**Creating Smart Machine Learning Models using Google Teachable Machine**

Google Teachable Machine is one of the easiest models that any layman can use to train machine learning models without any coding experience. Below are the step-by-step descriptions for using it:  
**1. Teachable Machine**

Teachable Machine has now been integrated into Google Sites – so the first step is to navigate to the Google Teachable Machine website.

Choose the type of project: Image, Audio, or Pose.

**2.Create a New Project**

To begin using the tool, look for the words “Get Started” or “Start a New Project”.

Choose the type of model you would want to train, (for instance Image Classification, Audio Classification).

**3.Add Classes**

Allocate the classes that you expect the model to identify and create categories (classes) to know the sequences. For instance, if developing an image project, name some of the categories are “Cat” and “Dog”.

Name each class clearly. Like EG: A, B, C, D or any but clearly to be name.  
**4. Train the Model**

**Collect Data:**

For each of the classes, either use your webcam or upload files to give examples.

Unlike human input, input from computers is preferred to be diverse and clear to make better model capture or upload.

**Review Data:**

Check whether you have enough of data of each class. Therefore, it is desirable to have a balanced dataset.

**5.Train Your Model**

Now click on the button named “Train Model”.

Just wait for the model to indicate how it has modified to accommodate the data you presented to it. This could take anything from a few seconds to a few minutes depending on the size of a given data set.

**6. Test Your Model**

Upon the training, try the model using direct feed or record, from a webcam, microphone or new samples uploaded by the user.

It is also needs to check that the model correctly assigns the input to the given categories or not.

**7.Export and Use the Model**

**Export Options:**

Download the model for locals use either in TensorFlow of TensorFlow Lite formats.

As for deployment: utilize Google Cloud.

Click here to copy the shareable link.

**Integrate the Model:**

After exporting the model, easily deploy it into applications such as websites, apps or any IoT device.  
**8. Success**

It is better to have diversity and representation to generalizeпіон

It is advisable to put the model through various tests in order to come up with the different areas a model is likely to have a weakness in.

In the case of poor performance change and train again to achieve better results.

Feel free to ask if you want a breakdown of any project type that you are interested in!

**Export code and file format created as working on project**

<div>Teachable Machine Image Model</div>

<button type="button" onclick="init()">Start</button>

<div id="webcam-container"></div>

<div id="label-container"></div>

<script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@latest/dist/tf.min.js"></script>

<script src="https://cdn.jsdelivr.net/npm/@teachablemachine/image@latest/dist/teachablemachine-image.min.js"></script>

<script type="text/javascript">

// More API functions here:

// https://github.com/googlecreativelab/teachablemachine-community/tree/master/libraries/image

// the link to your model provided by Teachable Machine export panel

const URL = "./my\_model/";

let model, webcam, labelContainer, maxPredictions;

// Load the image model and setup the webcam

async function init() {

const modelURL = URL + "model.json";

const metadataURL = URL + "metadata.json";

// load the model and metadata

// Refer to tmImage.loadFromFiles() in the API to support files from a file picker

// or files from your local hard drive

// Note: the pose library adds "tmImage" object to your window (window.tmImage)

model = await tmImage.load(modelURL, metadataURL);

maxPredictions = model.getTotalClasses();

// Convenience function to setup a webcam

const flip = true; // whether to flip the webcam

webcam = new tmImage.Webcam(200, 200, flip); // width, height, flip

await webcam.setup(); // request access to the webcam

await webcam.play();

window.requestAnimationFrame(loop);

// append elements to the DOM

document.getElementById("webcam-container").appendChild(webcam.canvas);

labelContainer = document.getElementById("label-container");

for (let i = 0; i < maxPredictions; i++) { // and class labels

labelContainer.appendChild(document.createElement("div"));

}

}

async function loop() {

webcam.update(); // update the webcam frame

await predict();

window.requestAnimationFrame(loop);

