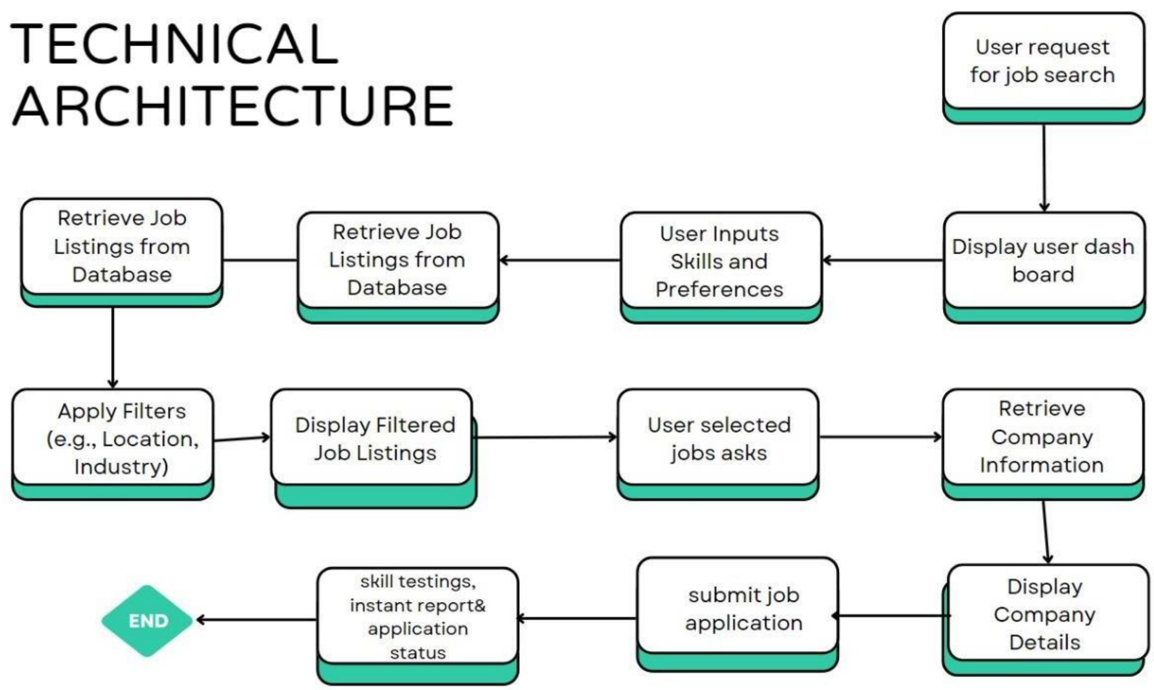
Rookies.IN

Introduction:

According to present situation many entry-level job seekers are facing many problems to get good package job or desired job .To solve this problem we came up with a new application to make it better and easy to prepare for companies like apple,google ,Microsoft,Bank of America,De sahw &co, Infosys , capegemini,Spotify…etc….by using this application the job seekers can get more information about present updated technologies and projects .

This helps the society to get the jobs easy and in an accessible and also comfortable. This web application can predict which technology recruits more for a company ,for example amazon required more frontend developers as compared to backend developers so a job seeker can apply to the frontend developer role to amazon compay .

Technical Architecture:



Prerequisites:

To complete this project, you must require the following software's, concepts, and packages

Anaconda Navigator is a free and open-source distribution of the Python and R programming languages for data science and machine learning-related applications. It can be installed on Windows, Linux, and macOS.Conda is an open-source, cross-platform, package management system. Anaconda comes with so very nice tools like JupyterLab, Jupyter Notebook,

Visual Studio Code, Html,Css ,Java Script , Boost strap,Python,…etc . For this project, we will be using Jupyter notebook and Spyder.

To install Anaconda navigator and to know how to use Jupyter Notebook & Spyder using Anaconda watch the video

Link: Click here to watch the video

1. **To build Machine learning models you must require the following packages**

- **Numpy**:
  - It is an open-source numerical Python library. It contains a multidimensional array and matrix data structures and can be used to perform mathematical operations

- **Scikit-learn:**
  - It is a free machine learning library for Python. It features various algorithms like support vector machine, random forests, and k-neighbors, and it also supports Python numerical and scientific libraries like NumPy and SciPy

- **Flask:**
  Web framework used for building Web applications

- **Python packages:**
  - open anaconda prompt as administrator
  - Type "pip install numpy" and click enter.
  - Type "pip install pandas" and click enter.
  - Type "pip install scikit-learn" and click enter.
  - Type "pip install tensorflow==2.3.2" and click enter.
  - Type "pip install keras==2.3.1" and click enter.

