# Live Capuchin Detection App: Technical Readme


## Author: Sai Rakshith Potluri (GT MSA 2023)
## Github Repo: https://github.com/raks097/live_capuchin_detection/


This application leverages cutting-edge machine learning techniques to accurately detect capuchin monkeys in various environments in real-time. Our solution is particularly valuable for wildlife monitoring, ecological research, and preserving biodiversity.

This readme is divided into three distinct sections, each detailing a critical component of the app's development and deployment:

- **Roboflow Dataset Storage and Augmentation:** This section covers the initial phase of the project, focusing on how to gather, store, and augment image data of capuchin monkeys. Using Roboflow, we explain the process of creating a robust dataset that serves as the foundation for our machine learning model.

- **YoloNAS-S Model Training using Super Gradients:** The second section dives into the specifics of training the YoloNAS-S model. We utilize Super Gradients and the prepared dataset from Roboflow to develop a model capable of recognizing capuchin monkeys with high accuracy. This part includes details on setting up the training environment, configuring the model, and generating the necessary .pth files.

- **Python-based App for Live Detection:** The final section outlines how to integrate the trained model into a Python-based application. This app uses the .pth files to detect capuchin monkeys in real-time. We provide instructions on setting up the app, loading the model, and deploying the solution for practical use.

## Roboflow Dataset Storage and Augmentation:

Roboflow is an advanced tool designed to streamline and optimize the process of preparing datasets for machine learning, particularly in the field of computer vision. It simplifies tasks such as image annotation, data augmentation, and dataset management, making it an invaluable resource for projects that involve image recognition, such as our Live Capuchin Detection App. By leveraging Roboflow's capabilities, we can efficiently prepare a robust dataset of capuchin monkey images, which is crucial for training an accurate and reliable detection model.

Steps for Using Roboflow in the Capuchin Detection App

1. Creating a Roboflow Account and Project:
    a. Sign up for a free account on Roboflow.

b. Create a *New Project* in the Roboflow dashboard, specifically tailored for object detection. Enter a Project Name as well.

2. Uploading and Annotating Images:
    a. Upload your collection of capuchin monkey images and the corresponding csv annotations to the project.
    b. If the images are not already annotated, use Roboflow Annotate to label them. Proper annotation is critical for effective model training.

3. Generating a Dataset Version:
    a. Navigate to the "Generate" section in your project to start creating a dataset version.
    b. This is where you apply preprocessing steps (like normalization, resizing) and choose augmentation techniques (like flipping, rotation, or brightness adjustments) to enhance the dataset. These augmentations help the model learn to recognize capuchins in various conditions and angles.

4. Exporting the Augmented Dataset:
    a. After selecting and configuring your augmentations, click "Version" to view the augmented dataset version.
    b. This dataset can now be used to train the detection model in Roboflow, or exported (click ‘Export Dataset’) for use in a custom training pipeline.

By following these steps, you can leverage Roboflow's powerful features to create a highly effective dataset for capuchin monkey detection, laying a solid foundation for the subsequent model training and application development phases of the project.

## YoloNAS-S Model Training using Super Gradients:

YoloNAS-S is a state-of-the-art neural network architecture designed for object detection tasks. It stands out due to its balance between accuracy and efficiency, making it ideal for real-time applications like animal detection. Super Gradients is a library that provides an optimized training environment, facilitating efficient training of deep learning models like YoloNAS-S.

Workflow Overview with Roboflow Integration

1. Preparing the Training Environment:
    a. The first step is to set up the training environment, which involves configuring the necessary libraries and dependencies. Super Gradients makes this process smoother by providing an optimized framework for training.

2. Importing the Dataset from Roboflow:

    a. The augmented dataset created in Roboflow is imported into the training environment. This integration ensures that the model trains on high-quality, diverse data, improving its ability to accurately detect capuchins in various scenarios.

3. Configuring YoloNAS-S Model Parameters:
    a. Before training begins, it's crucial to configure the YoloNAS-S model parameters. This includes setting the architecture specifics, learning rates, and other hyperparameters that influence the training process.

4. Training the Model:
    a. With the dataset and model parameters set, the training process commences. During this phase, the model learns to identify and localize capuchin monkeys from the input images.
    b. The training script typically includes several epochs of training, where the model iteratively improves its detection capabilities.

5. Monitoring and Optimizing Performance:
    a. Throughout the training, performance metrics such as loss and accuracy are monitored. Adjustments can be made to the training parameters if necessary, ensuring optimal model performance.