}

// run the webcam image through the image model

async function predict() {

// predict can take in an image, video or canvas html element

const prediction = await model.predict(webcam.canvas);

for (let i = 0; i < maxPredictions; i++) {

const classPrediction =

prediction[i].className + ": " + prediction[i].probability.toFixed(2);

labelContainer.childNodes[i].innerHTML = classPrediction;

}

}

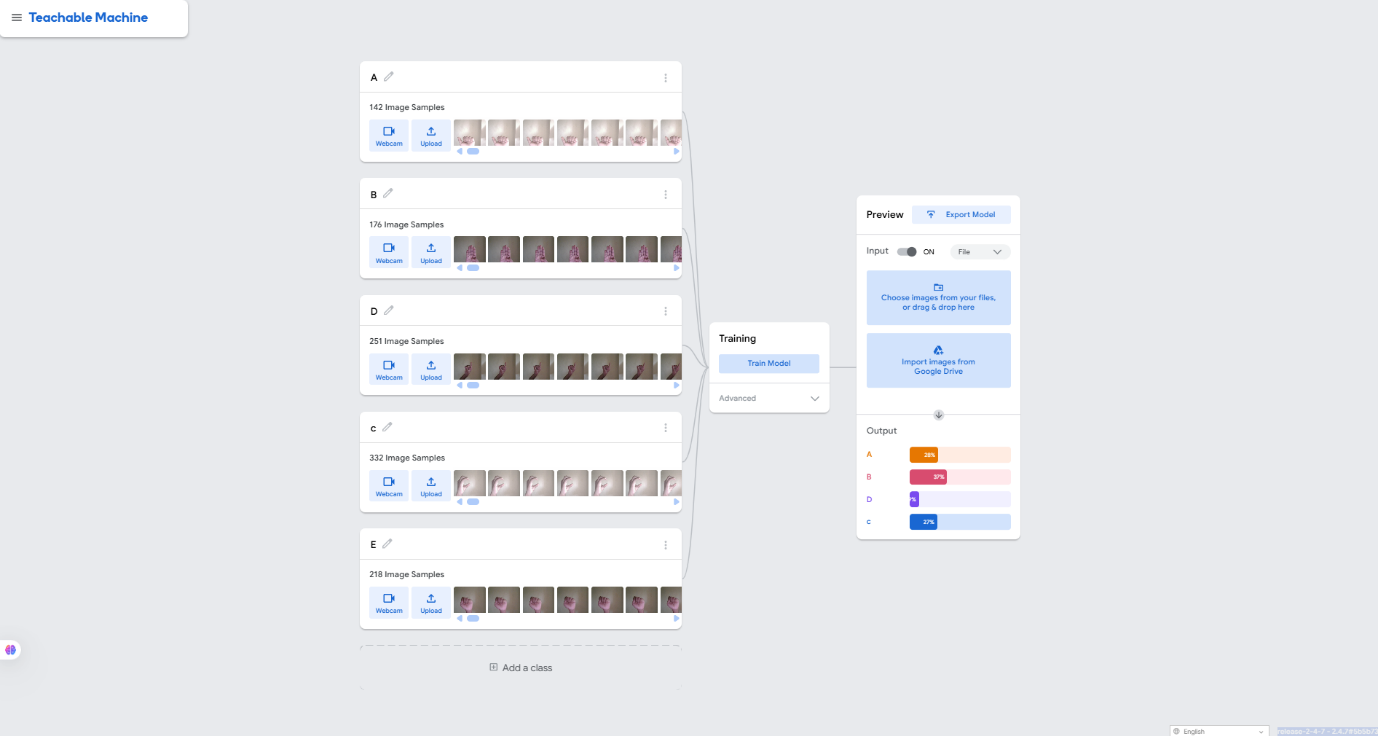
</script>

