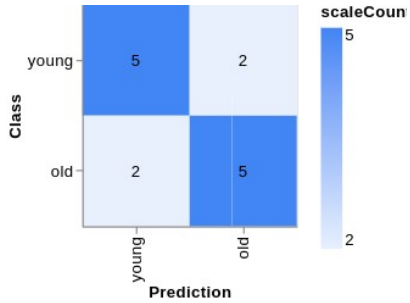


Figure 1.12: Teachable machine confusion matrix for the age model

Class Name	Precision	1-Precision	Recall	1-Recall	f1-score
young	0.714	0.286	0.714	0.286	0.714
old	0.714	0.286	0.714	0.286	0.714
Accuracy	0.714				
Misclassification Rate	0.286				
Macro-F1	0.714				
Weighted-F1	0.714				

Figure 1.13: Confusion matrix for the age model

Figure 1.14 is the confusion matrix for the validation of the age model and figure 1.15 are the validation's stats.

Validation			
TARGET / OUTPUT	young	old	SUM
young	12 40%	3 10%	15 80% 20%
old	3 10%	12 40%	15 80% 20%
SUM	15 80% 20%	15 80% 20%	24 / 30 80% 20%

Figure 1.14: The Confusion matrix of the age model

Class Name	Precision	1-Precision	Recall	1-Recall	f1-score
young	0.800	0.200	0.800	0.200	0.800
old	0.800	0.200	0.800	0.200	0.800
Accuracy	0.800				
Misclassification Rate	0.200				
Macro-F1	0.800				
Weighted-F1	0.800				

Figure 1.15: The stats of the age confusion matrix

Although, its based on limited data, the precision, accuracy and recall metrics in figure 1.13 are promising and meet my desired threshold. The same metrics in figure 1.15 are based on more data but are slightly out of the threshold. Although, I believe the model should perform to a good standard.

The age model is a complex model because the lines between the two classifiers, young and old are blurred. Distinguishing between the two classifier gets more difficult when meeting in the middle in terms of age. This can lead to outliers, like the one seen in figure 1.16. The man in figure 1.16 is labelled as old but incorrectly identified as young. In reality, some people will see him as middle aged or old, but again, its very subjective. If its subjective for us humans, it will also be difficult for our model to decide too. This edge case exist because the man can be seen as a middle aged man, and as such has aspects of him that are young.

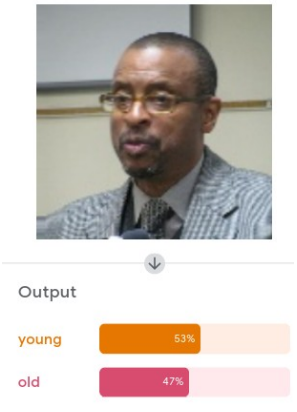


Figure 1.16: FP result form the age model

Empirical Evaluation

My agent and model are designed to work in tangent to classify:

- gender
- age
- wearing glasses or sunglasses

After assessing my agent and models, I can't say it can successfully classify my target characteristics because it lacked consistency.

Figure 2.1 is a screenshot proving the success of my model and agent. The Age and glasses model are fairly certain that the input from the webcam is a TP and correctly identifying my characteristics. However, the gender model is not certain of my gender based on the image from the webcam from the agent, but still manages to class me correctly.

Figure 2.1: A screenshot of my agent detecting my webcam and my model trying to classify my characteristics

However, the gender model classed me as a female and continued switching between male and female classifications, figure 2.2. The other model continued to be consistent in their classification.

Figure 2.2: A screenshot of my agent and the FN for my gender classification

It’s possible that my cap could be making it difficult for the model to correctly identify me, so I decided to take it off for further analysis.

Figure 2.3: the agent classifying me without my cap or glasses

Figure 2.4: Agent classifying me without a cap incorrectly, this time with two FN

As you can interpret from figure 2.3 and figure 2.4, the gender model continued to classify my gender inconsistently. The glasses model also showed some inconsistency in figure 2.4. To collect more data, I asked my mum to let my agent classify her too.

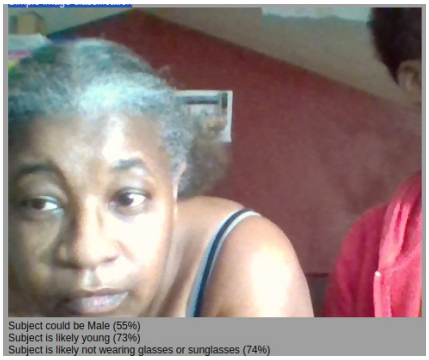


Figure 2.5: my agent incorrectly classifying my mum!

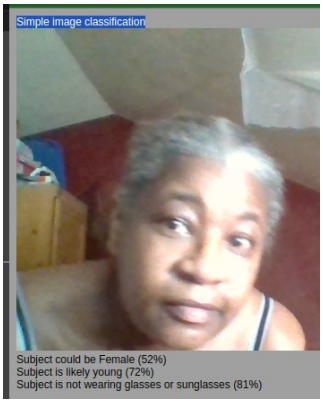


Figure 2.6

As seen in figure 2.5 and 2.6, the gender model is still inconsistent with my mum, producing some FPs. The sample size used to test the model could be the cause of the uncertainty displayed by the model. Although it can be seen as subjective, the age model also failed to correctly classify my mum, who can be considered middle aged/ old.

