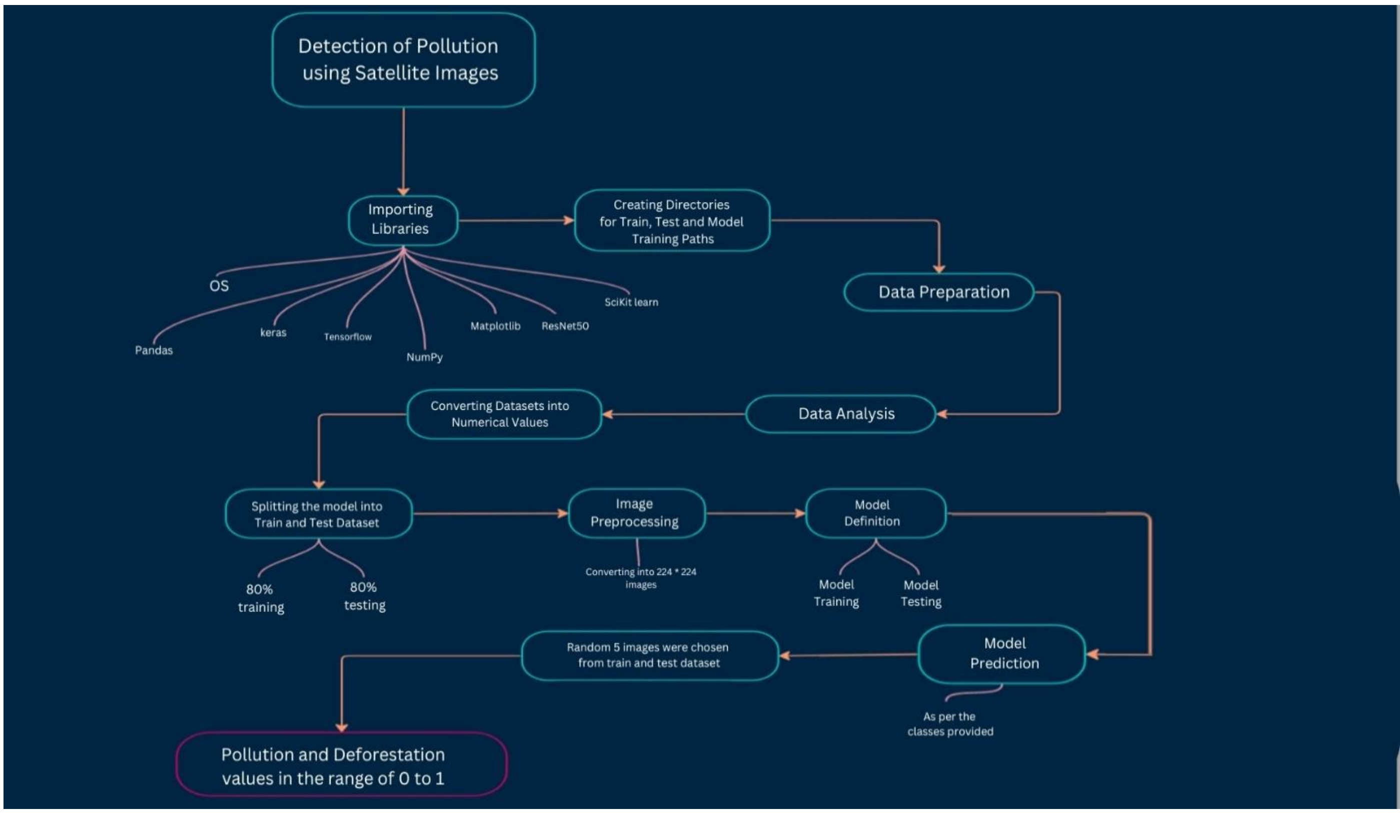
Project Block Diagram -


Detection of Pollution using Satellite Images
Importing Libraries
Creating Directories for Train, Test and Model Training Paths
OS
Pandas
keras
Tensorflow
NumPy
Matplotlib
ResNet50
SciKit learn
Data Preparation
Data Analysis
Converting Datasets into Numerical Values
Splitting the model into Train and Test Dataset
80% training
80% testing
Image Preprocessing
Converting into 224 * 224 images
Model Definition
Model Training
Model Testing
Model Prediction
As per the classes provided
Random 5 images were chosen from train and test dataset
Pollution and Deforestation values in the range of 0 to 1

Deep Learning Model:

It is the most Important part of the product to detect the suspicious Activity.

language – We are using the Python Language as it already contains most of the libraries mostly used for Machine Learning and Deep Learning.

IDE To be used – We are using Jupyter Notebook and Google Collab to write and debug the code as here we can execute the line by line code of the python program.

In google Collab we already get all the Libraries pre installed and GPU and TPU server in free.

Model To Be Used –

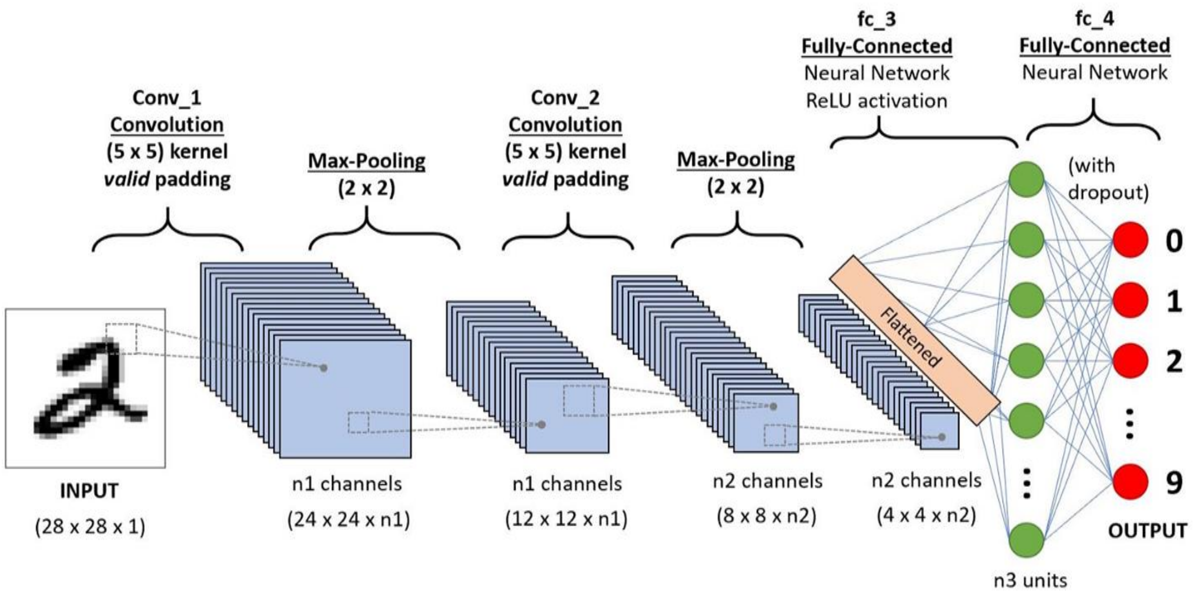
we are using **ResNet50** model for classification of images. As in our model simultaneously detects images so this model will give the better accuracy.

ResNet50 –

ResNet 50 is a powerful model for image classification tasks, and its transfer learning capabilities makes it suitable for a wide range of problems. By fine-tuning the pre-trained model with your specific dataset, you can achieve high accuracy in your classification tasks. The architecture includes 48 convolutional layers, one max pool layer, and one average pool layer. ResNet50 is known for its efficiency and accuracy in image classification tasks.

CNN –

A convolutional neural network (CNN or convnet) is a subset of machine learning. It is one of the various types of artificial neural networks which are used for different applications and data types. A CNN is a kind of network architecture for deep learning algorithms and is specifically used for image recognition and tasks that involve the processing of pixel data.



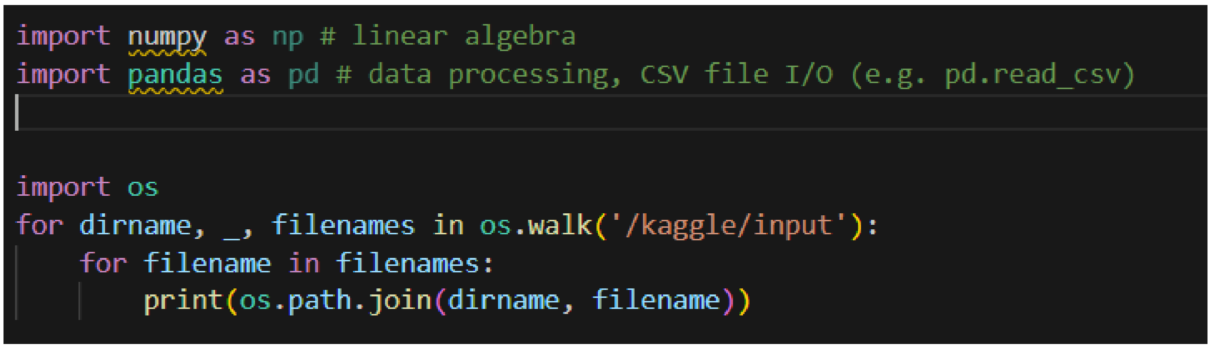
TensorFlow, Keras, Numpy , Pandas, MatplotLib, SKLearn –

Tensorflow and Keras – TensorFlow is an open-sourced end-to-end platform, a library for multiple machine learning tasks, while Keras is a high-level neural network library that runs on top of TensorFlow. Both provide high-level APIs used for easily building and training models, but Keras is more user-friendly because it's built-in Python.

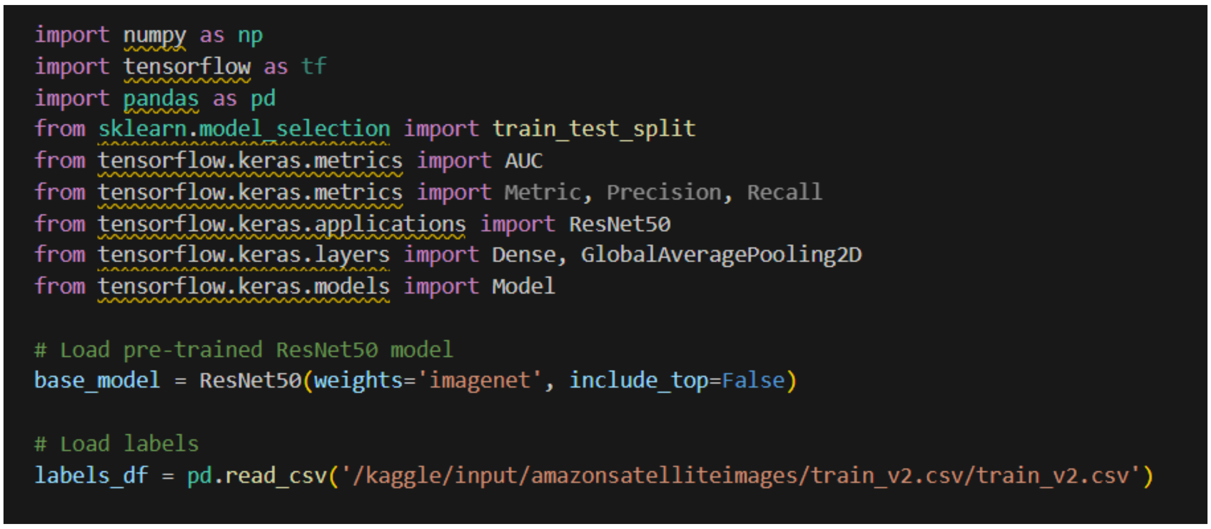
Code For the Project –

First we are importing the Important Libraries –

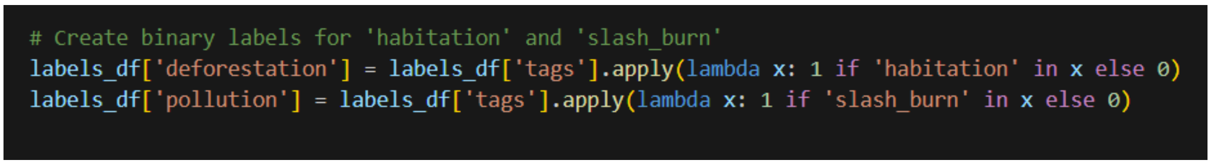
- Numpy, Pandas, Matplotlib, Sklearn, TensorFlow, Keras, Collection, OS, ResNet50, Matplotlib, Random



```python
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)


import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

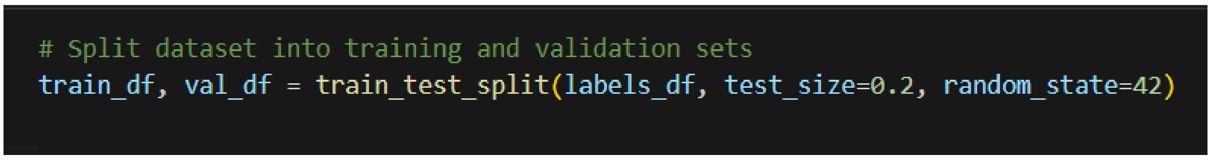


```python
import numpy as np
import tensorflow as tf
import pandas as pd
from sklearn.model_selection import train_test_split
from tensorflow.keras.metrics import AUC
from tensorflow.keras.metrics import Metric, Precision, Recall
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.models import Model

# Load pre-trained ResNet50 model
base_model = ResNet50(weights='imagenet', include_top=False)

# Load labels
labels_df = pd.read_csv('/kaggle/input/amazonsatelliteimages/train_v2.csv/train_v2.csv')
```

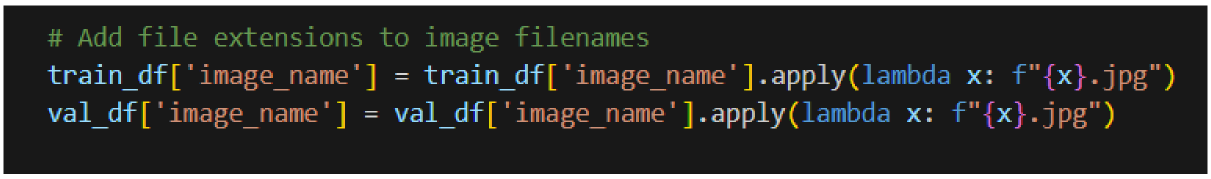


```python
# Create binary labels for 'habitation' and 'slash_burn'
labels_df['deforestation'] = labels_df['tags'].apply(lambda x: 1 if 'habitation' in x else 0)
labels_df['pollution'] = labels_df['tags'].apply(lambda x: 1 if 'slash_burn' in x else 0)
```

➢ Splitting a dataset in 80:20 ratio to train and test the model

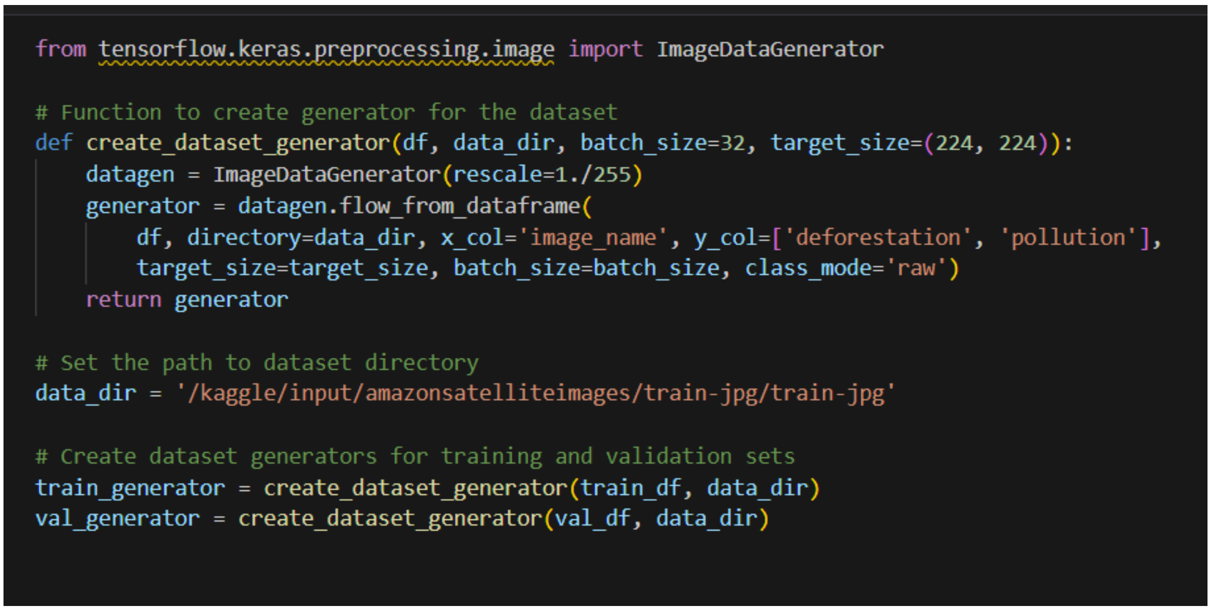


```python
# Split dataset into training and validation sets
train_df, val_df = train_test_split(labels_df, test_size=0.2, random_state=42)
```



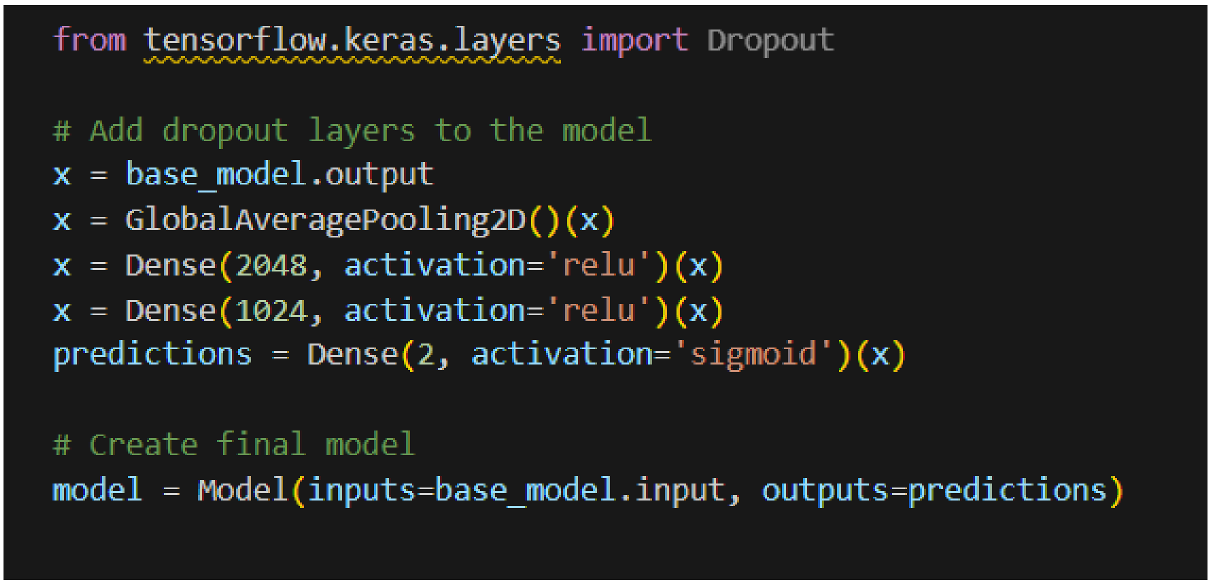
```python
# Add file extensions to image filenames
train_df['image_name'] = train_df['image_name'].apply(lambda x: f"{x}.jpg")
val_df['image_name'] = val_df['image_name'].apply(lambda x: f"{x}.jpg")
```

➢ Augmentation and preprocessing of images before training it :



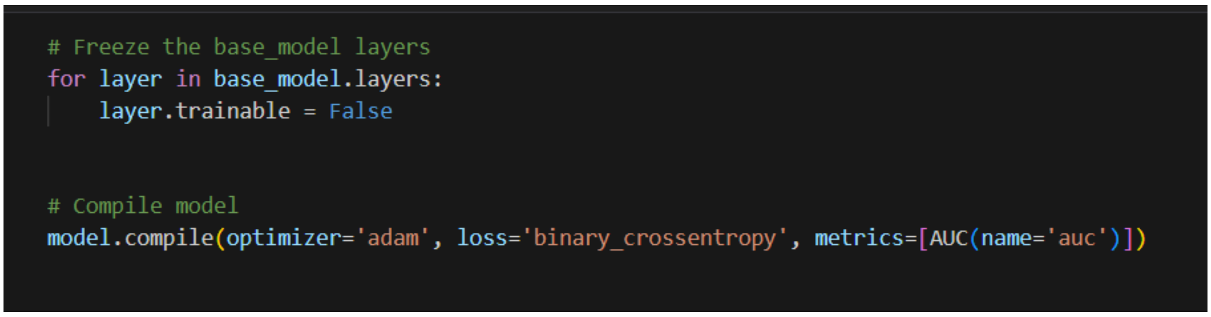
```python
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Function to create generator for the dataset
def create_dataset_generator(df, data_dir, batch_size=32, target_size=(224, 224)):
    datagen = ImageDataGenerator(rescale=1./255)
    generator = datagen.flow_from_dataframe(
        df, directory=data_dir, x_col='image_name', y_col=['deforestation', 'pollution'],
        target_size=target_size, batch_size=batch_size, class_mode='raw')
    return generator

# Set the path to dataset directory
data_dir = '/kaggle/input/amazonsatelliteimages/train-jpg/train-jpg'

# Create dataset generators for training and validation sets
train_generator = create_dataset_generator(train_df, data_dir)
val_generator = create_dataset_generator(val_df, data_dir)
```

➢ Now creating the Deep Learning Model with sigmoid as an activation function for an output –

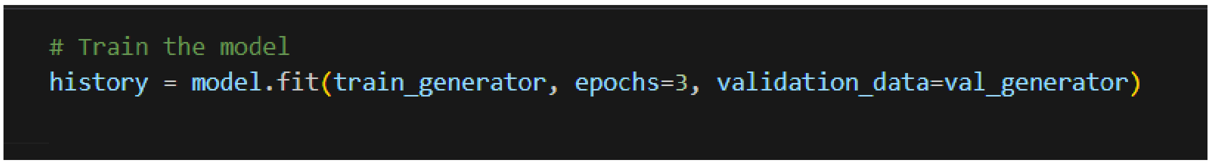


```python
from tensorflow.keras.layers import Dropout

# Add dropout layers to the model
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(2048, activation='relu')(x)
x = Dense(1024, activation='relu')(x)
predictions = Dense(2, activation='sigmoid')(x)

# Create final model
model = Model(inputs=base_model.input, outputs=predictions)
```

- To reduce the Loss, optimize the model and calculate the accuracy:
  If AUC < 0.5, model is behaving randomly

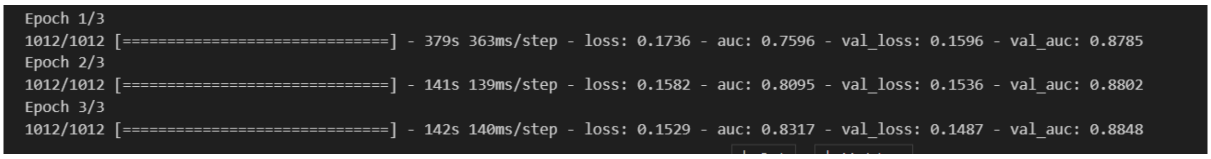


```python
# Freeze the base_model layers
for layer in base_model.layers:
    layer.trainable = False


# Compile model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=[AUC(name='auc')])
```



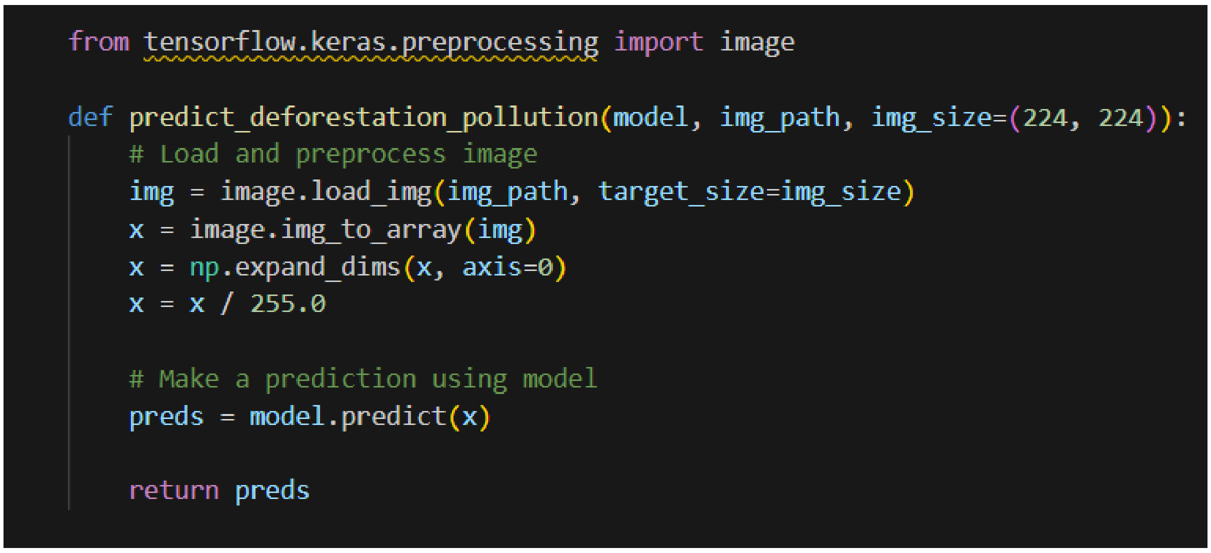
```python
# Train the model
history = model.fit(train_generator, epochs=3, validation_data=val_generator)
```

- Verifying test loss and test acuuracy: We used here only 3 epochs so that model doesn’t overfitt with the given test cases and hence we got the accuracy of 88.48% .



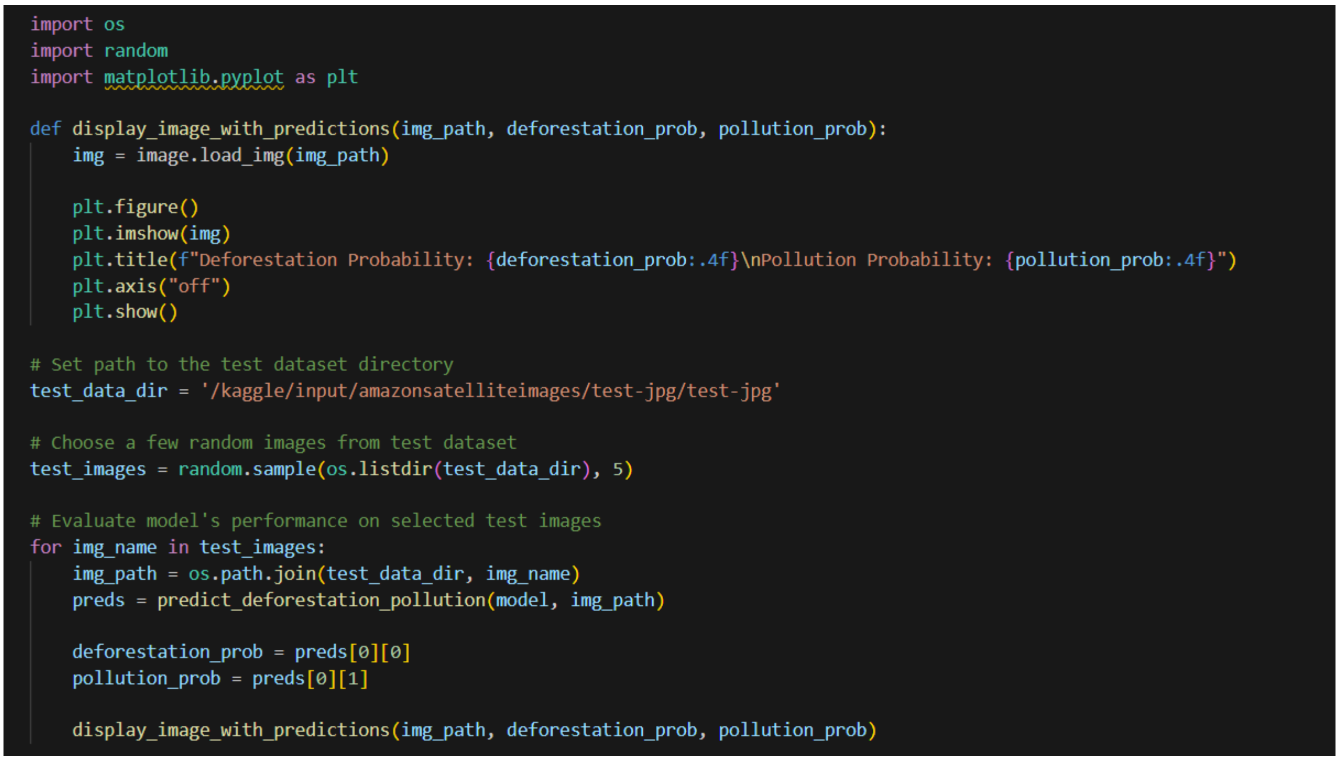
```
Epoch 1/3
1012/1012 [==============================] - 379s 363ms/step - loss: 0.1736 - auc: 0.7596 - val_loss: 0.1596 - val_auc: 0.8785
Epoch 2/3
1012/1012 [==============================] - 141s 139ms/step - loss: 0.1582 - auc: 0.8095 - val_loss: 0.1536 - val_auc: 0.8802
Epoch 3/3
1012/1012 [==============================] - 142s 140ms/step - loss: 0.1529 - auc: 0.8317 - val_loss: 0.1487 - val_auc: 0.8848
```

- Importing the images from the provided test and train dataset to test the model and to lowers its spatial resolution so that model computation takes much time to compute.



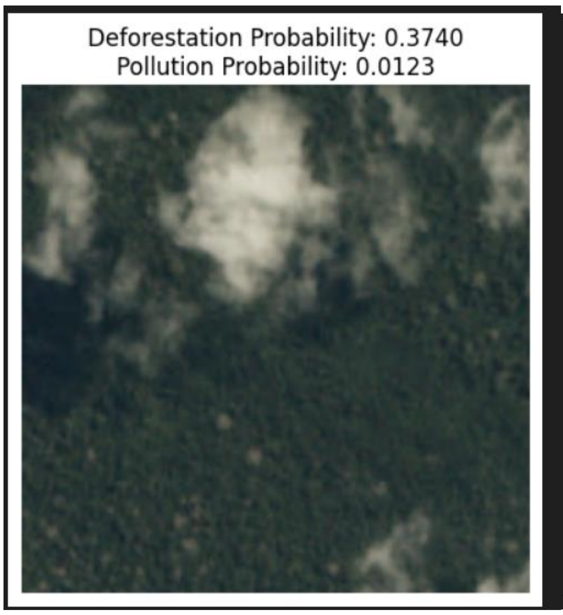
```python
from tensorflow.keras.preprocessing import image

def predict_deforestation_pollution(model, img_path, img_size=(224, 224)):
    # Load and preprocess image
    img = image.load_img(img_path, target_size=img_size)
    x = image.img_to_array(img)
    x = np.expand_dims(x, axis=0)
    x = x / 255.0

    # Make a prediction using model
    preds = model.predict(x)

    return preds
```

- Considering any 5 images in test dataset to generate the output:

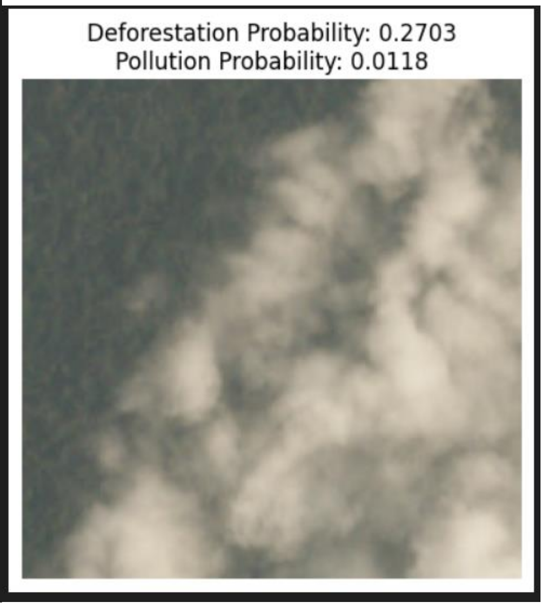


```python
import os
import random
import matplotlib.pyplot as plt

def display_image_with_predictions(img_path, deforestation_prob, pollution_prob):
    img = image.load_img(img_path)

    plt.figure()
    plt.imshow(img)
    plt.title(f"Deforestation Probability: {deforestation_prob:.4f}\nPollution Probability: {pollution_prob:.4f}")
    plt.axis("off")
    plt.show()

# Set path to the test dataset directory
test_data_dir = '/kaggle/input/amazonsatelliteimages/test-jpg/test-jpg'

# Choose a few random images from test dataset
test_images = random.sample(os.listdir(test_data_dir), 5)

# Evaluate model's performance on selected test images
for img_name in test_images:
    img_path = os.path.join(test_data_dir, img_name)
    preds = predict_deforestation_pollution(model, img_path)

    deforestation_prob = preds[0][0]
    pollution_prob = preds[0][1]

    display_image_with_predictions(img_path, deforestation_prob, pollution_prob)
```

- Output after training the model with the dataset and testing the model with test and train data

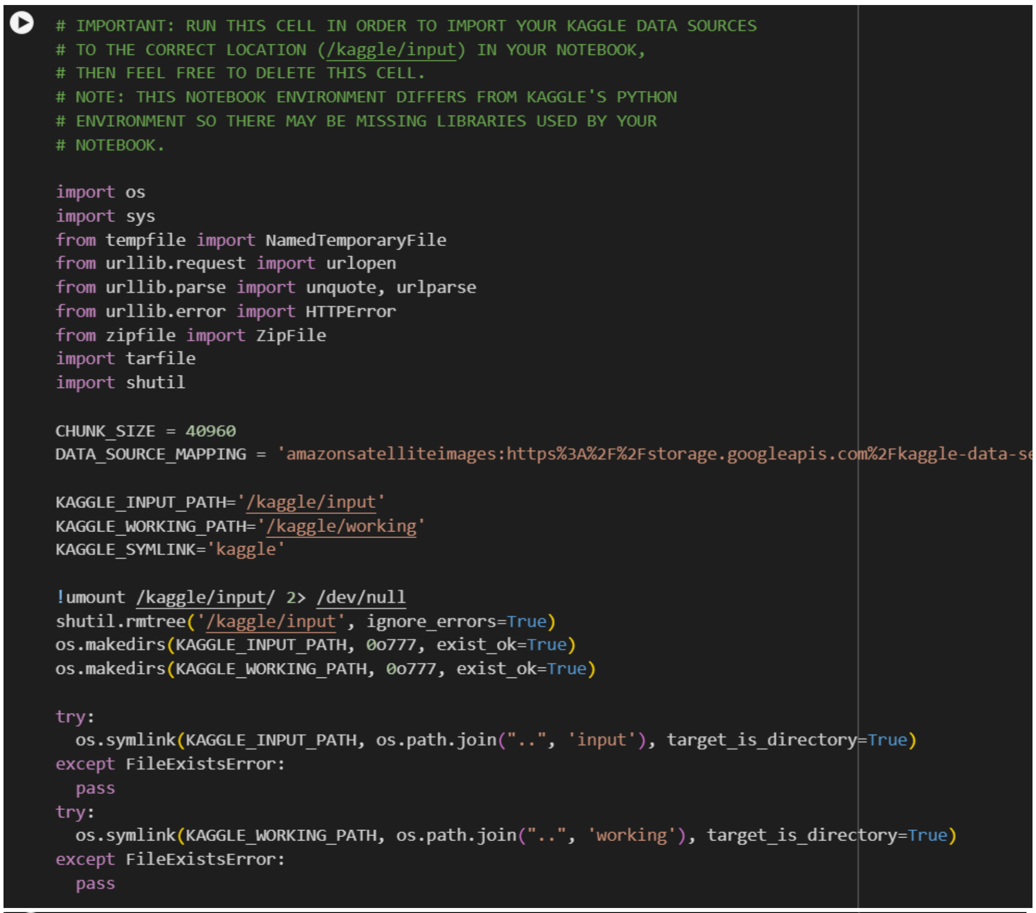


```
Deforestation Probability: 0.3740
Pollution Probability: 0.0123
```

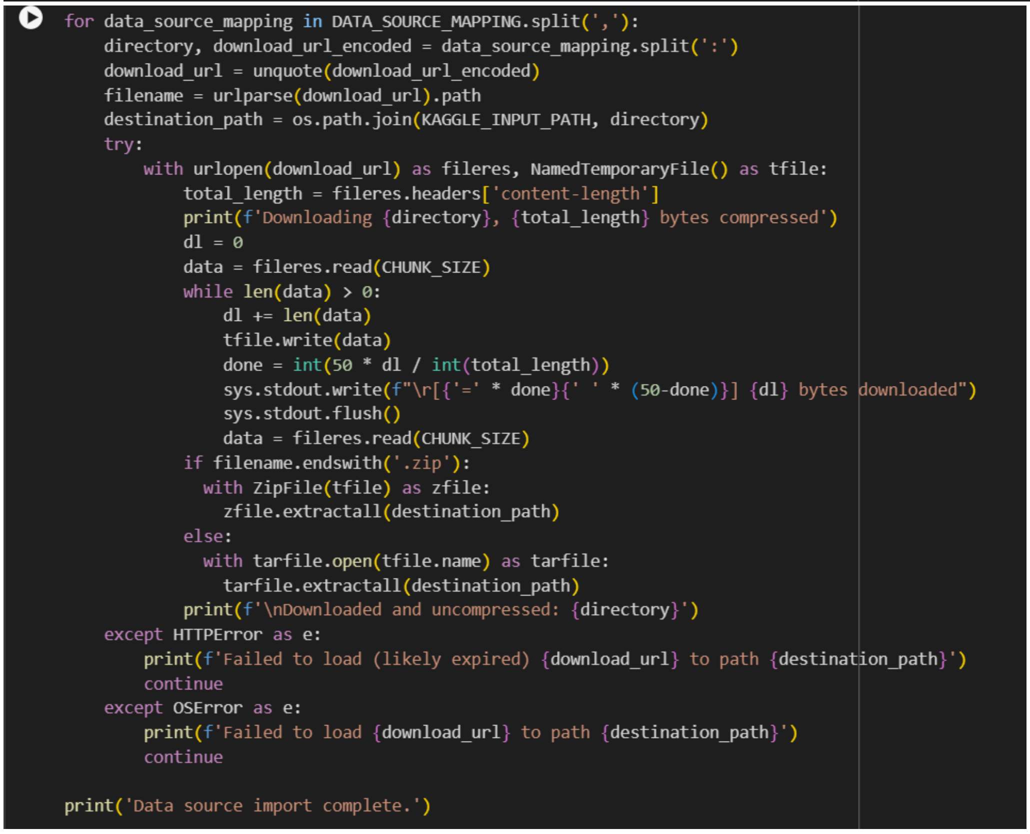


```
Deforestation Probability: 0.2703
Pollution Probability: 0.0118
```

Importing DL model from Kaggle to Google Collab to make the model work faster as the google collab has its own GPU power .



```python
# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES
# TO THE CORRECT LOCATION (/kaggle/input) IN YOUR NOTEBOOK,
# THEN FEEL FREE TO DELETE THIS CELL.
# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE'S PYTHON
# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR
# NOTEBOOK.

import os
import sys
from tempfile import NamedTemporaryFile
from urllib.request import urlopen
from urllib.parse import unquote, urlparse
from urllib.error import HTTPError
from zipfile import ZipFile
import tarfile
import shutil

CHUNK_SIZE = 40960
DATA_SOURCE_MAPPING = 'amazonsatelliteimages:https%3A%2F%2Fstorage.googleapis.com%2Fkaggle-data-s

KAGGLE_INPUT_PATH='/kaggle/input'
KAGGLE_WORKING_PATH='/kaggle/working'
KAGGLE_SYMLINK='kaggle'

!umount /kaggle/input/ 2> /dev/null
shutil.rmtree('/kaggle/input', ignore_errors=True)
os.makedirs(KAGGLE_INPUT_PATH, 0o777, exist_ok=True)
os.makedirs(KAGGLE_WORKING_PATH, 0o777, exist_ok=True)

try:
  os.symlink(KAGGLE_INPUT_PATH, os.path.join("..", 'input'), target_is_directory=True)
except FileExistsError:
  pass
try:
  os.symlink(KAGGLE_WORKING_PATH, os.path.join("..", 'working'), target_is_directory=True)
except FileExistsError:
  pass
```



```python
for data_source_mapping in DATA_SOURCE_MAPPING.split(','):
    directory, download_url_encoded = data_source_mapping.split(':')
    download_url = unquote(download_url_encoded)
    filename = urlparse(download_url).path
    destination_path = os.path.join(KAGGLE_INPUT_PATH, directory)
    try:
        with urlopen(download_url) as fileres, NamedTemporaryFile() as tfile:
            total_length = fileres.headers['content-length']
            print(f'Downloading {directory}, {total_length} bytes compressed')
            dl = 0
            data = fileres.read(CHUNK_SIZE)
            while len(data) > 0:
                dl += len(data)
                tfile.write(data)
                done = int(50 * dl / int(total_length))
                sys.stdout.write(f"\r[{'=' * done}{' ' * (50-done)}] {dl} bytes downloaded")
                sys.stdout.flush()
                data = fileres.read(CHUNK_SIZE)
            if filename.endswith('.zip'):
              with ZipFile(tfile) as zfile:
                zfile.extractall(destination_path)
            else:
              with tarfile.open(tfile.name) as tarfile:
                tarfile.extractall(destination_path)
            print(f'\nDownloaded and uncompressed: {directory}')
    except HTTPError as e:
        print(f'Failed to load (likely expired) {download_url} to path {destination_path}')
        continue
    except OSError as e:
        print(f'Failed to load {download_url} to path {destination_path}')
        continue

print('Data source import complete.')
```

Using Gradio to make the UI interface for an user to access this Deep Learning Model to predict the pollution through satellite images.

Gradio :

Gradio is a Python library that allows you to quickly create customizable UIs for your machine learning models. It's designed to make it easy for both developers and non-developers to interact with machine learning models, enabling tasks like model deployment, testing, and visualization without needing deep technical knowledge.

Here are some key features and aspects of Gradio:

Simple Interface: Gradio provides a simple and intuitive interface for creating UIs. You can define input and output components using Python functions and decorators.

Wide Range of Components: Gradio supports various input components like text boxes, sliders, dropdowns, and file uploads. Similarly, it supports output components like text, images, plots, and more.

Support for Multiple Frameworks: Gradio is framework-agnostic, meaning you can use it with popular machine learning frameworks like TensorFlow, PyTorch, and scikit-learn.

Live Updates: The UI updates in real-time as users interact with it, providing instant feedback.

Customizability: Gradio allows you to customize the appearance of UI components and style them according to your preferences.

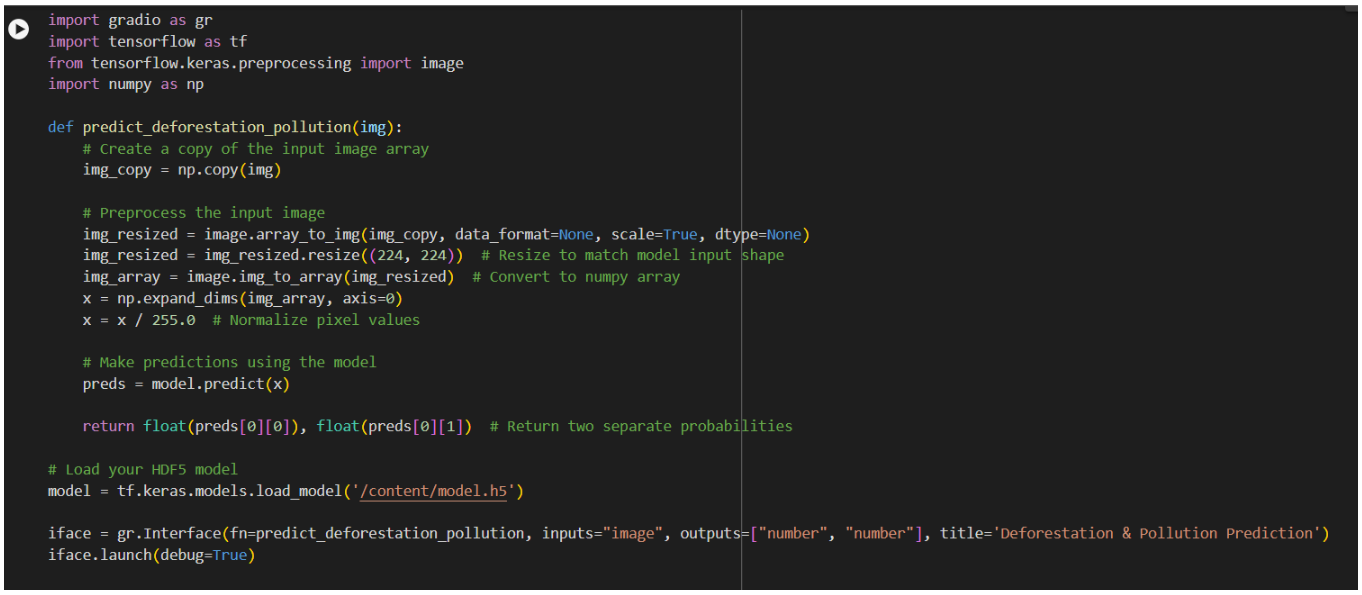
Deployment Options: You can deploy Gradio UIs locally on your machine, or you can share them online using Gradio's cloud hosting service. This makes it easy to share your models with others or integrate them into web applications.

Community Support: Gradio has an active community, with developers contributing new features, examples, and documentation regularly.

Integration with Model Zoo: Gradio integrates with the Gradio Model Zoo, a collection of pre-trained models across various domains, making it easy to use and deploy state-of-the-art models.

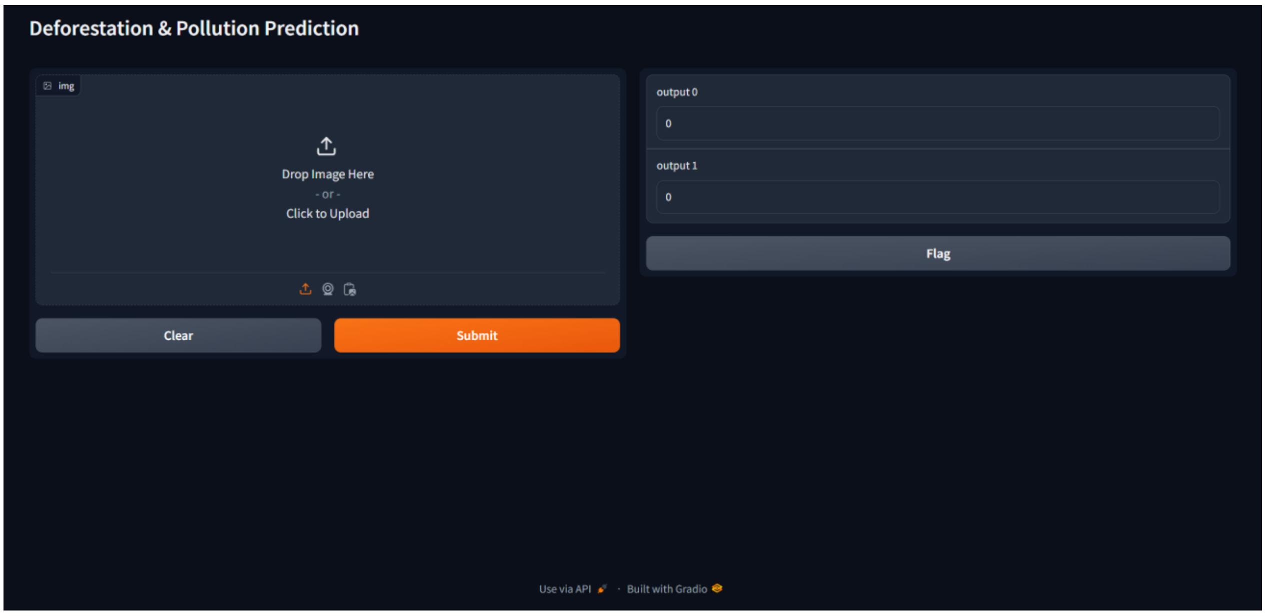
Overall, Gradio is a powerful tool for building interactive UIs for machine learning models, whether you're a seasoned developer or just getting started with machine learning. Its simplicity and versatility make it a popular choice for both prototyping and deploying ML applications.

- Integrating Model.h5 document which contains a DL model with a web interface to upload images to generate pollution and deforestation output based on user input.



```python
import gradio as gr
import tensorflow as tf
from tensorflow.keras.preprocessing import image
import numpy as np

def predict_deforestation_pollution(img):
    # Create a copy of the input image array
    img_copy = np.copy(img)

    # Preprocess the input image
    img_resized = image.array_to_img(img_copy, data_format=None, scale=True, dtype=None)
    img_resized = img_resized.resize((224, 224))  # Resize to match model input shape
    img_array = image.img_to_array(img_resized)  # Convert to numpy array
    x = np.expand_dims(img_array, axis=0)
    x = x / 255.0  # Normalize pixel values

    # Make predictions using the model
    preds = model.predict(x)

    return float(preds[0][0]), float(preds[0][1])  # Return two separate probabilities

# Load your HDF5 model
model = tf.keras.models.load_model('/content/model.h5')

iface = gr.Interface(fn=predict_deforestation_pollution, inputs="image", outputs=["number", "number"], title='Deforestation & Pollution Prediction')
iface.launch(debug=True)
```

- UI interface of Webpage:



output 0

```
0
```

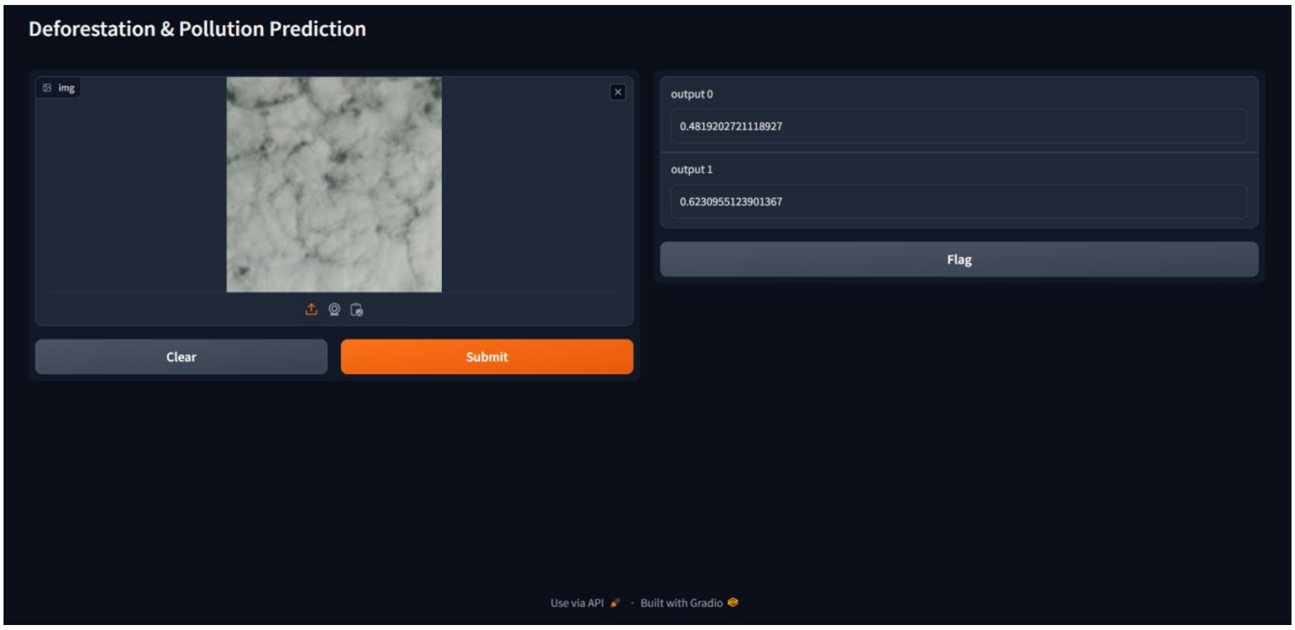
output 1

```
0
```

Output1 showcases: Deforestation level on a scale of 0-1

Output2 showcases: Pollution level on a scale of 0-1

- OUTPUT1: When a polluted image is given as an input



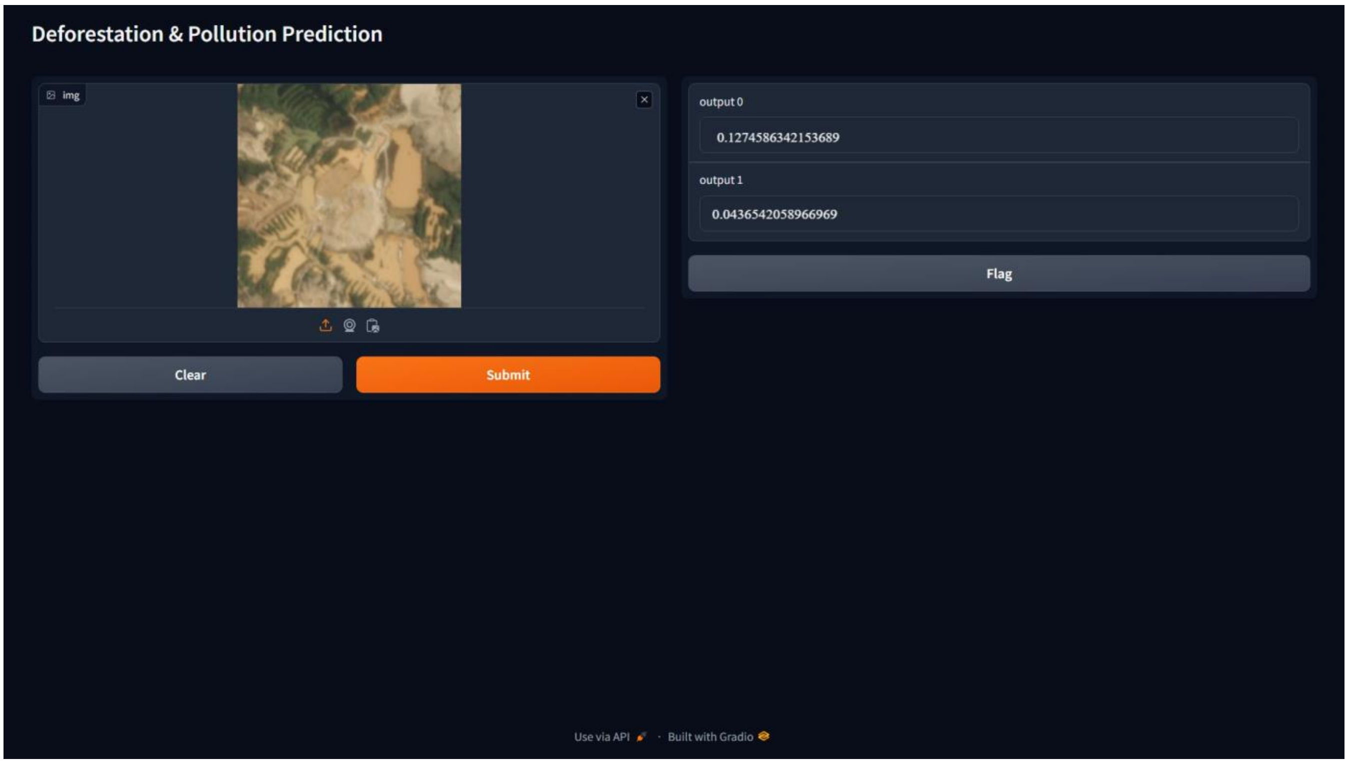
output 0

```
0.4819202721118927
```

output 1

```
0.6230955123901367
```

- OUTPUT2: When a green land surface is given as an input



Deforestation & Pollution Prediction

output 0

0.1274586342153689

output 1

0.0436542058966969

Conclusion –

We successfully completed the project with UI interface integrated with deep learning model which is being trained on more than 40,000 images of dataset which can be used to predict pollution and deforestation level in an area based on user input image.

This various deforestation and pollution levels can be used to regulate pollution and various environmental factors during numerous government projects such highway and industrialization.