Export file  

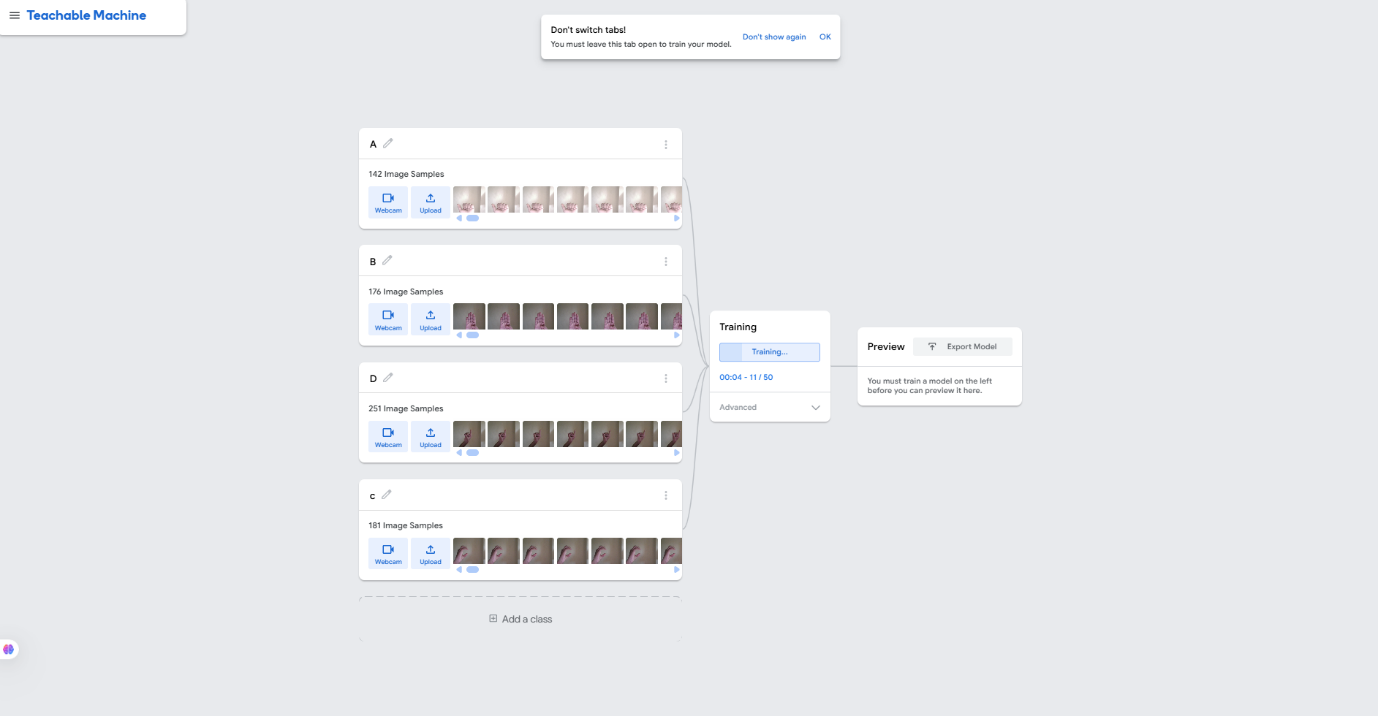

https://teachablemachine.withgoogle.com/models/ngazIUiMb/

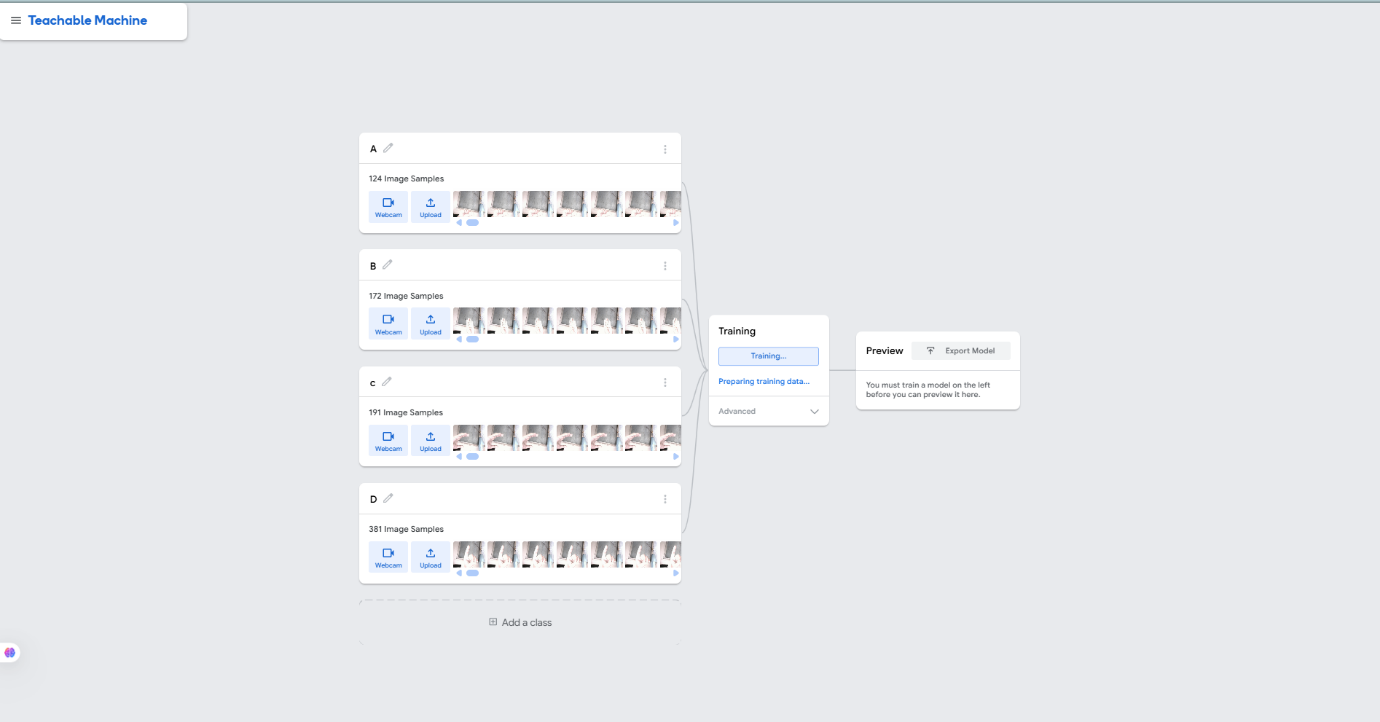