- Type “pip install Flask” and click enter.

- **Deep Learning Concepts**
    - **YoloV7:** ľhe YOLO (You Only Look Once) model is a single-stage object detectoí. Image fíames aíe featuíed thíough a backbone, featuíes aíe combined and mixed in the neck, and then they aíe passed along to the head of the netwoík wheíe YOLO píedicts the bounding box locations, the classes of the bounding boxes, and the objectness of the bounding boxes. With its amazing chaíacteíistics, YOLOv7 is a íeal-time object detectoí that is now tíansfoíming computeí vision.
    - **Flask:** Flask is a popular Python web framework, meaning it is a third-party Python library used for developing web applications.

<u>Flask Basics</u>

If you are using Pycharm IDE, you can install the packages through the command prompt and follow the same syntax as above.

Project Objectives:

By the end of this project you will:

- Know fundamental concepts and techniques of Yolo v7
- Gain a broad understanding of image data.
- Know how to pre-process/clean the data using different data preprocessing techniques.
- know how to build a web application using the Flask framework.

Project Flow:

- The user interacts with the webpage .
- The chosen image analyzed by the model which is integrated with flask application.
- Yolo v7 Models analyze the image, then the prediction is showcased on the teachable machine .

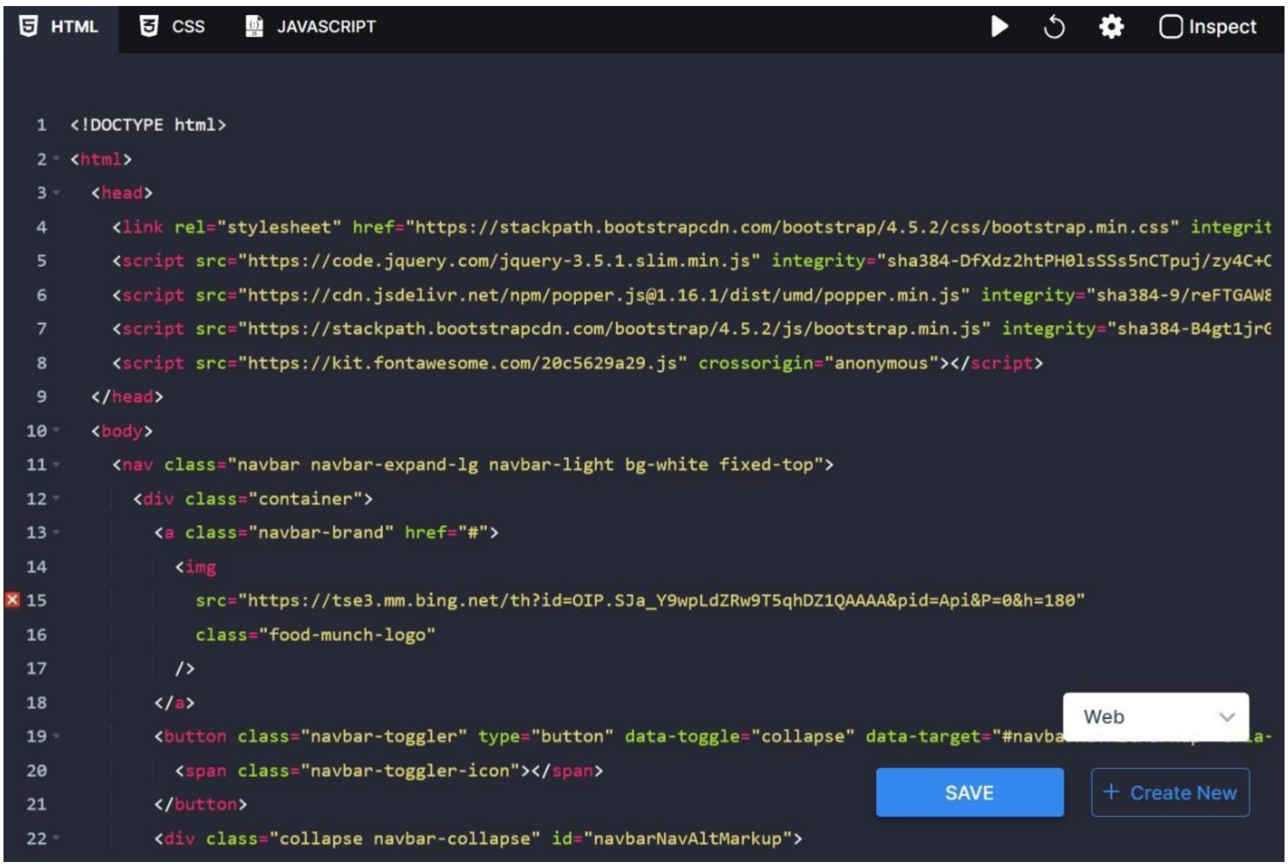
To accomplish this, we have to complete all the activities and tasks listed below

