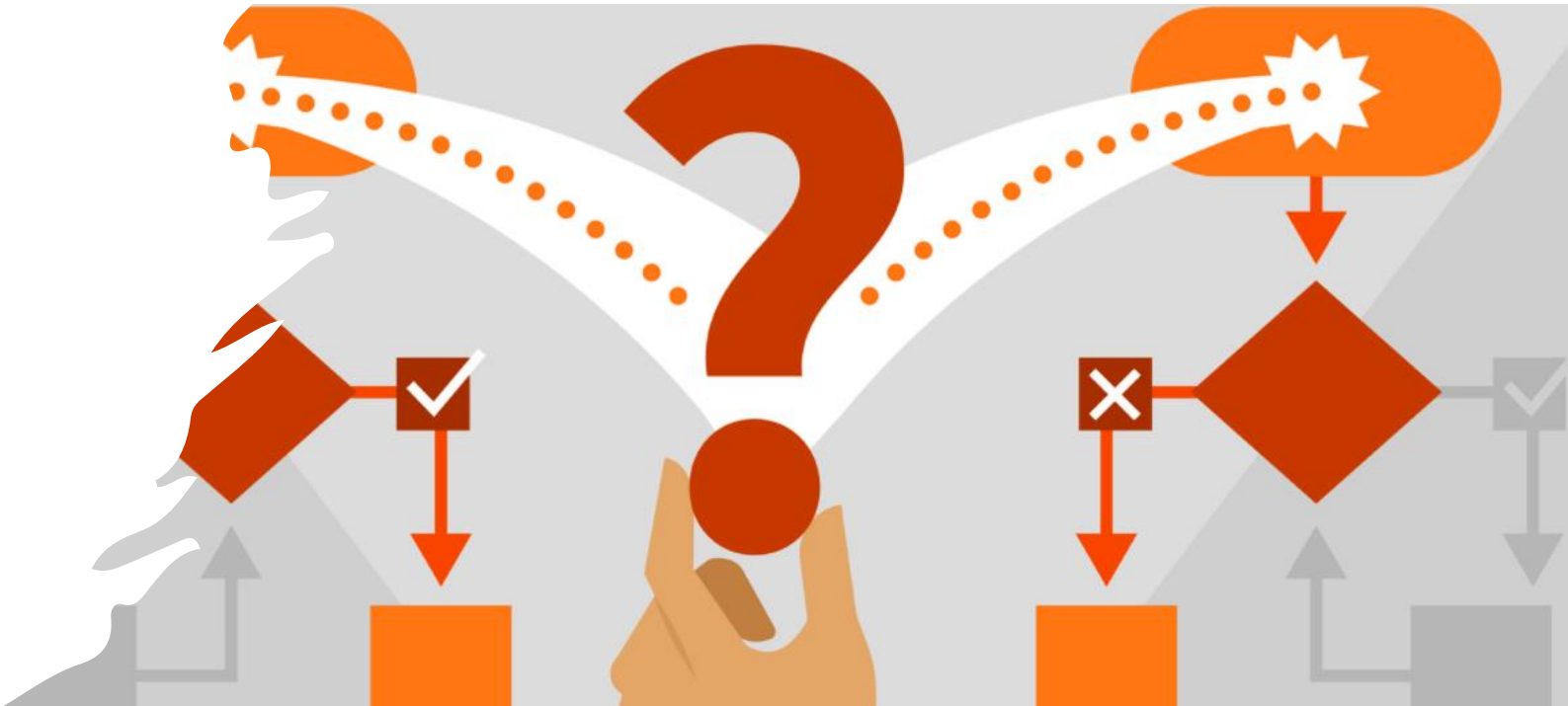
TEACHABLE MACHINE

IMAGE CLASSIFICATION SYSTEM



TEAM


Ahmed Abo Elyaized El Gohary


Mostafa Mohamed Ahmed


Yassa Adel Thabet


Mohamed Essam Saeed


Omar Ahmed Shwaqy


Eissa Islam Eissa

Supervised by

Eng. Sara Ashraf

Project Overview

Title

Building a Teachable Machine-Like Image Classification System.