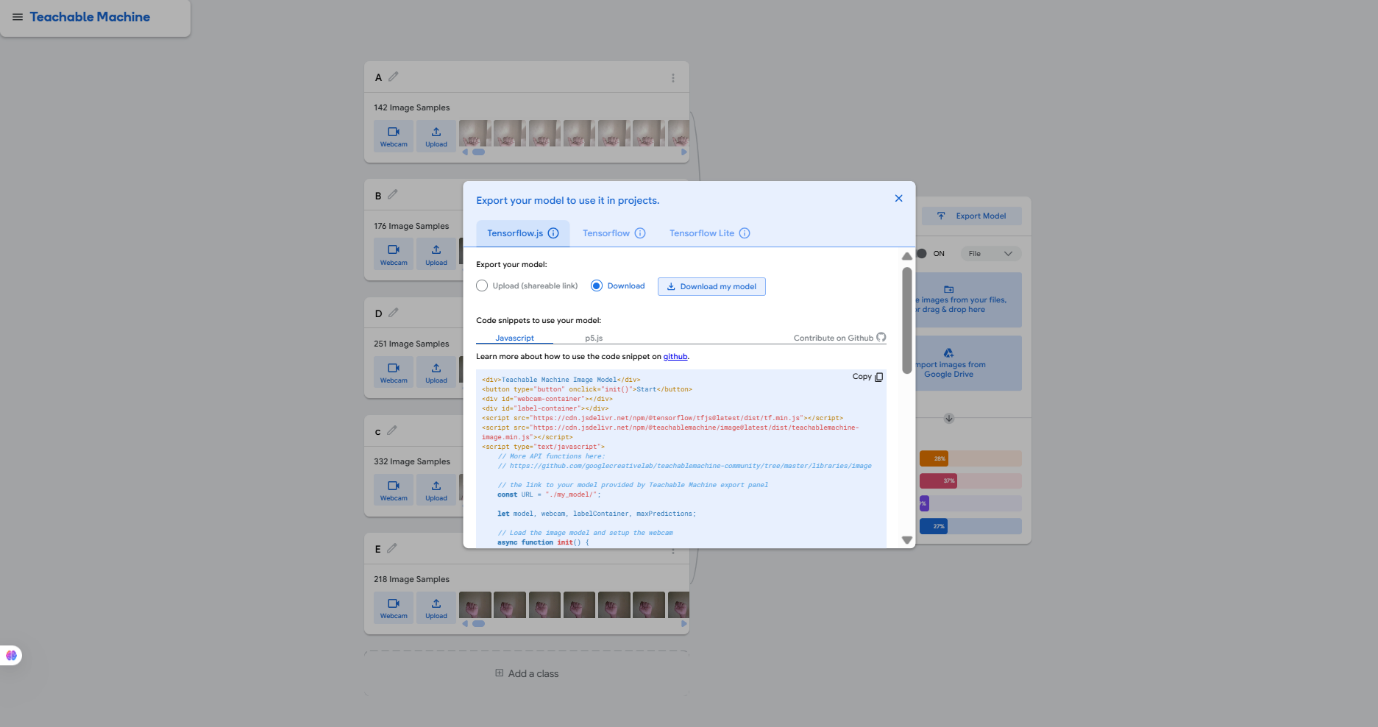
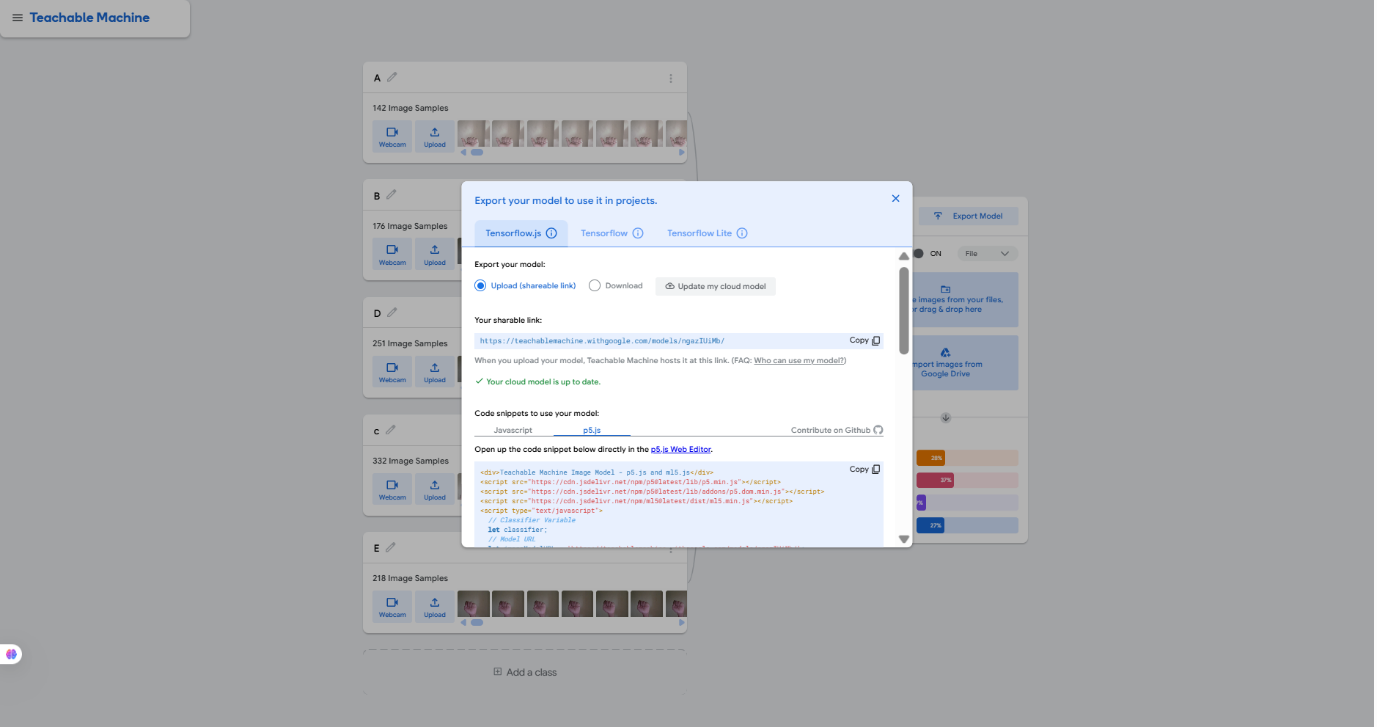
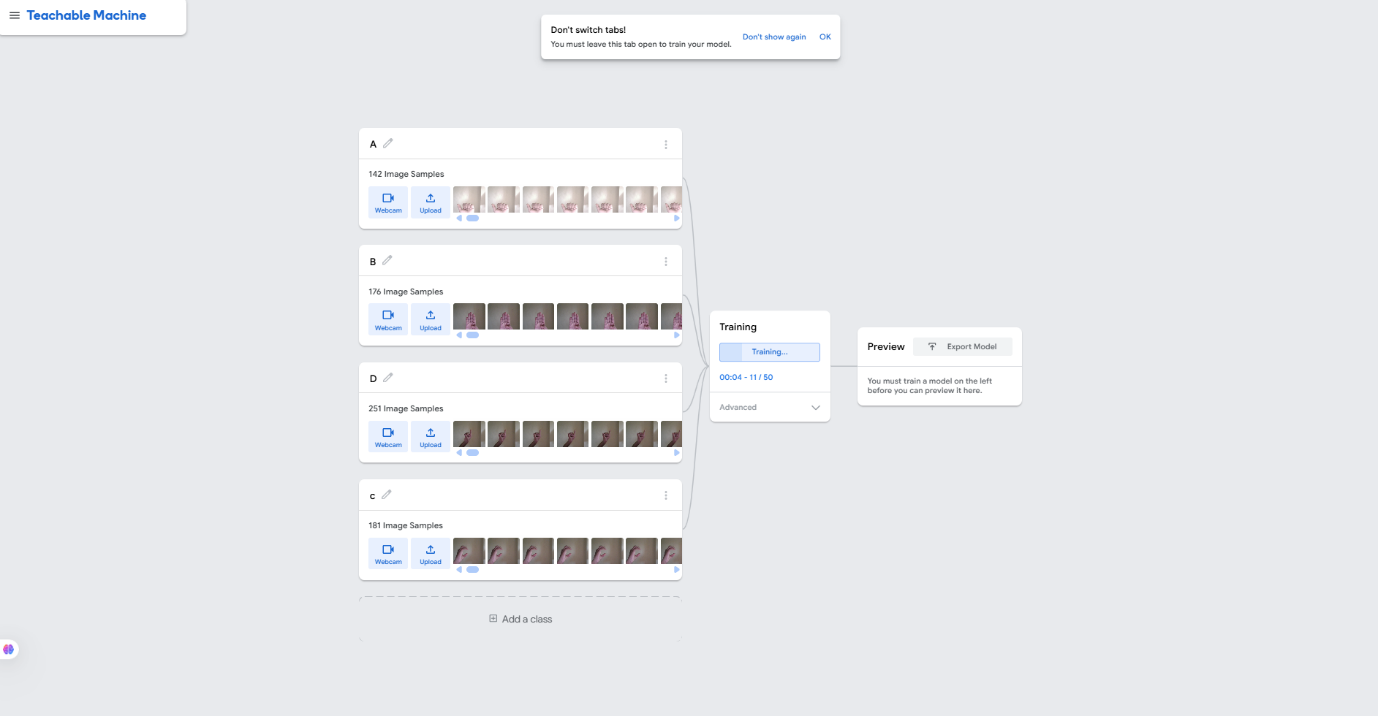
**Screen shots of working project**