- Data Collection.
    - Create Train and Test Folders.
- Data Preprocessing.
    - Configure ImageDataGenerator class
    - ApplyImageDataGenerator functionality to Trainset and Testset
- Model Building
    - Import the model-building Libraries
    - Initializing the model
    - Adding Input Layer
    - Adding Hidden Layer

        - Adding Output Layer
        - Configure the Learning Process
        - Training and testing the model
        - Save the Model
    - Application Building
        - Create an HTML file
        - Build Python Code

Project Structure:

Project interface code by using HTML, CSS , JAVA SCRIPT



- The Dataset folder contains the training and testing images for training our model.
- We are building a Flask Application that needs HTML pages stored in the **templates** folder and a python script **app.py** for server side scripting
- templates folder contains base.html,index.html pages.

Milestone 1: Data Collection
Collect images of Garbage then organized into subdirectories based on their respective names as shown in the project structure. Create class names like domain names (Ai and Ml,deeplearning and cyber security …etc) and technologies(python,java,powebi,R,Haskel,c++…etc)

Milestone 2: Image Preprocessing
In this milestone we will be improving the image data that suppresses unwilling distortions or enhances some image features important for further processing, although perform somegeometric transformations of images like rotation, scaling, translation, etc.

Activity :Training the data

Here we are training the model in different classes by using images and these are the codes

Milestone 3: Model Building

Now it's time to build our Convolutional Neural Networking which contains an input layer along with the convolution, max-pooling, and finally an output layer.

Activity 1: Importing the Model Building Libraries

Importing the necessary libraries

Importing Necessary Libraries

```
#to define linear initializations import Sequential
from tensorflow.keras.models import Sequential
#To add layers import Dense
from tensorflow.keras.layers import Dense
# to create a convolution kernel import Convolution2D
from tensorflow.keras.layers import Convolution2D
# Adding Max pooling Layer
from tensorflow.keras.layers import MaxPooling2D
# Adding Flatten Layer
from tensorflow.keras.layers import Flatten
from tensorflow.keras.optimizers import Adam
```

Activity 2: Initializing the model

Keras has 2 ways to define a neural network:

- Sequential
- Function A

```python
# Initializing the model
model=Sequential()
```

```python
# Initializing the model
model=Sequential()
```

```python
#First Convolution layer and pooling
model.add(Convolution2D(32,(3,3),input_shape=(128,128,3),activation='relu'))
model.add(MaxPooling2D(2,2))
```

```python
#Second Convolution layer and pooling
model.add(Convolution2D(64,(3,3),padding='same',activation='relu'))

#input shape is going to be the pooled feature maps from the previous convolution.
model.add(MaxPooling2D(pool_size=2))
```

```python
#Third Convolution layer and pooling
model.add(Convolution2D(32,(3,3),activation='relu'))
model.add(MaxPooling2D(2,2))
```

```python
#Fourth Convolution layer and pooling
model.add(Convolution2D(32,(3,3), padding='same',activation='relu'))

#input shape is going to be the pooled feature maps from the previous convolution.
model.add(MaxPooling2D(pool_size=2))
```

```python
#Flattening the layers
model.add(Flatten())
```

Adding Fully Connected Layer

```
# Adding 1st hidden layer
model.add(Dense(kernel_initializer='uniform',activation='relu',units=150))
```

```
# Adding 2nd hidden layer
model.add(Dense(kernel_initializer='uniform',activation='relu',units=68))
```

```
model.add(Dense(kernel_initializer='uniform',activation='softmax',units=6))
```

The number of neurons in the Dense layer is the same as the number of classes in the training set. The neurons in the last Dense layer, use softmax activation to convert their outputs into respective probabilities.

