National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Deepak Yadav
Student ID: x21219991

School of Computing
National College of Ireland

Supervisor: Mr. Hicham Rifai

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Deepak Yadav
Student ID:	x21219991
Programme:	Data Analytics
Year:	2023
Module:	MSc Research Project
Supervisor:	Mr. Hicham Rifai
Submission Due Date:	14/12/2023
Project Title:	Butterfly and Moth Species Detection and Classification Using Deep Learning
Word Count:	2006
Page Count:	10

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Deepak Yadav
Date:	30th January 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Deepak Yadav
x21219991

1 Introduction

This section serves as a manual to the software and hardware configuration employed in the development of the model presented in the thesis. It operates as a collection of directives for potential code reuse or rerun on varying systems. This project investigates the implementation of profound learning models for the prediction of butterfly species through an interactive web application powered by Streamlit. Streamlit, a Python library, enables the creation of user-friendly web applications with minimal code. The developed application permits users to upload butterfly images for real-time species classification using diverse pretrained models, such as EfficientNetB0, VGG19, ResNet50, and a ButterflyNet CNN model.

2 System Specifications

The system configuration provided below is utilized in the creation of the project. It is recommended that users consider using a configuration similar to the one outlined below to ensure optimal performance when running the provided code artifacts.

The code is executed on Google Colab, a cloud platform renowned for its capabilities in coding and running machine learning and deep learning models. Google Colab integrates TensorFlow and Keras, enhancing execution speed by leveraging GPU and TPU resources when necessary.

The table in 1 outlines the key specifications of the system used for the project.

Table 1: System Specifications

Processor	12th Gen Intel(R) Core(TM) i5-1240P 1.70 GHz
Installed RAM	16.0 GB (15.7 GB usable)
System type	64-bit operating system, x64-based processor

3 Software Requirements

This section provides an overview of the essential software components required to utilize and engage with the classification approach.

Python has been chosen as the implementation language for this project because of its accessibility, open-source nature, and extensive library support. This language specifically facilitates the implementation of advanced technologies such as TensorFlow, Keras, and

Streamlit, which play pivotal roles in the development and visualization of deep learning models.

To ensure smooth functionality on Google Colab and Streamlit, it is crucial to have the latest version of Google Chrome installed. This is imperative for achieving optimal performance when executing Colab notebooks and accessing web pages hosted by Streamlit.

In order to commence working with the models, a sample set of butterfly images is required for testing purposes.

Furthermore, the project temporarily stores the models in your local Google Drive. In order to accommodate this, it is necessary to possess a minimum of 2 GB of free space available in your Google Drive. This will guarantee a seamless experience while working with the developed models.

3.1 Google Colab

Google Colab is a cloud-based platform widely used for coding and running machine learning and deep learning models. It provides free access to GPU and TPU resources, significantly speeding up model training. To use Google Colab, follow these steps:

1. Open your browser and navigate to Google Colab (`https://colab.research.google.com/`).
2. Sign in with your Google account.
3. Upload the notebook file provided named **x21219991_Deepak_Yadav.ipynb**.

3.2 Libraries

This project leverages the following libraries for image classification. It's essential to install these libraries to ensure smooth execution.

The provided code snippet takes care of installing and importing all the required libraries into the user's Colab environment. Each import statement is accompanied by comments for better understanding.

Refer to Figure 1 for a visual representation.