Objective

Develop a web application allowing users to upload images Classes, set training parameters, train a classification model, and test it with new images.

Tech Stack

- **Frontend:**
  - Image Classification.html # UI for uploading images, training, and testing
  - Image Classification.css # Frontend styles
  - Image Classification.js # Frontend logic (JS/React/Vue.js)
- **Backend:**
  - App.py # FlaskI for server-side logic
  - Model.py # Machine learning model training and testing logic
- **Deployment**: Heroku/AWS/Google Cloud

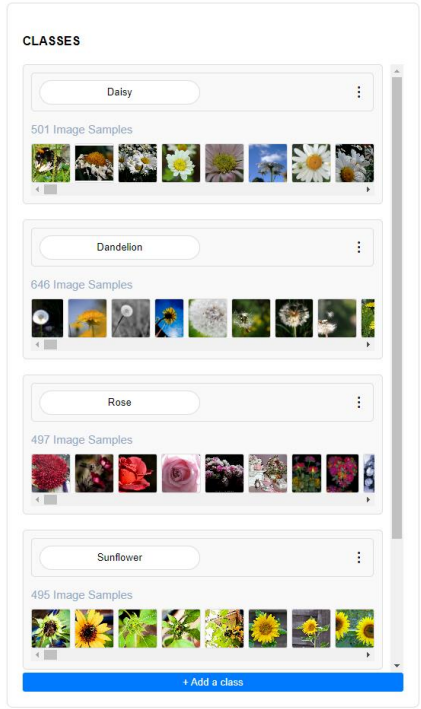
Frontend Setup

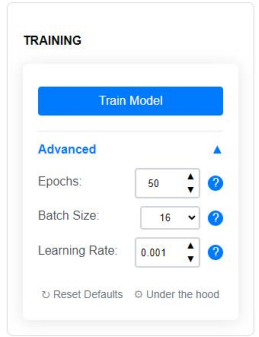
UI Components

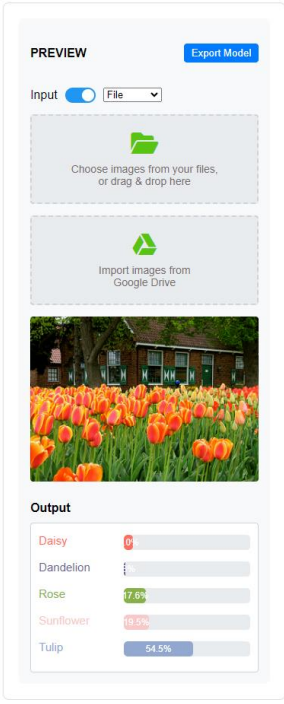
- **Class creation & Image Upload:**
  - Add Classes name.
  - Add Class image samples by upload or Webcam .
  - Use **JavaScript** to preview images and track uploads.
- **Training Parameters:**
  - Include fields for:
    - **Number of epochs** (number of training iterations).
    - **Batch size** (how many images to process at once).
    - **Learning rate** (how quickly the model learns).
- **Buttons:**
  - Add a "Train Model" button that triggers the backend /train endpoint.
- **Testing Section:**
  - Add another file upload input to let users upload images for model testing after training.

Apps/Tools:

Visual Studio Code for code editing.







Backend Development

set up APIs to manage the image uploads, model training, and testing.