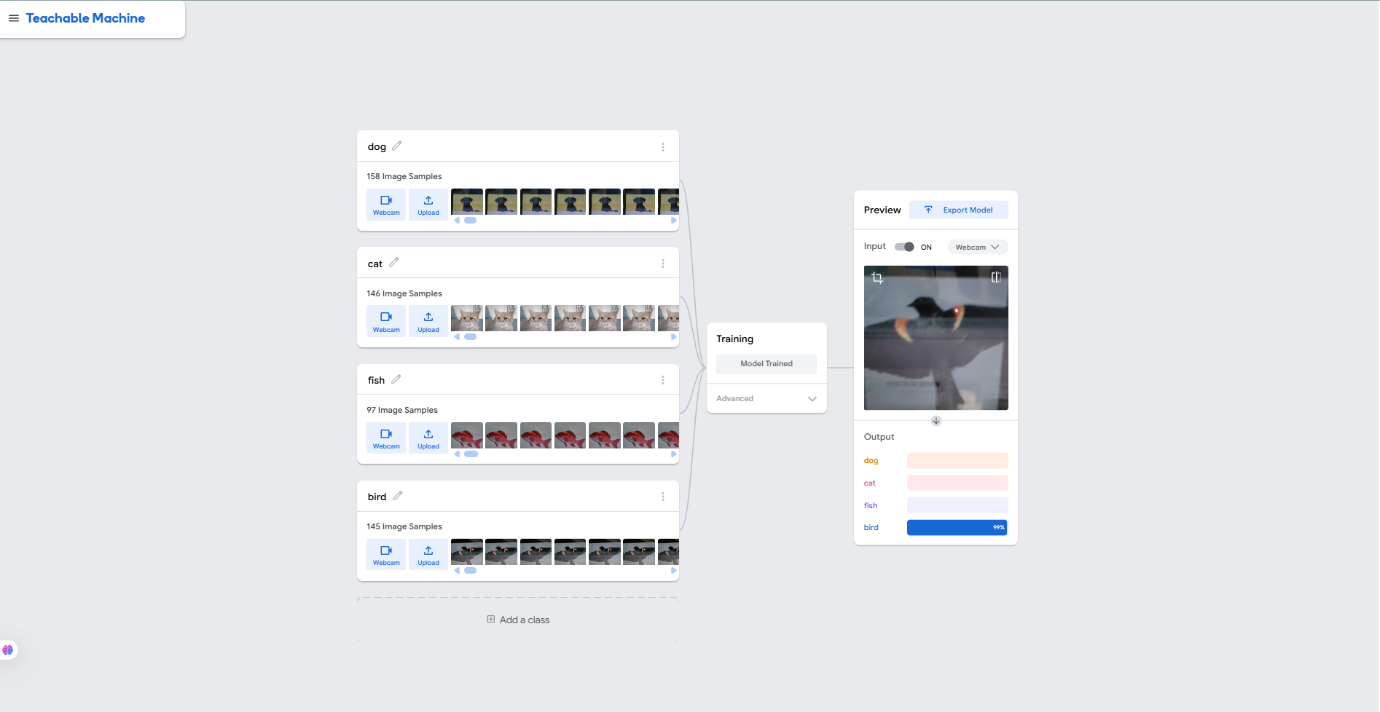
**Sign Language Translator**



**Process:**



**Animal Classifier for Wildlife Enthusiasts**

Tested T.m with different image of animals and accuracy level shown in screen shot.