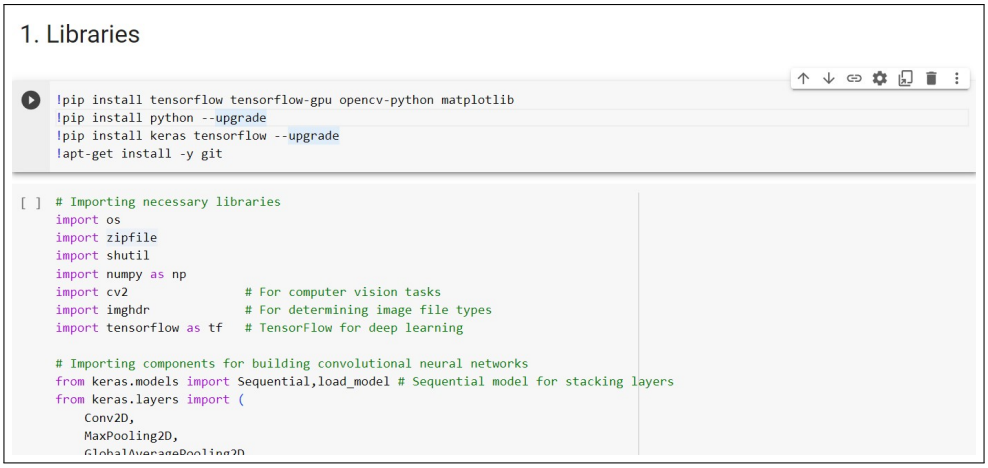


Figure 1: Visual Representation of Installed Libraries

4 Dataset Resources

For this research, the dataset comprises over 13,000 photos representing 100 distinct butterfly species, sourced from the Kaggle (2021) data repository. All images are uniformly sized at 224 x 224 x 3 pixels and saved in JPG format. The labels column provides the species label for each image file. A glimpse of sample images from the dataset is presented in Figure 2.

To ensure project robustness and independence from Kaggle's platform, the dataset was downloaded from Kaggle and subsequently uploaded to a public GitHub repository. This precautionary step guards against potential issues, such as file deletion by users or downtime on the Kaggle platform. The code accesses the dataset directly from GitHub, facilitating seamless integration into the Colab environment.

Refer to Figure 2 for a snapshot of the Kaggle Dataset page.

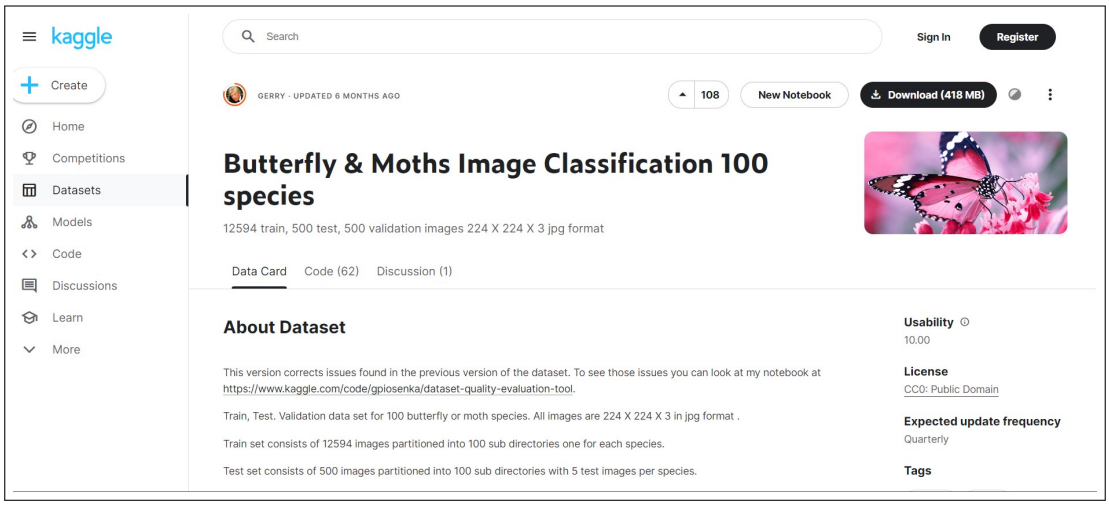


Figure 2: Snapshot of Kaggle Dataset Page

Refer to Figure 3 for a snapshot of the code for downloading the required dataset files.

Figure 3: Snapshot of Dataset GitHub Code

5 Utility Functions

In the course of this project, several key functions have been crafted to enhance efficiency and streamline various processes. Here's an overview of the essential functions utilized:

- `create_class_mapping(directory)`: Generates a mapping from class folder names to unique identifiers.
- `plot_images(tensorflow_dataset, class_mapping, num_batches=4)`: Visualizes images from a TensorFlow dataset using Matplotlib.
- `create_image_dataset(directory, batch_size=32, image_size=(224, 224))`: Creates a TensorFlow image dataset from a specified directory.
- `plot_loss(history)`: Plots training loss and validation loss over epochs using Plotly.
- `plot_accuracy(history)`: Plots training accuracy and validation accuracy over epochs using Plotly.
- `get_class_names(data_dir)`: Retrieves sorted class names from subfolder names within a directory.
- `evaluate_test(model, modelName, test, test_dir)`: Evaluates a model on a test dataset, printing various metrics and visualizations.

