**Yoga Pose Classification**

**Project Description:**

Yoga is a 5000-year-old practice developed in ancient India by the Indus-Sarasvati civilization. The word yoga means deep association and union of mind with the body. It is used to keep both mind and body in equilibration in all flip-flops of life by means of asana, meditation, and several other techniques. Nowadays, yoga has gained worldwide attention due to increased stress levels in the modern lifestyle, and there are numerous methods or resources for learning yoga. Yoga can be practiced in yoga centers, through personal tutors, and can also be learned on one’s own with the help of the Internet, books, recorded clips, etc. In fast-paced lifestyles, many people prefer self-learning because the abovementioned resources might not be available all the time. But in self-learning, one may not find an incorrect pose. Incorrect posture can be harmful to one’s health, resulting in acute pain and long-term chronic concerns.  In this Project Deep Learning based techniques are developed to detect yoga pose by uploading an image or by real time video using computer vision technique. It detects the pose and tells exactly the pose name.

Transfer learning has become one of the most common techniques that has achieved better performance in many areas, especially in medical image analysis and classification. We used Transfer Learning techniques like Inception V3,Resnet50,Xception, VGG19 that are more widely used as a transfer learning method in medical image analysis and they are highly effective.

**Technical Architecture:**

# 

**Prerequisites:**

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

* **Anaconda navigator and PyCharm / Spyder:**
  + Refer the link below to download anaconda navigator
  + Link (PyCharm) : <https://youtu.be/1ra4zH2G4o0>
  + Link (Spyder) : <https://youtu.be/5mDYijMfSzs>
* **Python packages:**
  + Open anaconda prompt as administrator
  + Type “pip install numpy” and click enter.
  + Type “pip install pandas” 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.

# Prior Knowledge:

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

* **Deep Learning Concepts** 
  + **CNN:** <https://towardsdatascience.com/basics-of-the-classic-cnn-a3dce1225add>
  + **VGG16:** <https://medium.com/@mygreatlearning/what-is-vgg16-introduction-to-vgg16-f2d63849f615>
  + **ResNet-50:** <https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33>
  + **Inception-V3:** <https://iq.opengenus.org/inception-v3-model-architecture/>
  + **Xception:** <https://pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/>
* **Computer Vision**
  + <https://docs.opencv.org/4.x/dd/d43/tutorial_py_video_display.html>
* **Flask:** Flask is a popular Python web framework, meaning it is a third-party Python library used for developing web applications.

Link: [**https://www.youtube.com/watch?v=lj4I\_CvBnt0**](https://www.youtube.com/watch?v=lj4I_CvBnt0)

# Project Objectives:

# By the end of this project you’ll understand:

# ● preprocessing the images.

# ● Applying Transfer learning algorithms on the dataset.

# ● How deep neural networks detect the disease.

# ● You will be able to know how to find the accuracy of the model.

# ● you will be able to build web applications using the Flask framework.

# 

# Project Flow:

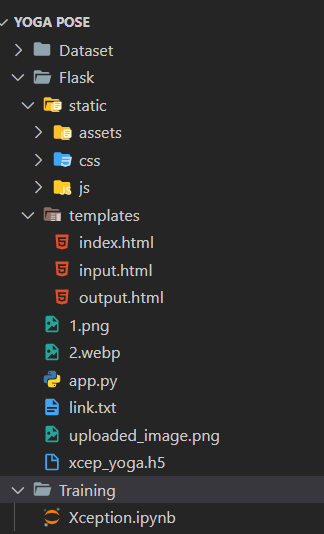
* The user interacts with the UI (User Interface) to choose the image.
* The chosen image analyzed by the model which is integrated with flask application.
* The Xception Model analyzes the image, then the prediction is showcased on the Flask UI.

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

* Data Collection.
  + Create a Train and Test path.
* Data Pre-processing.
* Import the required library
* Configure ImageDataGenerator class
* Apply ImageDataGenerator functionality to Trainset and Testset
* Model Building
  + Pre-trained CNN model as a Feature Extractor
  + Adding Dense Layer
  + Configure the Learning Process
  + Train the model
  + Save the Model
  + Test the model
* Application Building
  + Create an HTML file

# Project Structure:

Create a Project folder which contains files as shown below



● The Dataset folder contains the images for training and validating 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

● we need the model which is saved and the saved model in this content is a **xcep\_yoga.h5**

● templates folder contains index.html, input.html, output.html pages.

# Milestone 1: Data Collection

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

Yoga is a very well-known practice to curb anxiety and relieve stress and overall wellbeing. There are many yoga poses but the very well-known ones are the downward dog pose, goddess pose, tree pose, plank pose and the warrior pose. Smart technology can be used to classify between them.

The dataset is divided 5 sub folders in each directory corresponding to the 5 classes of yoga poses such as Goodess, Plank, Tree, Downdog, Warrior 2. The images are extracted from Bing using their API functionality and then they are manually cleaned, resized and compressed. You can download the dataset used in this project using the below link.

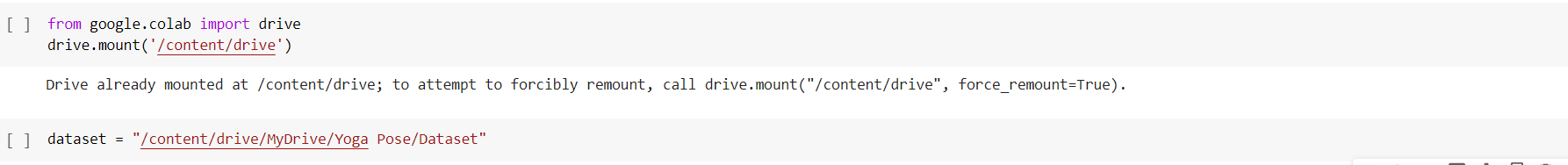
Dataset: - <https://www.kaggle.com/datasets/ujjwalchowdhury/yoga-pose-classification>

# Note: For better accuracy train on more images

We are going to build our training model on Google colab.

# Upload the dataset into google drive and connect the google colab with drive using the below code

# Once mounted the drive create a folder with project name and move the dataset to that folder and mount to that folder.



# Milestone 2: Model Building

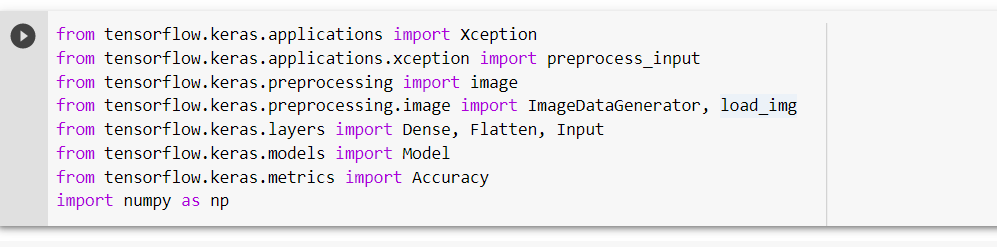
Now it's time to build our model. Let’s use the pre-trained model which is Xception, one of the convolution neural net (CNN) architecture which is considered as a very good model for Image classification.

Deep understanding on the Xception model – Link is referred to in the prior knowledge section. Kindly refer to it before starting the model building part.

**Activity 1: Importing the libraries**

Import the necessary libraries as shown in the image

.



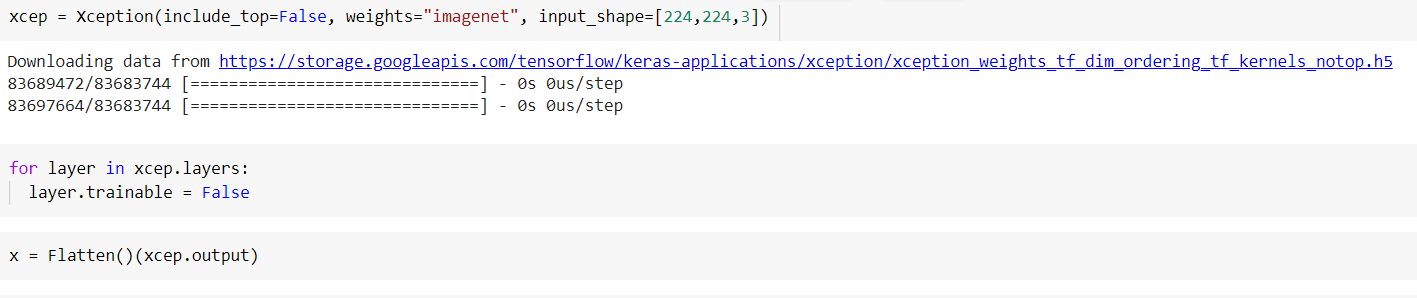
**Activity 2: Loading the model and adding Flatten Layers**

For one of the models, we will use it as a simple feature extractor by freezing all the five convolution blocks to make sure their weights don’t get updated after each epoch as we train our own model.

Here, we have considered images of dimension (224, 224, 3).

