Fake/Real Logo Detection using Deep learning

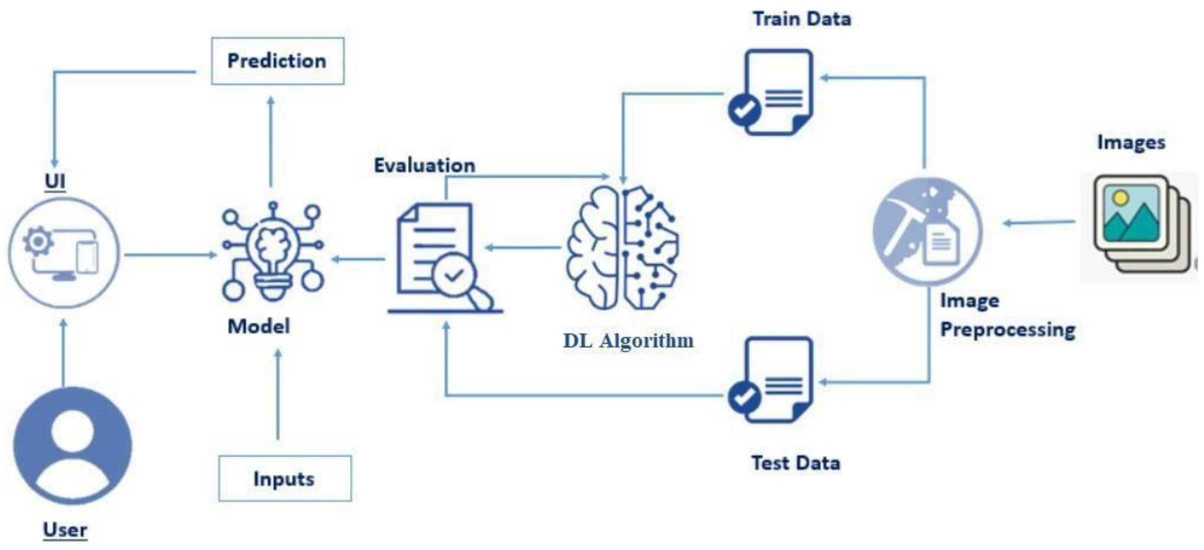
Introduction:

Logo detection and authentication are important tasks in the digital age, where counterfeit and fraudulent activities are prevalent. Detecting fake or real logos can help protect brands, prevent fraud, and ensure the authenticity of products or services. Recent advances in deep learning, specifically convolutional neural networks (CNNs), have proven effective in image recognition tasks, including logo detection. VGG19 is a popular CNN architecture known for its accuracy in image classification tasks. Leveraging VGG19 for fake/real logo detection involves a two-step process: training and inference. The model learns to extract features from logo images and differentiate between real and fake versions based on the provided labels.

Once the VGG19 model is trained, it can be used for inference on new logo images to predict whether they are real or fake. The logo images are fed into the trained model, which processes them through its layers and generates predictions. The model outputs probabilities or confidence scores indicating the likelihood of the logo being real or fake. The predictions can be further analyzed and thresholded to make a binary decision (real or fake) based on a predetermined threshold value. This enables automated logo authentication and detection, aiding in the identification of counterfeit products or unauthorized logo usage.

By leveraging the power of deep learning and CNNs, this approach enables automated and accurate identification of real and fake logos, contributing to a safer and more secure environment for businesses and consumers.

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,

QtConsole, Spyder, Glueviz, Orange, Rstudio, Visual Studio Code. For this project, we will be using Jupyter notebook and VS code.

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**:
  - o 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:**
  - o 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:**
  - o open anaconda prompt as administrator
  - o Type "pip install numpy" and click enter.
  - o Type "pip install pandas" and click enter.
  - o Type "pip install scikit-learn" and click enter.
  - o Type "pip install tensorflow==2.12.0" and click enter.
  - o Type "pip install keras==2.12.0" and click enter.
  - o Type "pip install Flask" and click enter.

- **Deep Learning Concepts**
    - **CNN:** a convolutional neural network is a class of deep neural networks, most commonly applied to analyzing visual imagery.
      CNN Basic
    - **VGG19:** VGG19 is a deep convolutional neural network architecture for image classification, consisting of 19 layers with small convolution filters.
      VGG19
    - **Flask:** Flask is a popular Python web framework, meaning it is a third-party Python library used for developing web applications.
      Flask Basics

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 Convolutional Neural Network.
- 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 UI (User Interface) to choose the image.
- The chosen image is analyzed by the model which is integrated with flask application.
- CNN Models analyze the image, then prediction is showcased on the Flask

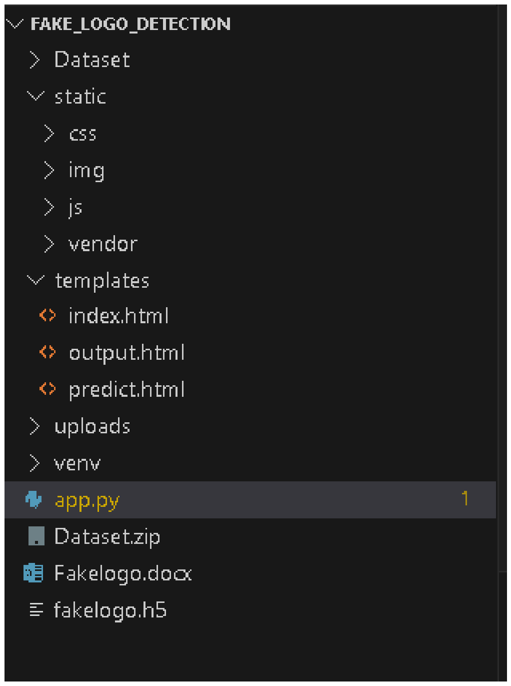
UI. To accomplish this, we must complete all the activities and tasks listed below.

- Data Collection.
    - Create Train and Test Folders.
- Data Preprocessing.
    - Import the ImageDataGenerator library
    - Configure ImageDataGenerator class
    - ApplyImageDataGenerator functionality to Train dataset and Test dataset
- Model Building
    - Import the model building Libraries.
    - Importing the VGG19.
    - Initializing the model
    - Adding Fully connected Layer
    - Configure the Learning Process

    - o Training and Testing the model.
    - o Save the Model
- o Application Building
    - o Create an HTML file
    - o Build Python Code

Project Structure:

Create a Project folder which contains files as shown below.



```
FAKE_LOGO_DETECTION
  > Dataset
  v static
    > css
    > img
    > js
    > vendor
  v templates
    index.html
    output.html
    predict.html
  > uploads
  > venv
  app.py                1
  Dataset.zip
  Fakelogo.docx
  fakelogo.h5
```

- 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
- we need the model which is saved and the saved model in this content is a **fakelogo.h5**
- templates folder contains index.html, predict.html & output.html pages.

<u>Milestone 1</u>: Define Problem / Problem Understanding

Activity 1: Specify the business problem.

Refer Project Description

Activity 2: Business requirements

Here are some potential business requirements for Fake/Real Logo Identification:

a. **Accurate Prediction:**
The predictor must be able to accurately classify the images of Fake/Real Logos. So that there will be no misclassification which decreases the accuracy of the model.

b. **Real-time data acquisition:**
The predictor must be able to acquire real-time data from the various sources. The data acquisition must be seamless and efficient to ensure that the predictor is always up-to-date with the latest information.

c. **User-friendly interface:**
The predictor must have a user-friendly interface that is easy to navigate and understand. The interface should present the results of the predictor in a clear and concise manner.

d. Report generation:
Generate a report outlining the predicted Logos. By analyzing data and applying advanced algorithms, the project generates detailed reports that include key findings, disease classifications, and relevant insights. The report should be presented in a clear and concise manner, with appropriate insights to help patients confirm their results.

Activity 3: Literature Survey (Student Will Write)

A literature survey would involve researching and reviewing existing studies, articles, and other publications on the topic of the project. The survey would aim to gather information on current systems, their strengths and weaknesses, and any gaps in knowledge that the project could address. The literature survey would also look at the methods and techniques used in previous projects, and any relevant data or findings that could inform the design and implementation of the current project.

Activity 4: Social or Business Impact.

The Fake/Real Logos Prediction project can have both social and business impacts.

Social Impact:
Fake/real logo prediction helps protect consumers from counterfeit products or services. By accurately identifying fake logos, individuals can make informed purchasing decisions, avoiding potentially harmful or low-quality counterfeit goods. Fake logos can damage the reputation of established brands. Logo prediction systems enable brands to detect and take action against unauthorized use or reproduction of their logos, safeguarding their reputation and maintaining trust among customers. Logo prediction technology can be applied in anti-fraud efforts, such as identifying fraudulent websites or unauthorized sellers. This helps reduce online scams, phishing attempts, and unauthorized use of logos for deceptive purposes.

Business Impact:
Authenticity is a crucial factor for brand success. Fake/real logo prediction systems help companies establish and maintain brand authenticity, reinforcing consumer trust and loyalty. Authenticity is a crucial factor for brand success. Fake/real logo prediction systems help companies establish and maintain brand authenticity, reinforcing consumer trust and loyalty. Counterfeit products pose a significant challenge to businesses across various industries. Accurate logo prediction enables companies to identify counterfeit goods, take legal actions against counterfeiters, and protect their market share.

<u>Milestone 2</u>: Data Collection & Image Preprocessing:

In this milestone First, we will collect images of Fake/Real Logos then organized into subdirectories based on their respective names as shown in the project structure. Create folders of types of Colon Diseases that need to be recognized. In this project, we have collected images of 63 types of Images like Adidas, Amazon, Android, Apple, Ariel, BMW, Bic, Burger King, Cadbury, Chevrolet, Chrome, Coca Cola, Cowbell, Dominos, Fila, Gillete, Google, Goya oil, Guinness, Heinz, Honda, Hp, Huawei, Instagram, Kfc, Krisspy Kreme, Lays, Levis, Lg, Lipton, Mars, Marvel, McDonald, Mercedes Benz, Microsoft, MnM, Mtn, Mtn dew, NASA, Nescafe, Nestle, Nestle milo, Netflix, Nike, Nutella, Oral b, Oreo, Paypal, Peak milk, Pepsi, PlayStation, Pringles, Puma, Reebok, Rolex, Samsung, Sprite, Starbucks, Tesla, Tiktok, Twitter, Youtube, Zara they are saved in the respective sub directories with their respective names.

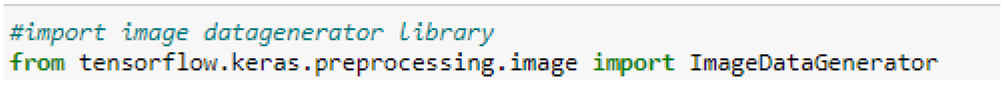
Download the Dataset -
https://www.kaggle.com/datasets/prosperchuks/fakereal-logo-detection-dataset

In Image Processing, 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.

Activity 1: Import the ImageDataGenerator library.

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.



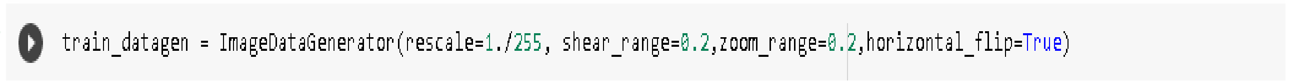
```
#import image datagenerator library
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

Activity 2: 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.



```
train_datagen = ImageDataGenerator(rescale=1./255, shear_range=0.2,zoom_range=0.2,horizontal_flip=True)
```

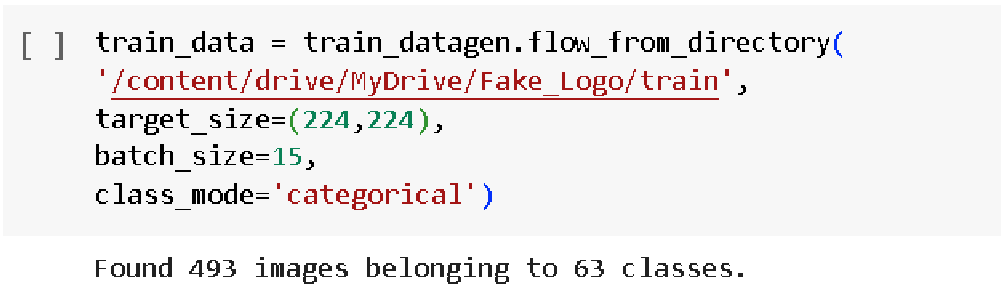
Activity 3: Apply ImageDataGenerator functionality to Trainset and Testset

Let us apply ImageDataGenerator functionality to 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 Adidas, Amazon, Android, Apple, Ariel, BMW, Bic, Burger King, Cadbury, Chevrolet, Chrome, Coca Cola, Cowbell, Dominos, Fila, Gillete, Google, Goya oil, Guinness, Heinz, Honda, Hp, Huawei, Instagram, Kfc, Krisspy Kreme, Lays, Levis, Lg, Lipton, Mars, Marvel, McDonald, Mercedes Benz, Microsoft, MnM, Mtn, Mtn dew, NASA, Nescafe, Nestle, Nestle milo, Netflix, Nike, Nutella, Oral b, Oreo, Paypal, Peak milk, Pepsi, PlayStation, Pringles, Puma, Reebok, Rolex, Samsung, Sprite, Starbucks, Tesla, Tiktok, Twitter, Youtube, Zara, together with labels 0 to 62.

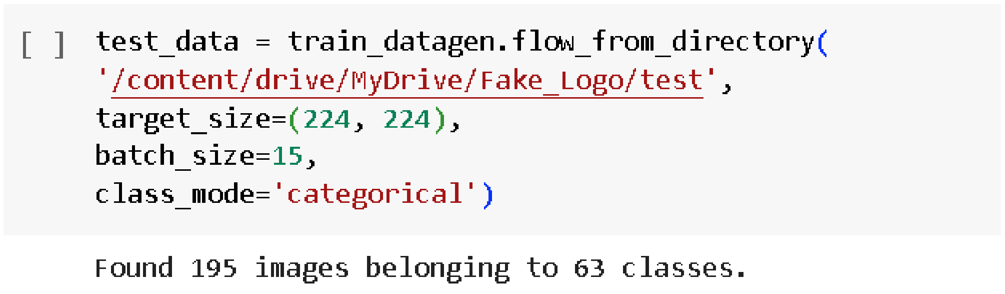
Arguments:

- 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 15.
- 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).



```
train_data = train_datagen.flow_from_directory(
'/content/drive/MyDrive/Fake_Logo/train',
target_size=(224,224),
batch_size=15,
class_mode='categorical')
```

```
Found 493 images belonging to 63 classes.
```



```
test_data = train_datagen.flow_from_directory(
'/content/drive/MyDrive/Fake_Logo/test',
target_size=(224, 224),
batch_size=15,
class_mode='categorical')
```

```
Found 195 images belonging to 63 classes.
```

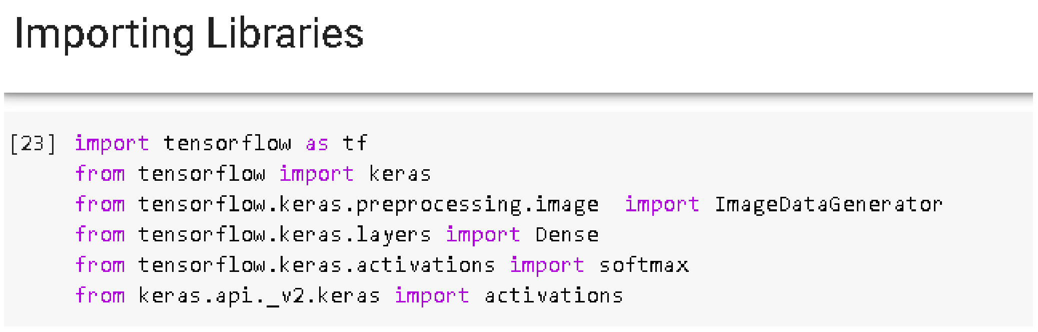
We notice that 493 images belong to 63 classes for training and 195 images belong to 63 classes for testing purposes.

Milestone 3: Model Building
Now it's time to build our Convolutional Neural Networking using vgg19 which contains an input layer along with the convolution, max-pooling, and finally an output layer.

Link: https://thesmartbridge.com/documents/spsaimldocs/CNNflow.pdf

Activity 1: Importing the Model Building Libraries

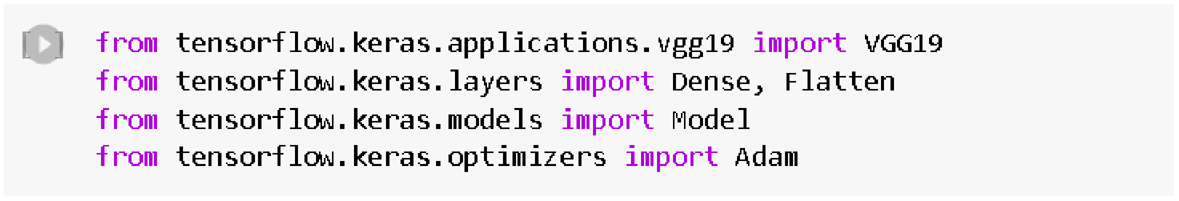
Importing the necessary libraries


Importing Libraries

```
[23] import tensorflow as tf
     from tensorflow import keras
     from tensorflow.keras.preprocessing.image  import ImageDataGenerator
     from tensorflow.keras.layers import Dense
     from tensorflow.keras.activations import softmax
     from keras.api._v2.keras import activations
```

Activity 2: Importing the VGG16 model:

To initialize the VGG16 model, the weights are usually pre-trained on the ImageNet dataset, which is a large-scale dataset of images belonging to 1,000 different categories. These pre-trained weights can be downloaded from the internet, and they can be used as a starting point to fine-tune the model for a specific task, such as object recognition or classification.



```
from tensorflow.keras.applications.vgg19 import VGG19
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
```

Activity 3: Initializing the model:

- The model will be initialized with the pre-trained weights from the ImageNet dataset, and the last fully connected layer will be excluded from the model architecture.

- The loop that follows freezes the weights of all the layers in the VGG19 model by setting `i.trainable=False` for each layer in the model. This is done to prevent the weights from being updated during training, as the model is already pre-trained on a large dataset.

- Finally, a `Flatten()` layer is added to the output of the VGG19 model to convert the output tensor into a 1D tensor.

- The resulting model can be used as a feature extractor for transfer learning or as a starting point for building a new model on top of it.

```
[ ]  Image_size=[224,224]
```

```
[ ]  sol=VGG19(input_shape=Image_size + [3] , weights='imagenet' , include_top = False)
```

```
[ ]  for i in sol.layers:
       i.trainable = False
```

```
[ ]  y=Flatten()(sol.output)
```

Activity 4: Adding Fully connected Layers

- For information regarding CNN Layers refer to the link Link: https://victorzhou.com/blog/intro-to-cnns-part-1/
- As the input image contains three channels, we are specifying the input shape as (128,128,3).

- We are adding a output layer with activation function as "softmax" .

```
[ ]  from keras.api._v2.keras import activations
     final = Dense(63, activation = 'softmax')(y)
```

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

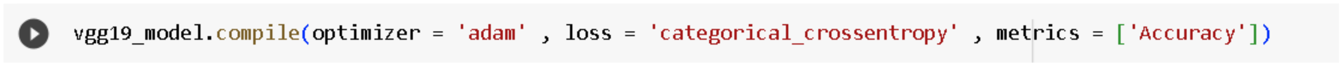
Understanding the model is a very important phase to properly use it for training and prediction purposes. Keras provides a simple method, summary to get the full information about the model and its layers.

```python
vgg19_model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0         
                                                                 
 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792      
                                                                 
 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928     
                                                                 
 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0         
                                                                 
 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856     
                                                                 
 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584    
                                                                 
 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0         
                                                                 
 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168    
                                                                 
 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080    
                                                                 
 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080    
                                                                 
 block3_conv4 (Conv2D)       (None, 56, 56, 256)       590080    
                                                                 
 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0         
                                                                 
 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160   
```

Activity 5: 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.



```
vgg19_model.compile(optimizer = 'adam' , loss = 'categorical_crossentropy' , metrics = ['Accuracy'])
```

Activity 6: Train The model

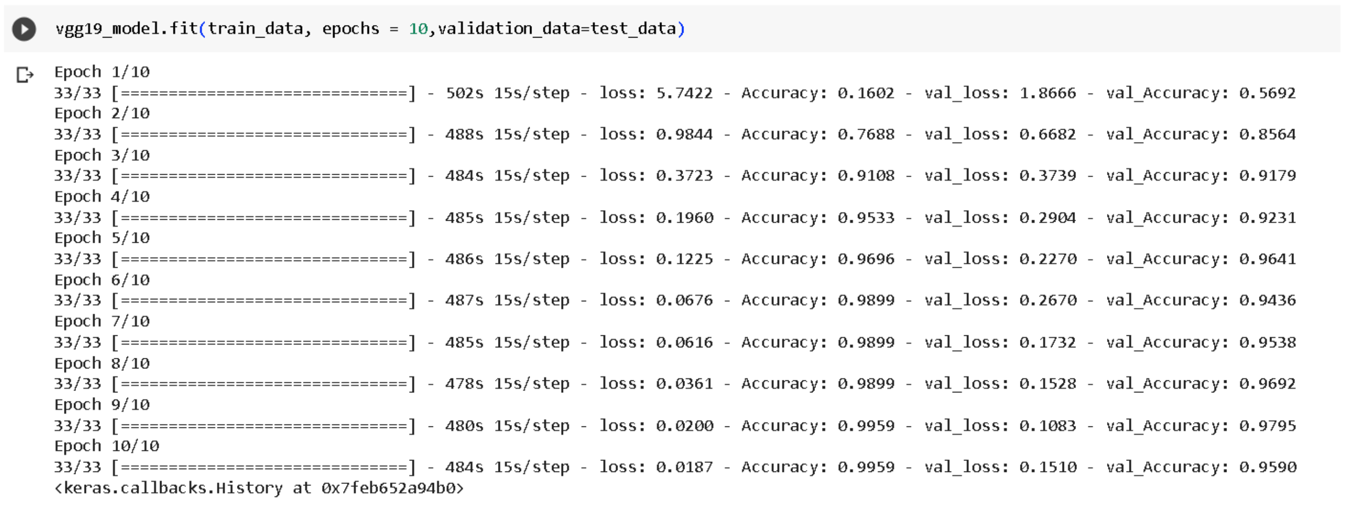
Now, let us train our model with our image dataset. The model is trained for 10 epochs and after every epoch, the current model state is saved if the model has the least loss encountered till that time. We can see that the training loss decreases in almost every epoch till 30 epochs and probably there 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.

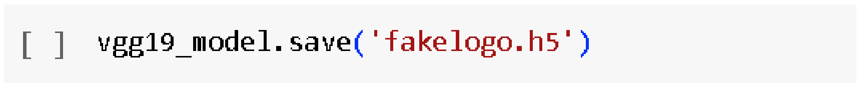


```
vgg19_model.fit(train_data, epochs = 10,validation_data=test_data)
```

```
Epoch 1/10
33/33 [==============================] - 502s 15s/step - loss: 5.7422 - Accuracy: 0.1602 - val_loss: 1.8666 - val_Accuracy: 0.5692
Epoch 2/10
33/33 [==============================] - 488s 15s/step - loss: 0.9844 - Accuracy: 0.7688 - val_loss: 0.6682 - val_Accuracy: 0.8564
Epoch 3/10
33/33 [==============================] - 484s 15s/step - loss: 0.3723 - Accuracy: 0.9108 - val_loss: 0.3739 - val_Accuracy: 0.9179
Epoch 4/10
33/33 [==============================] - 485s 15s/step - loss: 0.1960 - Accuracy: 0.9533 - val_loss: 0.2904 - val_Accuracy: 0.9231
Epoch 5/10
33/33 [==============================] - 486s 15s/step - loss: 0.1225 - Accuracy: 0.9696 - val_loss: 0.2270 - val_Accuracy: 0.9641
Epoch 6/10
33/33 [==============================] - 487s 15s/step - loss: 0.0676 - Accuracy: 0.9899 - val_loss: 0.2670 - val_Accuracy: 0.9436
Epoch 7/10
33/33 [==============================] - 485s 15s/step - loss: 0.0616 - Accuracy: 0.9899 - val_loss: 0.1732 - val_Accuracy: 0.9538
Epoch 8/10
33/33 [==============================] - 478s 15s/step - loss: 0.0361 - Accuracy: 0.9899 - val_loss: 0.1528 - val_Accuracy: 0.9692
Epoch 9/10
33/33 [==============================] - 480s 15s/step - loss: 0.0200 - Accuracy: 0.9959 - val_loss: 0.1083 - val_Accuracy: 0.9795
Epoch 10/10
33/33 [==============================] - 484s 15s/step - loss: 0.0187 - Accuracy: 0.9959 - val_loss: 0.1510 - val_Accuracy: 0.9590
<keras.callbacks.History at 0x7feb652a94b0>
```

Activity 7: Save the Model

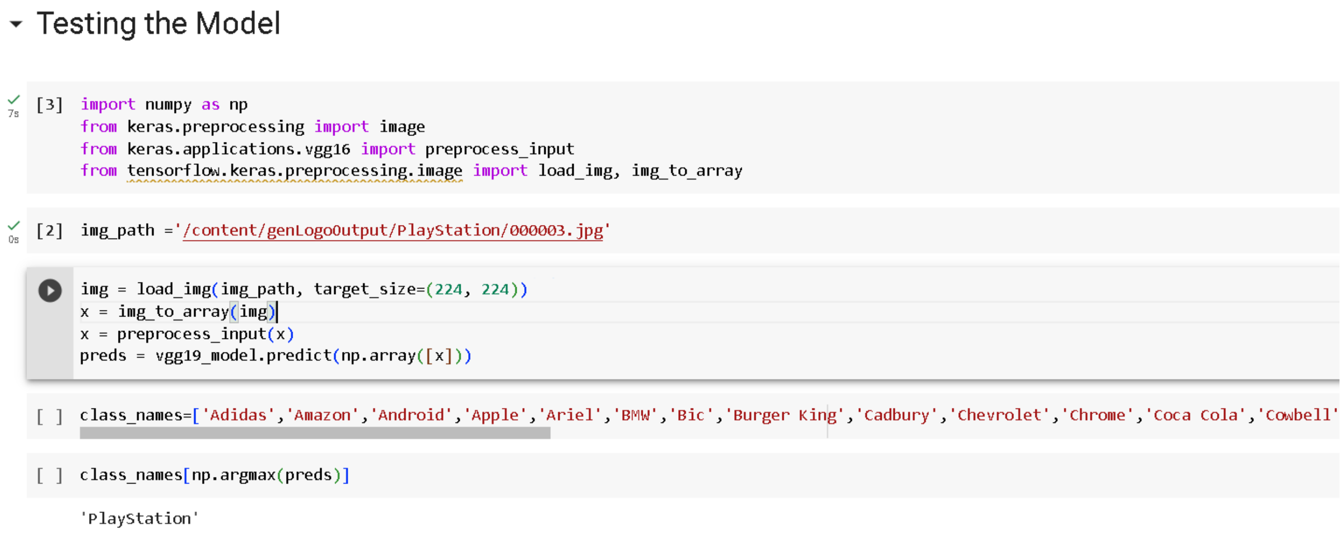
The model is saved with .h5 extension as follows.
An H5 file is a data file saved in the Hierarchical Data Format (HDF). It contains multidimensional arrays of scientific data.



```
[ ]  vgg19_model.save('fakelogo.h5')
```

Activity 8: 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.



Testing the Model

```
[3]  import numpy as np
     from keras.preprocessing import image
     from keras.applications.vgg16 import preprocess_input
     from tensorflow.keras.preprocessing.image import load_img, img_to_array
```

```
[2]  img_path ='/content/genLogoOutput/PlayStation/000003.jpg'
```

```
     img = load_img(img_path, target_size=(224, 224))
     x = img_to_array(img)
     x = preprocess_input(x)
     preds = vgg19_model.predict(np.array([x]))
```

```
[ ]  class_names=['Adidas','Amazon','Android','Apple','Ariel','BMW','Bic','Burger King','Cadbury','Chevrolet','Chrome','Coca Cola','Cowbell'
```

```
[ ]  class_names[np.argmax(preds)]

     'PlayStation'
```

Taking an image as input and checking the results
By using the model we are predicting the output for the given input image.
The predicted class index name will be printed here.

Milestone 4: Application Building

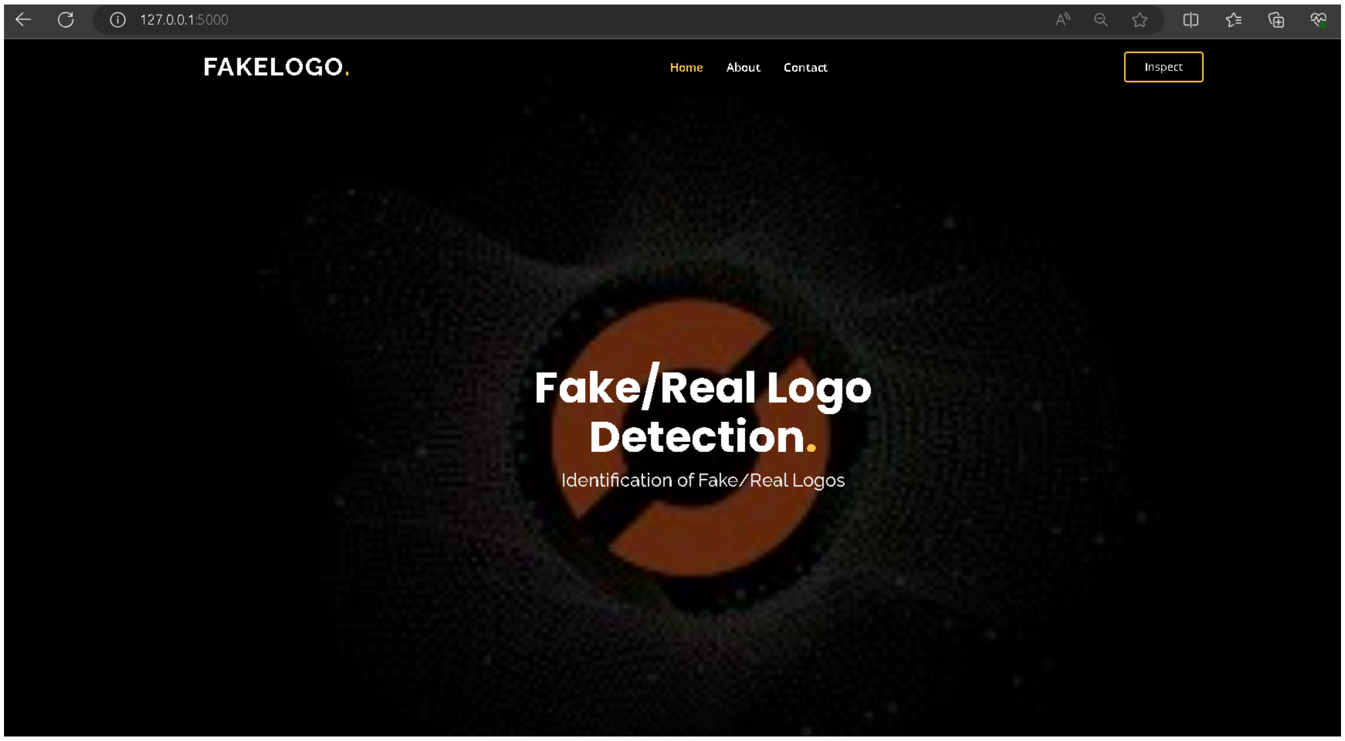
Now that we have trained our model, let us build our flask application which will be running in our local browser with a user interface.

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 Colon Diseases and showcased on the HTML page to notify the user. Whenever the user interacts with the UI and selects the "Inspect" button, the next page is opened where the user chooses the image and predicts the output.

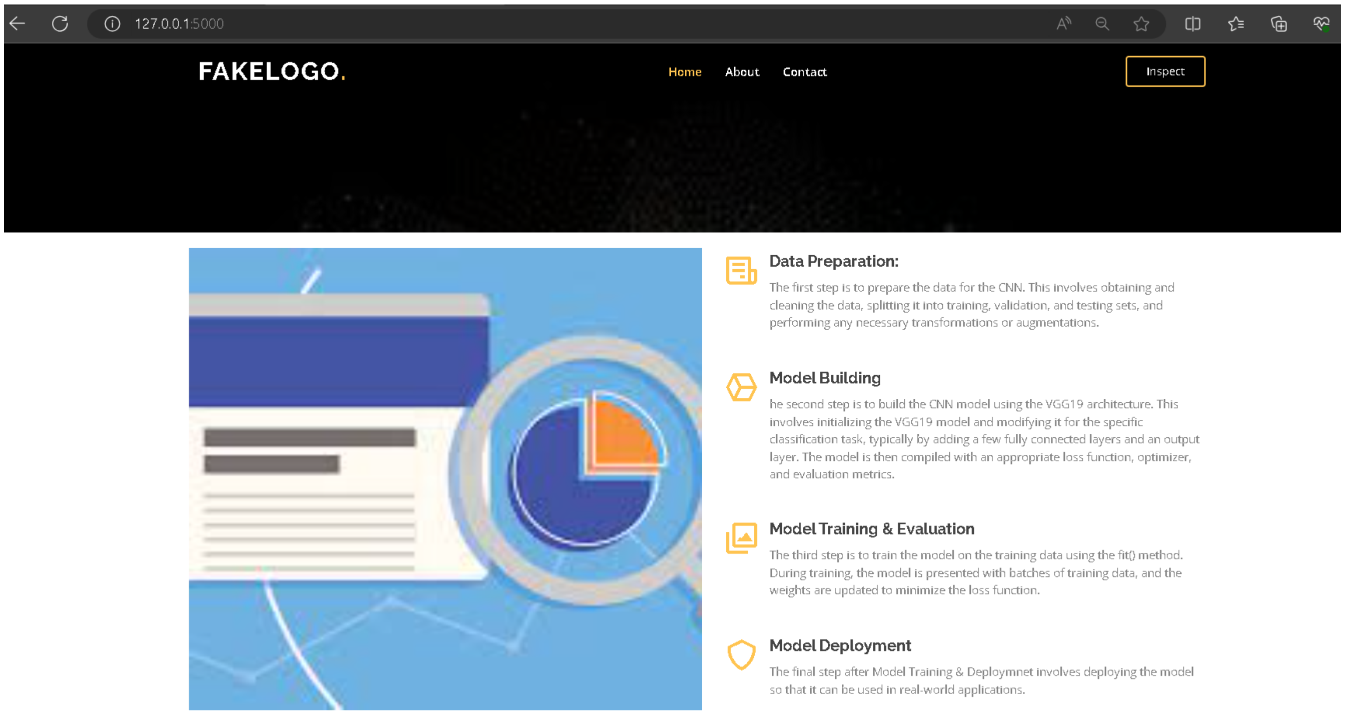
Activity 1 : Create HTML Pages

- We use HTML to create the front end part of the web page.
- Here, we have created 3 HTML pages- index.html, predict.html, and output.html
- home.html displays the home page.
- index.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


127.0.0.1:5000
FAKELOGO.
Home
About
Contact
Inspect
Fake/Real Logo Detection.
Identification of Fake/Real Logos

About Section:-



Data Preparation:

The first step is to prepare the data for the CNN. This involves obtaining and cleaning the data, splitting it into training, validation, and testing sets, and performing any necessary transformations or augmentations.

Model Building

he second step is to build the CNN model using the VGG19 architecture. This involves initializing the VGG19 model and modifying it for the specific classification task, typically by adding a few fully connected layers and an output layer. The model is then compiled with an appropriate loss function, optimizer, and evaluation metrics.

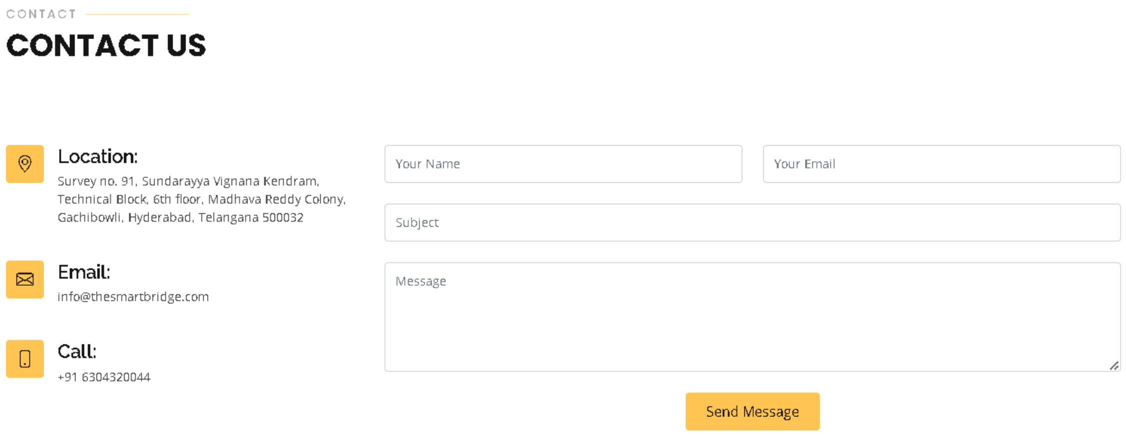
Model Training & Evaluation

The third step is to train the model on the training data using the fit() method. During training, the model is presented with batches of training data, and the weights are updated to minimize the loss function.

Model Deployment

The final step after Model Training & Deploymnet involves deploying the model so that it can be used in real-world applications.

Contact Us:-



CONTACT

CONTACT US

Location:
Survey no. 91, Sundarayya Vignana Kendram, Technical Block, 6th floor, Madhava Reddy Colony, Gachibowli, Hyderabad, Telangana 500032

Email:
info@thesmartbridge.com

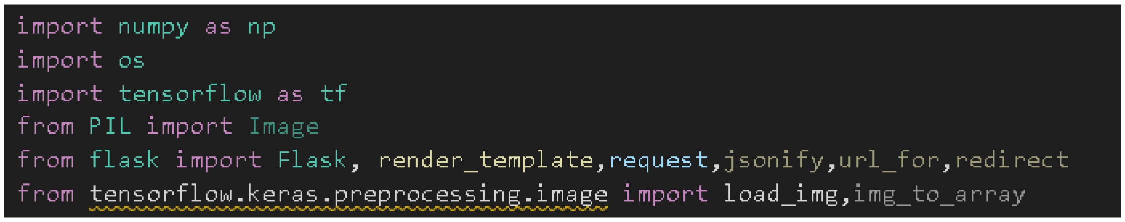
Call:
+91 6304320044

Activity 2: Build python code

Task 1: Importing Libraries

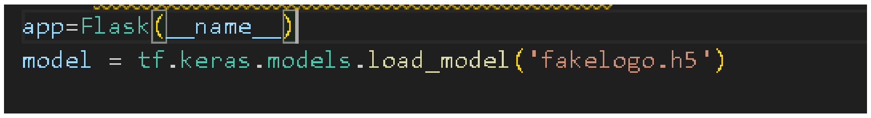
The first step is usually importing the libraries that will be needed in the program.

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.



```
import numpy as np
import os
import tensorflow as tf
from PIL import Image
from flask import Flask, render_template,request,jsonify,url_for,redirect
from tensorflow.keras.preprocessing.image import load_img,img_to_array
```

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

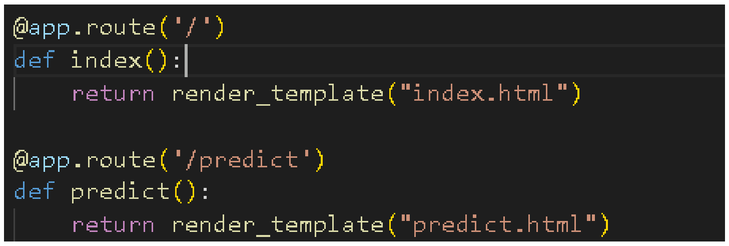


```
app=Flask(__name__)
model = tf.keras.models.load_model('fakelogo.h5')
```

Task 3: Routing to the html Page

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

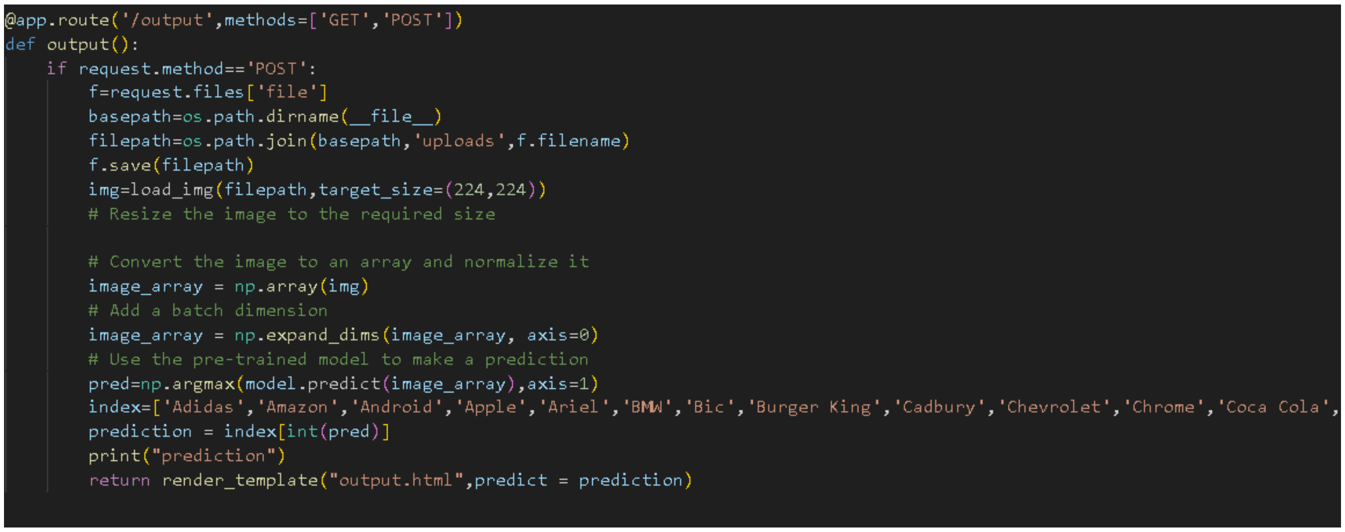
In the above example, ‘/’ URL is bound with index.html function. Hence, when the home page of a web server is opened in the browser, the html page will be rendered. Whenever you browse an image from the html page this photo can be accessed through POST or GET Method.



```
@app.route('/')
def index():
    return render_template("index.html")

@app.route('/predict')
def predict():
    return render_template("predict.html")
```

Showcasing prediction on UI:



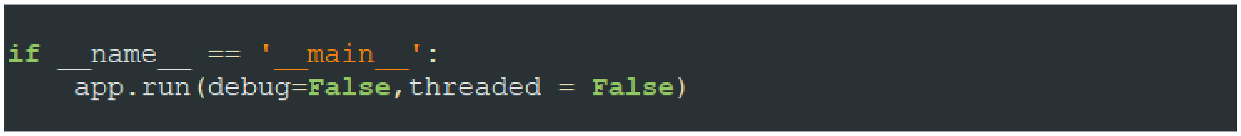
```
@app.route('/output',methods=['GET','POST'])
def output():
    if request.method=='POST':
        f=request.files['file']
        basepath=os.path.dirname(__file__)
        filepath=os.path.join(basepath,'uploads',f.filename)
        f.save(filepath)
        img=load_img(filepath,target_size=(224,224))
        # Resize the image to the required size

        # Convert the image to an array and normalize it
        image_array = np.array(img)
        # Add a batch dimension
        image_array = np.expand_dims(image_array, axis=0)
        # Use the pre-trained model to make a prediction
        pred=np.argmax(model.predict(image_array),axis=1)
        index=['Adidas','Amazon','Android','Apple','Ariel','BMW','Bic','Burger King','Cadbury','Chevrolet','Chrome','Coca Cola',
        prediction = index[int(pred)]
        print("prediction")
        return render_template("output.html",predict = prediction)
```

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 to 62.) 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 prediction variable used in the html page.

Finally, Run the application

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

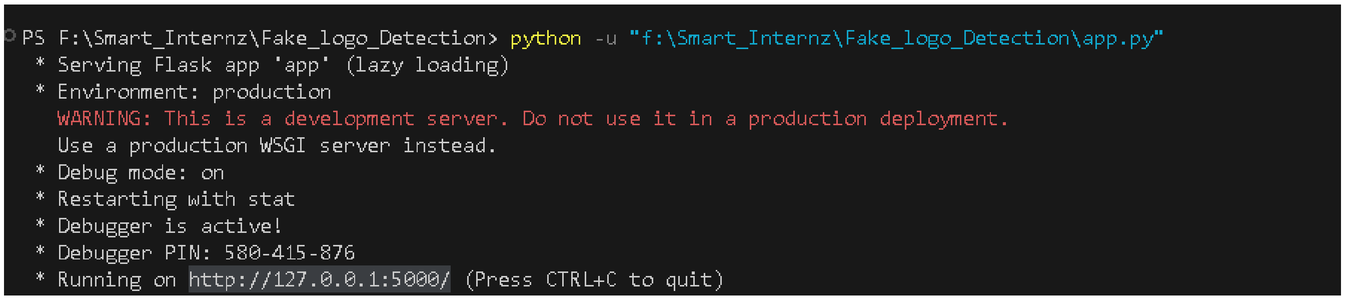


```
if __name__ == '__main__':
    app.run(debug=False,threaded = False)
```

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.

Then it will run on localhost: 5000

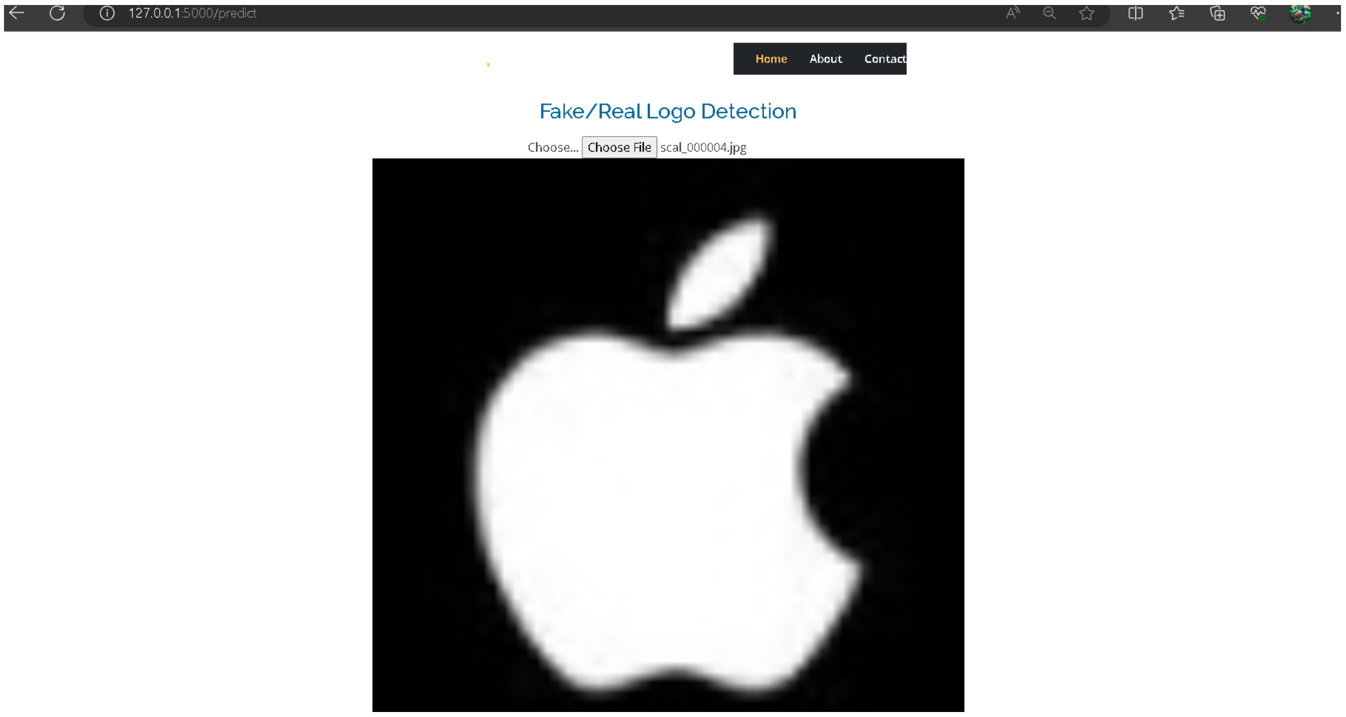


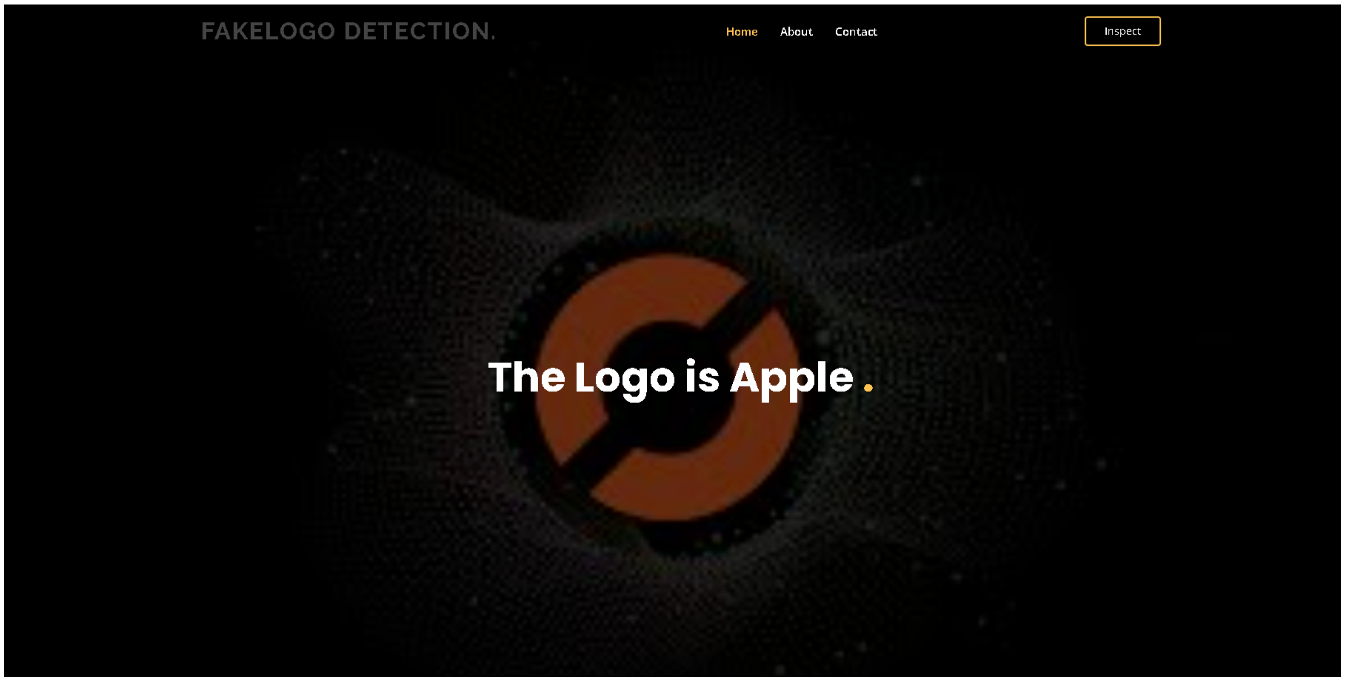
```
PS F:\Smart_Internz\Fake_logo_Detection> python -u "f:\Smart_Internz\Fake_logo_Detection\app.py"
 * Serving Flask app 'app' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 580-415-876
 * 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:

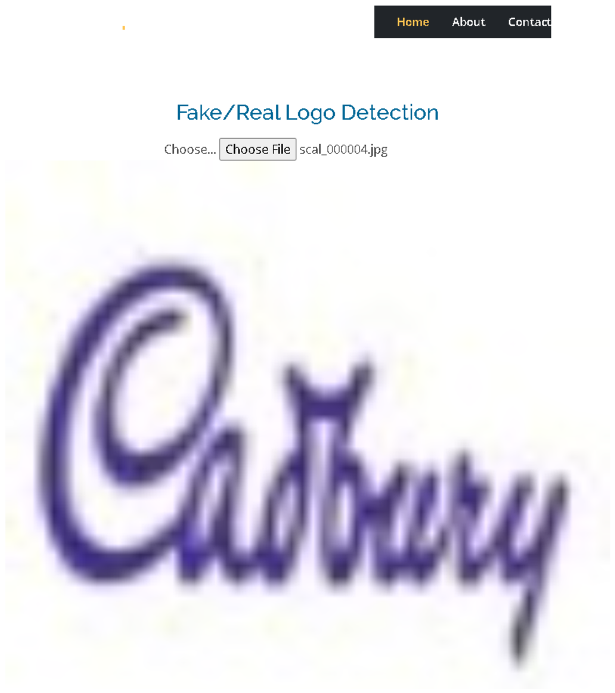
Output 1:

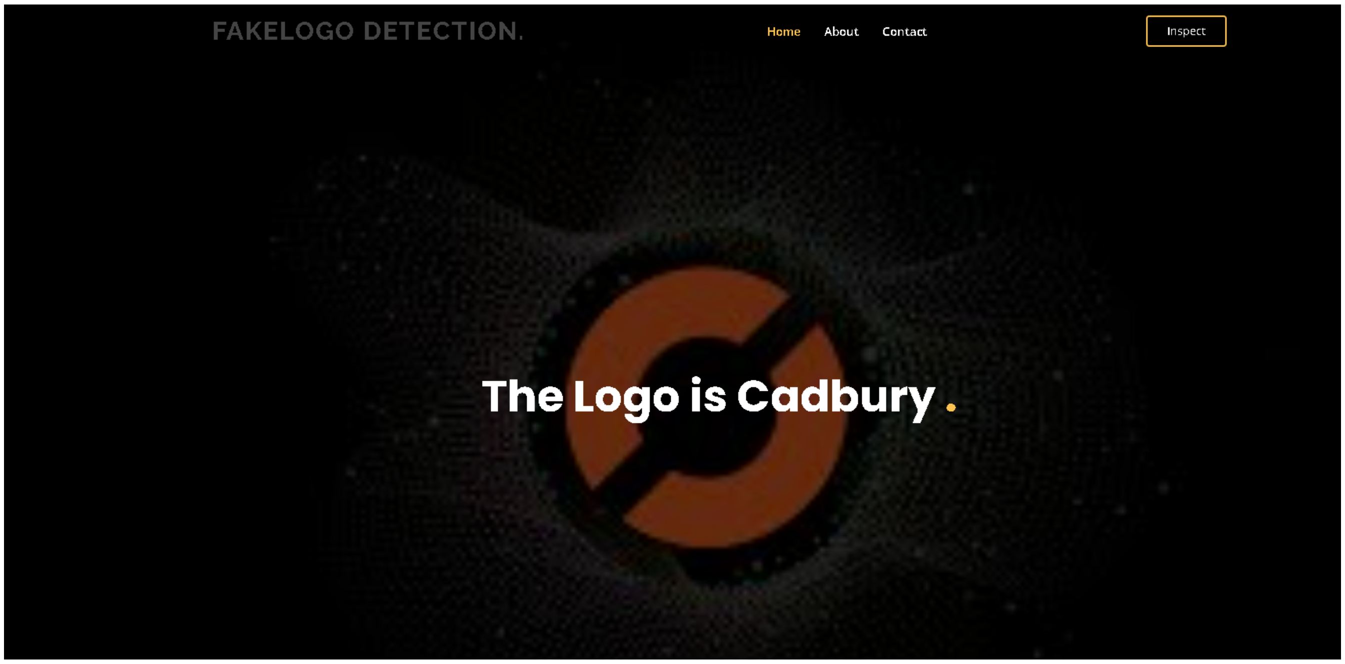




The Logo is Apple .

Output 2:


Home
About
Contact
Fake/Real Logo Detection
Choose...
Choose File
scal_000004.jpg


FAKELOGO DETECTION.
Home
About
Contact
Inspect
The Logo is Cadbury .