These functions play pivotal roles in tasks ranging from data preparation to model evaluation, contributing to the overall success of the project.

5.1 Additional Functions

- `filter_images(dir_list, image_exts)`: Filters images in specified directories based on allowed image extensions. Removes images with extensions not in the specified list. Prints information about removed images.
- `count_images_in_dir(directory, image_extensions)`: Counts the number of images in a given directory with specified image extensions. Returns the count.
- `generate_directory_stats(dir_list, image_extensions)`: Generates statistics for each directory in the list. Returns total folder count, total image count, folder names, subfolder counts, and image counts.
- `display_directory_stats(dir_list, image_extensions)`: Displays directory statistics using Plotly. Returns total folder count and total image count.

These additional functions play a crucial role in image preprocessing, data exploration, and statistical analysis within the project.

6 Exploratory Data Analysis

The provided dataset is pre-divided into training, testing, and validation sets. To conduct Exploratory Data Analysis (EDA), the project examined the image count and class balance, visualizing the distribution through a bar plot.

Refer to the Exploratory Data Analysis section in the notebook file for detailed insights. Additionally, Figure 4 offers a snapshot of the bar plot illustrating the image count in the training dataset.

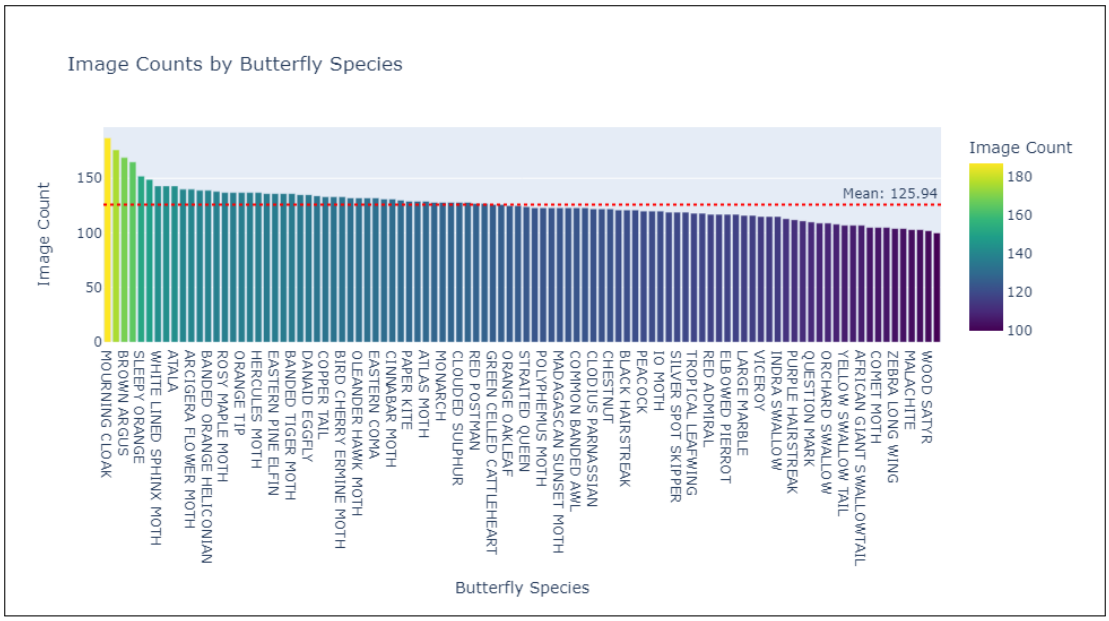


Figure 4: Snapshot of Image count in the train dataset

7 Deep Learning Models

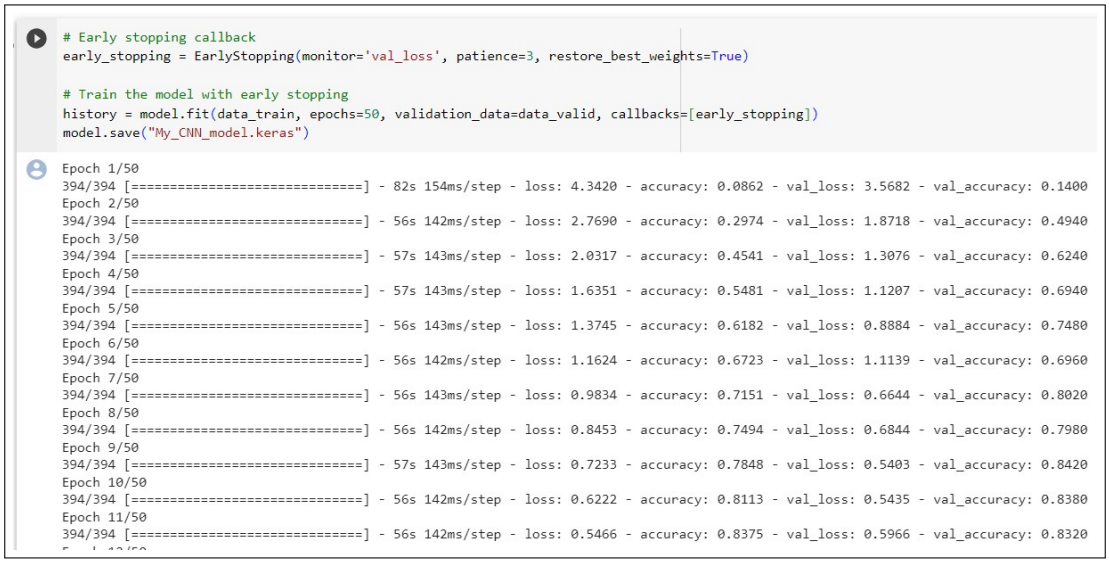
To comprehensively assess the researcher's model, this project has developed various alternative models for comparative analysis. The models share consistent parameters, including patience and epochs, ensuring a fair evaluation. Evaluation metrics are visualized using the dedicated functions: `plot_loss(history)` and `plot_accuracy(history)`.

7.1 ButterflyNet Model

The image classification Convolutional Neural Network (CNN) designed follows a sequential model structure incorporating key elements for effective feature extraction and regularization. The architecture initiates with basic feature extraction and progressively captures intricate patterns through additional convolutional layers. Batch normalization ensures training stability, max pooling reduces spatial dimensions, and dropout minimizes overfitting risks. The model is compiled using the Adam optimizer with a learning rate scheduler, complemented by dense layers and dropout for robustness. The project's summary provides a comprehensive overview of the network's architecture and parameters.

In the "Build Deep Learning Model" section of the notebook, the code for the CNN model, including the utilization of the early stopping callback, can be found. The code

snippet illustrates the implementation of early stopping, configured to monitor validation loss with a patience of 3 epochs. The model undergoes 50 epochs, and the resulting training history is stored. The trained model is saved as "ButterflyNet.keras" (see Figure 5 for a snapshot of the CNN model training).



```
# Early stopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

# Train the model with early stopping
history = model.fit(data_train, epochs=50, validation_data=data_valid, callbacks=[early_stopping])
model.save("My_CNN_model.keras")
```

```
Epoch 1/50
394/394 [==============================] - 82s 154ms/step - loss: 4.3420 - accuracy: 0.0862 - val_loss: 3.5682 - val_accuracy: 0.1400
Epoch 2/50
394/394 [==============================] - 56s 142ms/step - loss: 2.7690 - accuracy: 0.2974 - val_loss: 1.8718 - val_accuracy: 0.4940
Epoch 3/50
394/394 [==============================] - 57s 143ms/step - loss: 2.0317 - accuracy: 0.4541 - val_loss: 1.3076 - val_accuracy: 0.6240
Epoch 4/50
394/394 [==============================] - 57s 143ms/step - loss: 1.6351 - accuracy: 0.5481 - val_loss: 1.1207 - val_accuracy: 0.6940
Epoch 5/50
394/394 [==============================] - 56s 143ms/step - loss: 1.3745 - accuracy: 0.6182 - val_loss: 0.8884 - val_accuracy: 0.7480
Epoch 6/50
394/394 [==============================] - 56s 142ms/step - loss: 1.1624 - accuracy: 0.6723 - val_loss: 1.1139 - val_accuracy: 0.6960
Epoch 7/50
394/394 [==============================] - 56s 143ms/step - loss: 0.9834 - accuracy: 0.7151 - val_loss: 0.6644 - val_accuracy: 0.8020
Epoch 8/50
394/394 [==============================] - 56s 142ms/step - loss: 0.8453 - accuracy: 0.7494 - val_loss: 0.6844 - val_accuracy: 0.7980
Epoch 9/50
394/394 [==============================] - 57s 143ms/step - loss: 0.7233 - accuracy: 0.7848 - val_loss: 0.5403 - val_accuracy: 0.8420
Epoch 10/50
394/394 [==============================] - 56s 142ms/step - loss: 0.6222 - accuracy: 0.8113 - val_loss: 0.5435 - val_accuracy: 0.8380
Epoch 11/50
394/394 [==============================] - 56s 142ms/step - loss: 0.5466 - accuracy: 0.8375 - val_loss: 0.5966 - val_accuracy: 0.8320
```

Figure 5: Snapshot of ButterflyNet Model Training

7.2 EfficientNetB0 Model

To harness transfer learning, the project integrates the pre-trained EfficientNetB0 model for image classification. The model's layers are frozen to retain pre-trained weights. A custom sequential model is then created, extending with flattened layers, dense layers, batch normalization, dropout, and a softmax output for 100 classes. The model is compiled using the Adam optimizer with a learning rate scheduler. The summary provides insights into architecture and parameters. EfficientNetB0 enhances feature extraction capabilities, valuable for image classification. The model is trained with an early stopping callback, and the training history is saved as "custom_model_EfficientNetB0.keras".

7.3 VGG19 Model

The project incorporates the VGG19 pre-trained model for image classification. All layers of the VGG19 model are frozen to preserve pre-trained weights. A custom sequential model is constructed, adding flattened layers, dense layers, batch normalization, dropout, and a softmax output for 100 classes. The model is compiled using the Adam optimizer. The summary provides an overview of architecture and parameters. VGG19 contributes to robust feature extraction. The model is trained with an early stopping callback, and the training history is saved as "vgg_custom_model.keras".

7.4 ResNet50 Model

The ResNet50 pre-trained model is integrated into the project for image classification. All layers of the ResNet50 model are frozen to maintain pre-trained weights. A custom sequential model is crafted, incorporating flattened layers, dense layers, batch nor-

malization, dropout, and a softmax output layer for 100 classes. The model is compiled using the Adam optimizer, and the summary provides insights into its architecture and parameters. ResNet50 enhances feature extraction capabilities. The model undergoes training with an early stopping callback, and the training history is saved as "resnet_custom_model.keras".

8 Model Evaluation

The table 2 presents the evaluation metrics for four distinct deep learning models applied to a test dataset. Notably, the **ResNet50** model stands out with a significantly lower test loss of 0.2285 and an impressive accuracy of 95.00%. Meanwhile, the **ButterflyNet Model** achieved a test loss of 0.5081 with an accuracy of 85.40%. The **VGG19 model** demonstrated competitive results with a test loss of 0.2771 and an accuracy of 92.80%, and similarly, the **EfficientNetB0** model recorded a test loss of 0.2349 and an accuracy of 93.60%. This comparison underscores the effectiveness of ResNet50 in achieving superior accuracy, while the other models exhibit commendable performance in image classification tasks.

Table 2: Test Results Summary for All Models

Model	**Test Loss**	**Test Accuracy**
ResNet50 Model	0.2285	95.00%
EfficientNetB0 Model	0.2349	93.60%
VGG19 Model	0.2771	92.80%
ButterflyNet Model	0.5081	85.40%

9 Streamlit

Streamlit is a robust Python library that streamlines the development of interactive web applications. With a concise codebase, users can effortlessly convert data scripts into shareable web apps. Streamlit's user-friendly design empowers developers, including those with limited web development experience, to seamlessly create and deploy applications for data visualization, machine learning, and beyond.

In this project, Streamlit plays a crucial role in tunneling the webpage to an external site, facilitating users to upload their butterfly images for species prediction. The `app.py` file under the `Streamlit` section encapsulates the code for the web page.

9.1 Obtaining External IP

- Execute the cell located at the end of the presented notebook after successfully running all preceding cells.

  Consult Figure 6 for a visual representation of the code.

- Copy the URL provided as "External URL" from the cell's output by right-clicking on the link and selecting "Copy link address."

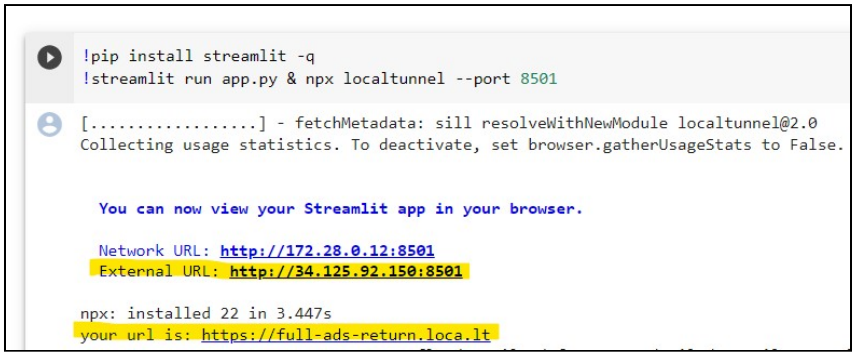


Figure 6: Snapshot of Obtaining External IP

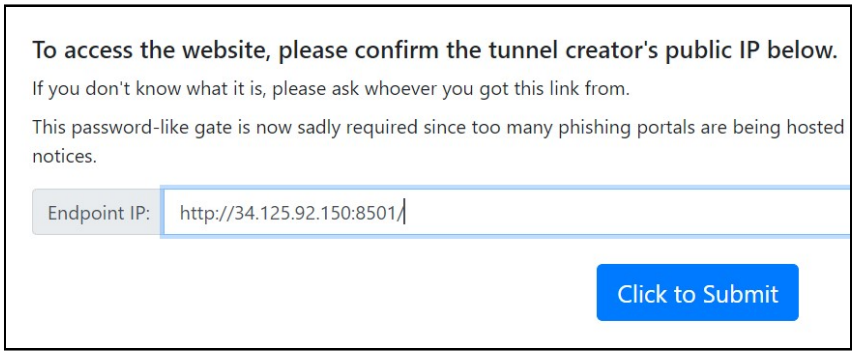


Figure 7: Snapshot of Endpoint IP

- Click on the link labeled "your url is:" and paste the copied URL as depicted in Figure 7.
- Remove the "http://" and the number after ":" as illustrated in Figure 8.

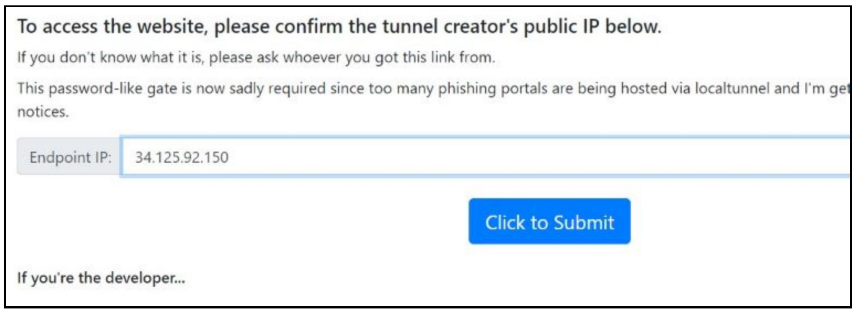


Figure 8: Snapshot of Final Endpoint IP

- Click on the "Click to Submit" button.

9.2 Homepage

Users are directed to a page designed by `app.py` and can view the webpage, as seen in Figure 9. Click the "Let's Go" button.

Figure 9: Snapshot of Landing WebPage

Users are guided to another page where they can upload a test image using the "Browse files" button, as depicted in Figure 10. The "Exit" button returns to the homepage.

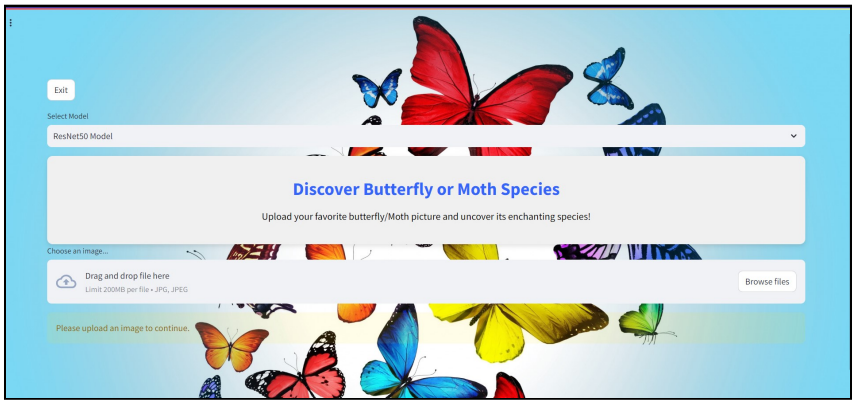


Figure 10: Snapshot of Uploading Test Image Page

Upon uploading an image, the page processes and predicts the species, displaying the output at the bottom. The species name and model confidence percentage are presented, along with the uploaded image, as shown in Figure 11.

9.3 Selection of Models

Users can select different models for the same image by using the dropdown menu available at the top of the page, below the "Exit" button.

Users can click and choose the model, as represented in Figure 12.

For a seamless experience, close the tab or stop the Colab cell to exit the webpage.

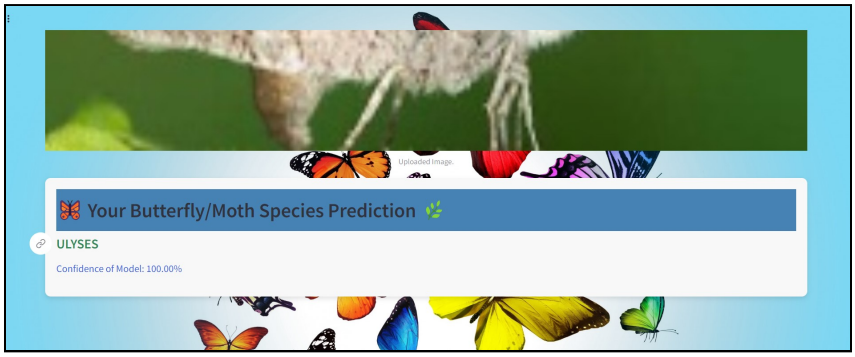


Figure 11: Snapshot of Species Prediction

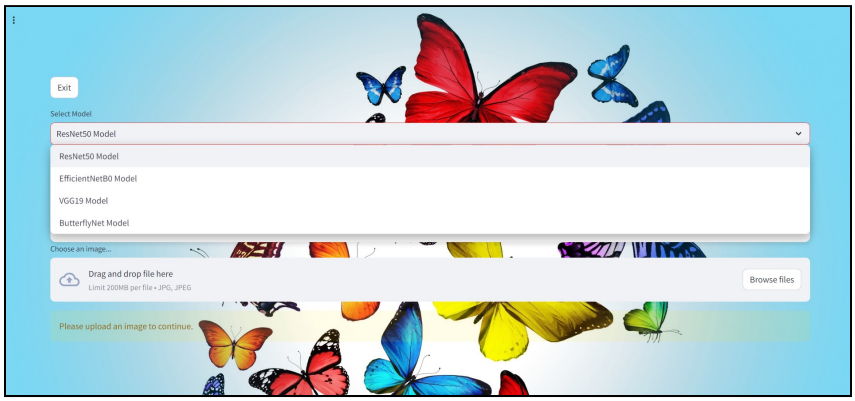


Figure 12: Snapshot of Model selection

References

Kaggle (2021), 'Butterfly & moths image classification 100 species', `https://www.kaggle.com/datasets/gpiosenka/butterfly-images40-species`.