Also, we have assigned include\_top = False because we are using convolution layer for features extraction and wants to train fully connected layer for our images classification(since it is not the part of ImageNet dataset)

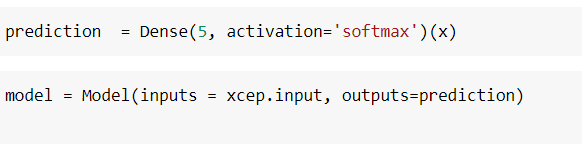
Flatten layer flattens the input. Does not affect the batch size.



**Activity 3: Adding Output Layer and Creating Model Object**

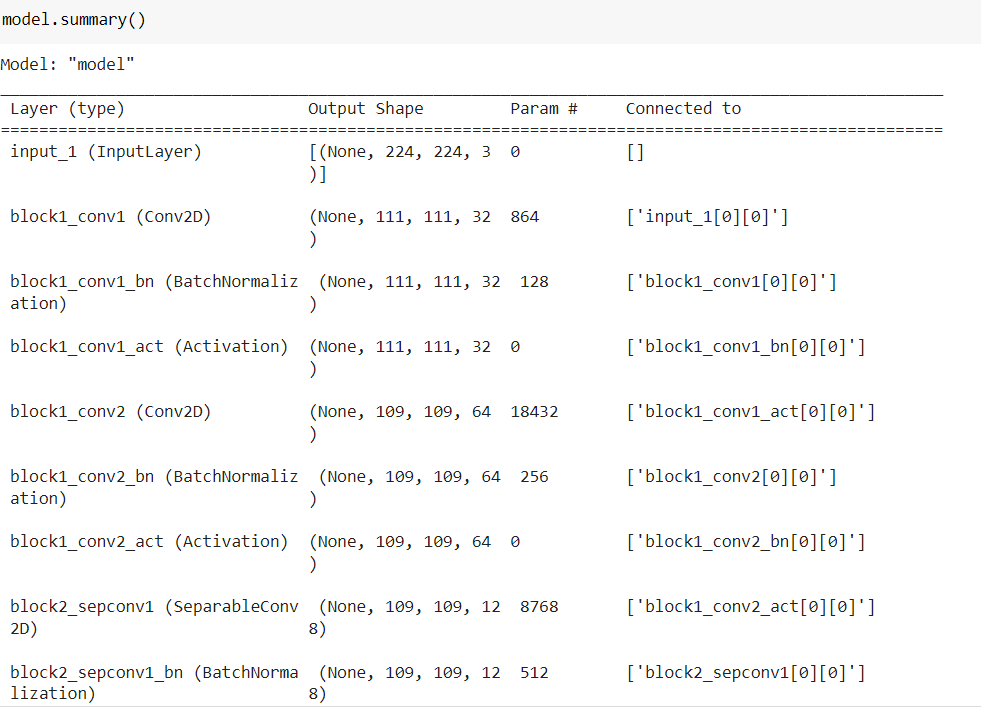
A dense layer is a deeply connected neural network layer. It is the most common and frequently used layer.

Let us create a model object named model with inputs as xception.input and output as dense layer.



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.

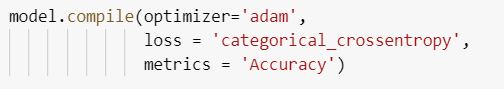


**Activity 4: Configure the Learning Process**

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 in the 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



**Activity 5: Import the ImageDataGenerator library**

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

Image data augmentation is a technique that can be used to artificially expand the size of a training dataset by creating modified versions of images in the dataset.

The Keras deep learning neural network library provides the capability to fit models using image data augmentation via the ImageDataGenerator class.

Let us import the ImageDataGenerator class from tensorflow Keras

import keras image

**Activity 6: Configure ImageDataGenerator class**

ImageDataGenerator class is instantiated and the configuration for the types of data augmentation

There are five main types of data augmentation techniques for image data; specifically:

* Image shifts via the width\_shift\_range and height\_shift\_range arguments.
* The image flips via the horizontal\_flip and vertical\_flip arguments.
* Image rotations via the rotation\_range argument
* Image brightness via the brightness\_range argument.
* Image zoom via the zoom\_range argument.

An instance of the ImageDataGenerator class can be constructed for train and test.



**Activity 7: Apply ImageDataGenerator functionality to data set**

Let us apply ImageDataGenerator functionality to the Train set and Test set by using the following code. For Training set using flow\_from\_directory function.

This function will return batches of images from the subdirectories

Arguments:

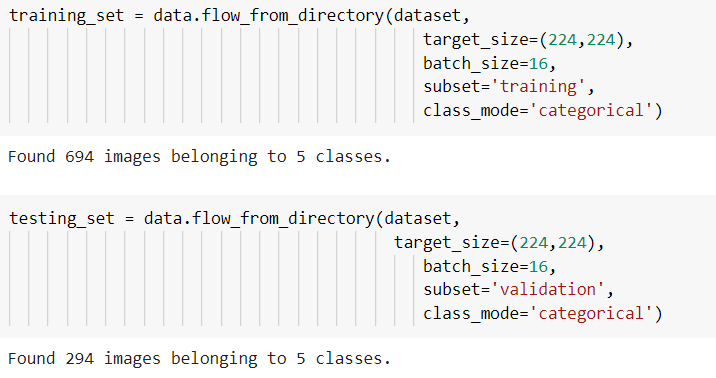
* Directory: Directory where the data is located. If labels are "inferred", it should contain subdirectories, each containing images for a class. Otherwise, the directory structure is ignored.
* batch\_size: Size of the batches of data which is 32.
* target\_size: Size to resize images after they are read from disk.
* class\_mode:

- ‘int': means that the labels are encoded as integers (e.g. for sparse\_categorical\_crossentropy loss).

- 'categorical' means that the labels are encoded as a categorical vector (e.g. for categorical\_crossentropy loss).

- 'binary' means that the labels (there can be only 2) are encoded as float32 scalars with values 0 or 1 (e.g. for binary\_crossentropy).

- None (no labels).



Total the dataset is having 694 train images and 294 test images divided under 5 classes

**Milestone 3: Training**

**Activity 1: Train the model**

Now, let us train our model with our image dataset. The model is trained for 25 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 10 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

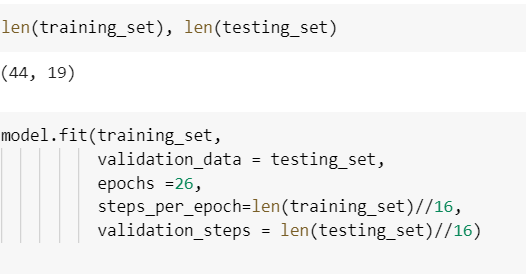
the loss and metrics for any model after any epoch has ended.

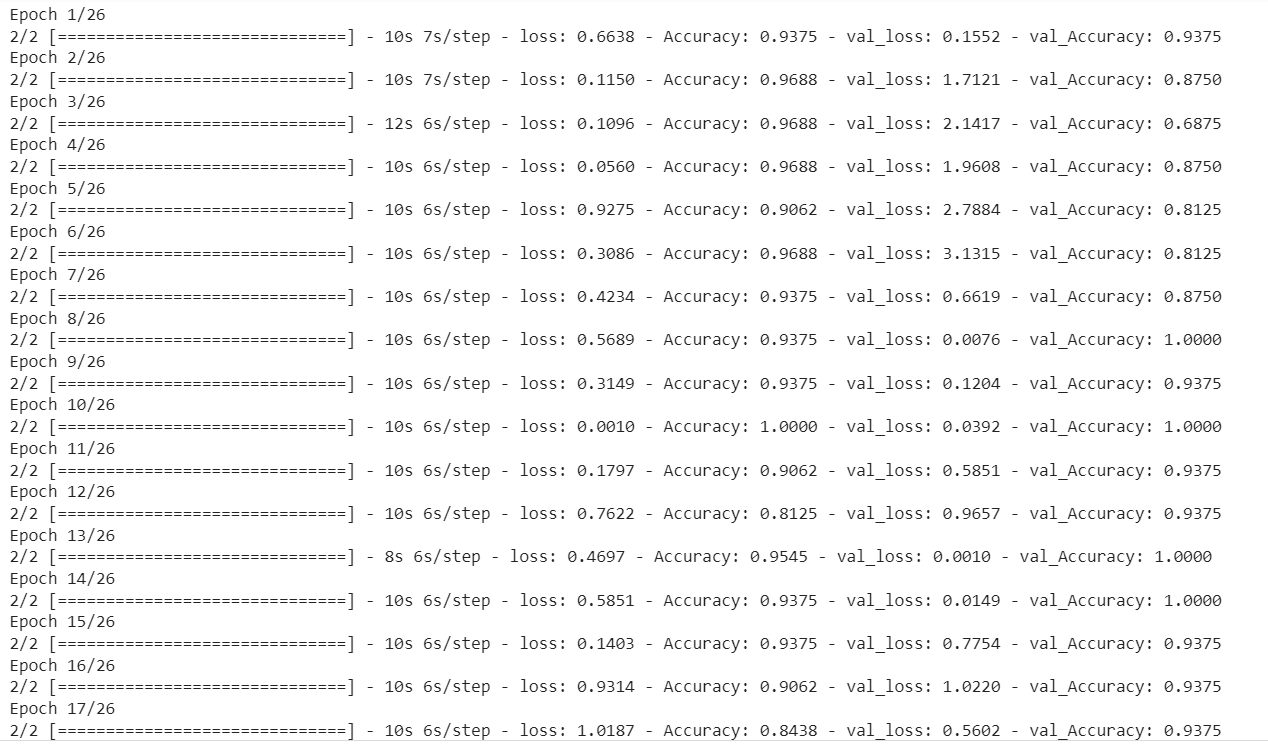
* validation\_steps: only if the validation\_data is a generator then only this argument