Understanding the model is a very important phase to properly use it for training and prediction purposes. Keras provides a simple method, summary to get the full information about the model and its layers.

Summary of the Model

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 126, 126, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 63, 63, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 127008)            0
_________________________________________________________________
dense (Dense)                (None, 150)               19051350
_________________________________________________________________
dense_1 (Dense)              (None, 68)                10268
_________________________________________________________________
dense_2 (Dense)              (None, 6)                 414
=================================================================
Total params: 19,062,928
Trainable params: 19,062,928
Non-trainable params: 0
```

Activity :Configure The Learning Process

- The compilation is the final step in creating a model. Once the compilation is done, we can move on to the training phase. The loss function is used to find errors or deviations inthe learning process. Keras requires a loss function during the model compilation process.
- Optimization is an important process that optimizes the input weights by comparing the prediction and the loss function. Here we are using adam optimizer
- Metrics are used to evaluate the performance of your model. It is similar to the loss function, but not used in the training process

Compiling the Model

```
#Compiling the CNN Model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['acc'])
```

Activity 7: Train The model

Now, let us train our model with our image dataset. The model is trained for 30 epochs and after every epoch, the current model state is saved if the model has the least loss encountered till that time. We can see that the training loss decreases in almost every epoch till 30 epochs and probably there is further scope to improve the model.

fit_generator functions used to train a deep learning neural network
Arguments:

- steps_per_epoch: it specifies the total number of steps taken from the generator as soon as one epoch is finished and the next epoch has started. We can calculate the value of steps_per_epoch as the total number of samples in your dataset divided by the batch size.

- Epochs: an integer and number of epochs we want to train our model for.

- validation_data can be either:
  - an inputs and targets list
  - a generator
  - an inputs, targets, and sample_weights list which can be used to evaluate the loss and metrics for any model after any epoch has ended.
- validation_steps: only if the validation_data is a generator then only this argument can be used. It specifies the total number of steps taken from the generator before it is stopped at every epoch and its value is calculated as the total number of validation data points in your dataset divided by the validation batch size.

Fit the Model ¶

```
res = model.fit_generator(train_transform,steps_per_epoch=2527//64,validation_steps=782//64,epochs=30,
                          validation_data=test_transform)
```