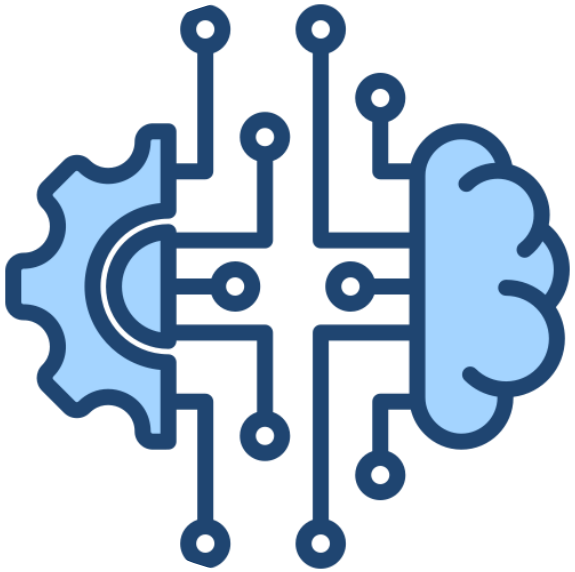
- **Endpoints**:
  - **/upload**: This endpoint should handle file uploads. Store images in a directory or cloud storage (e.g., AWS S3).
  - **/train:** This endpoint initiates the training of the CNN model using the uploaded images.
    It will take parameters like (epochs, learning rate , batch size.) .
  - **/predict**: This endpoint loads the trained model and makes predictions on new images.

- **Model Training:**
  - Use **TensorFlow/Keras** to create a CNN model:
    - Start with Conv2D layers, MaxPooling2D, and Dense layers.
    - Make the model configurable by user inputs (epochs, learning rate , batch size.).
  - Implement callbacks like early stopping to prevent overfitting.

Apps/Tools:

- **Flask** (for building APIs)
- **Postman** (to test API endpoints).

Model Training Process



The training logic needs to:

1. Preprocessing: Resize images, normalize data.
2. Model Creation: Build the CNN model architecture using TensorFlow/Keras.
3. Training: Customize training with epochs, batch size, learning rate.
4. Callbacks: Implement early stopping and model checkpoints.
5. Compile the model with a loss function (categorical cross-entropy for classification) and optimizer (like Adam).
6. Train the model and store it.

Store the trained model in:

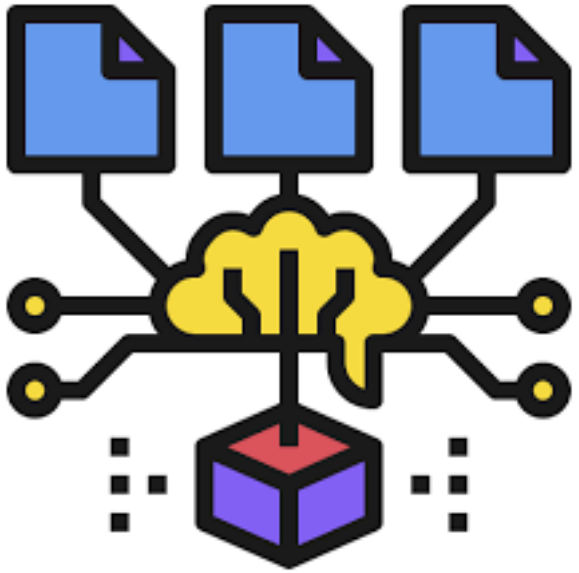
HDF5 format for TensorFlow.

Load and save model checkpoints during training.

Apps:

Google Colab for running Python and TensorFlow

Testing and Predictions



Once the model is trained, the /predict endpoint can:

- Load the trained model.
- Preprocess the test image to match the model input.
- Predict on new images uploaded by users using model.predict() .
- Display confidence scores and class predictions on the frontend.

User Interaction

- Simple drag-and-drop for image uploads.
- Real-time display of results on the application interface.

Deployment and Accessibility

After everything is set, package the application for deployment:

1.Backend:

1. Set up a Procfile (for Heroku) or an AWS Elastic Beanstalk environment for deployment.
2. Use Docker for more flexible and consistent deployment environment.

2.Frontend:

1. Host frontend on a service like Netlify or include it in the backend deployment (using Flask or FastAPI to serve static files).

Apps/Tools:

Docker (for containerized deployments).
Heroku CLI, AWS CLI, or Google Cloud CLI (for cloud deployment).

Future Enhancements:

Adding more complex models.
Enabling model export for use outside the web app.
Expanding to more types of machine learning tasks.