**List of files and folders in GitHub Repository with brief description of the item.**

**1. Repository Setup**

**Name:**

An easy-to-understand short description of your repository that gives an idea of what the project is about or what is it for. Do not use the name of organizational units that are too vague or ambiguous.

**Description:**

Explain the purpose of repository in not more than three statements. This is located under the repository and it makes users able to grasp what the project contains.

**Repository Type:**

Decide if the repository should be open to any users which means that all users can find and join it or it should be the project only where some users can join collaborating with its creators.

**License:**

Include a license file (an example may be MIT, Apache 2.0, and GPL) this will allow a person who downloads your project to understand the terms of use, ways and conditions to share or alter it.

**2. Essential Files**

**README.md:**

An extensive explanation of your work. Include:

The need, objectives and tasks of the project.

Setup instructions for users.

Examples of use and features of the model.

Any precondition and dependency if in place.

**CONTRIBUTING.md:**

**Guidelines for contributors, detailing:**

Where to report bugs and how to take part in the project development.

Programming Language Standards or Style Guides.

The approach that should be used in manners to integrate effort.

.

**CODE\_OF\_CONDUCT.md:**

It gives directions concerning the acceptable behavior and the expected interaction in a given community. It guarantees that no contributor feels uncomfortable and has no reason to stop contributing to the material.

**LICENSE:**

Let contain a file that indicates the license that this project will follow. As important for open source projects to define usage rights and restrictions.

.**gitignore:**

A file containing patterns which file or directories Git should be attuned not to look for while tracking changes (e.g., build file, logs, sensitive information files).

**CHANGELOG.md**:  
A chronological list of notable changes to the project. Helps users and contributors keep track of updates across versions.

**3. Code Structure**

**Directory Organization:**

Keep the directory structure simple, neat, well thought conceived and easily understandable to the users about your project. Examples:

src/ for source code.

docs/ for documentation.

tests/ for test scripts.

folder that is commonly named as asets/ for images, logos or some other resources.

**Modular Code:**

This means writing code that modular, reusable, and maintainable. Maintain associating functions of similar or related nature together.

**Documentation:**

Inline comments and docstrings may be used to explain about the functionality and logic of the piece of code for better contribution from others.

**4. Version Control**

**Branching Strategy:**

Adopt a branching strategy, such as:

primary or pinnacle for code that is fit to go live on the production environment.

feature/ for new features.

bugfix/ for bug fixes.

release/ for working on a new version of ,

**Tags:**

Includes specific pointing which may be particular moments of the repository development (release version: v1.0.0, v1.1.0 and etc.).

**Commit Messages:**

Remember to commit useful and brief commit messages. Follow a format such as:

**feat:** for new features.

**fix:** for bug fixes.

**docs:** for documentation changes.

**refactor:** for code refactoring.

**5. Collaboration**

**Issues:**

Have options for the bug reports, feature requests, and questions. They should be labeled (priority) such as bug, enhancement, documentation, etc.

**Pull Requests:**

Ensure that there are templates in pull requests for contributors to use when coming up with them. Include:

A summary of the changes.

Testing procedures in relation to the imposed changes.

Associated issue numbers.

**Assignments:**

Spike issues and pull requests so members see which ones they are supposed to work on in order to have clear tasks.

**6. Automation and Enhancements**

**CI/CD Setup:**

On GitHub, or with any other CI/CD tooling, automated testing, building and deploying should be done.

**Badges:**

Save badges (e.g., build status, license, dependencies) into the README file to read more about the project quicker.

**Code Review:**

Make a practice of reviewing the code on the pull requests before switching to merge in order to keep the quality.

Git hub link  
https://github.com/jagdishkumarhembrom/sign-language-translator.git

Thanks you