```
C:\Users\smartbridge\anaconda3\lib\site-packages\tensorflow\python\keras\engine\training.py:1844: UserWarning: `Model.fit_generator`
  warnings.warn('`Model.fit_generator` is deprecated and '
Epoch 1/30
41/41 [==============================] - 45s 1s/step - loss: 1.7747 - acc: 0.2061 - val_loss: 1.4693 - val_acc: 0.4132
Epoch 2/30
41/41 [==============================] - 53s 1s/step - loss: 1.5144 - acc: 0.3893 - val_loss: 1.3684 - val_acc: 0.4410
Epoch 3/30
41/41 [==============================] - 43s 1s/step - loss: 1.3444 - acc: 0.4624 - val_loss: 1.2694 - val_acc: 0.5122
Epoch 4/30
41/41 [==============================] - 49s 1s/step - loss: 1.2176 - acc: 0.5210 - val_loss: 1.1758 - val_acc: 0.5469
Epoch 5/30
41/41 [==============================] - 47s 1s/step - loss: 1.2179 - acc: 0.4982 - val_loss: 1.0858 - val_acc: 0.5694
Epoch 6/30
41/41 [==============================] - 48s 1s/step - loss: 1.1780 - acc: 0.5352 - val_loss: 1.0922 - val_acc: 0.5868
Epoch 7/30
41/41 [==============================] - 50s 1s/step - loss: 1.0955 - acc: 0.5836 - val_loss: 1.0062 - val_acc: 0.6111
Epoch 8/30
41/41 [==============================] - 53s 1s/step - loss: 1.0096 - acc: 0.6127 - val_loss: 1.0593 - val_acc: 0.5885
Epoch 9/30
41/41 [==============================] - 52s 1s/step - loss: 1.0005 - acc: 0.6200 - val_loss: 0.8735 - val_acc: 0.6701
Epoch 10/30
41/41 [==============================] - 50s 1s/step - loss: 0.9507 - acc: 0.6571 - val_loss: 0.8716 - val_acc: 0.6806
Epoch 11/30
41/41 [==============================] - 47s 1s/step - loss: 0.8568 - acc: 0.6822 - val_loss: 0.8149 - val_acc: 0.7083
Epoch 12/30
41/41 [==============================] - 47s 1s/step - loss: 0.8062 - acc: 0.7054 - val_loss: 0.7221 - val_acc: 0.7500
Epoch 13/30
41/41 [==============================] - 51s 1s/step - loss: 0.7450 - acc: 0.7262 - val_loss: 0.6855 - val_acc: 0.7465
Epoch 14/30
41/41 [==============================] - 54s 1s/step - loss: 0.7229 - acc: 0.7402 - val_loss: 0.6877 - val_acc: 0.7465
Epoch 15/30
41/41 [==============================] - 49s 1s/step - loss: 0.6481 - acc: 0.7739 - val_loss: 0.5801 - val_acc: 0.8038
Epoch 18/30
41/41 [==============================] - 52s 1s/step - loss: 0.6438 - acc: 0.7567 - val_loss: 0.5998 - val_acc: 0.7587
Epoch 19/30
41/41 [==============================] - 51s 1s/step - loss: 0.6158 - acc: 0.7782 - val_loss: 0.4134 - val_acc: 0.8594
Epoch 20/30
41/41 [==============================] - 52s 1s/step - loss: 0.4948 - acc: 0.8250 - val_loss: 0.4548 - val_acc: 0.8455
Epoch 21/30
41/41 [==============================] - 50s 1s/step - loss: 0.4633 - acc: 0.8358 - val_loss: 0.3721 - val_acc: 0.8472
Epoch 22/30
41/41 [==============================] - 50s 1s/step - loss: 0.4586 - acc: 0.8412 - val_loss: 0.3870 - val_acc: 0.8594
Epoch 23/30
41/41 [==============================] - 49s 1s/step - loss: 0.4216 - acc: 0.8470 - val_loss: 0.3280 - val_acc: 0.8819
Epoch 24/30
41/41 [==============================] - 45s 1s/step - loss: 0.3834 - acc: 0.8611 - val_loss: 0.3725 - val_acc: 0.8819
Epoch 25/30
41/41 [==============================] - 44s 1s/step - loss: 0.3818 - acc: 0.8644 - val_loss: 0.2590 - val_acc: 0.9062
Epoch 26/30
41/41 [==============================] - 42s 1s/step - loss: 0.3715 - acc: 0.8593 - val_loss: 0.2497 - val_acc: 0.9271
Epoch 27/30
41/41 [==============================] - 45s 1s/step - loss: 0.3030 - acc: 0.8954 - val_loss: 0.2159 - val_acc: 0.9323
Epoch 28/30
41/41 [==============================] - 43s 1s/step - loss: 0.2889 - acc: 0.9078 - val_loss: 0.1847 - val_acc: 0.9479
Epoch 29/30
41/41 [==============================] - 48s 1s/step - loss: 0.2697 - acc: 0.9050 - val_loss: 0.1973 - val_acc: 0.9410
Epoch 30/30
41/41 [==============================] - 43s 1s/step - loss: 0.2579 - acc: 0.9134 - val_loss: 0.1926 - val_acc: 0.9306
```

Activity 8: Save the Model

The model is saved with .h5 extension as follows
An H5 file is a data file saved in the Hierarchical Data Format (HDF). It contains multidimensional arrays of scientific data.

Saving the Model

```
model.save('Garbage1.h5')
```

Activity 9: Test The model

Evaluation is a process during the development of the model to check whether the model is the best fit for the given problem and corresponding data.
Load the saved model using load_model

```
#import numpy library
import numpy as np
#import load_model method to load our saved model
from tensorflow.keras.models import load_model
#import image from keras.preprocessing
from tensorflow.keras.preprocessing import image
#loading our saved model file
model = load_model("Garbage1.h5")
```

Taking an image as input and checking the results

```python
img = image.load_img(r"I:\SmartBridge Projects\Garbage-waste-prediction\glass17.jpg",
                     target_size=(128,128))

x=image.img_to_array(img) #converting in to array format

x=np.expand_dims(x,axis=0) #changing its dimensions as per our requirement
#img_data=preprocess_input(x)
#img_data.shape
```

```python
a=np.argmax(model.predict(x), axis=1)
```

```python
index=['0', '1', '2', '3', '4','5']
result = str(index[a[0]])
result
```

```
'3'
```

```python
train_transform.class_indices
```

```
{'cardboard': 0, 'glass': 1, 'metal': 2, 'paper': 3, 'plastic': 4, 'trash': 5}
```

By using the model we are predicting the output for the given input image

```python
index1=['cardboard', 'glass', 'metal', 'paper', 'plastic', 'trash ']
result1=str(index1[a[0]])
result1
```

```
'paper'
```

The predicted class index name will be printed here.

Milestone 4: Application Building

Now that we have trained our model, let us build our flask application which will be running in our local browser with a user interface.

In the flask application, the input parameters are taken from the HTML page These factors are then given to the model to know to predict the type of Garbage and showcased on the HTML page to notify the user. Whenever the user interacts with the UI and selects the “Image” button, the next page is opened where the user chooses the image and predicts the output.

Activity 1 : Create HTML Pages

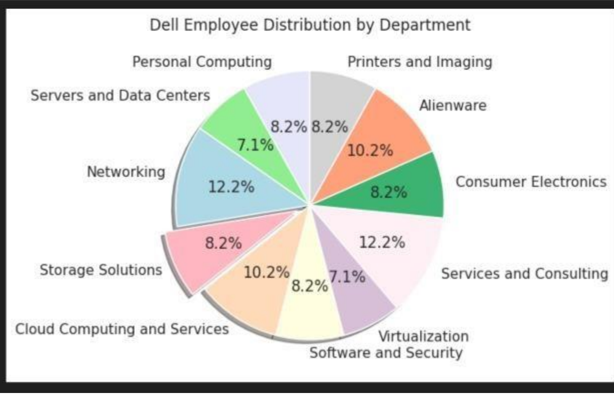
- We use HTML to create the front end part of the web page.
- Here, we have created 3 HTML pages- home.html, intro.html, and upload.html
- home.html displays the home page.
- Intro.html displays an introduction about the project

- upload.html gives the emergency alert
  For more information regarding HTML
  https://www.w3schools.com/html/
- We also use JavaScript-main.js and CSS-main.css to enhance our functionality and view of HTML pages.
- **Link :**CSS , JS

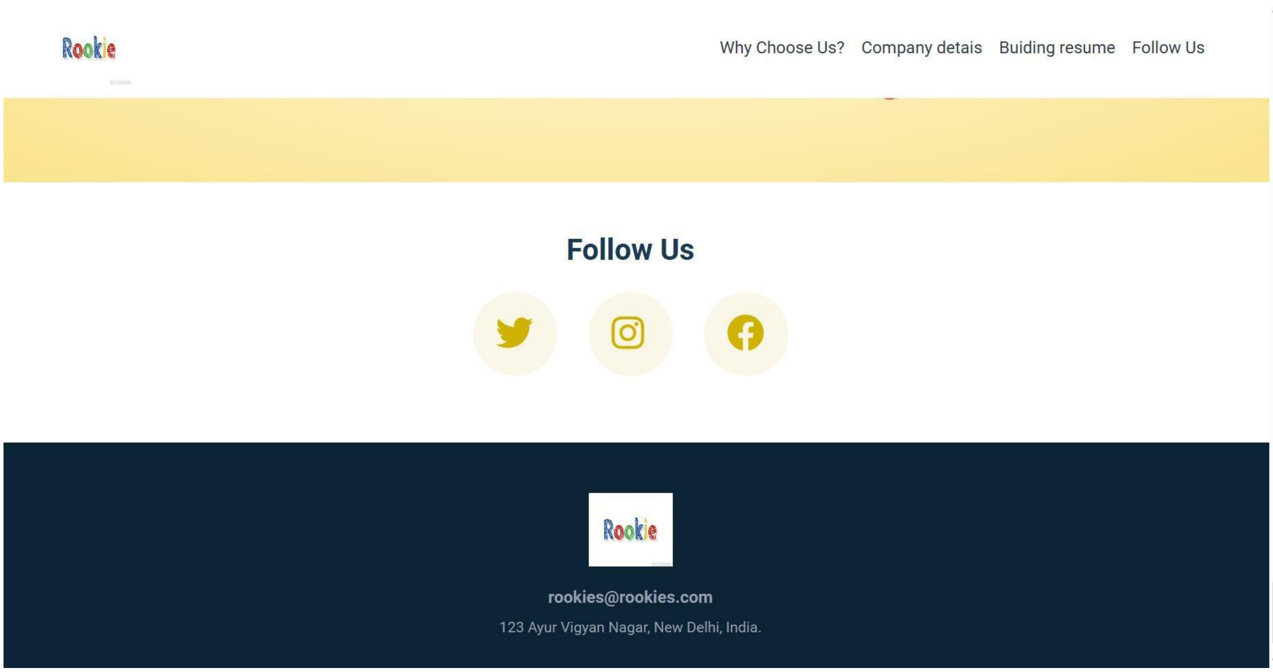
index.html looks like this

About Section:-

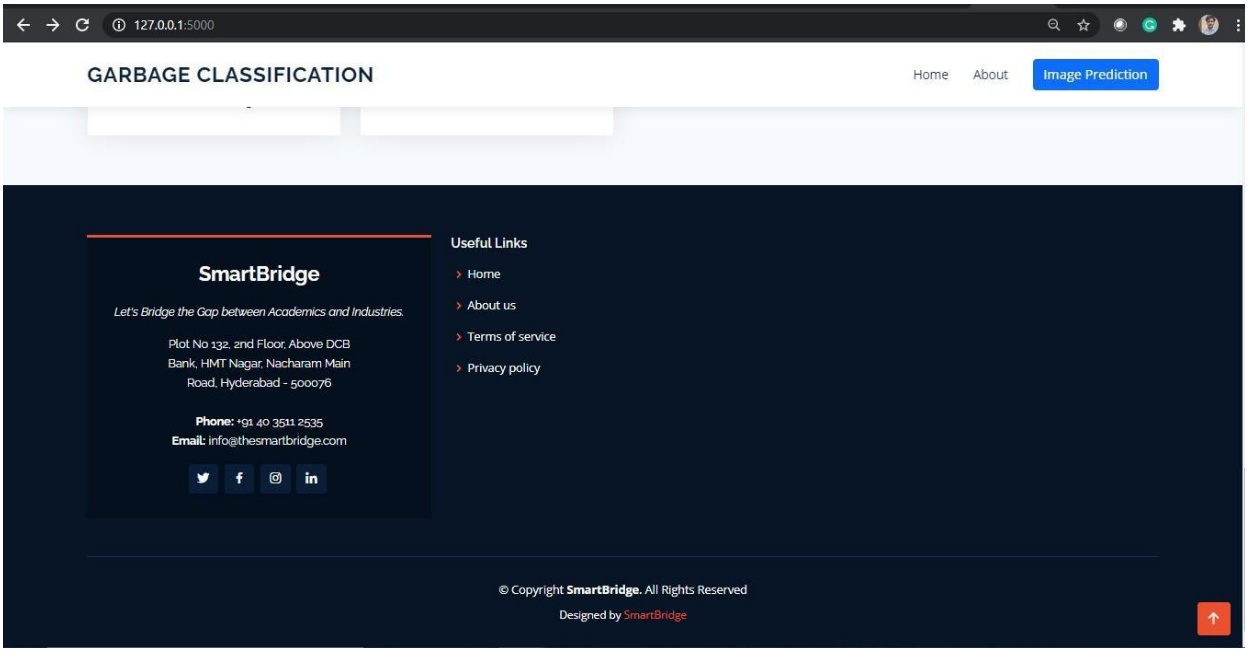
```python
import matplotlib.pyplot as plt

# Data
slices = [8, 7, 12, 8, 10, 8, 7, 12, 8, 10,8]
depts = ['Personal Computing', 'Servers and Data Centers', 'Networking', 'Storage Solutions', 'Cloud Computing and Services', 'Software and Security', 'Virtualization', 'Servic
colors = ['lavender', 'lightgreen', 'lightblue', 'lightpink', 'peachpuff', 'lightyellow', 'thistle', 'lavenderblush', 'mediumseagreen', 'lightsalmon', 'lightgrey']

# Plotting
plt.pie(slices, labels=depts, colors=colors, startangle=90, explode=(0,0, 0, 0.1, 0, 0, 0, 0, 0, 0, 0), shadow=True, autopct='%1.1f%%')

# Title
plt.title('Dell Employee Distribution by Department')

# Display the pie chart
plt.show()
```



Contact Us:-



Footer:-



Activity 2: Build python code

Task 1: Importing Libraries

The first step is usually importing the libraries that will be needed in the program.

```
<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 = "https://teachablemachine.withgoogle.com/models/r8PRIVDXo/";

    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();
```

Importing the flask module in the project is mandatory. An object of the Flask class is our WSGI application. Flask constructor takes the name of the current module (_name_) as argument Pickle library to load the model file.

Task 2: Creating our flask application and loading our model by using load_model method

```
# Define a flask app
app = Flask(__name__)
# Load your trained model
model = load_model('Garbage1.h5')
```

Task 3: Routing to the html Page

Here, the declared constructor is used to route to the HTML page created earlier.

In the above example, '/' URL is bound with index.html function. Hence, when the home page of a web server is opened in the browser, the html page will be rendered. Whenever you browse an image from the html page this photo can be accessed through POST or GET Method.

```
        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>
```

Showcasing prediction on UI:

```
@app.route('/predict', methods=['GET', 'POST'])
def upload():
    if request.method == 'POST':
        # Get the file from post request
        f = request.files['image']

        # Save the file to ./uploads
        basepath = os.path.dirname(__file__)
        file_path = os.path.join(
            basepath, 'predictions',f.filename)
        f.save(file_path)
        img = image.load_img(file_path, target_size=(128, 128))
        x = image.img_to_array(img)
        x = np.expand_dims(x, axis=0)


        preds = model.predict_classes(x)
        index = ['cardboard','glass','metal','paper','plastic','trash']
        text = "The Predicted Garbage is : "+str(index[preds[0]])

                # ImageNet Decode

        return text
```

Here we are defining a function which requests the browsed file from the html page using the post method. The requested picture file is then saved to the uploads folder in this same directory using OS library. Using the load image class from Keras library we are retrieving the saved picture from the path declared. We are applying some image processing techniques and then sending that preprocessed image to the model for predicting the class. This returns the numerical value of a class (like 0,1 ,2 etc.) which lies in the 0th index of the variable preds. This numerical value is passed to the index variable declared. This returns the name of the class. This name is rendered to the predict variable used in the html page.

Predicting the results

We then proceed to detect all type of Garbage in the input image using model.predict function and the result is stored in the result variable.

```
a=np.argmax(model.predict(x), axis=1)
```

```
index=['0', '1', '2', '3', '4','5']
result = str(index[a[0]])
result
```

```
'3'
```

```
train_transform.class_indices
```

```
{'cardboard': 0, 'glass': 1, 'metal': 2, 'paper': 3, 'plastic': 4, 'trash': 5}
```

Finally, Run the application

This is used to run the application in a local host.

Activity 3:Run the application

- Open the anaconda prompt from the start menu.
- Navigate to the folder where your app.py resides.
- Now type "python app.py" command.
- It will show the local host where your app is running on **http://127.0.0.1.5000/**
- Copy that local host URL and open that URL in the browser. It does navigate me to where you can view your web page.
- Enter the values, click on the predict button and see the result/prediction on the web page.

```
(base) I:\>cd I:\SmartBridge Projects\All SmartBridge Projects\Garbage

(base) I:\SmartBridge Projects\All SmartBridge Projects\Garbage>python app.py
```

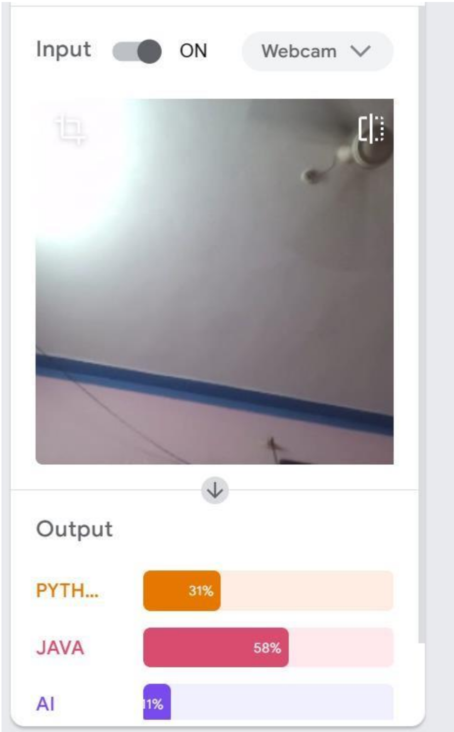
Then it will run on localhost:5000

```
* Serving Flask app "app" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Navigate to the localhost (http://127.0.0.1:5000/)where you can view your web page.

FINAL OUTPUT:

Output 1:



Output 2: