**Image Processing Using Transfer learning**

**Project Description:**

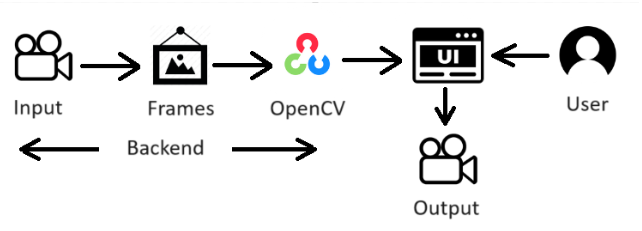
This project aims to develop a system for automatically classifying brain tumors using VGG-16, a pre-trained convolutional neural network (CNN) model

Early and accurate diagnosis of brain tumors is crucial for effective treatment planning and improved patient outcomes. Traditional methods often rely on expert analysis of medical images, which can be time-consuming, prone to human error, and subjective.

This project leverages the power of deep learning to:

* Automate brain tumor classification: The VGG-16 model will be trained on a dataset of brain MRI scans labeled with different types of tumors (e.g., benign, malignant). This trained model will then be able to analyze new MRI scans and predict the presence and type of tumor with high accuracy.
* Reduce human workload and improve efficiency: Automating tumor classification can free up radiologists' time for more complex tasks, potentially leading to faster diagnoses and improved patient care.
* Provide objective and consistent results: Deep learning models can learn complex patterns from data, potentially achieving higher consistency and objectivity in classification compared to traditional methods.

**Technical Architecture:**



**Pre-requisites:**

**To complete this project, you must require the following software, concepts, and packages.**

1. **IDE Installation**:

VS Code IDE is Ideal to complete this project

To install **VS Code**, please refer to [VS Code IDE Installation Steps](https://www.youtube.com/watch?v=naL0cZNQh1g)

1. **Python Packages**

If you are using **anaconda navigator**, follow the below steps to download the required packages:

Open the Anaconda prompt

* Type “pip install tensorflow==2.15.0” and click enter.
* Type “pip install numpy ==1.26.3” and click enter
* Type "pip install Flask” and click enter.

**Prior Knowledge:**

You must have prior knowledge of the following topics to complete this project.

* OpenCV - <https://www.youtube.com/watch?v=WQeoO7MI0Bs>
* Flask - <https://www.youtube.com/watch?v=lj4I_CvBnt0>

# Project Objectives:

By the end of this project, you will:

* Know fundamental concepts and techniques used for computer vision.
* Gain knowledge of CNN and transfer learning.

# Project Flow:

* The user interacts with the UI to enter the input.
* Entered input is analyzed by the model which is integrated.
* Once the model analyses the input the summary is showcased on the UI

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

**Create app.py python file :**

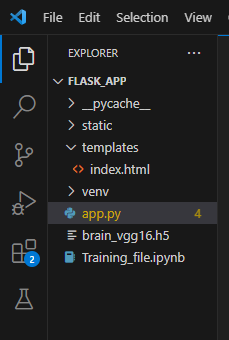
* Import the required libraries

**UI Integration:**

* Building HTML Pages
* Build Python code
* Run the application

**Project Structure:**

The Template folder contains HTML pages. The app.py contains the python code used . you'll find HTML pages, while app.py comprises the python code responsible for counting the pushups.



**Milestone 1:** **Create app.py python file**

**Activity 1: Import the required libraries**

We will be importing the necessary packages initially.



**Activity 2: Image Processing experience with Python:**

**Explanation of below code :**  
**Imports:**

**1. Importing Libraries:**

* Image and model-related libraries:
* tensorflow.keras.layers: Provides layers for building neural networks.
* tensorflow.keras.models: Houses the model class for defining architectures.
* tensorflow.keras.preprocessing.image: Offers tools for image preprocessing.
* tensorflow.keras.applications.vgg16: Contains the pre-trained VGG16 model.
* Sequential: A simple model type for linear stacking of layers.

Data manipulation and visualization:

* glob: Used for finding files matching a pattern.
* numpy: Provides numerical computation capabilities.
* matplotlib.pyplot: Used for plotting graphs.

**2. Loading the VGG16 Model:**

**Image size and paths:**

* imageSize set to [224, 224], the input size for VGG16.
* train\_path and test\_path store the paths to training and testing image folders.

**Loading VGG16:**

vgg = VGG16(...): Creates a VGG16 model instance, excluding the top layer.

weights='imagenet': Loads pre-trained weights trained on ImageNet.

include\_top=False: Excludes the top classification layers for customization.

**3. Adding Custom Layers:**

**Freezing pre-trained layers:**

for layer in vgg.layers: layer.trainable = False: Prevents weights updates for pre-trained layers.

**Flattening:**

x = Flatten()(vgg.output): Flattens the convolutional output into a 1D vector.

**Output layer:**

prediction = Dense(...): Adds a fully connected layer with appropriate output neurons for the number of classes.

activation='softmax': Makes predictions probabilities for multi-class classification.

**4. Creating the Complete Model:**

**Model construction:**

model = Model(inputs=vgg.input, outputs=prediction): Defines the model with VGG16's input and the custom output layer.

**Model summary:**

model.summary(): Prints an overview of the model's architecture and layers.

**5. Compiling the Model:**

Specifying loss, optimizer, and metrics:

model.compile(...): Configures the model's learning process.

loss='categorical\_crossentropy': Suitable for multi-class classification.

optimizer='adam': A common and efficient optimization algorithm.

metrics=['accuracy']: Measures accuracy during training and evaluation.

**6. Data Augmentation:**

**Creating image generators:**

train\_datagen and test\_datagen for training and testing data, respectively.

rescale: Scales pixel values for better training.

shear\_range, zoom\_range, horizontal\_flip (for training): Apply random transformations to increase data diversity and prevent overfitting.

7. Loading Data:

**Using ImageDataGenerator:**

flow\_from\_directory: Reads and loads images from directories, automatically classifying them based on folder structure.

8. Training the Model:

**Initiating training:**

model.fit\_generator(...): Trains the model using the loaded image generators.

epochs: Number of training iterations over the entire dataset.

9. Plotting Loss and Accuracy:

**Visualizing training progress:**

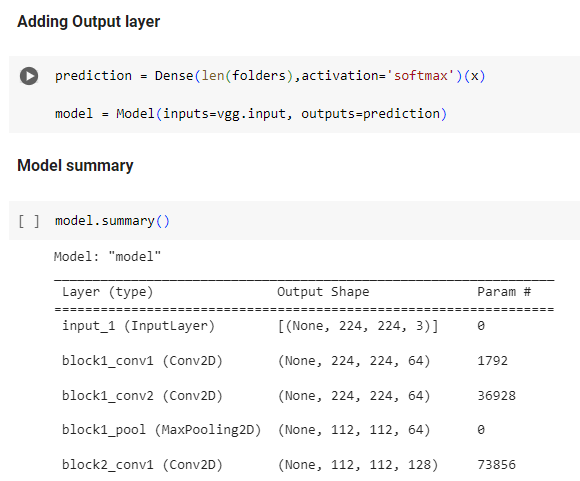
plt.plot(...): Plots loss and accuracy curves for both training and validation data.

10. Saving the Trained Model:

**Preserving trained weights:**

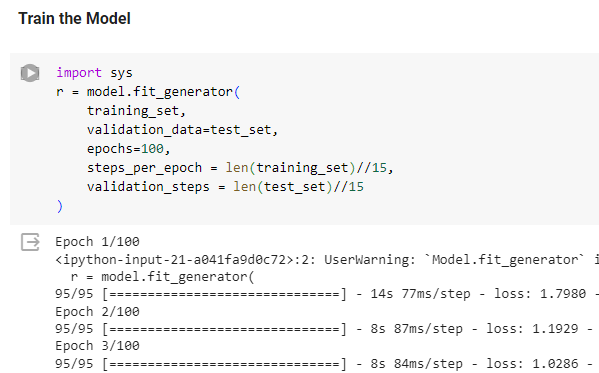
model.save('brain\_vgg16.h5'): Saves the model for future use.

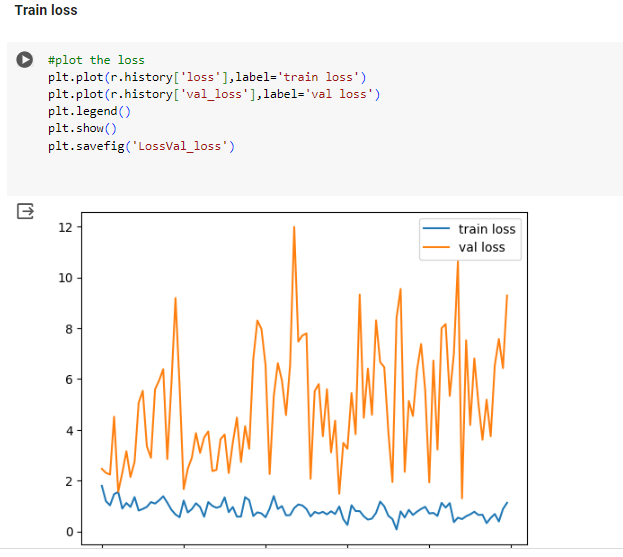




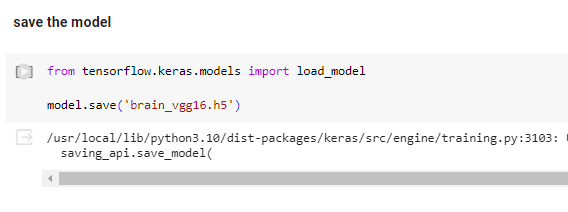












# Milestone 2: UI Integration

In this section, we will be building a web application that is integrated into the model we built. A UI is provided for the users where he/she has to navigate to open the web cam.

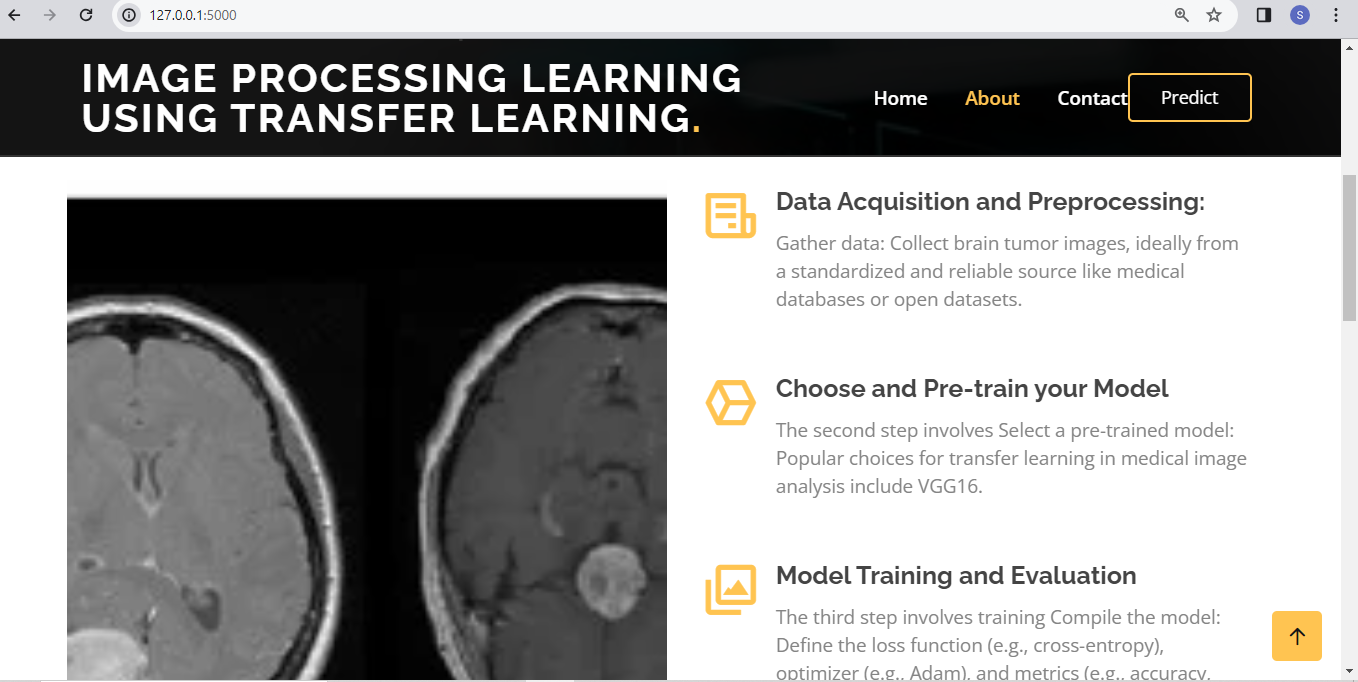
This section has the following tasks

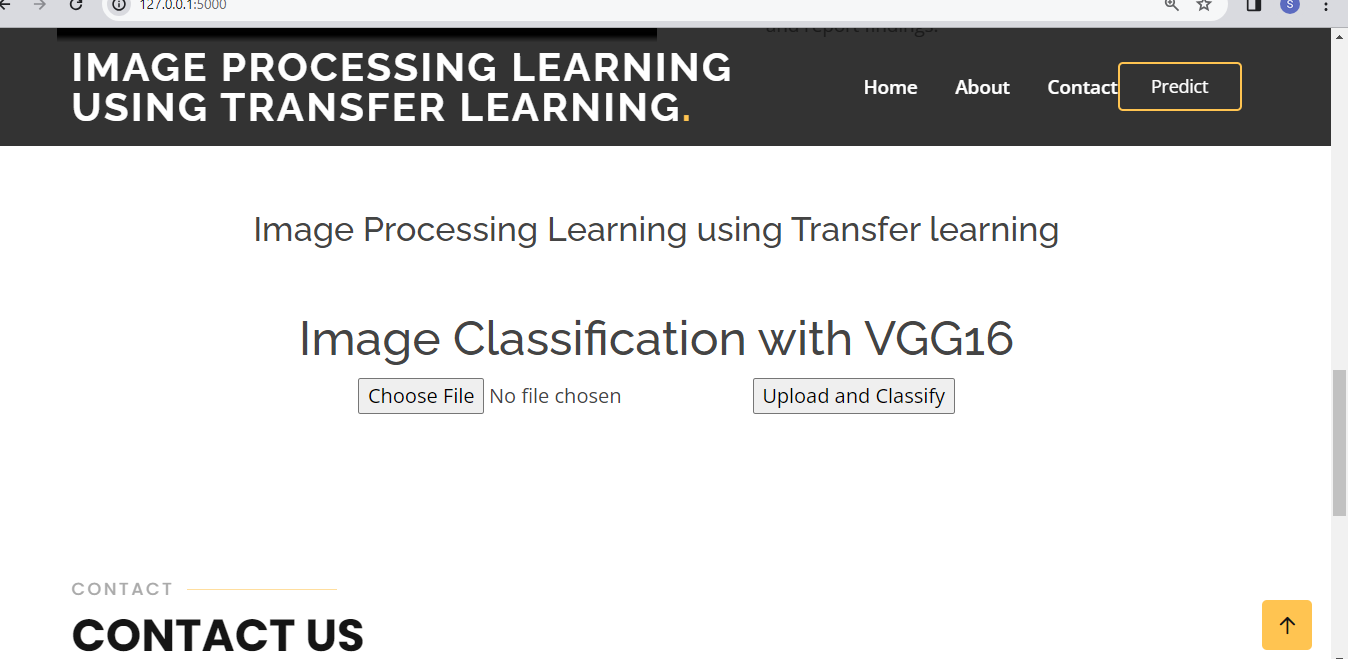
* Building HTML Pages
* Building server-side script

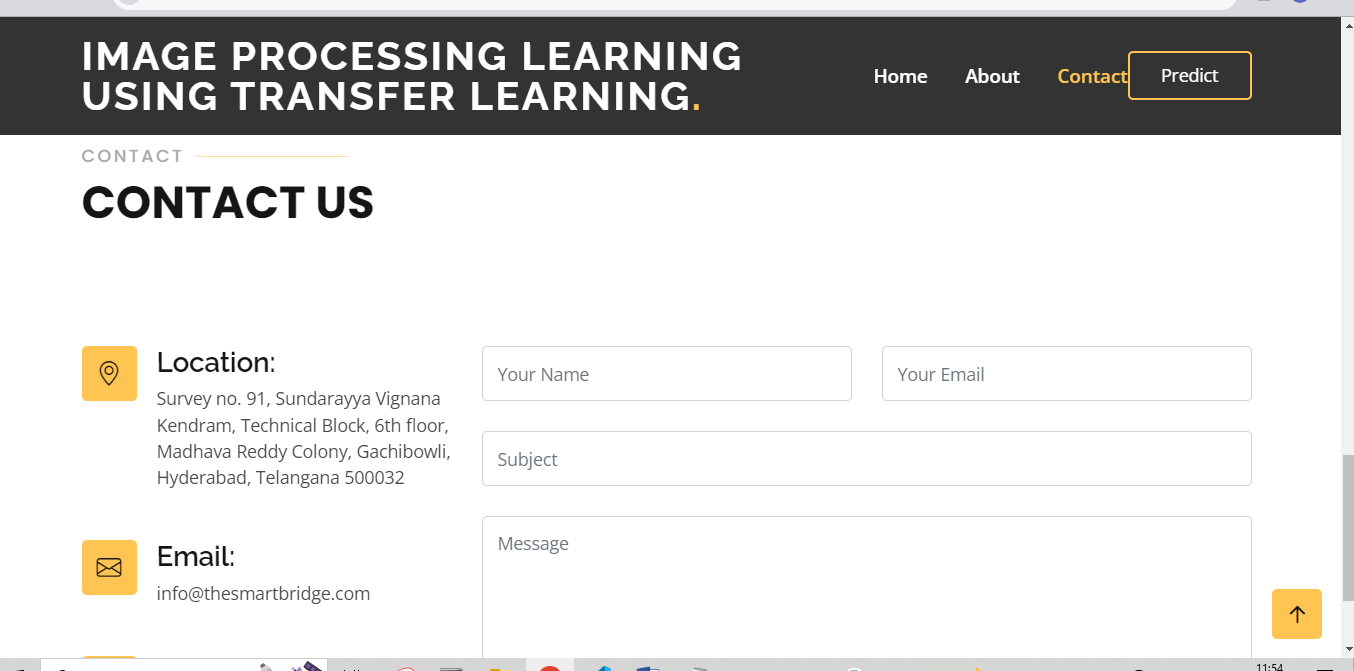
**Activity1: Building Html Pages:**

We have created HTML files for this project and saved them in the templates folder.

Let’s see how those html pages looks like:

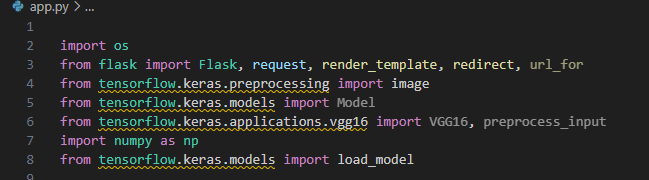




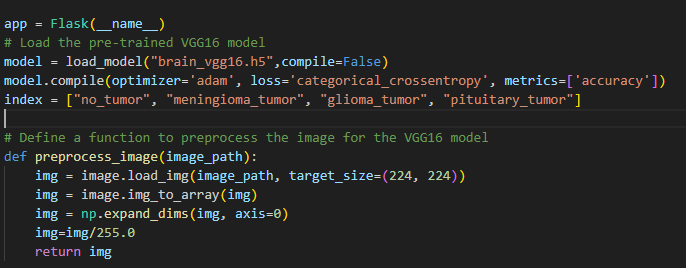


**Activity 2: Build Python code:**

* Import the libraries

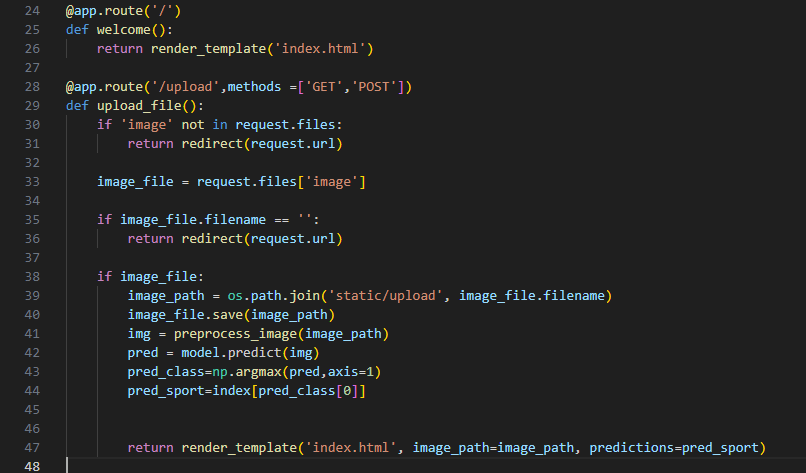


* Importing the Flask module into the project is mandatory. An object of the Flask class is our WSGI application. The Flask constructor takes the name of the current module (\_\_name\_\_) as an argument.

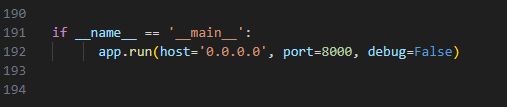


Render HTML page:

* Here we will be using the declared constructor to route to the HTML page that we have created earlier. In the above example, the ‘/’ URL is bound with the index.html function. Hence, when the home page of the web server is opened in the browser, the HTML page will be rendered.

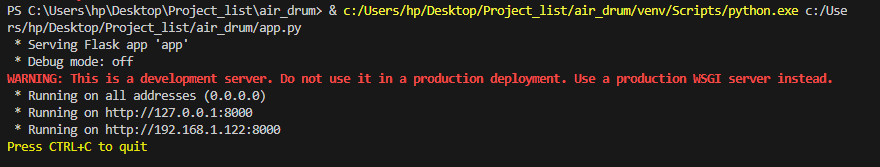


Main Function:



**Activity 3: Run the application**

* Open the anaconda prompt from the start menu
* Navigate to the folder where your Python script is.
* Now type the “python app.py” command
* Navigate to the localhost where you can view your web page.
* Click on the predict button from the top right corner, enter the inputs, click on the submit button, and see the result/prediction on the web.



Output:

