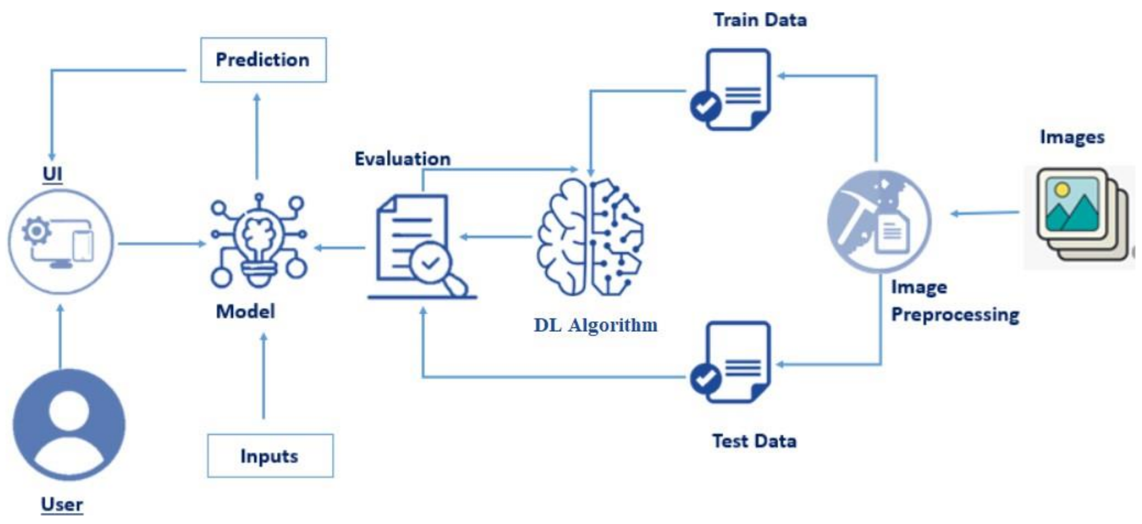
Crime Vision: Advanced Crime Classification with Deep Learning

Project Description:

Crime identification using deep learning is a technique that involves applying deep learning techniques, specifically deep learning, to analyze images and video footage of crime scenes or incidents and identify and classify different types of crimes. Deep learning involves training convolutional neural networks on large amounts of data to recognize patterns and make predictions or decisions.
By using deep learning, it is possible to analyze images and video footage of crime scenes or incidents and classify different types of crimes based on the type of activity depicted in the images. This can be useful in a variety of criminal justice and law enforcement contexts, including crime scene investigation, forensic analysis, and surveillance.
Deep learning algorithms can be trained to recognize patterns and features in images and video that are relevant to identifying different types of crimes. They can also be used to analyze large amounts of data, such as surveillance footage, to identify trends and patterns in crime data. This can allow law enforcement agencies to develop strategies and interventions to prevent crime.

Technical Architecture:



Project Flow:

- The user interacts with the UI to choose an image.
- The chosen image is processed by a transfer learning deep learning model.
- The transfer learning model is integrated with a Flask application.
- The transfer learning model analyzes the image and generates predictions.
- The predictions are displayed on the Flask UI for the user to see.
- This process enables users to input an image and receive accurate predictions quickly.

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

- Data Collection.
  - Create a Train and Test path.
- Image Pre-processing.
  - Import the required library
  - Configuration of Images and preprocessing
  - Apply Image_Dataset_from_directory functionality to Train set and Test set
- Model Building
  - Create Transfer Learning Function
  - Adding Dense Layer
  - Configure the Learning Process
  - Train the model
  - Save the Model
  - Test the model
- Application Building
  - Create an HTML file
  - Build Python Flask Code

Prior Knowledge:

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

Deep Learning Concepts

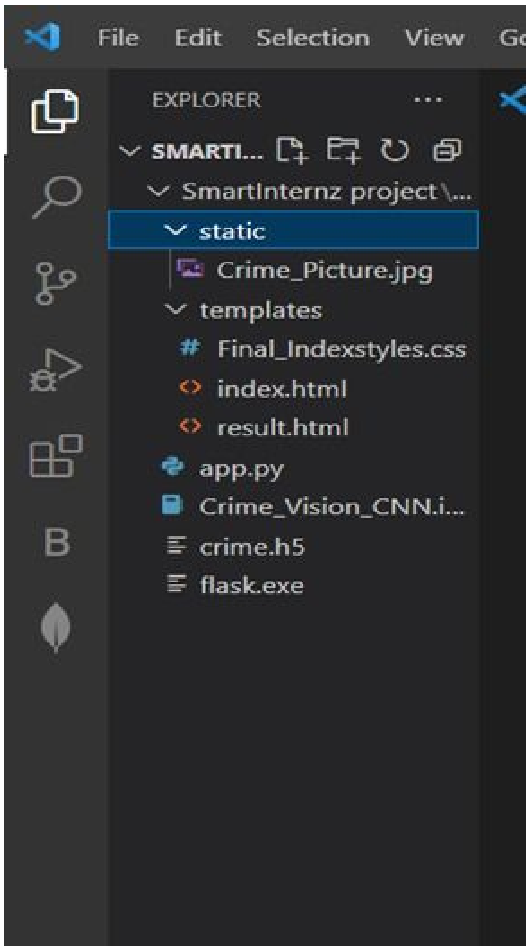
- **CNN:** https://towardsdatascience.com/basics-of-the-classic-cnn-a3dce1225add
- **TransferLearning:**https://www.analyticsvidhya.com/blog/2021/10/understanding-transfer-learning-for-deep-learning/

- **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

Project Structure:

Create a Project folder which contains files as shown below



- The Dataset folder contains the training and testing images for training our model.
- For building a Flask Application we needs HTML pages stored in the **templates** folder,CSS for styling the pages stored in the static folder and a python script **app.py** for server side scripting
- The IBM folder consists of a trained model notebook on IBM Cloud.
- Training folder consists of crime classification.ipynb model training file & crime.h5 is saved model

Milestone 1: Data Collection

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

Activity 1: Download the dataset

The dataset contains images extracted from every video from the UCF Crime Dataset.
Every 10th frame is extracted from each full-length video and combined for every video in that class.
All the images are of size 64*64 and in .png format
The dataset has a total of 14 Classes :

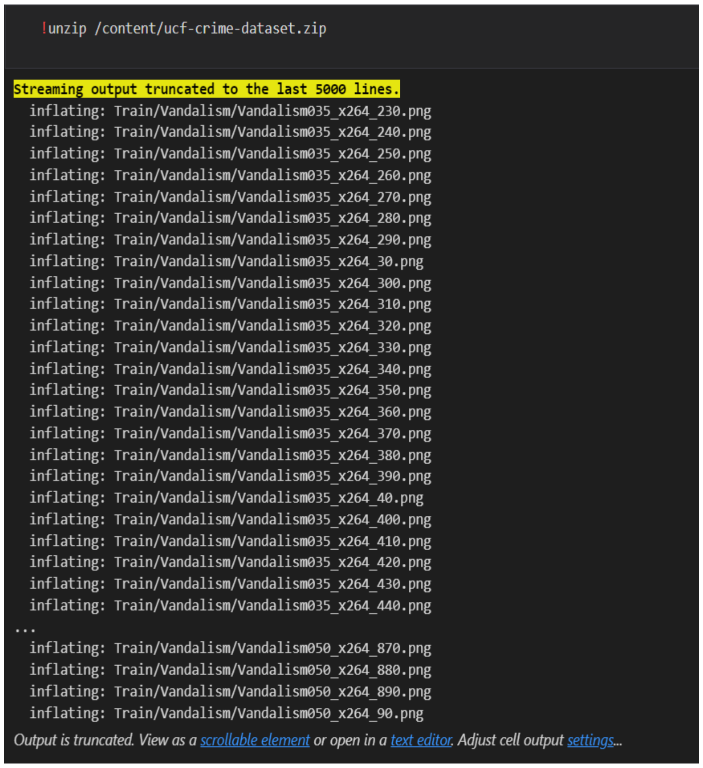
You can download the dataset used in this project using the below link Dataset:- : https://www.kaggle.com/datasets/odins0n/ucf-crime-dataset

Note: For better accuracy train on more images

We are going to build our training model on Google colab so we have to upload a dataset zip file on Google colab.

To upload a dataset zip file to Google Colab and then unzip it, you can follow these steps:

- Open Google Colab and create a new notebook.
- Click on the "Files" icon on the left-hand side of the screen.
- Click on the "Upload" button and select the zip file you want to upload.
- Wait for the upload to complete. You should see the file appear in the "Files" section.
- To unzip the file, you can use the following command:

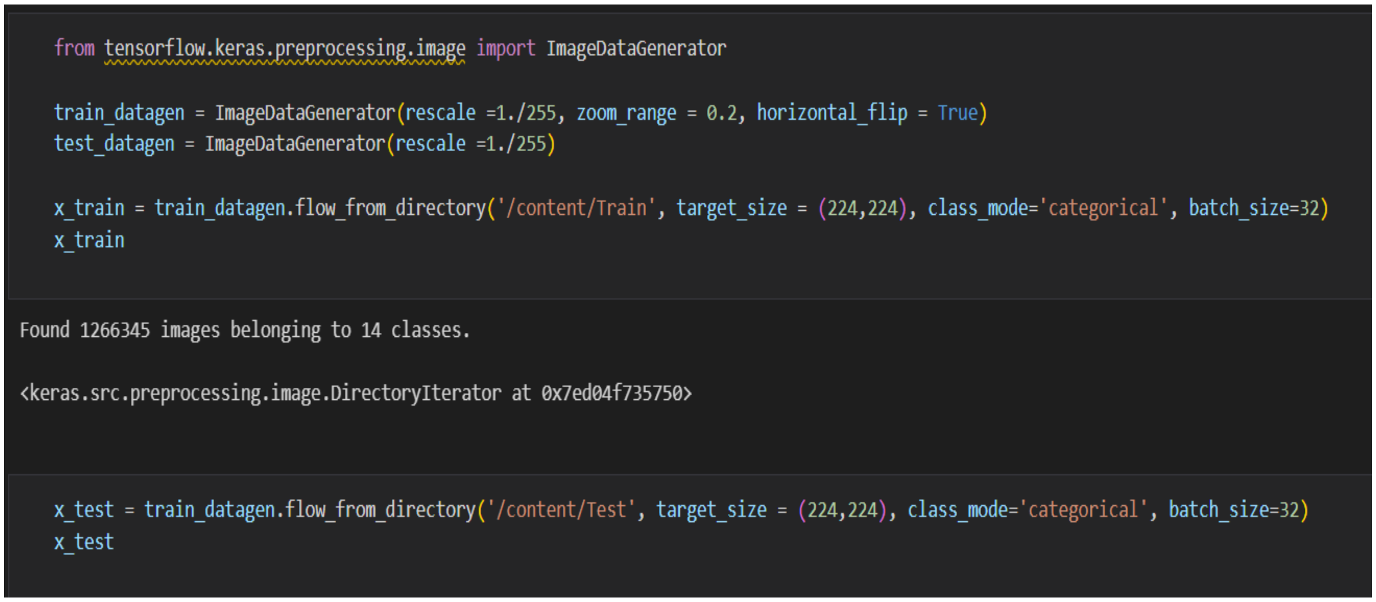


```
!unzip /content/ucf-crime-dataset.zip
```

```
Streaming output truncated to the last 5000 lines.
  inflating: Train/Vandalism/Vandalism035_x264_230.png
  inflating: Train/Vandalism/Vandalism035_x264_240.png
  inflating: Train/Vandalism/Vandalism035_x264_250.png
  inflating: Train/Vandalism/Vandalism035_x264_260.png
  inflating: Train/Vandalism/Vandalism035_x264_270.png
  inflating: Train/Vandalism/Vandalism035_x264_280.png
  inflating: Train/Vandalism/Vandalism035_x264_290.png
  inflating: Train/Vandalism/Vandalism035_x264_30.png
  inflating: Train/Vandalism/Vandalism035_x264_300.png
  inflating: Train/Vandalism/Vandalism035_x264_310.png
  inflating: Train/Vandalism/Vandalism035_x264_320.png
  inflating: Train/Vandalism/Vandalism035_x264_330.png
  inflating: Train/Vandalism/Vandalism035_x264_340.png
  inflating: Train/Vandalism/Vandalism035_x264_350.png
  inflating: Train/Vandalism/Vandalism035_x264_360.png
  inflating: Train/Vandalism/Vandalism035_x264_370.png
  inflating: Train/Vandalism/Vandalism035_x264_380.png
  inflating: Train/Vandalism/Vandalism035_x264_390.png
  inflating: Train/Vandalism/Vandalism035_x264_40.png
  inflating: Train/Vandalism/Vandalism035_x264_400.png
  inflating: Train/Vandalism/Vandalism035_x264_410.png
  inflating: Train/Vandalism/Vandalism035_x264_420.png
  inflating: Train/Vandalism/Vandalism035_x264_430.png
  inflating: Train/Vandalism/Vandalism035_x264_440.png
...
  inflating: Train/Vandalism/Vandalism050_x264_870.png
  inflating: Train/Vandalism/Vandalism050_x264_880.png
  inflating: Train/Vandalism/Vandalism050_x264_890.png
  inflating: Train/Vandalism/Vandalism050_x264_90.png
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Activity 2: Create training and testing dataset

To build a DL model we have to split training and testing data into two separate folders. But In the project dataset folder training and testing folders are presented. So, in this case we just have to assign a variable and pass the folder path to it.



```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

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

x_train = train_datagen.flow_from_directory('/content/Train', target_size = (224,224), class_mode='categorical', batch_size=32)
x_train
```

```
Found 1266345 images belonging to 14 classes.

<keras.src.preprocessing.image.DirectoryIterator at 0x7ed04f735750>
```

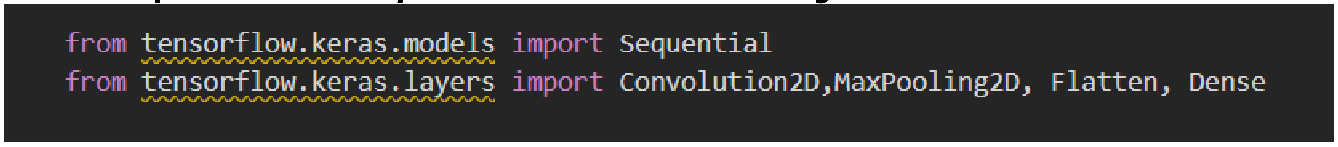
```
x_test = train_datagen.flow_from_directory('/content/Test', target_size = (224,224), class_mode='categorical', batch_size=32)
x_test
```

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 performing some geometric transformations of images like rotation, scaling, translation, etc.

Activity 1: Importing the libraries

Import the necessary libraries as shown in the image



```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Convolution2D,MaxPooling2D, Flatten, Dense
```

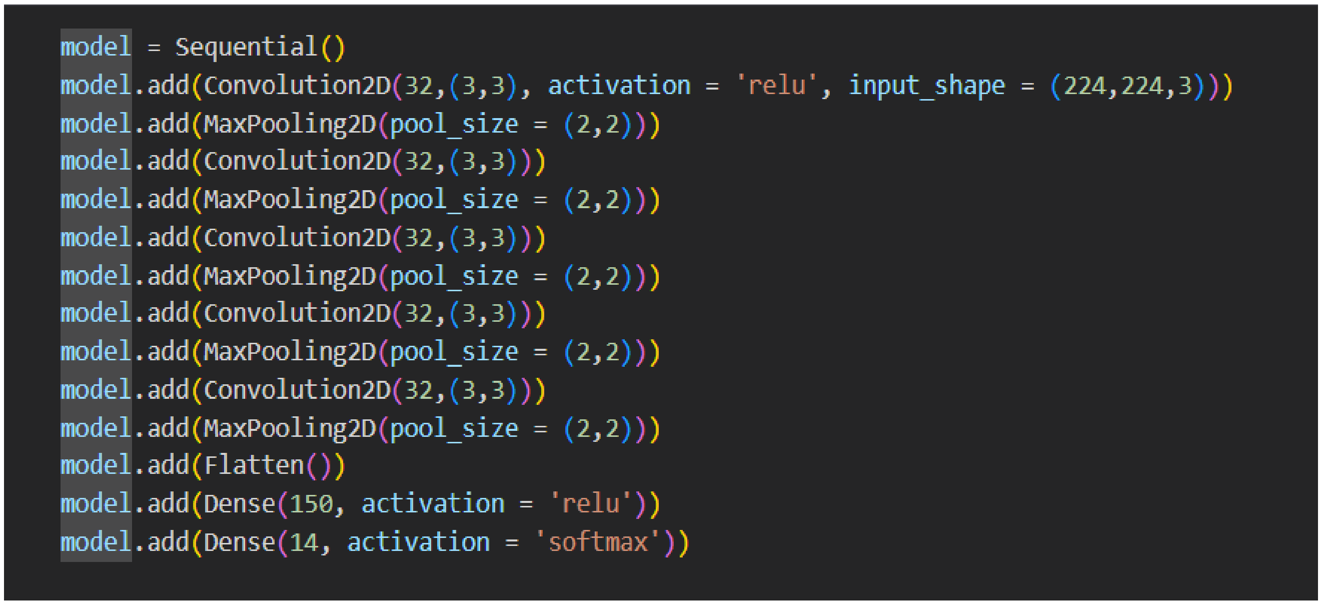
To understand the above imported libraries:-

- **Image_dataset_from_directory :** is a function in the tensorflow.keras.preprocessing module of TensorFlow, which allows you to create a TensorFlow Dataset from a directory containing image files. This function can be useful for training deep learning models on large image datasets.
- **Keras:** Keras is a high-level neural network API written in Python that allows for fast experimentation and prototyping of deep learning models.
- **DenseNet121**:It has been trained on large-scale image classification datasets such as ImageNet and has achieved state-of-the-art performance on a range of benchmark tasks. It has also been used as a pre-trained model for transfer learning in various computer vision applications.
- **Global average pooling 2D (GAP 2D):** It is a type of pooling operation commonly used in convolutional neural networks (CNNs) for image classification tasks.
- **Dense Layer:** A dense layer in neural networks is a fully connected layer where each neuron in the layer is connected to every neuron in the previous layer, and each connection has a weight associated with it.
- **Flatten Layer:** A flatten layer in neural networks is a layer that reshapes the input tensor into a one-dimensional array, which can then be passed to a fully connected layer.
- **Input Layer:** The input layer in neural networks is the first layer of the network that receives the input data and passes it on to the next layer for further processing.
- **Maxpooling 2D :** is a downsampling operation that reduces the spatial dimensions (height and width) of an input tensor while preserving the number of channels. The operation takes a window of a fixed size, typically 2x2, and outputs the maximum value within that window. Max Pooling helps reduce the computational cost of the network while also increasing its robustness to small translations of the input.
- **Convolution 2D:** It is a linear operation that applies a set of learnable filters (also called kernels or weights) to an input tensor to extract features. The filters slide over the input tensor, computing a dot product between their values and the values of the input tensor at each position. The output of a convolutional layer is a set of feature maps, each corresponding to a specific filter. Convolution 2D is the main building block of CNNs and is used to learn representations of the input data.

- **image:** from tensorflow.keras.preprocessing import image imports the image module from Keras' tensorflow.keras.preprocessing package. This module provides a number of image preprocessing utilities, such as loading images, converting images to arrays, and applying various image transformations.
- **load_img:** load_img is a function provided by the tensorflow.keras.preprocessing.image module that is used to load an image file from the local file system. It takes the file path as input and returns a PIL (Python Imaging Library) image object.
- **Dropout :** is a regularization technique that randomly drops out a fraction of the neurons in a neural network during training. This helps to prevent overfitting, which is when a model performs well on the training data but poorly on new data. Dropout forces the network to learn more robust features by preventing any one neuron from becoming too important in the network's predictions.
- **Numpy:** It is for performing mathematical functions
- **Matplotlib:**Matplotlib is a data visualization library in Python that is widely used for creating high-quality, publication-ready plots and charts.

- **clear_output:** command is used to clear the output of a Jupyter notebook cell. This can be useful when you want to update the output of a cell with new information, or when you want to remove previous output that is no longer relevant.

Activity 2: Configuration of Images and preprocessing



```
model = Sequential()
model.add(Convolution2D(32,(3,3), activation = 'relu', input_shape = (224,224,3)))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Convolution2D(32,(3,3)))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Convolution2D(32,(3,3)))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Convolution2D(32,(3,3)))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Convolution2D(32,(3,3)))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Flatten())
model.add(Dense(150, activation = 'relu'))
model.add(Dense(14, activation = 'softmax'))
```

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.

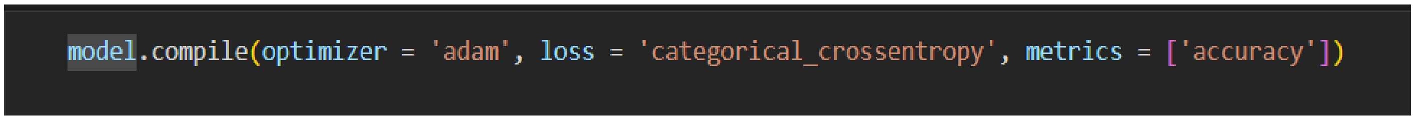
batch size: Size of the batches of data which is 64.

target size: Size to resize images after they are read from disk.

Learning Rate (LR): The learning rate is a hyperparameter that determines the step size at each iteration during gradient descent optimization. Gradient descent is the most common optimization algorithm used in machine learning update the weights of a neural network during training. The learning rate controls how quickly or slowly the weights are updated in response to the error gradient. A high learning rate can cause the algorithm to converge too quickly, whereas a low learning rate can cause slow convergence or even prevent the model from converging.

Seeds: Seeds are used in machine learning to ensure that results are reproducible. A seed is a random number that is used to initialize the random number generator before training the model. By setting the seed value, we can ensure that the same sequence of random numbers is generated every time the code is run. This is important when developing models as it allows us to compare results between different runsand ensure that any changes to the model or hyperparameters are actually improving performance.

Now it is time to Build input and output layers for Transfer Learning model



```
model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
```

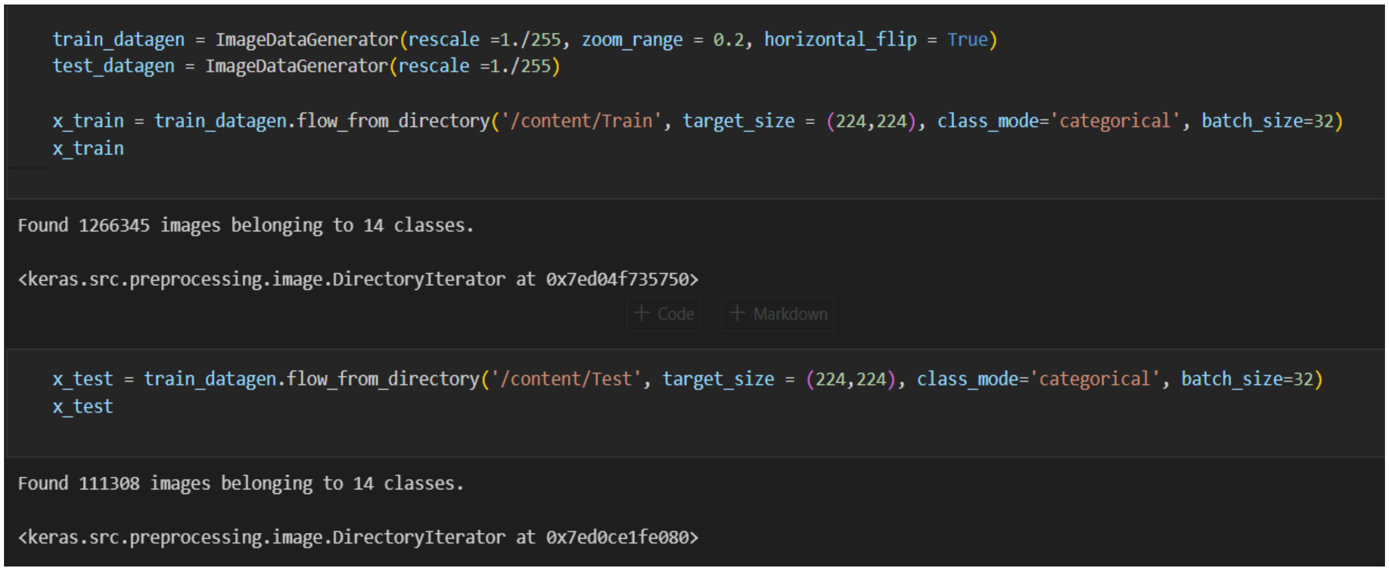
Hidden layers freeze because they have trained sequence, so changing the input and output layers

Activity 3: Apply Image_Dataset_from_directory functionality to Train set and Test set

ImageDataset.from_directory() is a function from the TensorFlow library used to load images from a directory and create a dataset object that can be used for machine learning tasks, such as image classification or object detection.

The function takes the following arguments:

- **directory**: A string representing the directory where the images are located.
- **labels**: A list of strings representing the class labels for the images.
- **batch_size**: An integer representing the number of images to load in each batch.
- **image_size**: A tuple representing the size to which the images should be resized.
-



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

x_train = train_datagen.flow_from_directory('/content/Train', target_size = (224,224), class_mode='categorical', batch_size=32)
x_train
```

```
Found 1266345 images belonging to 14 classes.

<keras.src.preprocessing.image.DirectoryIterator at 0x7ed04f735750>
```

```
x_test = train_datagen.flow_from_directory('/content/Test', target_size = (224,224), class_mode='categorical', batch_size=32)
x_test
```

```
Found 111308 images belonging to 14 classes.

<keras.src.preprocessing.image.DirectoryIterator at 0x7ed0ce1fe080>
```

Milestone 3: Model Building

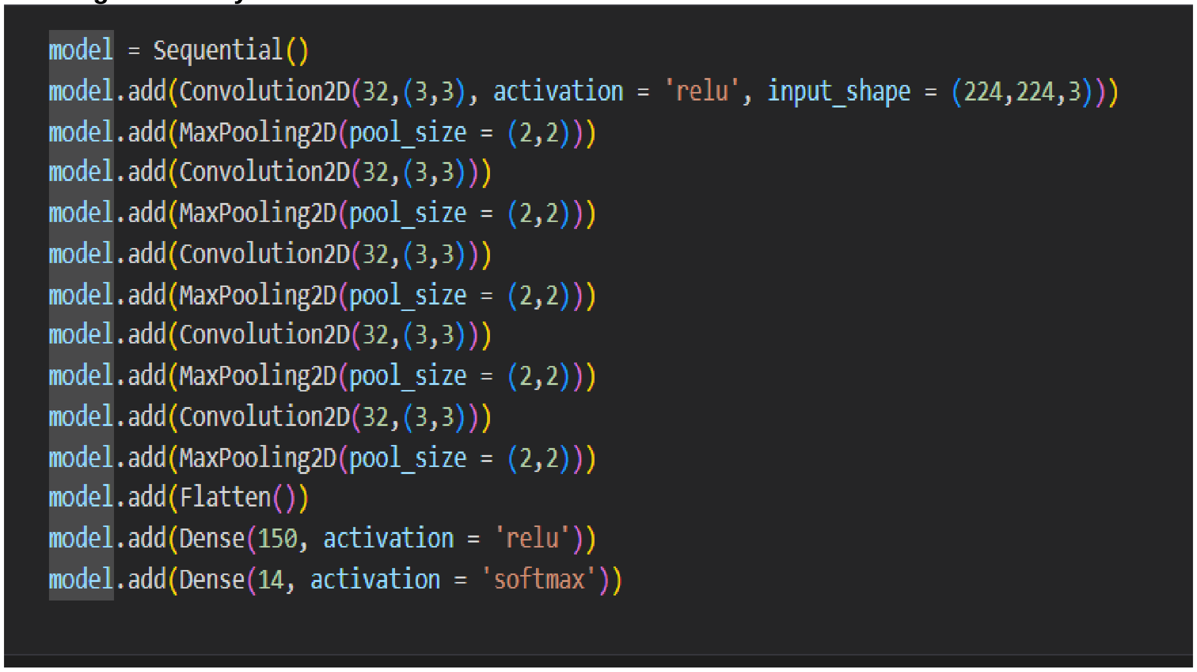
Activity 1: Create Transfer Learning Function
Now, let us create transfer learning function with DenseNet121 with parameters include_top, and weights imagenet with mentioned input shape. Also, we are setting threshold at 149.

DenseNet is a convolutional neural network where each layer is connected to all other layers that are deeper in the network, that is, the first layer is connected to the 2nd, 3rd, 4th and so on, the second layer is connected to the 3rd, 4th, 5th and so on.

- **include_top**: whether to include the fully-connected layer at the top of the network.
- **weights**: one of None (random initialization), 'imagenet' (pre-training on ImageNet), or the path to the weights file to be loaded.
- **input_shape**: optional shape tuple, only to be specified if include_top is False (otherwise the input shape has to be (224, 224, 3)

Activity 2:

Adding Dense Layers

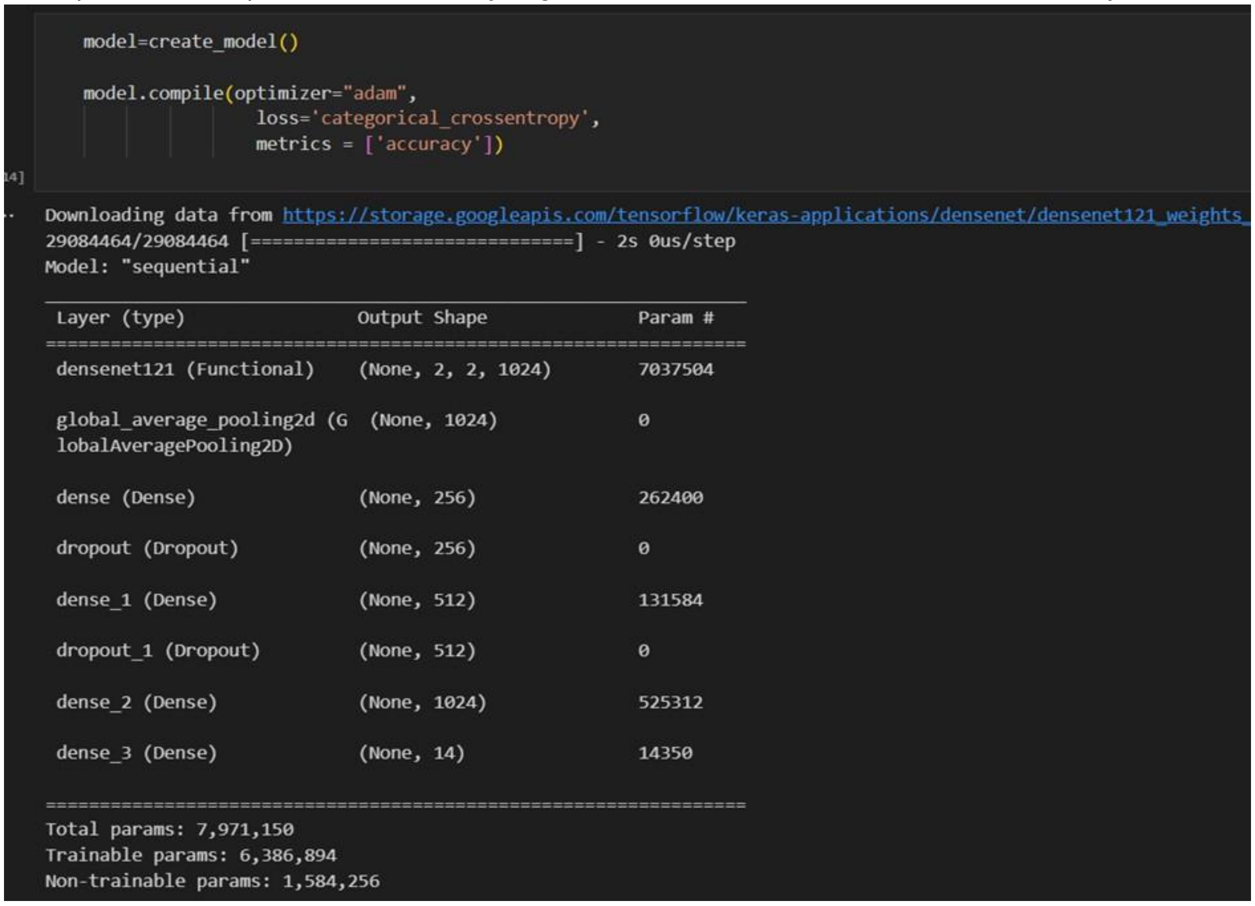


```
model = Sequential()
model.add(Convolution2D(32,(3,3), activation = 'relu', input_shape = (224,224,3)))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Convolution2D(32,(3,3)))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Convolution2D(32,(3,3)))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Convolution2D(32,(3,3)))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Convolution2D(32,(3,3)))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Flatten())
model.add(Dense(150, activation = 'relu'))
model.add(Dense(14, activation = 'softmax'))
```

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.



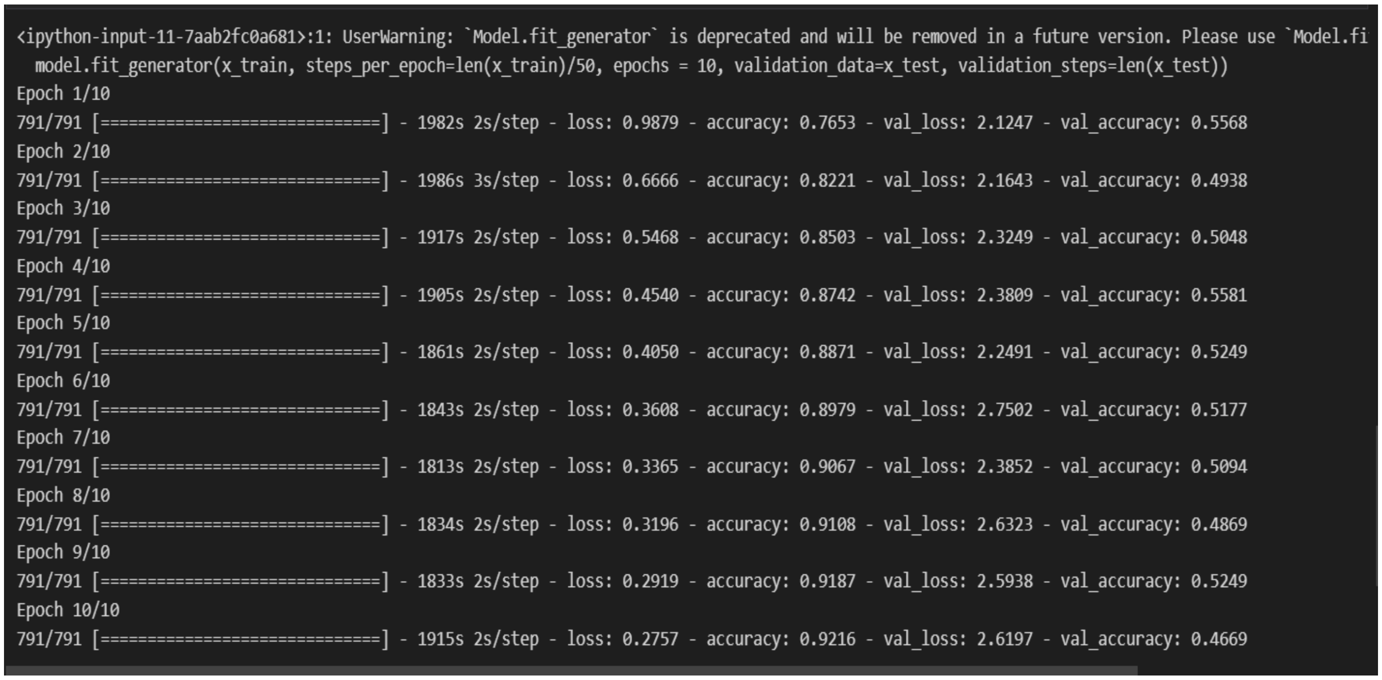
```
model=create_model()

model.compile(optimizer="adam",
                loss='categorical_crossentropy',
                metrics = ['accuracy'])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/densenet/densenet121_weights_
29084464/29084464 [==============================] - 2s 0us/step
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 densenet121 (Functional)    (None, 2, 2, 1024)        7037504

 global_average_pooling2d (G  (None, 1024)             0
 lobalAveragePooling2D)

 dense (Dense)               (None, 256)               262400

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 512)               131584

 dropout_1 (Dropout)         (None, 512)               0

 dense_2 (Dense)             (None, 1024)              525312

 dense_3 (Dense)             (None, 14)                14350

=================================================================
Total params: 7,971,150
Trainable params: 6,386,894
Non-trainable params: 1,584,256
```

Activity 3: 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



```
<ipython-input-11-7aab2fc0a681>:1: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fi
  model.fit_generator(x_train, steps_per_epoch=len(x_train)/50, epochs = 10, validation_data=x_test, validation_steps=len(x_test))
Epoch 1/10
791/791 [==============================] - 1982s 2s/step - loss: 0.9879 - accuracy: 0.7653 - val_loss: 2.1247 - val_accuracy: 0.5568
Epoch 2/10
791/791 [==============================] - 1986s 3s/step - loss: 0.6666 - accuracy: 0.8221 - val_loss: 2.1643 - val_accuracy: 0.4938
Epoch 3/10
791/791 [==============================] - 1917s 2s/step - loss: 0.5468 - accuracy: 0.8503 - val_loss: 2.3249 - val_accuracy: 0.5048
Epoch 4/10
791/791 [==============================] - 1905s 2s/step - loss: 0.4540 - accuracy: 0.8742 - val_loss: 2.3809 - val_accuracy: 0.5581
Epoch 5/10
791/791 [==============================] - 1861s 2s/step - loss: 0.4050 - accuracy: 0.8871 - val_loss: 2.2491 - val_accuracy: 0.5249
Epoch 6/10
791/791 [==============================] - 1843s 2s/step - loss: 0.3608 - accuracy: 0.8979 - val_loss: 2.7502 - val_accuracy: 0.5177
Epoch 7/10
791/791 [==============================] - 1813s 2s/step - loss: 0.3365 - accuracy: 0.9067 - val_loss: 2.3852 - val_accuracy: 0.5094
Epoch 8/10
791/791 [==============================] - 1834s 2s/step - loss: 0.3196 - accuracy: 0.9108 - val_loss: 2.6323 - val_accuracy: 0.4869
Epoch 9/10
791/791 [==============================] - 1833s 2s/step - loss: 0.2919 - accuracy: 0.9187 - val_loss: 2.5938 - val_accuracy: 0.5249
Epoch 10/10
791/791 [==============================] - 1915s 2s/step - loss: 0.2757 - accuracy: 0.9216 - val_loss: 2.6197 - val_accuracy: 0.4669
```

Metrics are used to evaluate the performance of your model. It is similar to the loss function, but not used in the training process

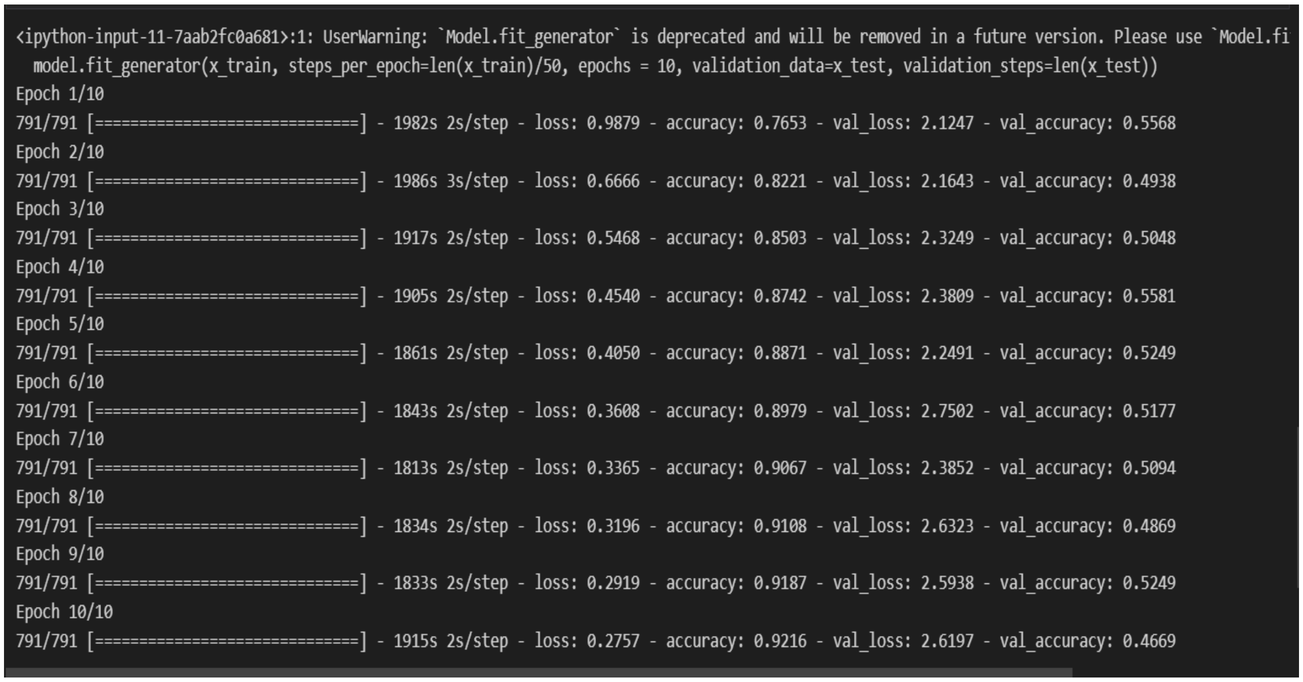
Activity 4: Train the model

Now, let us train our model with our image dataset. The model is trained for 5 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.

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.



```
<ipython-input-11-7aab2fc0a681>:1: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fi
  model.fit_generator(x_train, steps_per_epoch=len(x_train)/50, epochs = 10, validation_data=x_test, validation_steps=len(x_test))
Epoch 1/10
791/791 [==============================] - 1982s 2s/step - loss: 0.9879 - accuracy: 0.7653 - val_loss: 2.1247 - val_accuracy: 0.5568
Epoch 2/10
791/791 [==============================] - 1986s 3s/step - loss: 0.6666 - accuracy: 0.8221 - val_loss: 2.1643 - val_accuracy: 0.4938
Epoch 3/10
791/791 [==============================] - 1917s 2s/step - loss: 0.5468 - accuracy: 0.8503 - val_loss: 2.3249 - val_accuracy: 0.5048
Epoch 4/10
791/791 [==============================] - 1905s 2s/step - loss: 0.4540 - accuracy: 0.8742 - val_loss: 2.3809 - val_accuracy: 0.5581
Epoch 5/10
791/791 [==============================] - 1861s 2s/step - loss: 0.4050 - accuracy: 0.8871 - val_loss: 2.2491 - val_accuracy: 0.5249
Epoch 6/10
791/791 [==============================] - 1843s 2s/step - loss: 0.3608 - accuracy: 0.8979 - val_loss: 2.7502 - val_accuracy: 0.5177
Epoch 7/10
791/791 [==============================] - 1813s 2s/step - loss: 0.3365 - accuracy: 0.9067 - val_loss: 2.3852 - val_accuracy: 0.5094
Epoch 8/10
791/791 [==============================] - 1834s 2s/step - loss: 0.3196 - accuracy: 0.9108 - val_loss: 2.6323 - val_accuracy: 0.4869
Epoch 9/10
791/791 [==============================] - 1833s 2s/step - loss: 0.2919 - accuracy: 0.9187 - val_loss: 2.5938 - val_accuracy: 0.5249
Epoch 10/10
791/791 [==============================] - 1915s 2s/step - loss: 0.2757 - accuracy: 0.9216 - val_loss: 2.6197 - val_accuracy: 0.4669
```

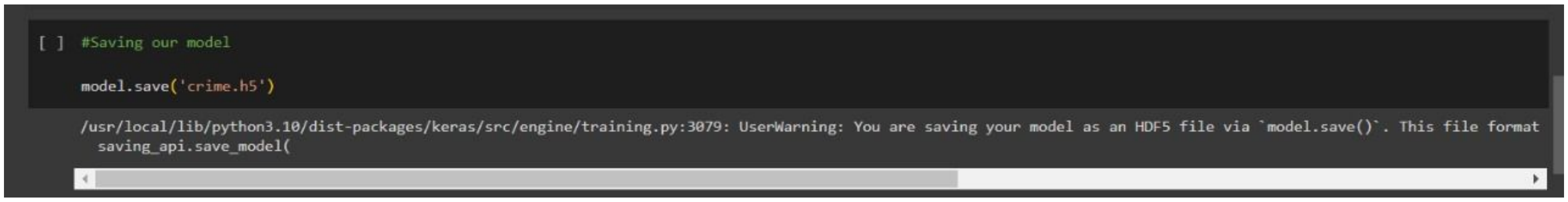
- Epochs: an integer and number of epochs we want to train our model for.

- 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.

Accuracy of the model after 5 epochs.

Milestone 4: Save the Model

The model is saved with .h5 extension as follows



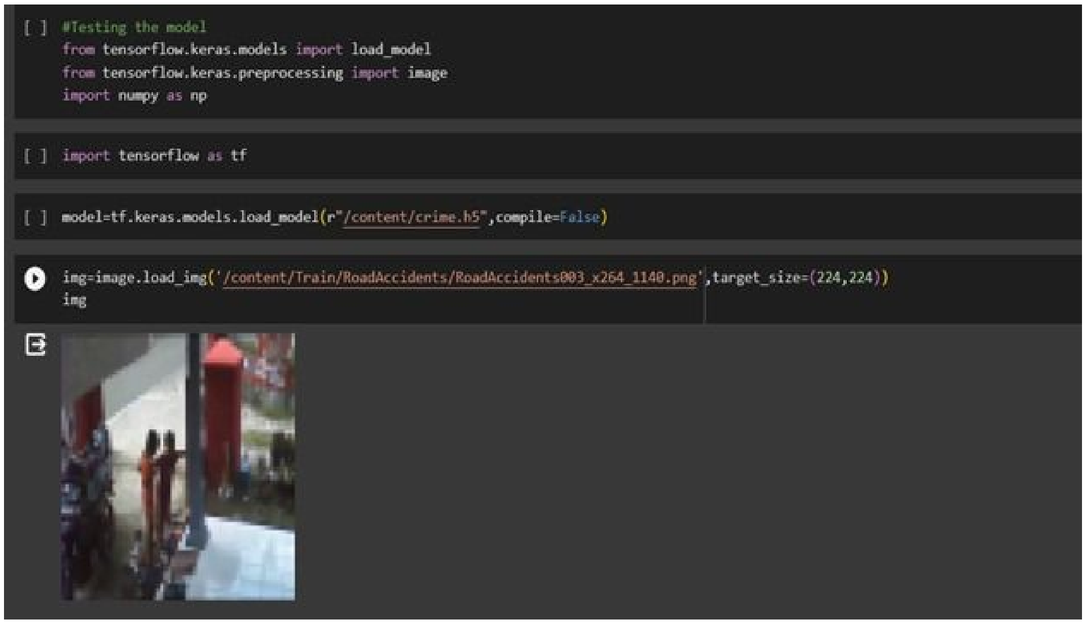
```
#Saving our model

model.save('crime.h5')

/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3079: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format
  saving_api.save_model(
```

An H5 file is a data file saved in the Hierarchical Data Format (HDF). It contains multidimensional arrays of scientific data.

Testing 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

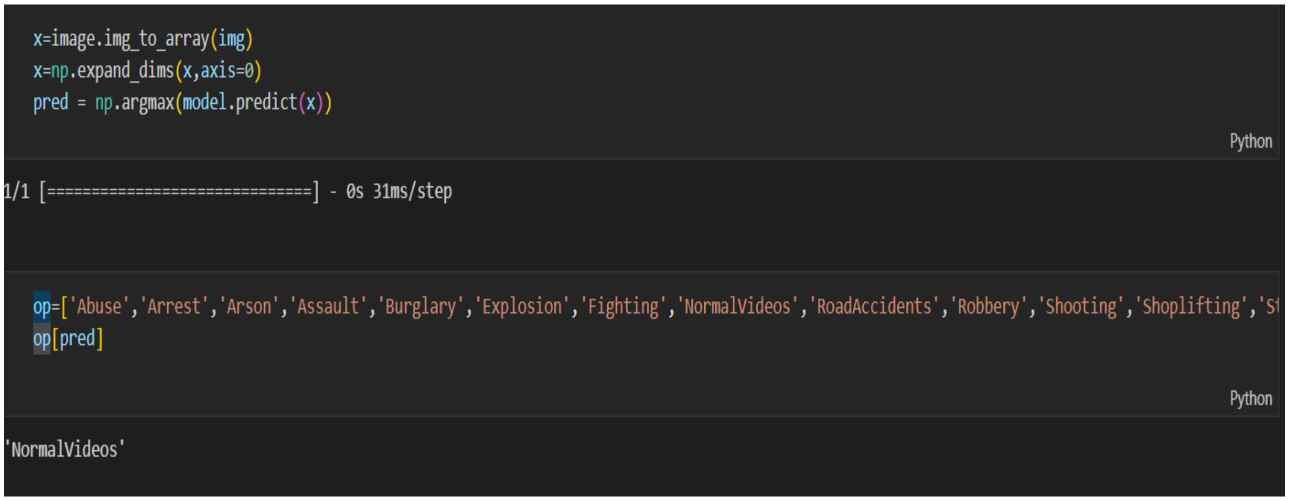


```
#Testing the model
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
import numpy as np
```

```
import tensorflow as tf
```

```
model=tf.keras.models.load_model(r"/content/crime.h5",compile=False)
```

```
img=image.load_img('/content/Train/RoadAccidents/RoadAccidents003_x264_1140.png',target_size=(224,224))
img
```



```
x=image.img_to_array(img)
x=np.expand_dims(x,axis=0)
pred = np.argmax(model.predict(x))
```

Python

```
1/1 [==============================] - 0s 31ms/step
```

```
op=['Abuse','Arrest','Arson','Assault','Burglary','Explosion','Fighting','NormalVideos','RoadAccidents','Robbery','Shooting','Shoplifting','St
op[pred]
```

Python

```
'NormalVideos'
```

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

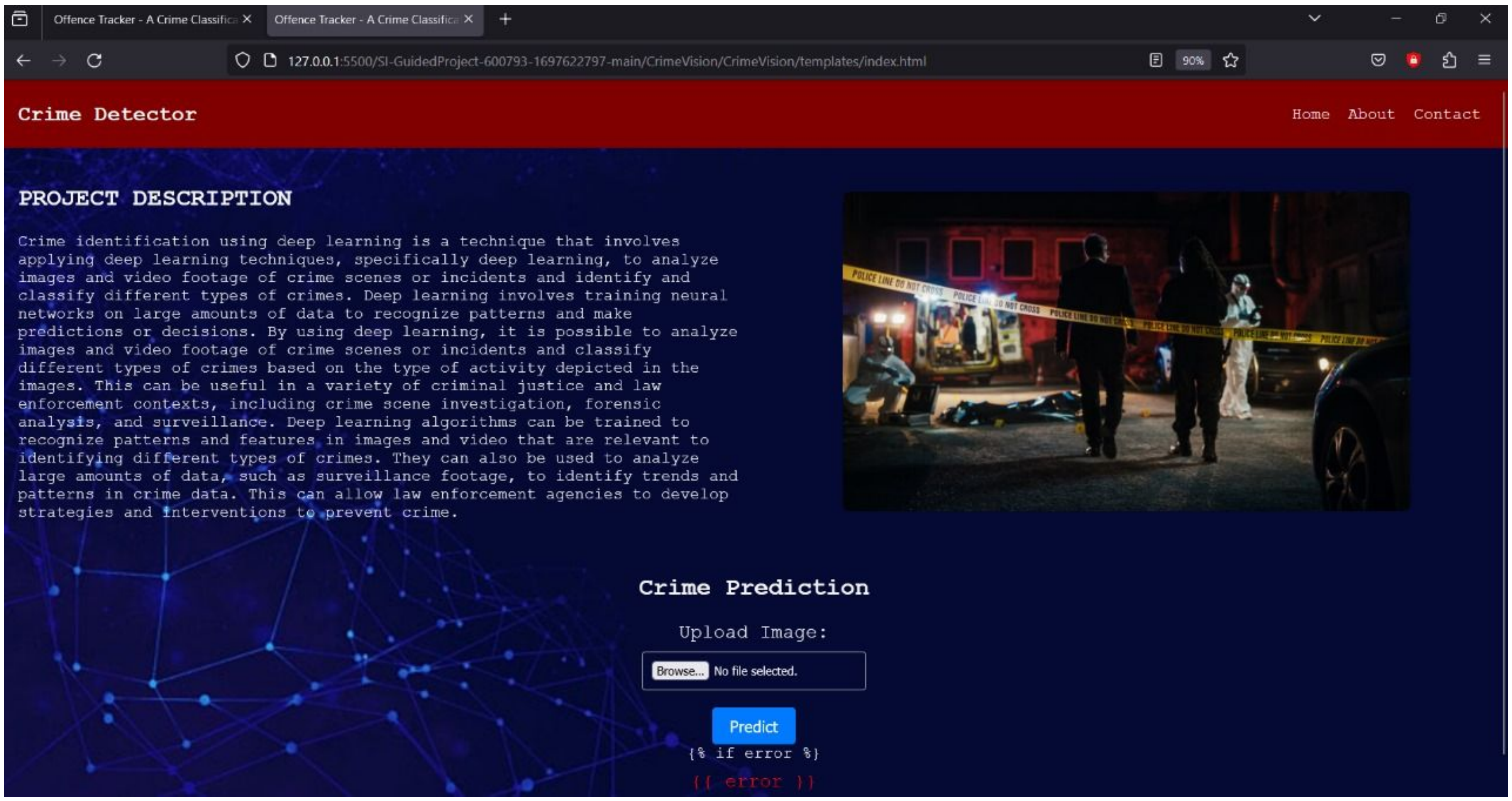
- Building HTML Pages
- Building python code

Activity1: Building Html Pages:

For this project create one HTML file namely

- index.html
- result.html

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


Offence Tracker - A Crime Classifica
Offence Tracker - A Crime Classifica
127.0.0.1:5500/SI-GuidedProject-600793-1697622797-main/CrimeVision/CrimeVision/templates/index.html
90%
Crime Detector
Home
About
Contact
POLICE LINE DO NOT CROSS
Browse...
No file selected.
Predict

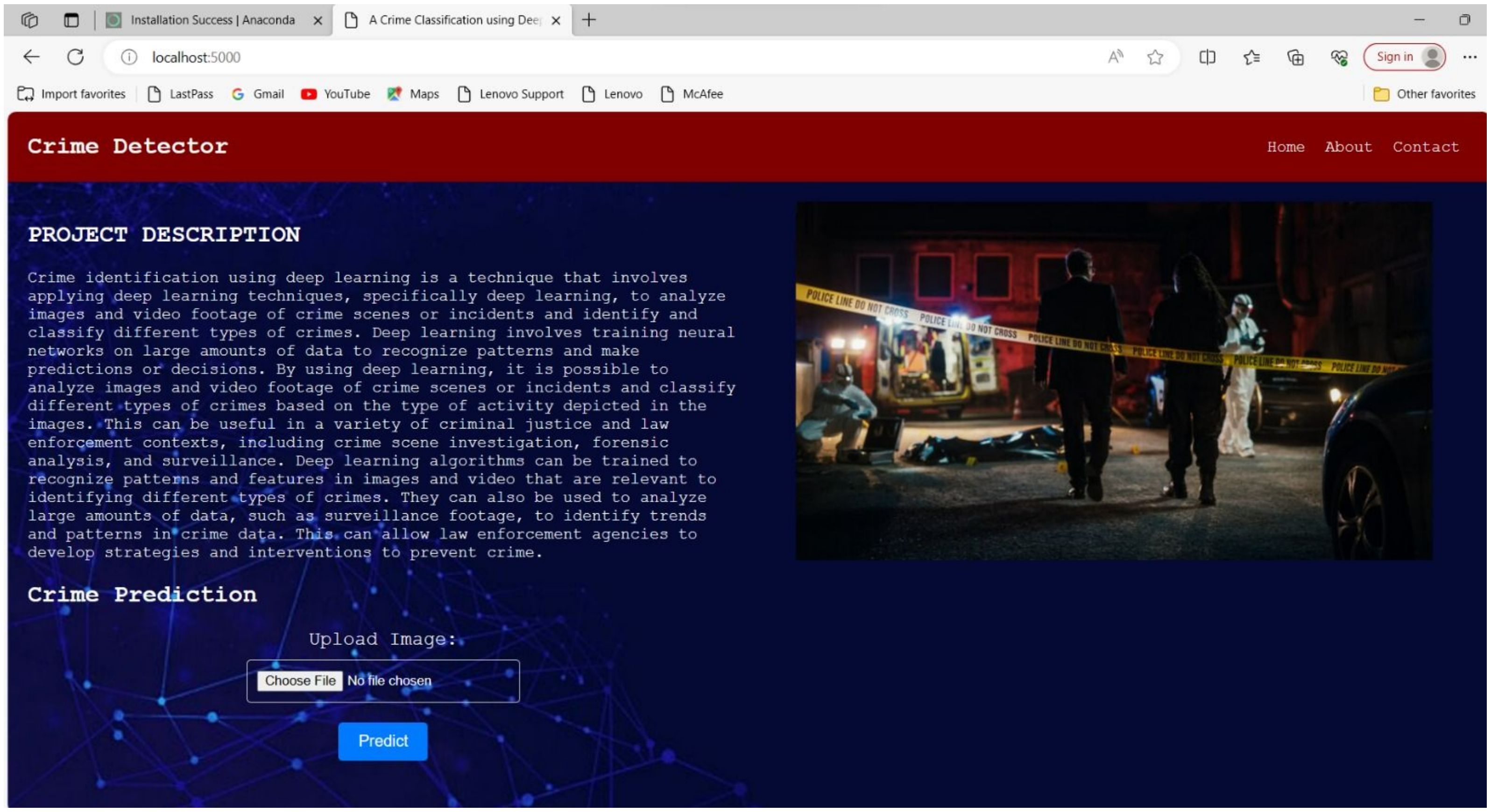
PROJECT DESCRIPTION

Crime identification using deep learning is a technique that involves applying deep learning techniques, specifically deep learning, to analyze images and video footage of crime scenes or incidents and identify and classify different types of crimes. Deep learning involves training neural networks on large amounts of data to recognize patterns and make predictions or decisions. By using deep learning, it is possible to analyze images and video footage of crime scenes or incidents and classify different types of crimes based on the type of activity depicted in the images. This can be useful in a variety of criminal justice and law enforcement contexts, including crime scene investigation, forensic analysis, and surveillance. Deep learning algorithms can be trained to recognize patterns and features in images and video that are relevant to identifying different types of crimes. They can also be used to analyze large amounts of data, such as surveillance footage, to identify trends and patterns in crime data. This can allow law enforcement agencies to develop strategies and interventions to prevent crime.

Crime Prediction

Upload Image:

{% if error %}
{{ error }}



PROJECT DESCRIPTION

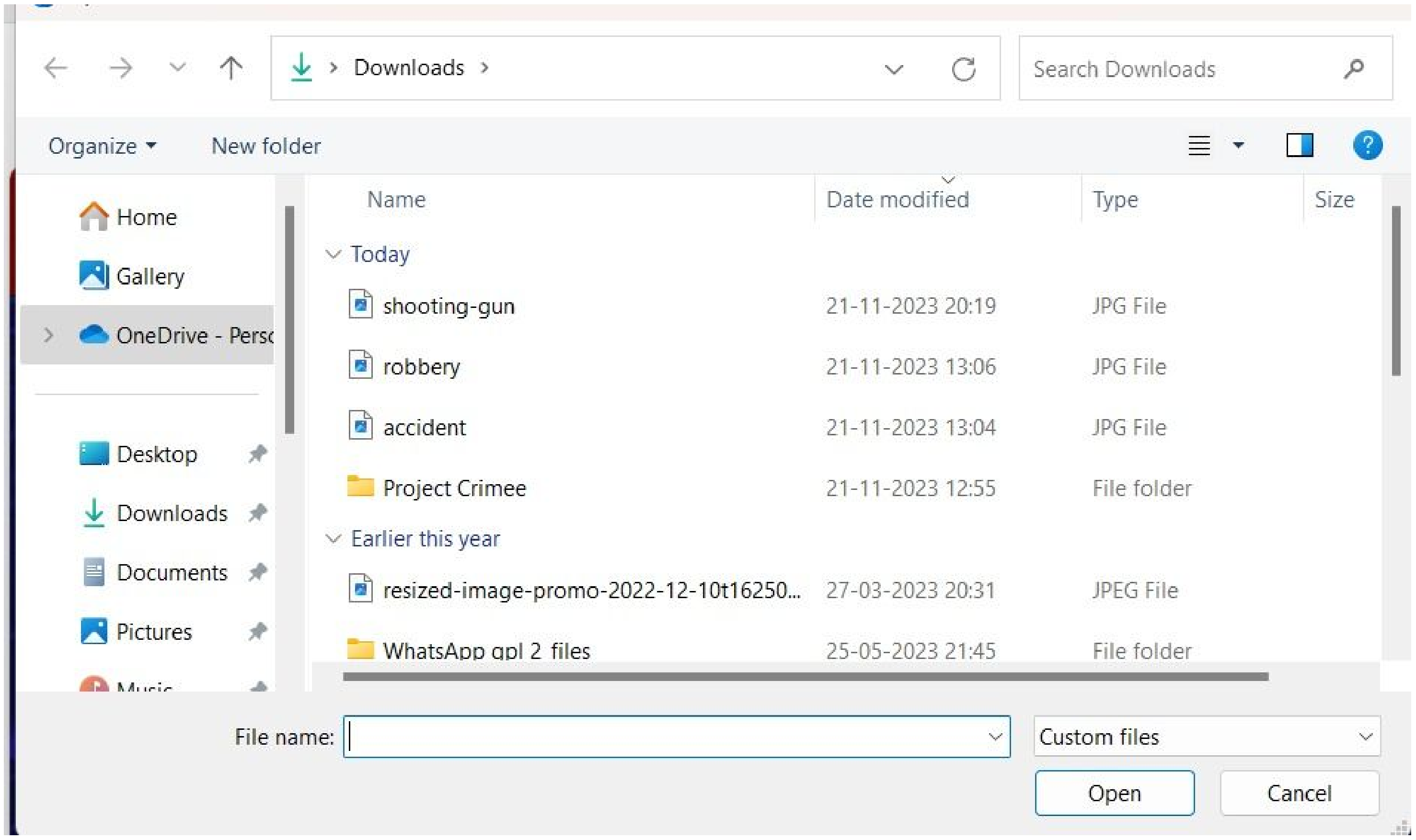
Crime identification using deep learning is a technique that involves applying deep learning techniques, specifically deep learning, to analyze images and video footage of crime scenes or incidents and identify and classify different types of crimes. Deep learning involves training neural networks on large amounts of data to recognize patterns and make predictions or decisions. By using deep learning, it is possible to analyze images and video footage of crime scenes or incidents and classify different types of crimes based on the type of activity depicted in the images. This can be useful in a variety of criminal justice and law enforcement contexts, including crime scene investigation, forensic analysis, and surveillance. Deep learning algorithms can be trained to recognize patterns and features in images and video that are relevant to identifying different types of crimes. They can also be used to analyze large amounts of data, such as surveillance footage, to identify trends and patterns in crime data. This can allow law enforcement agencies to develop strategies and interventions to prevent crime.

Crime Prediction

Upload Image:

When you click on the Choose file button, it will redirect you to the below page:

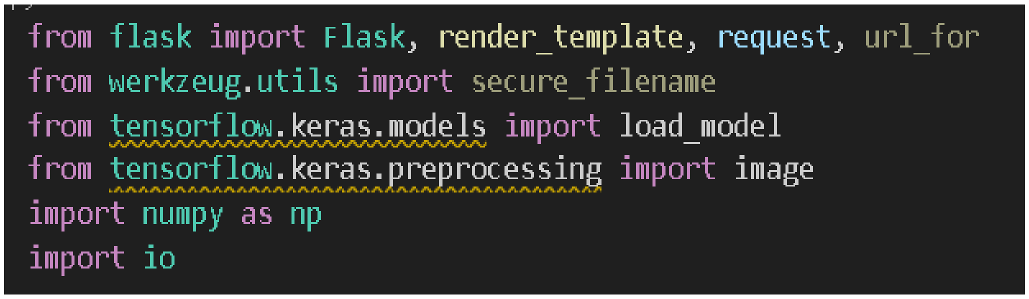
From here you can choose different images for getting prediction



Name	Date modified	Type	Size
Today			
shooting-gun	21-11-2023 20:19	JPG File	
robbery	21-11-2023 13:06	JPG File	
accident	21-11-2023 13:04	JPG File	
Project Crimee	21-11-2023 12:55	File folder	
Earlier this year			
resized-image-promo-2022-12-10t16250...	27-03-2023 20:31	JPEG File	
WhatsApp gpl 2 files	25-05-2023 21:45	File folder	

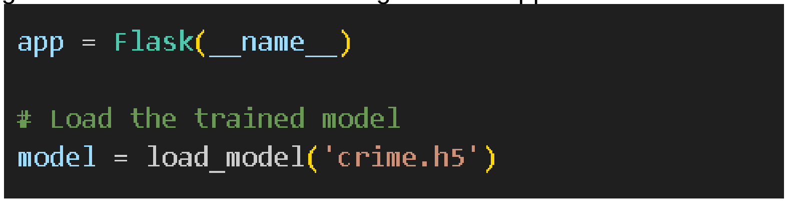
Activity 2: Build Python code:

Import the libraries



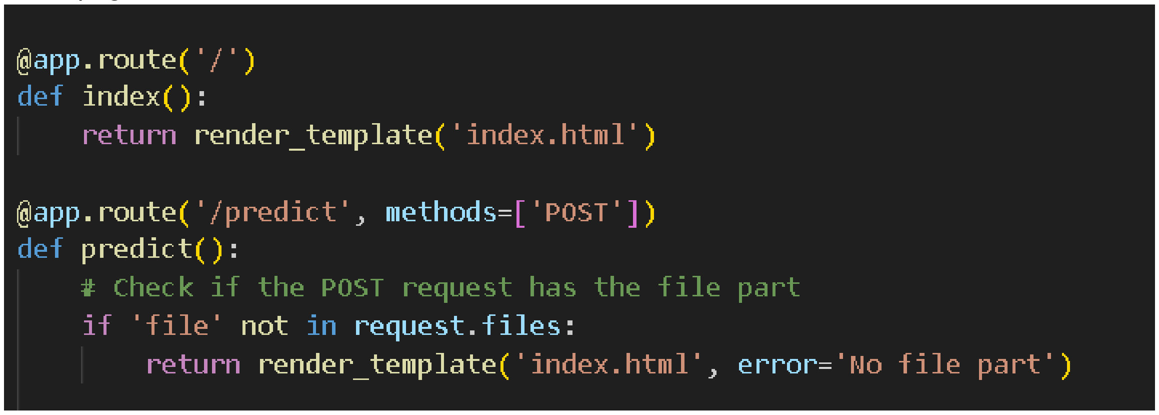
```python
from flask import Flask, render_template, request, url_for
from werkzeug.utils import secure_filename
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
import numpy as np
import io
```

Loading the saved model and initializing the flask app



```python
app = Flask(__name__)

# Load the trained model
model = load_model('crime.h5')
```

Render HTML pages:

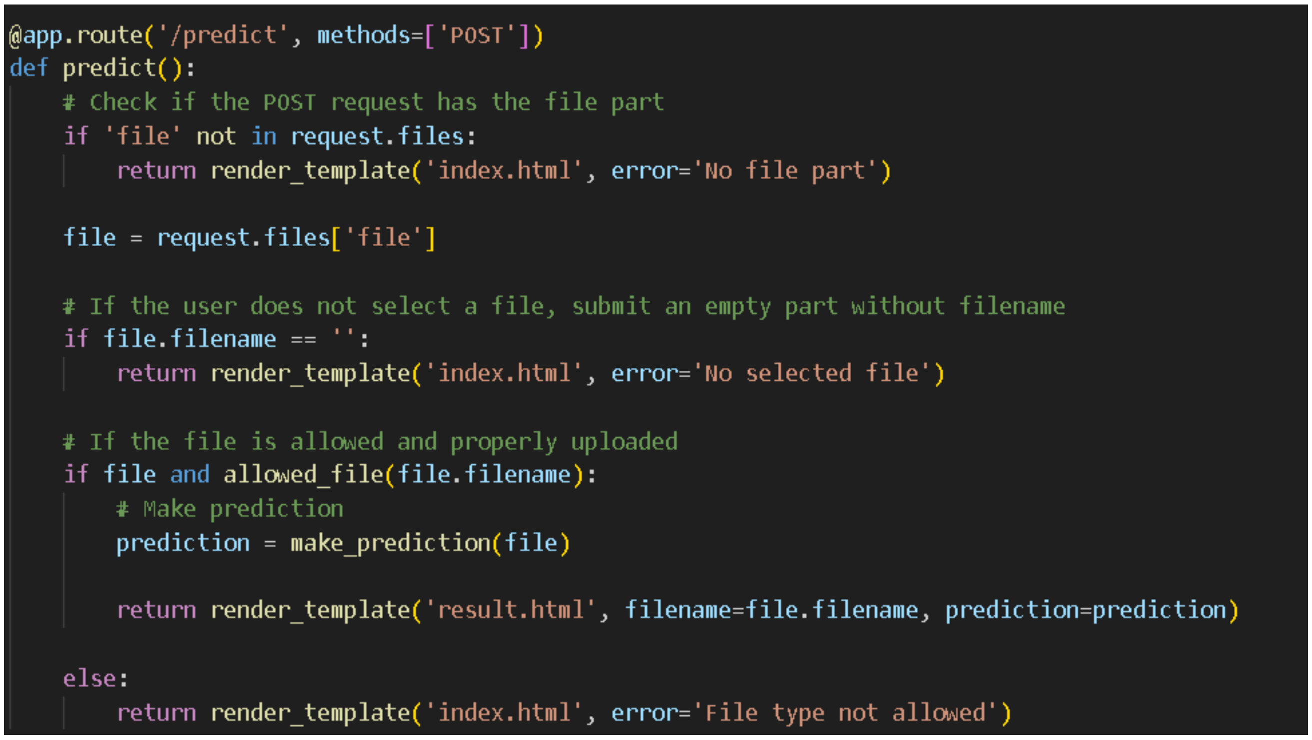


```python
@app.route('/')
def index():
    return render_template('index.html')

@app.route('/predict', methods=['POST'])
def predict():
    # Check if the POST request has the file part
    if 'file' not in request.files:
        return render_template('index.html', error='No file part')
```

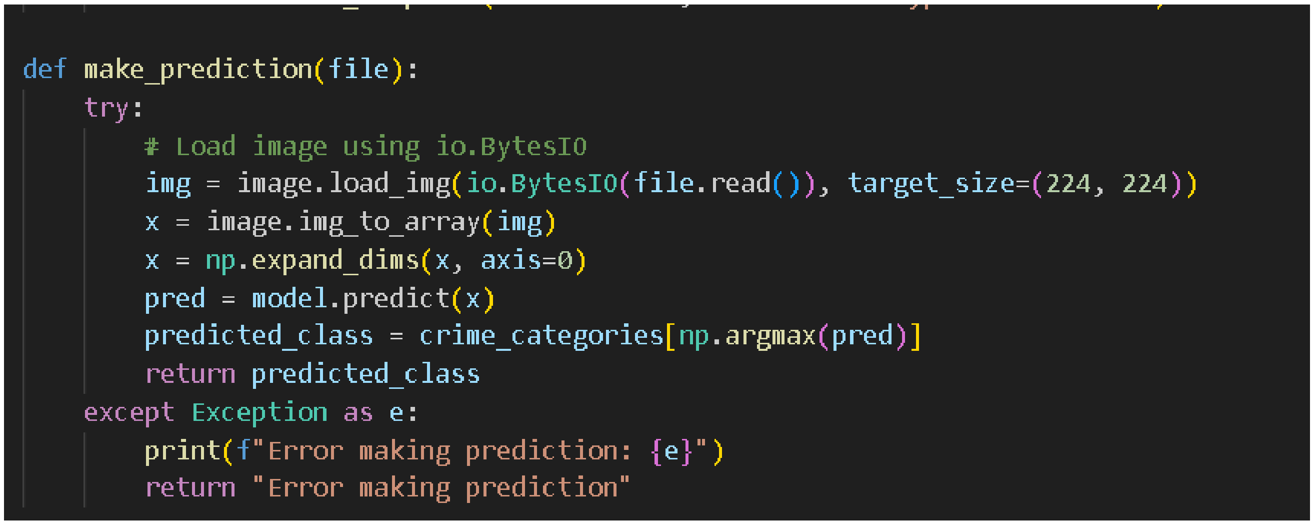
Once we uploaded the file into the app, then verifying the file uploaded properly or not. Here we will be using declared constructor to route to the HTML page which we have created earlier.

In the above example, '/' URL is bound with prediction.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.

Retrieves the value from UI:



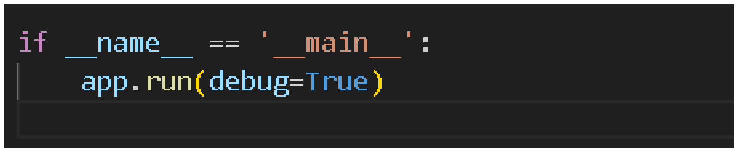
```python
@app.route('/predict', methods=['POST'])
def predict():
    # Check if the POST request has the file part
    if 'file' not in request.files:
        return render_template('index.html', error='No file part')

    file = request.files['file']

    # If the user does not select a file, submit an empty part without filename
    if file.filename == '':
        return render_template('index.html', error='No selected file')

    # If the file is allowed and properly uploaded
    if file and allowed_file(file.filename):
        # Make prediction
        prediction = make_prediction(file)

        return render_template('result.html', filename=file.filename, prediction=prediction)

    else:
        return render_template('index.html', error='File type not allowed')
```



```python
def make_prediction(file):
    try:
        # Load image using io.BytesIO
        img = image.load_img(io.BytesIO(file.read()), target_size=(224, 224))
        x = image.img_to_array(img)
        x = np.expand_dims(x, axis=0)
        pred = model.predict(x)
        predicted_class = crime_categories[np.argmax(pred)]
        return predicted_class
    except Exception as e:
        print(f"Error making prediction: {e}")
        return "Error making prediction"
```

Here we are routing our app to predict() 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 be rendered to the text that we have mentioned in the submit.html page earlier

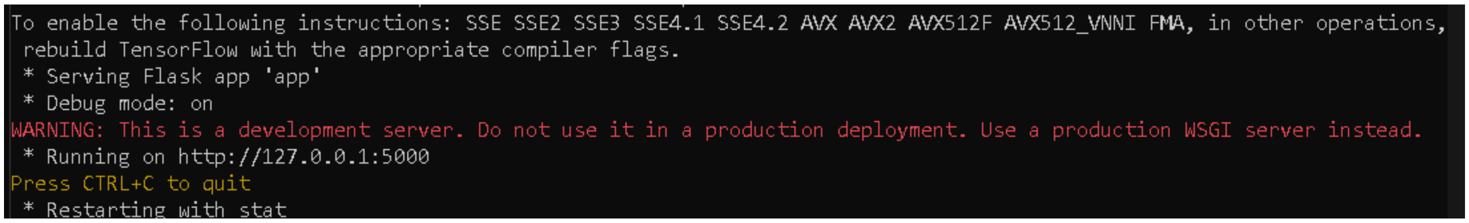
Main Function:



```python
if __name__ == '__main__':
    app.run(debug=True)
```

Activity 3: Run the application

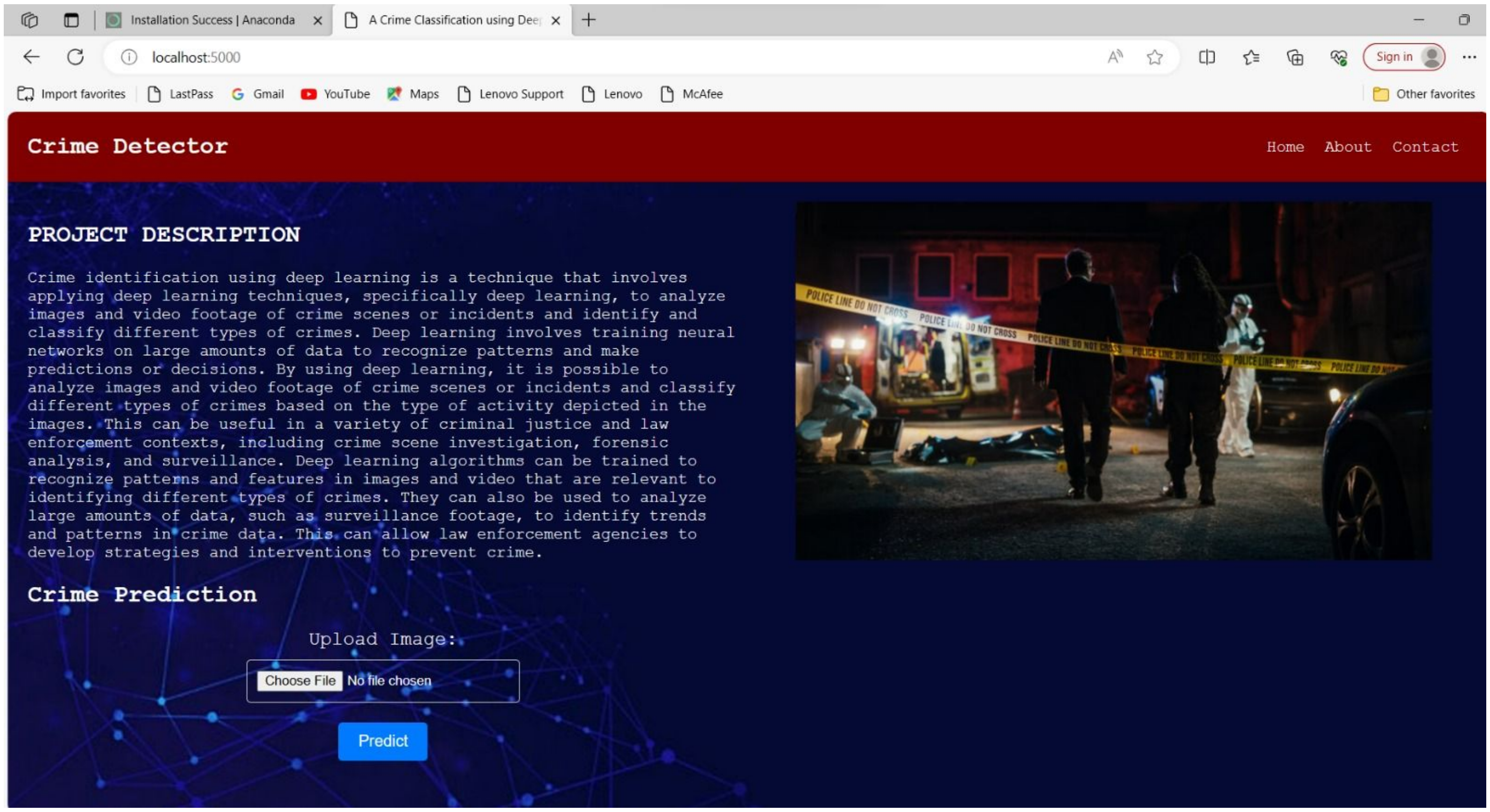
- Open anaconda prompt from the start menu
- Navigate to the folder where your python script is.
- Now type “python app.py” command
- Navigate to the localhost where you can view your web page.
- Click on the predict button from the top left corner, enter the inputs, click on the submit button, and see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



```
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX AVX2 AVX512F AVX512_VNNI FMA, in other operations,
 rebuild TensorFlow with the appropriate compiler flags.
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
```

Now, Go the web browser and write the localhost url (http://127.0.0.1:5000) to get the below result

Index page:



Crime Detector

Home About Contact

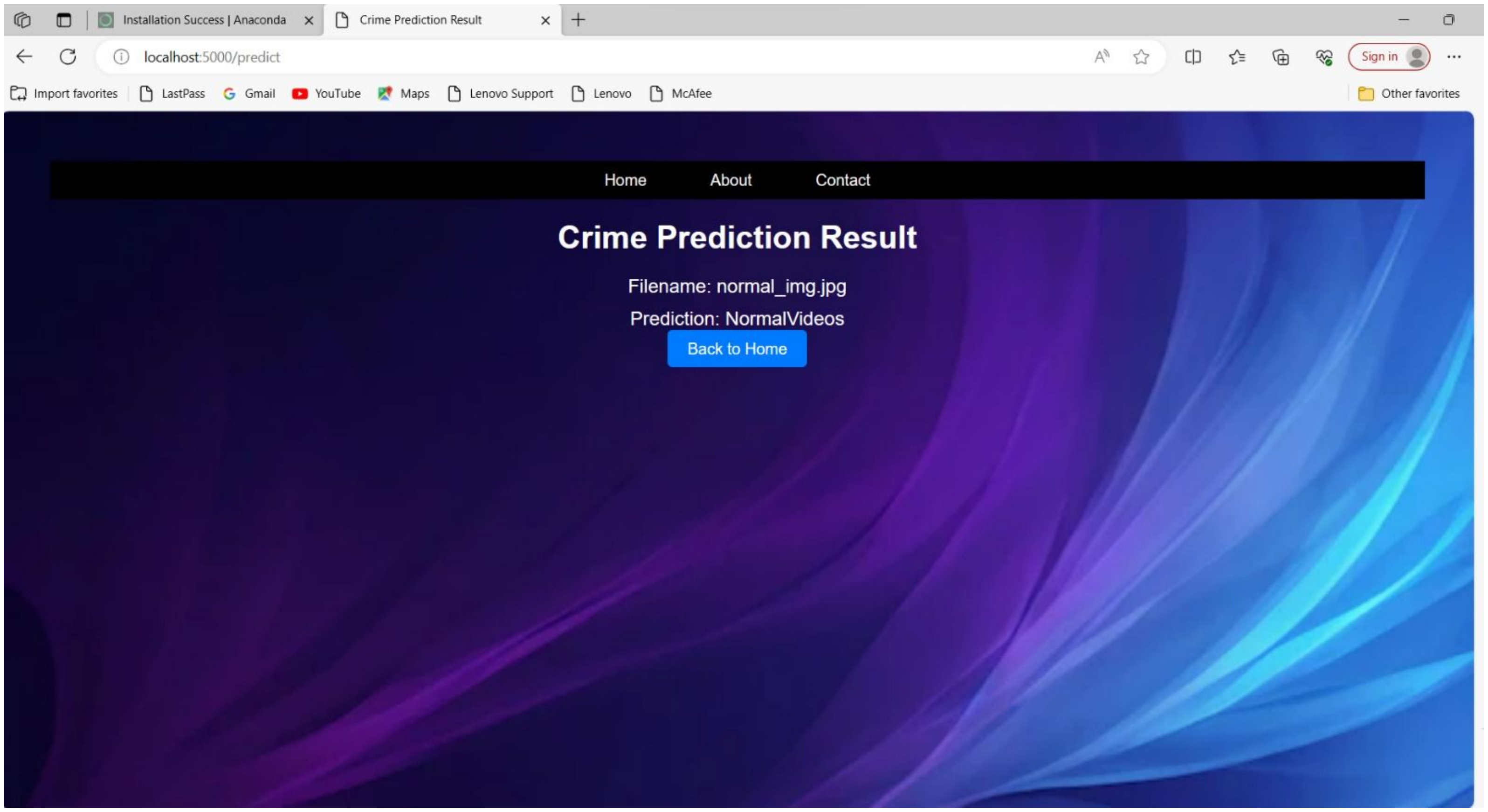
PROJECT DESCRIPTION

Crime identification using deep learning is a technique that involves applying deep learning techniques, specifically deep learning, to analyze images and video footage of crime scenes or incidents and identify and classify different types of crimes. Deep learning involves training neural networks on large amounts of data to recognize patterns and make predictions or decisions. By using deep learning, it is possible to analyze images and video footage of crime scenes or incidents and classify different types of crimes based on the type of activity depicted in the images. This can be useful in a variety of criminal justice and law enforcement contexts, including crime scene investigation, forensic analysis, and surveillance. Deep learning algorithms can be trained to recognize patterns and features in images and video that are relevant to identifying different types of crimes. They can also be used to analyze large amounts of data, such as surveillance footage, to identify trends and patterns in crime data. This can allow law enforcement agencies to develop strategies and interventions to prevent crime.

Crime Prediction

Upload Image:

Prediction page:



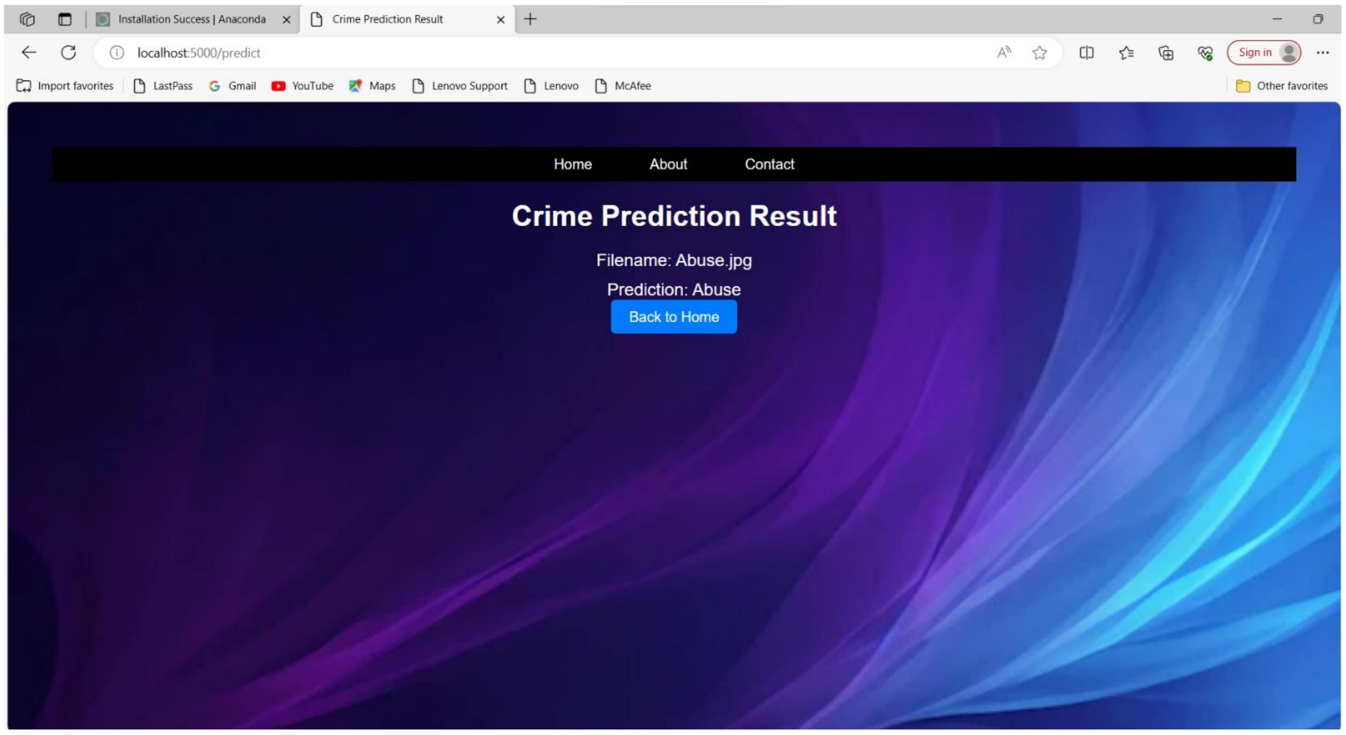
Crime Prediction Result

Filename: normal_img.jpg

Prediction: NormalVideos

Predicting with various input images

Input 1 and output:

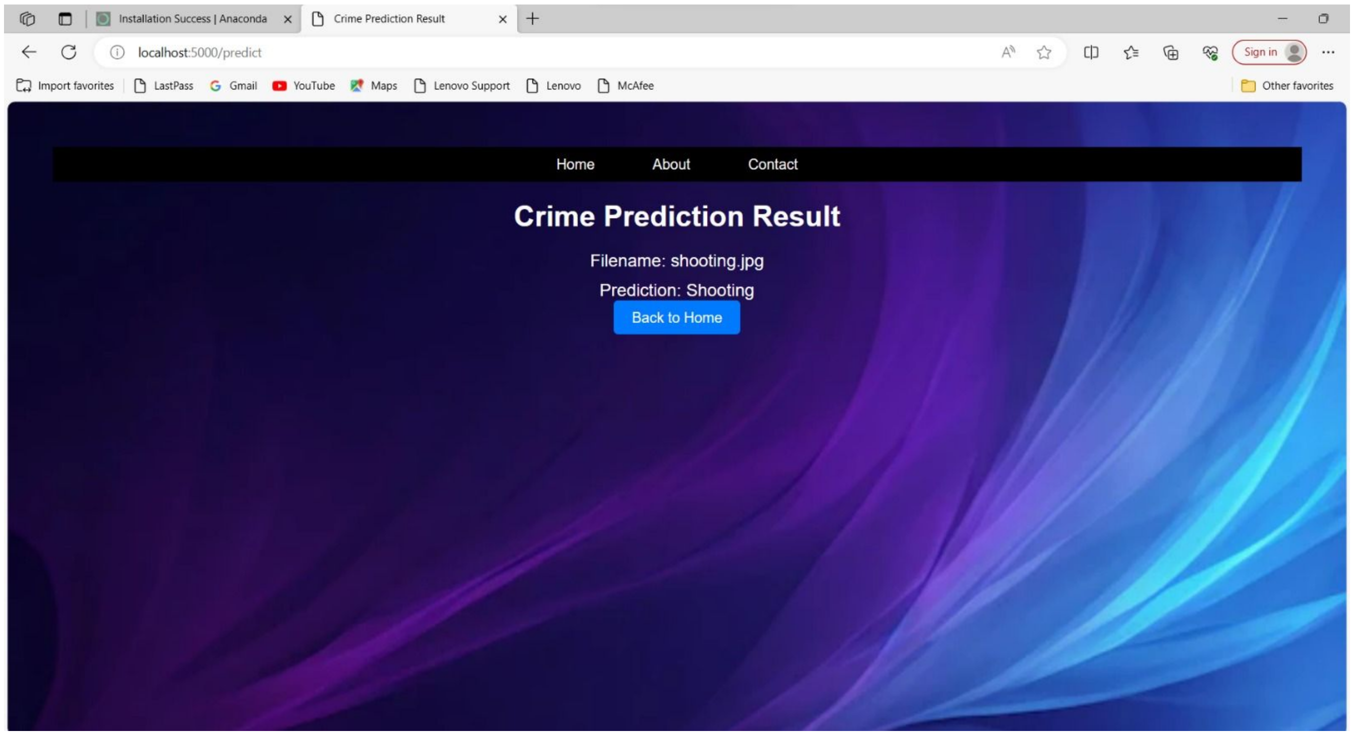


Crime Prediction Result

Filename: Abuse.jpg

Prediction: Abuse

Input 2 and output:

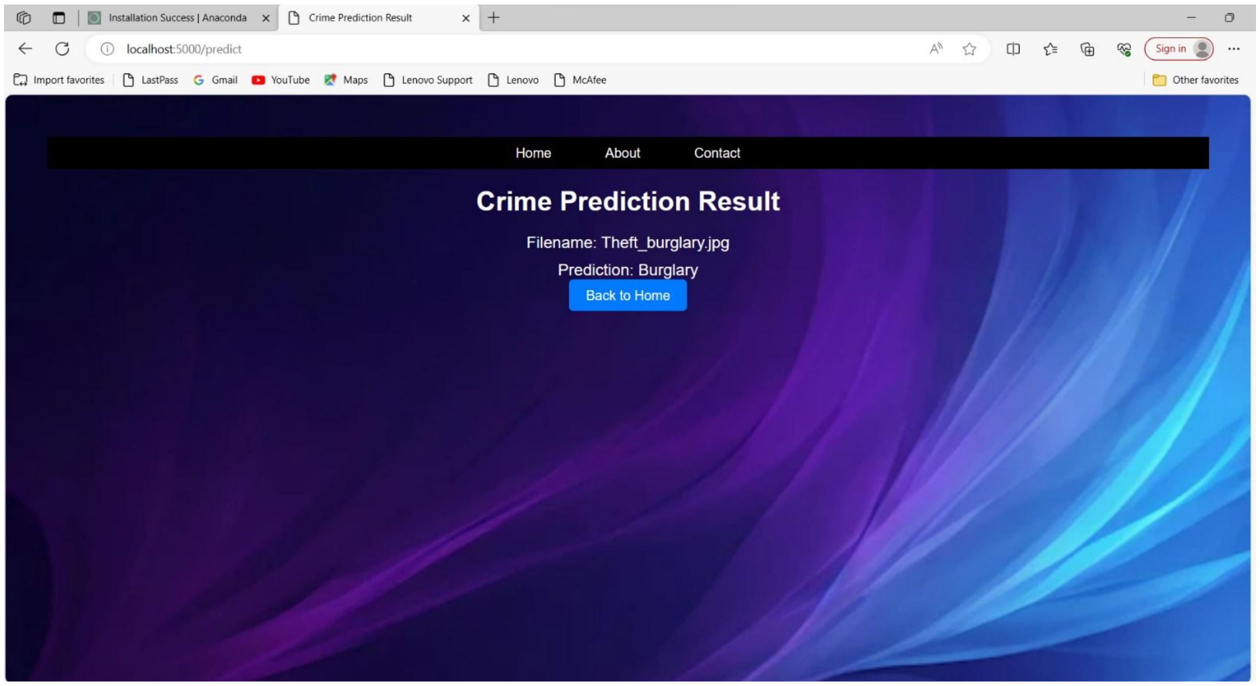


Crime Prediction Result

Filename: shooting.jpg

Prediction: Shooting

Input 3 and output :

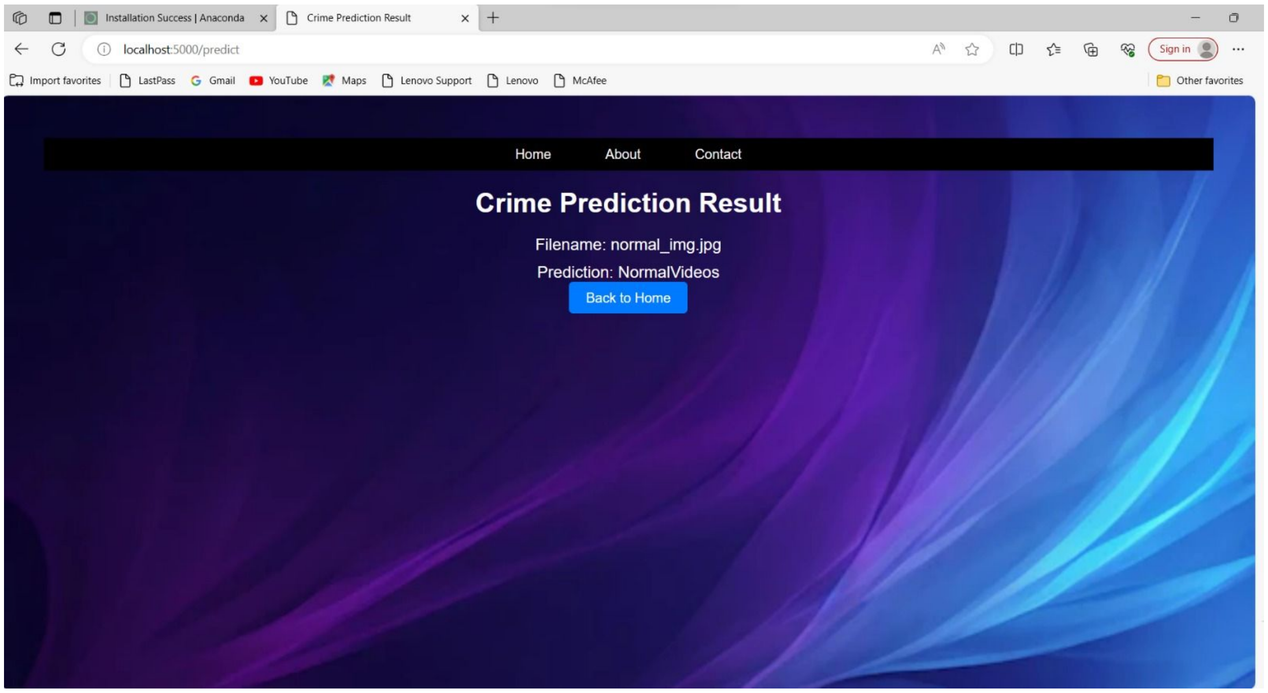


Crime Prediction Result

Filename: Theft_burglary.jpg

Prediction: Burglary

Input 4 and output:



Crime Prediction Result

Filename: normal_img.jpg

Prediction: NormalVideos