6. Generating and Saving .pth Files:
    a. Once the model has been trained to satisfaction, the final step is to save the trained model weights as .pth files. These files encapsulate the learned weights of the neural network.
    b. The .pth files are crucial for deploying the model in the Python-based app for live detection.

## Python Application for Live Capuchin Detection

This section details the development of a Python-based application for real-time capuchin monkey detection. The app leverages the pre-trained YoloNAS-S model, utilizing the .pth files generated in the previous step. The application is designed to run on CPU and does not require internet access, making it suitable for various environments.

## Setting Up the Environment

1. Installation of Necessary Packages:
    a. The application requires several Python libraries. It's recommended to use a package manager like Conda to create a virtual environment and manage dependencies.

b. Key libraries include torch, super-gradients for leveraging PyTorch functionalities, opencv-python for image processing, and others specified in a requirements.txt file.

2. Using Conda and requirements.txt:
    a. Create a new Conda environment to ensure package versions are compatible and isolated from the global Python installation.
    b. Use the requirements.txt file to install all necessary libraries with a single command: *conda install --file requirements.txt*.

## Detailed Workflow of the Python Application

1. Initial Setup:
    a. The script begins by importing necessary libraries such as torch, cv2 (OpenCV), and various utility functions. This setup ensures that all the required functionalities for image processing and model inference are available.
2. Loading the Pre-trained Model:
    a. The application loads the YoloNAS-S model with pre-trained weights from the .pth files. This step is crucial for enabling the model to detect capuchins based on the learning it has already acquired.
    b. Download weights from (https://drive.google.com/drive/folders/1--3LoLeV2sVSZqSMAlp13oNCv92SO0Co?usp=drive_link) and store it in the weights folder
3. Configuration for CPU Usage:
    a. The script configures the model to use the CPU for inference. This decision makes the app widely accessible, as it doesn't require a GPU for running. It's particularly beneficial for field deployment where high-end GPUs might not be available.
4. Setting Up Video Capture:
    a. The app uses OpenCV to capture live video feed. This can be from a connected camera or a video file. It's set up to process each frame of the video for real-time detection.
5. Frame Processing:

    a. Each frame from the video is processed to detect capuchins. The process involves resizing the frame, converting it into a tensor, and passing it through the model.
    b. The model outputs detection results, which include the coordinates of the bounding boxes and the confidence scores.
6. Displaying the Results:
    a. The application then draws bounding boxes around detected capuchins in each frame and displays these frames in real-time. This feature provides immediate visual feedback on the detections.
7. Handling Frame Storage with process_images:
    a. Currently, the process_images function is set up to store the frames. However, this function has the potential for expansion.
    b. In future versions, it could include additional cognitive tasks, like analyzing detected animals or implementing more targeted post-processing techniques.
8. Offline Functionality:
    a. The application operates independently of an internet connection. It relies solely on the local computational resources and the pre-trained model weights.
    b. This feature is particularly advantageous for use in remote or wilderness areas where internet access is limited or non-existent.

## Three Modes of Making Detections in the Live Capuchin Detection App

The Python-based application for live capuchin detection is designed to offer flexibility in how it processes and detects capuchins. This flexibility is realized through three distinct operational modes, each catering to different use cases and data sources. These modes are:

1. **Webcam Mode (process_webcam):**
    a. This mode is designed for real-time detection using a webcam.
    b. The application captures video feed from the webcam and processes each frame sequentially.
    c. For every frame, it applies the model to detect capuchins and displays the processed frame in a window.
    d. The user can terminate the process by pressing the 'q' key, which closes the video stream and the display window.

2. **Images Folder Mode (process_images_folder):**
    a. In this mode, the application processes a batch of images stored in a specified folder.
    b. It iterates over each image file (JPG format) in the folder, applying the detection model to each.
    c. This mode is particularly useful for analyzing a collection of still images, like camera trap photos in wildlife research.

3. **Videos Folder Mode (process_videos_folder):**
    a. Similar to the Images Folder mode, this mode handles multiple video files stored in a designated folder.
    b. The app processes each video file (MP4 format) individually, going through each frame to detect capuchins.
    c. It's useful for analyzing pre-recorded video footage, such as wildlife documentaries or archived video data.

## Deployment and Usage

The app is initiated through a simple Python command *python app.py*.
Ensure that all prerequisites are met, and the .pth files are correctly placed in the designated directory.