Overall, the evidence points to my models being able to classify people based on characteristics, just not consistently. The agent however, is working as designed because its detecting its environment via the webcam and producing an output based on the logic provided by the Teachable machine models.

CNN- Convolutional Neural Network:

Artificial neural network (ANN) is a subfield of machine learning designed to make predictions based on input data. Similar to supervised learning, the model is trained on labelled data before a production model can be used to make predictions. However, the architecture used to process data and learn differs from supervised learning. With ANN, its architecture mirrors a brain, using the concept of a network of interconnected artificial neurons to process data.

The architecture consists of 3 components:

- Input layer- This takes an input and passes it along to the hidden layer. If the consumed data is an image (which would be the case in a convolutional neural network), it will convert the image into pixels and feed it into a neuron in the input layer. Each neuron in the input layer will be interconnected to a neuron in the hidden layer and will pass the data there for it to be processed
- Hidden layer- ANN has at least one layer hidden layer, with the potential to have more. The data is received from the input layer by neurons and processed here. Each neuron in this layer will pass on the data to neurons interconnected in another hidden layer, if one exist, or the output layer. However, the data will only be moved if the neuron's computed value, based on its received inputs, (more on this below) meets or is greater than a predefined threshold for this neuron to fire
- Output layer- This is where the prediction or yhat is provided. The yhat is based on the computed value received from the final interconnected hidden layer (more on this below). The greatest value received by a neuron in this layer will determine the prediction given. That prediction is compared to the actual output while training a model.

Data that is passed between interconnected neurons go through paths referred to as channels. Every channel has a value associated to, known as a weight, that is used to determine an inputs importance when the data is processed. The weight assigned to each channel is usually unique to ensure differences in how the data gets processed in each neuron. For each input in a neuron, its value is calculated with the associated weight from its channel and added together. A bias is then added to that sum to determine the yhat. If that yhat is equal to or exceeds a defined threshold, the neuron is fired and its value is passed to the next neuron.

When training some iterations of this model, comparisons between the expected output and the prediction are done to find the error. The function responsible for this is called the cost function. The insight found in the error is used to adjust the weights of channels, which will change the outcome and produce more accurate results, reducing loss. When automated, this process is known as backpropagation. This is conceptually similar to punishments in reinforcement learning.

A convolutional neural network, also know as ConvNets is a type of Artificial neural network that works on grid-structure inputs, like images. This approach can be used for image classification, as demonstrated above, and facial recognition where facials features are converted to values to be used as inputs. This sets up a feature mapping for detecting features.

CNN's have many strengths:

- CNN's are transfer invariant, they don't care where the object/feature exists in the image as long as it exists. This Is great for image recognition
- Models pre-trained on large datasets can be leveraged and fine-tuned, reducing the amount of data and training time

- Well established and great results in tasks like object detection and image classification

CNN's have many challenges too:

- CNN's often require large diverse dataset to be effective
- Models with extensive layers can be computationally expensive
- Configuring a CNN can take a lot of experimentation because hyper-parameters that significantly impact the performance of the model can take a while to get right
- Troubleshooting CNN's can be tricky because they're like black boxes. We won't understand the feature detection logic that identifies features

CNN's have brought on significant advancement in technology and computer vision from its ability to detect features using data but the data, computational and configuration requirements should be considered when using the technology.

Code deep dive:

The class shown in figure 3.1 is instantiated for each modelused in the agent.

The results attribute on line 6 stores all the details from the model, including the name of each label and its results in an array. The 'setResults' method on line 12 is used to set the value of the results attribute on line 6.

The model attribute on line 7 is used to store the model and the 'setModel' attribute on line 12 is used to set the value of the model for the objected, seen on line 7.

The 'modelPrediction' method will return nothing if no classification results are exists in the object. If results do exists, the results array are looped through and the object with the greatest confidence is returned from the array. Each object represents a label and possible classification.

```
3  class TeachableMachineModel {
4      constructor (url) {
5          this.imageModelURL = url;
6          this.results = undefined;
7          this.model = undefined;
8      }
9      setModel (tmModel) {
10         this.model = tmModel;
11     }
12     setResults (results) {
13         this.results = results;
14     }
15
16     modelPrediction () {
17         if(!this.results) return;
18         const modelPredictionData = this.results.reduce((accumulator, currentValue) => accumulator.confidence > currentValue.confidence? accumulator:currentValue);
19         return  modelPredictionData;
20     }
21 }
```

Figure 3.1: Class for each machine model

On line 23 to line 25 we store the url for each Teachable Machine model that we use to create an instance of the class seen in figure 3.1. Each object will store the details specific to the model instantiated explained above (line 27 to 29). We group each model in an array on line 31.

```
23  const genderModelURL = 'https://teachablemachine.withgoogle.com/models/7zojvxBV6/';
24  const ageModelURL = 'https://teachablemachine.withgoogle.com/models/7b1fjdwZX/';
25  const glassesModelURL = 'https://teachablemachine.withgoogle.com/models/pPsE50dcv/';
26
27  const genderMachine = new TeachableMachineModel(genderModelURL);
28  const ageMachine = new TeachableMachineModel(ageModelURL);
29  const glassesMachine = new TeachableMachineModel(glassesModelURL);
30
31  const teachableMachineModels = [genderMachine, ageMachine, glassesMachine];
```

Figure 3.2: Invariant variables for the models

'cam' on line 33 is used to store the webcam object. The preload function is called automatically when the p5 library is loaded, but only once. In the preload function, the 'teachableMachine' array is looped through. For each object in the array, we use the url of the object to load a model and store that model in the object.

The setup function is called only once automatically after the preload function. In this function, we create a canvas that will contain the output from the camera then attach it to a HTML element. We set the frame rate for the camera. We receive input from the camera and hide its default display. Finally we call the classify function to start the process of classifying the input from the camera using the models. More on the classify function below.

```
33  let cam;
34
35
36  function preload() {
37
38      for (model of teachableMachineModels) {
39          const classifier = ml5.imageClassifier(model.imageModelURL + 'model.json');
40          model.setModel(classifier);
41      }
42  }
43
44
45  function setup() {
46      const viewport = createCanvas(480, 360);
47      viewport.parent('video_container');
48      frameRate(24);
49      cam = createCapture(VIDEO);
50      cam.hide();
51      classify(teachableMachineModels);
52  }
```

Figure 3.3

The classify function, as the name implies, is used to call the model on each object and classify the input coming from the camera. This is done only when the page is in focus, to preserve the performance and computational power of the device when the agent is not in use/needed.
After looping through the array storing the teachable machine model objects that is passed in as an argument, a callback function, 'processResults', is bound to the model (more on this later). Each model's classify method is invoked which runs the associated model and classifies the input from the camera. The method takes the camera object and the bound callback function 'processResults' as arguments. After a 400 milisecond delay, the classify method calls itself again, repeating the process.

The callback function 'processResults' will display a message to the console if there is an error when classifying the input from the camera. Otherwise, the models results are stored as an array on the object associated to the model. This is done with help from the the bind method, shown on line 59. The bind method associates the this object referenced on line 73 to the object we pass as the argument for the bind method.

```
55  function classify(teachableMachineModelsArr) {
56
57          if (document.hasFocus()) {
58              for (model of teachableMachineModelsArr) {
59                  const processor = processResults.bind(model);
60                  model.model.classify(cam, processor);
61              }
62          }
63      setTimeout(() => classify(teachableMachineModels, 400));
64  }
65
66  function processResults(error, results) {
67      if (error) {
68          console.error("classifier error: " + error);
69      } else {
70          const modelResults = results.reduce((resultsAccumulation, result) => {
71              return [...resultsAccumulation, {label: result.label, confidence: result.confidence}]
72          },[])
73          this.setResults(modelResults);
74      }
75  }
```

Figure 3.4: method used to classify input from the camera

The draw function shown in figure 3.5 is called on every frame and is effectively the agents output. In the function:

- The background colour is set
- The feed from the camera is stored in the canvas (see figure 3.3)
- A result for each model is displayed

```
78  function draw() {
79      background("#c0c0c0");
80      image(cam, 0, 0);
81      formatResponse(teachableMachineModels, confidenceInPrediction);
82  }
```

Figure 3.5: teachable machine draw function- the agents output

Finally, we have the helper functions (figure 3.7) which assists in formatting the response from the agent based on classification from the models:

- 'confidenceInPrediction' takes in the confidence of the predicted classifier and produces a calculated probability of its accuracy based on a percentage.
- 'formatResponse is where we display the results of the model to the HTML page. For each model, we get the prediction and the prediction's confidence percentage from the model's object and the prediction's accuracy probability from the 'confidenceInPrediction' helper function to structure a message that we pass into a HTML element

```
function confidenceInPrediction(confidencePercentage) {
    if (confidencePercentage < 60) {
        return "could be";
    } else if (confidencePercentage < 80) {
        return "is likely";
    } else {
        return "is";
    }
}

function formatResponse (teachableMachineModelsArr, confidenceInPredictionGenerator) {
    teachableMachineModelsArr.forEach((model, index) => {
        const modelResult = model.modelPrediction();
        if (!modelResult) return "Please wait";
        const resultConfidencePercentage = (modelResult.confidence * 100).toFixed(0);
        const elementText = `Subject ${confidenceInPredictionGenerator(resultConfidencePercentage)} ${modelResult.label} (${resultConfidencePercentage}%)`;
        document.getElementById(`results${index + 1}`).innerHTML = elementText;

    });

}
```

Figure 3.7. Helper functions that assist in structuring a response from each model's classification