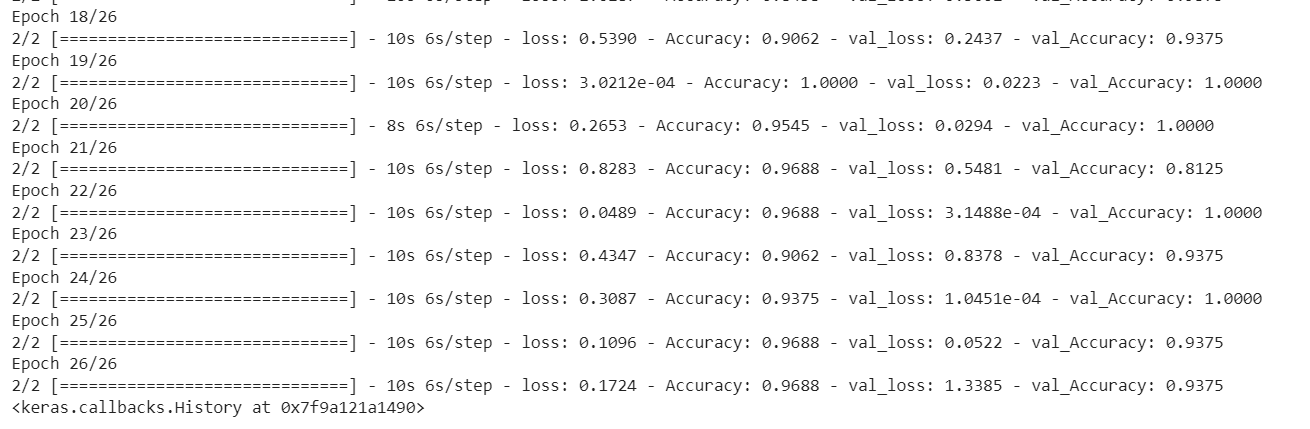
can be used. It specifies the total number of steps taken from the generator before it is

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.





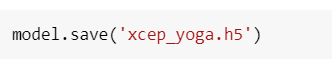


From the above run time, we can easily observe that at 26th epoch the model is giving the better accuracy as we can see Training Accuracy is 96% and Validation Accuracy is 93% .

# Activity 2: 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.

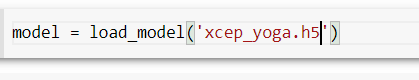


**Milestone 4: Testing**

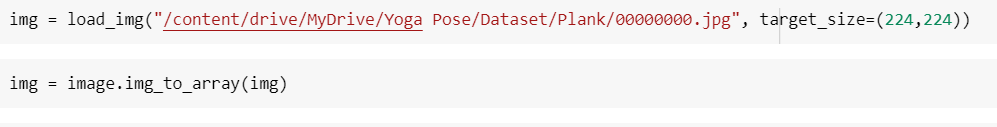
**Activity 1: Test the model**

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.

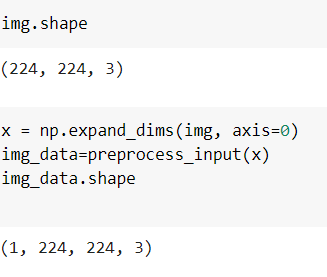
Load the saved model using load\_model



Taking an image as input and checking the results and preprocessing the input image as required for Xception model.



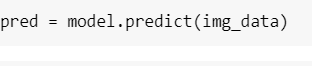
* Converting image to array by using img\_to\_array because from image we are getting the pixels and making those pixel into array for the model.



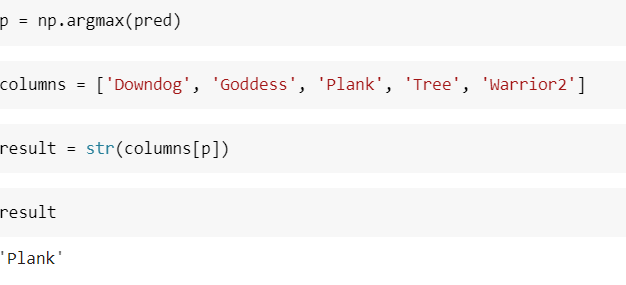
* By passing the array of image to no.expand\_dims we are converting the 3d array in 4d, because to tell with how many images we have in that array.
* preprocess\_inpur we are using to preprocess the input that into the required format of xception model.

**Note: We used Xception, for image is 224,224, so change according to the model**

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



The predicted class index name will be printed here.



# Milestone 5: Application Building

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.

This section has the following tasks

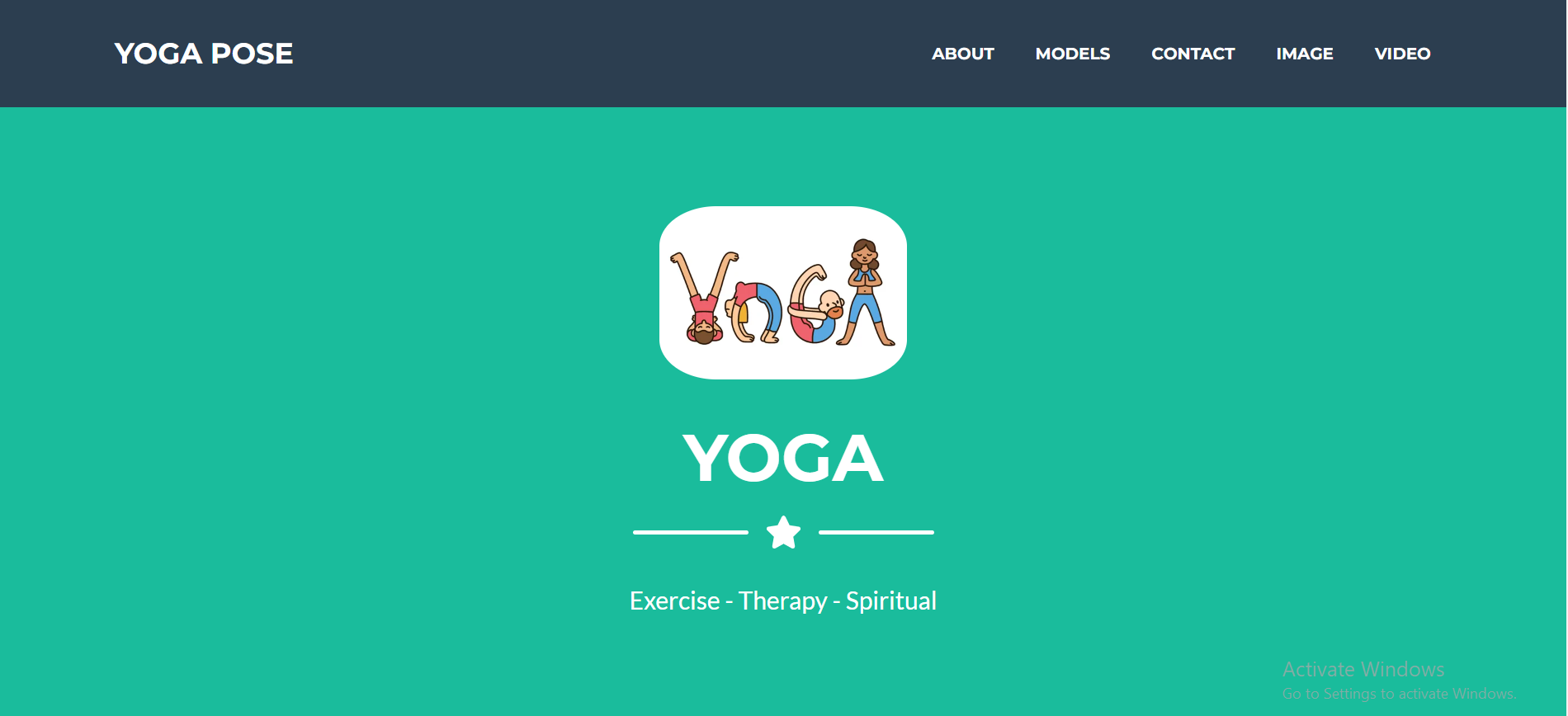
* Building HTML Pages
* Building server side script

**Activity 1: Building Html Pages:**

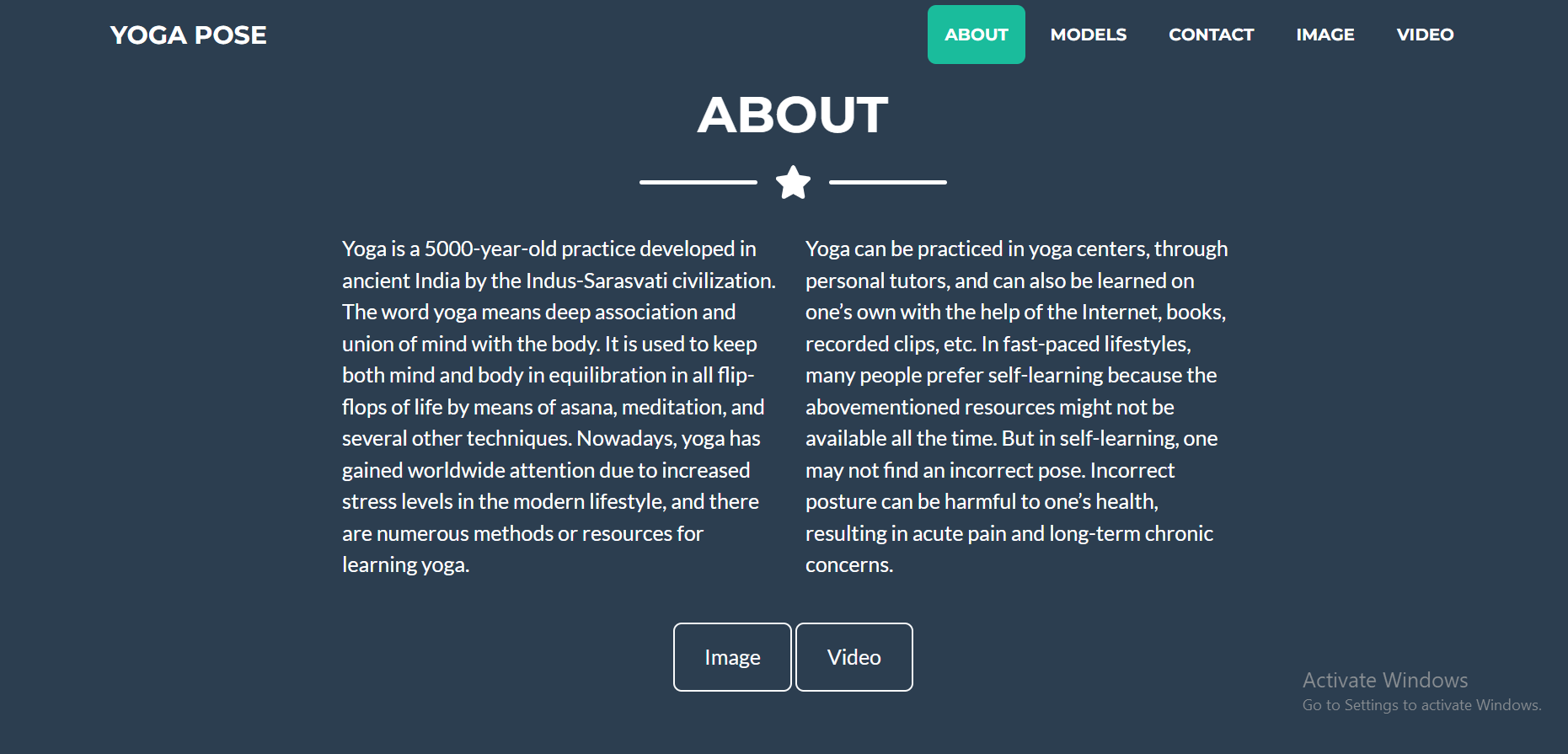
For this project create one HTML file namely

* Index.html

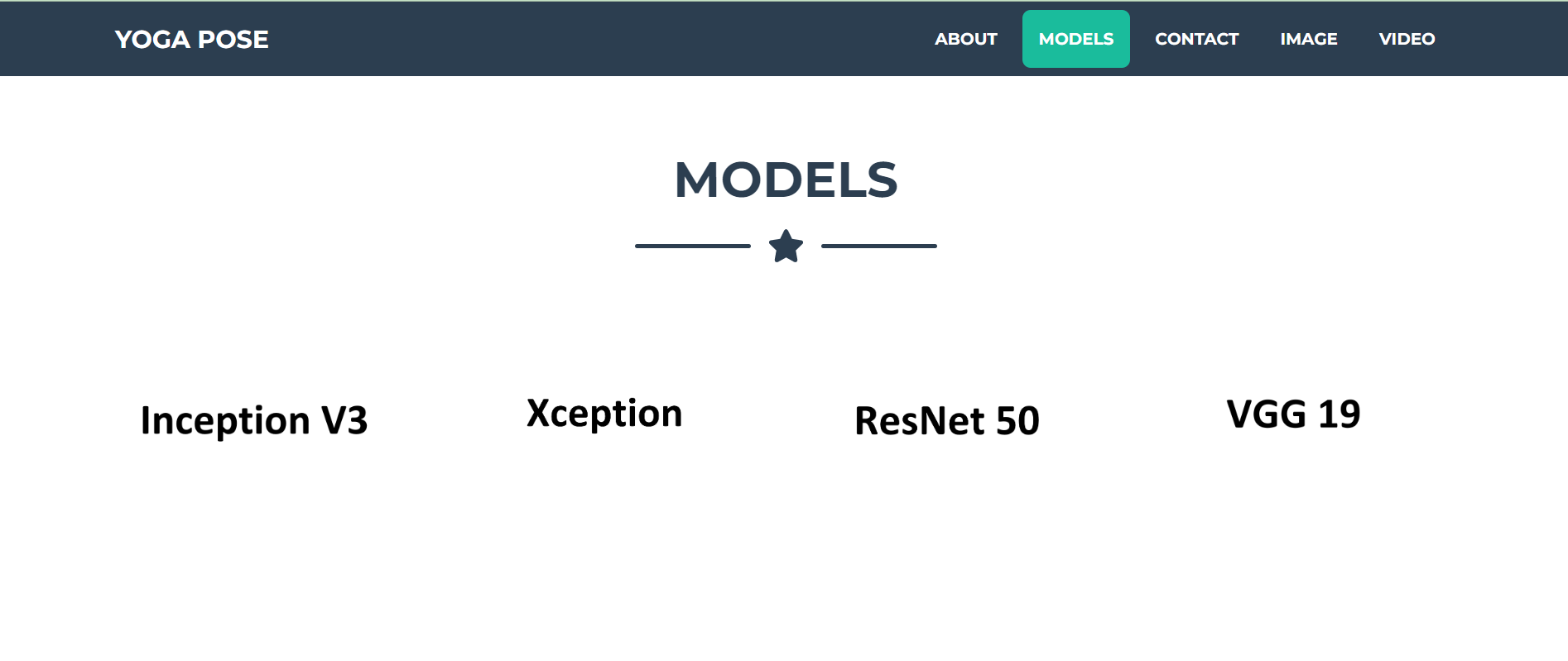
Let’s see how our index.html page looks like:

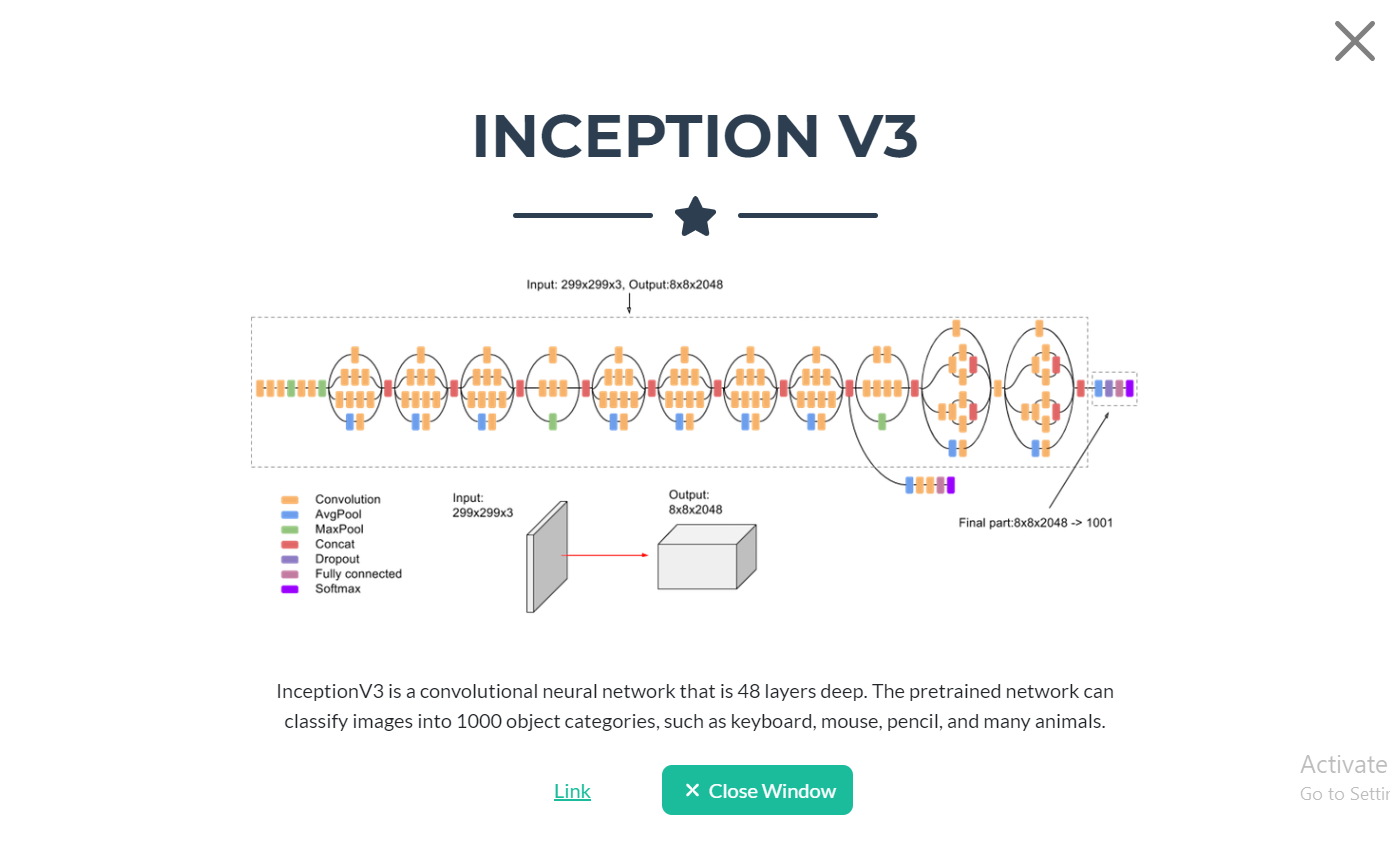


When you click on about button on the top, you will be redirecting to the following page

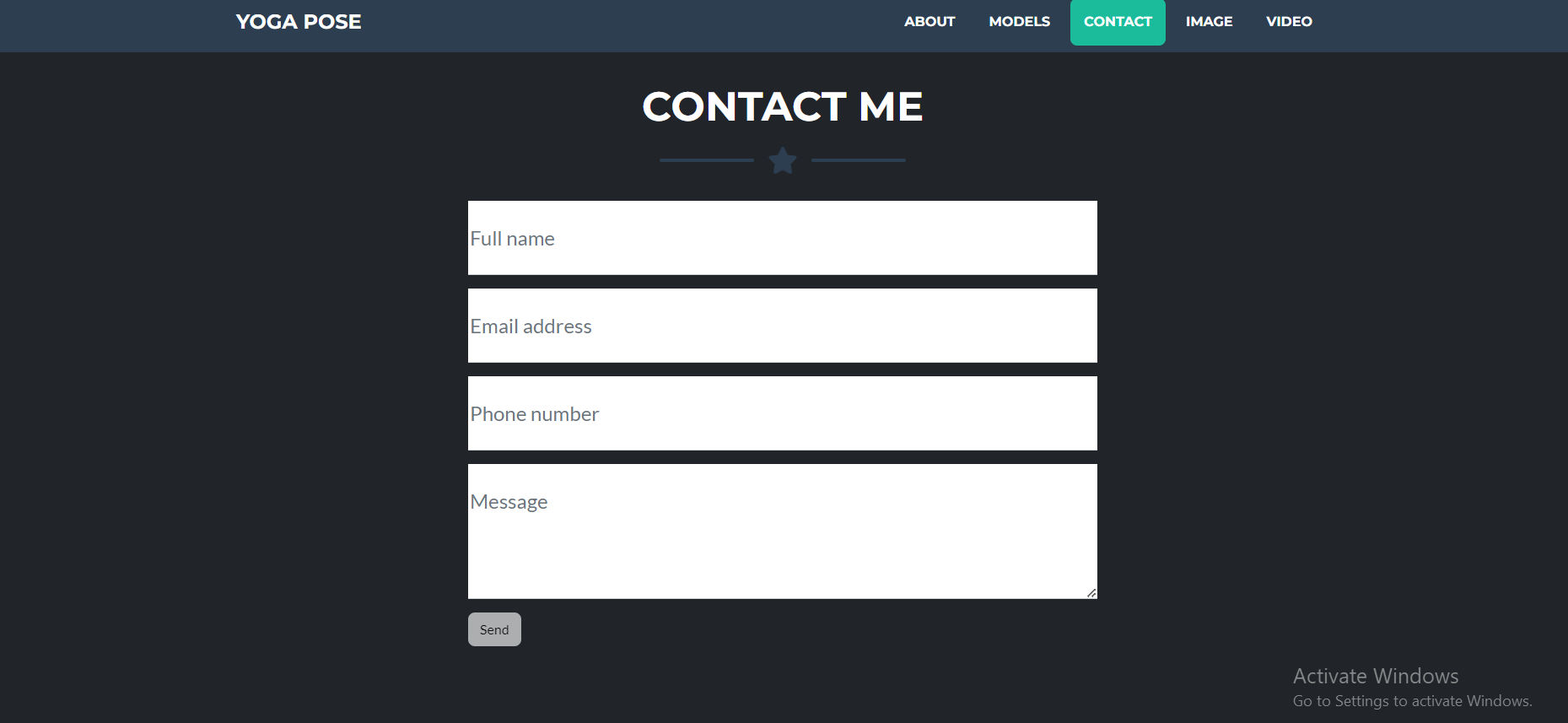


When you click on the models button, it will redirect you to the below page, when hover on the each model name you can expand and see the architecture and description of the model and link to study about the model.

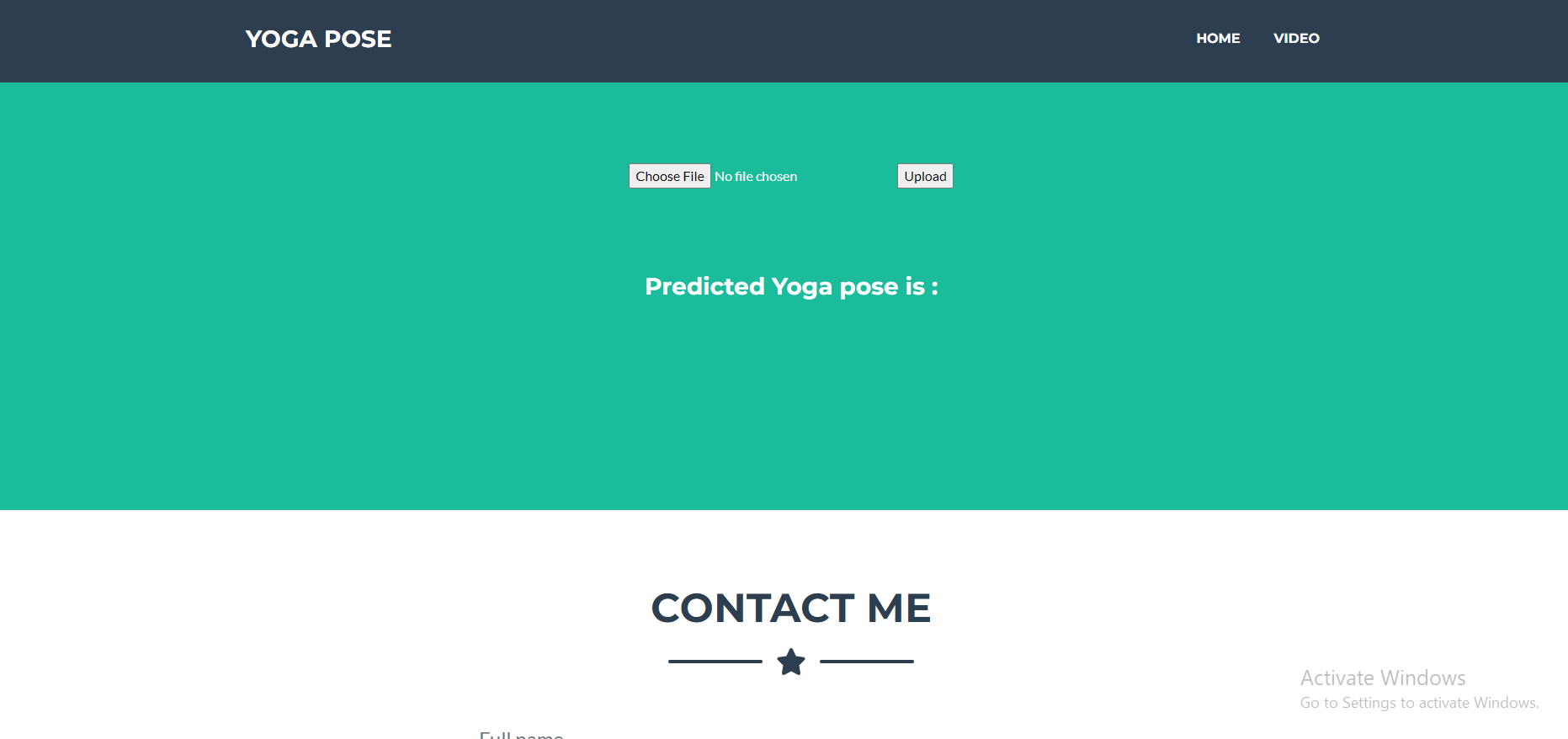




When you click on vaccinations, it will direct you the below page, where you can find the form to enter your details.

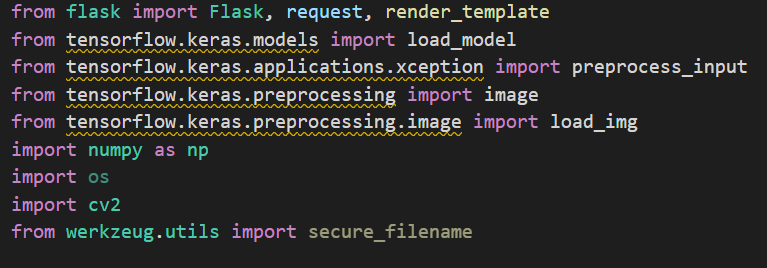


When you click on the Image button, it will display the below page, Where you can upload your image and get the prediction by clicking upload button.

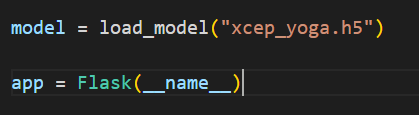


**Activity 2: Build Python code:**

Import the libraries

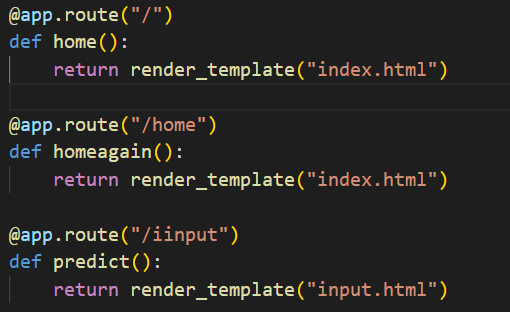


Loading the saved model and initializing the flask app

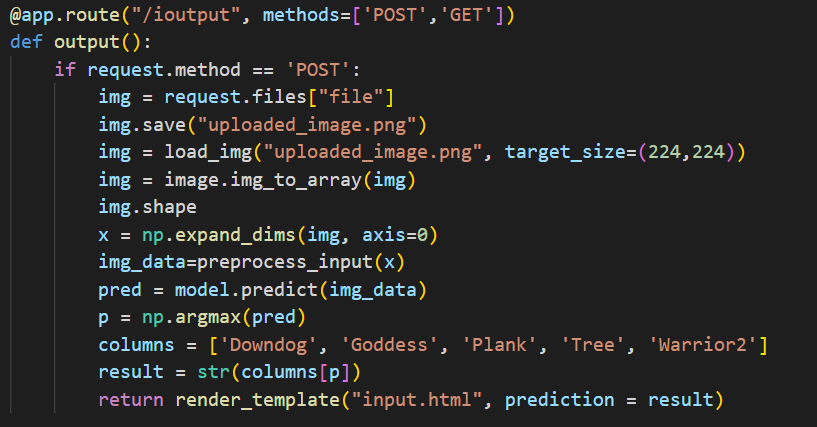


Render HTML pages:

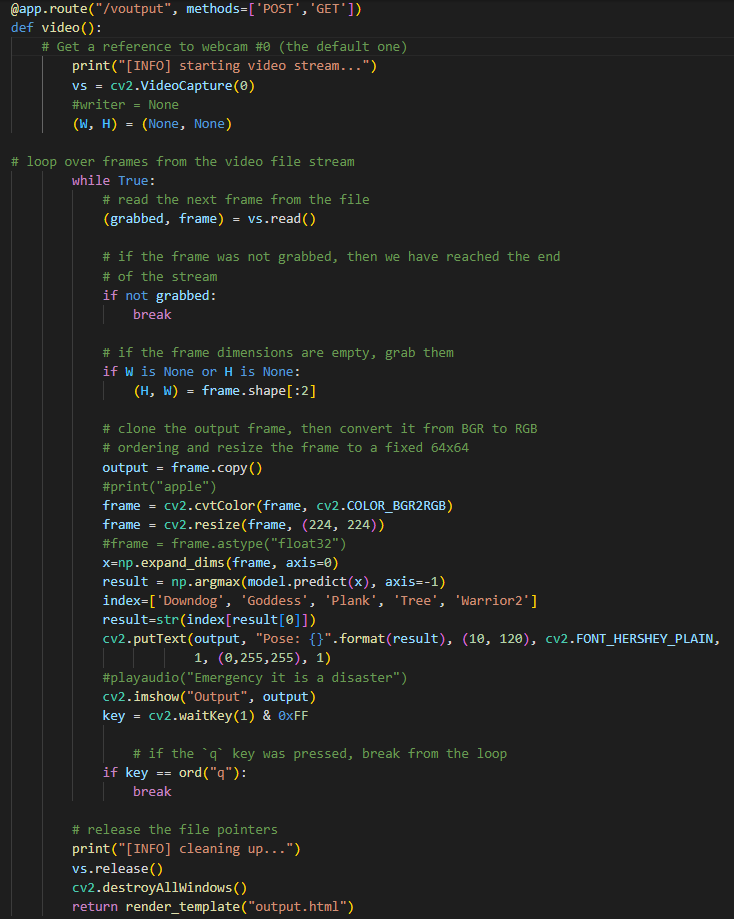
Once we uploaded the file into the app, then verifying the file uploaded properly or not. Here we will be using declared constructor to route to the HTML page which we have created earlier.



In the above example, ‘/’ URL is bound with index.html function. Hence, when the home page of the web server is opened in browser, the html page will be rendered. Whenever you enter the values from the html page the values can be retrieved using POST Method.

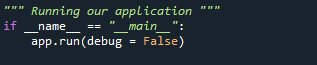


Here we are routing our app to res function. This function retrieves all the values from the HTML page using Post request. That is stored in an array. This array is passed to the model.predict () function. This function returns the prediction. And this prediction value will rendered to the text that we have mentioned in the input.html page earlier.



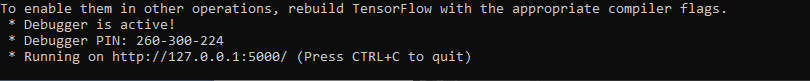
Here we are using Open CV to open the camera and take video, Video is nothing but the series of images, so we send it to the model for prediction and we display the predicted output in the camera its self. After closing the camera window by pressing q, it will redirect to output.html.

**Main Function:**

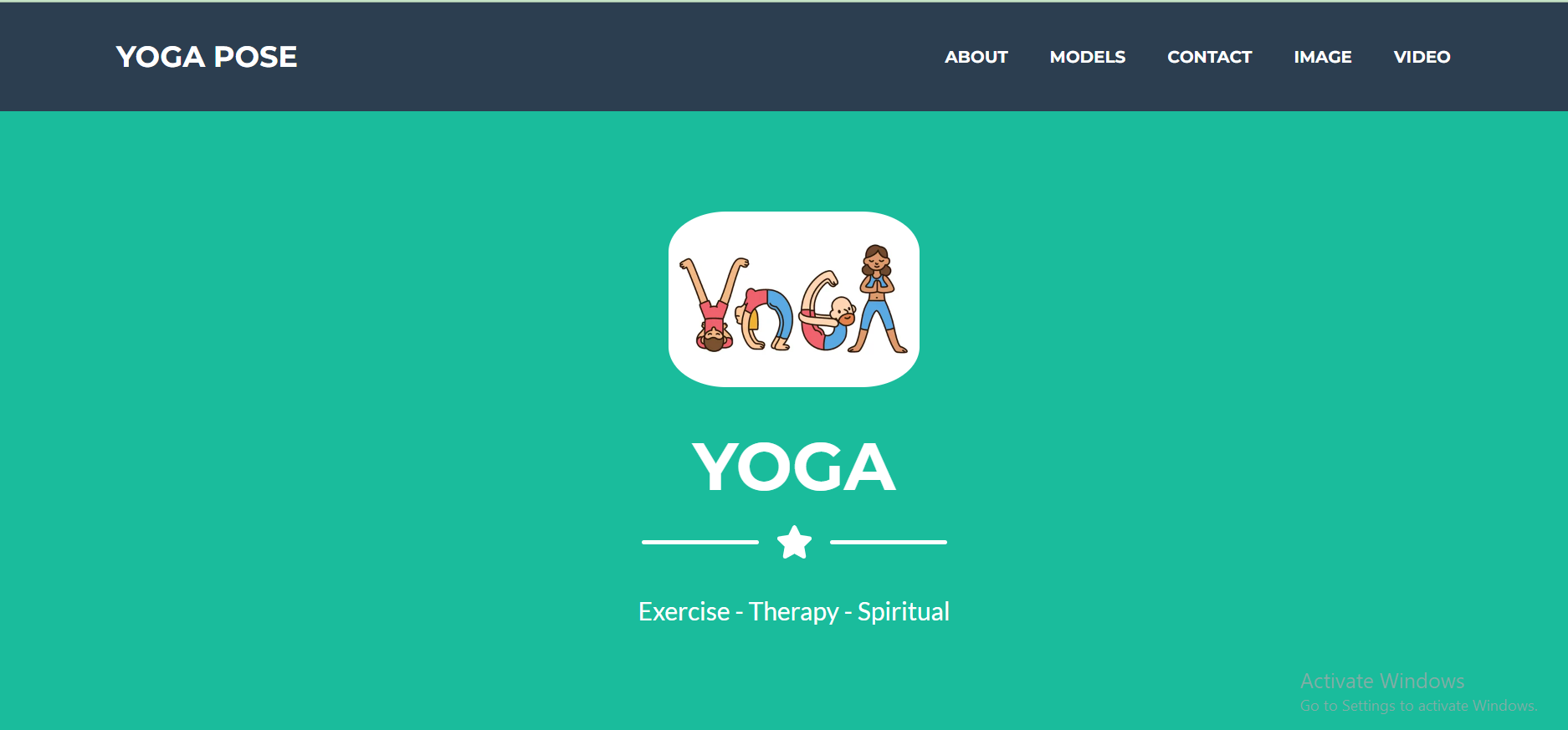


**Activity 3: Run the application**

* 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.

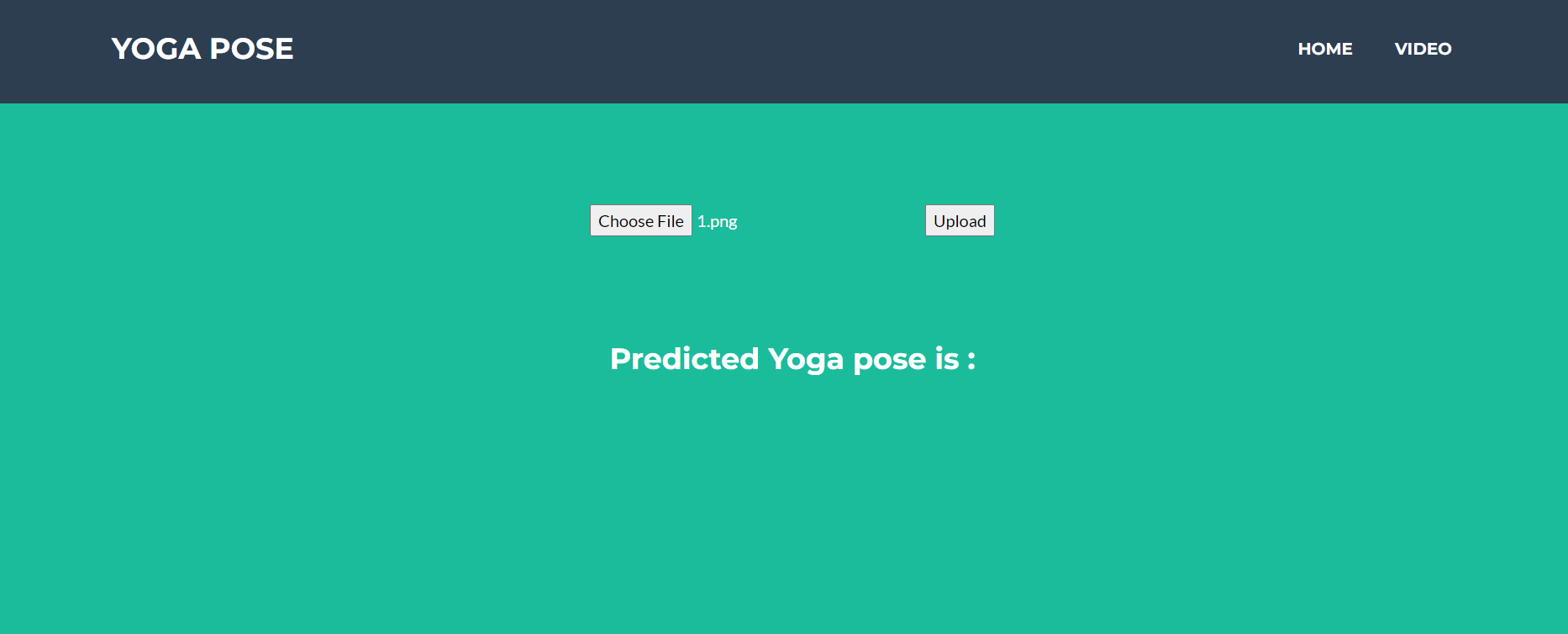


The home page looks like this. When you click on the button **Image** you’ll be redirected to the predict section



Click on image button

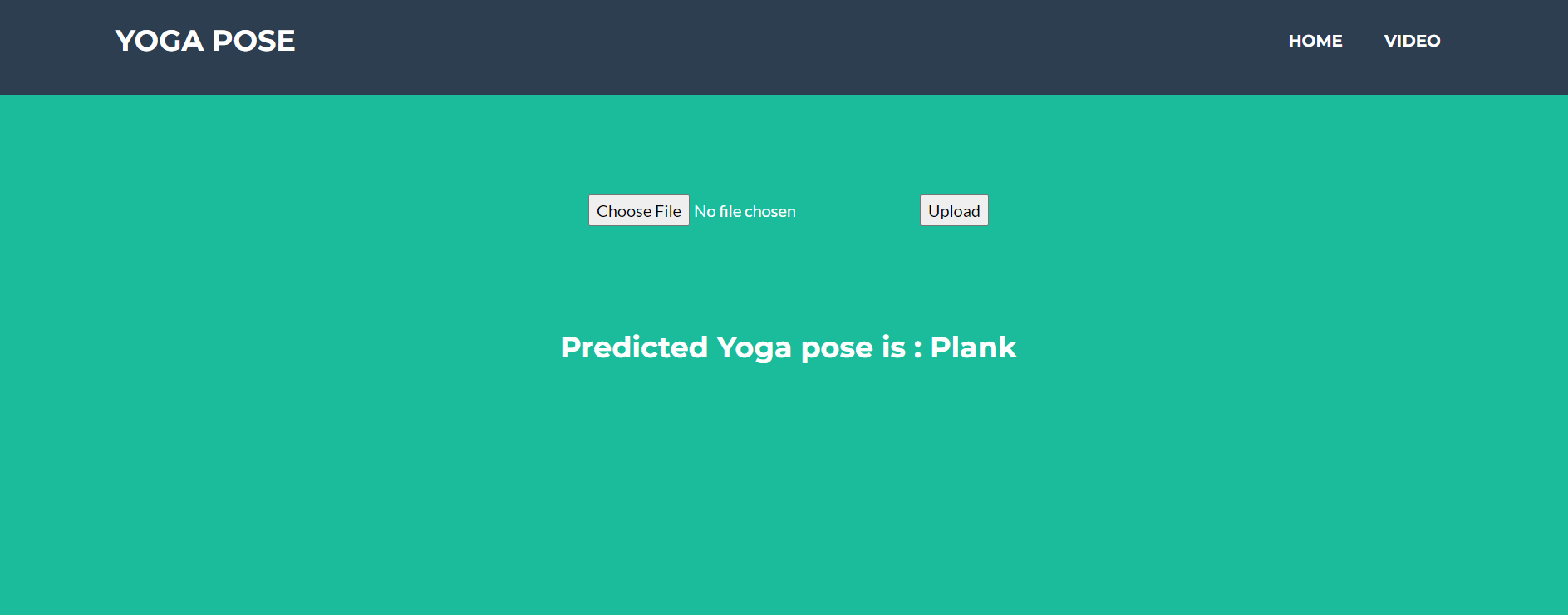
**Input 1:**



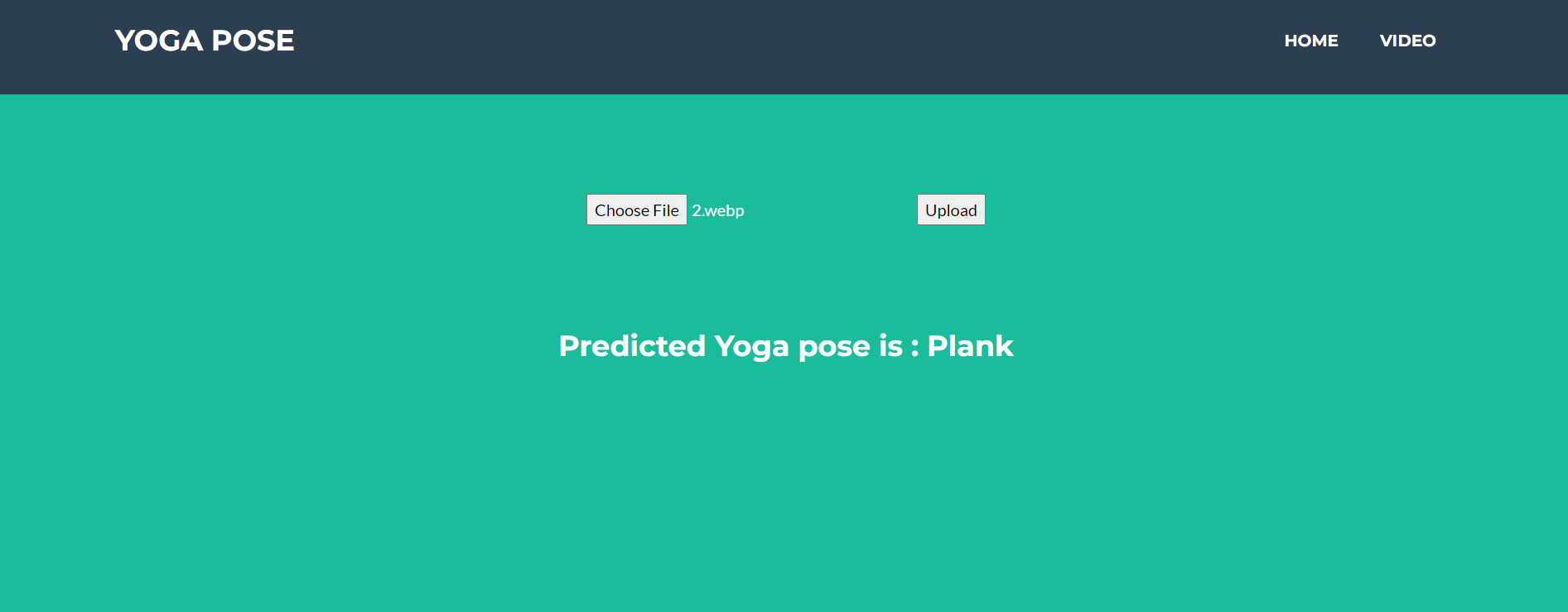
Once you upload the image and click on submit button, the output will be displayed in the below

Page

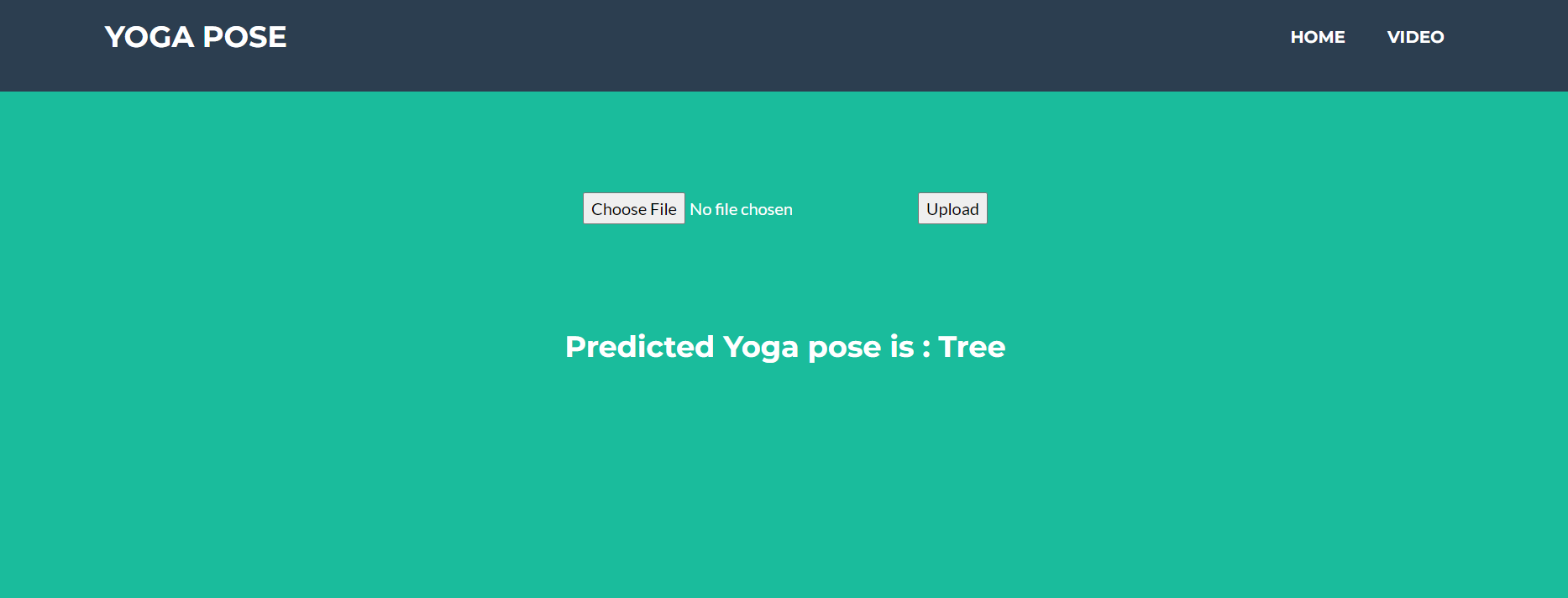
**Output: 1**



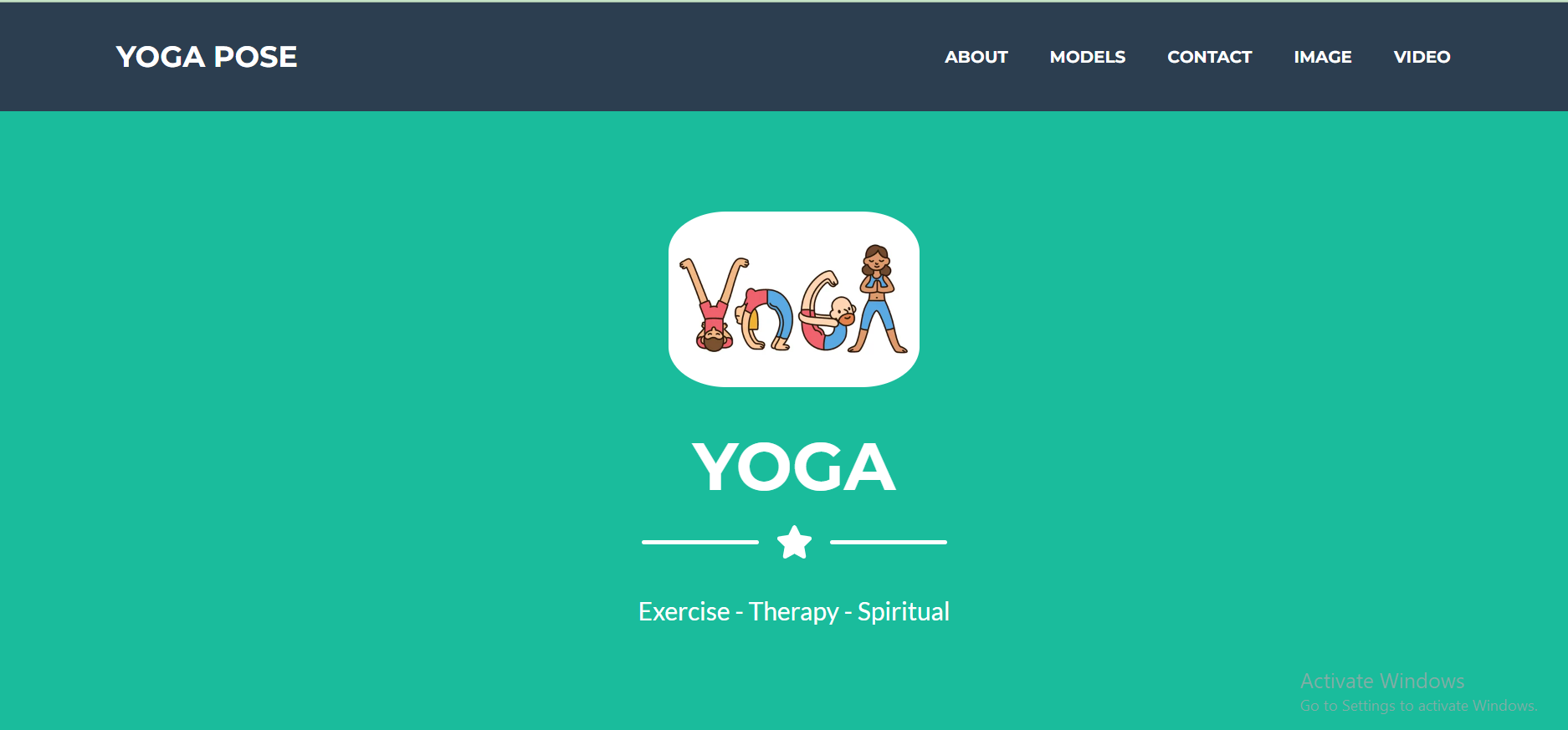
**Input: 2**



**Output: 2**



The home page looks like this. When you click on the button **Video** you’ll be redirected to the Video predict section



Click on Video button

**Input & output 1:**

