**Slide 1: Title Slide**

**Title:** Detecting and Blurring Picture Objects Using YOLO Object Classification  
**Subtitle:** An AI-Powered Solution for Privacy Protection  
**Prepared By:** Silas Phillips, Keegan Nohavec, and Christoph Guenther

**Slide 2: Introduction**

**Content:**

* **Problem Statement:** Personal photos, art, and other picture objects in images and videos can raise privacy concerns.
* **Solution:** Use an Artificial Intelligence (AI) model to automatically detect these objects and obfuscate them .
* **Technology Stack:** Roboflow, YOLO, Streamlit, Python.

**Slide 3: Project Goals**

**Content:**

* **Primary Goal:** Develop an AI model to detect and blur picture objects in images and videos.
* **Sub-goals:**
  + High-accuracy detection of picture objects.
  + Efficient blurring that maintains image quality.
  + Fast processing of images/videos without sacrificing accuracy.
  + User interface for uploading and processing images/videos.

**Slide 4: Data Collection and Preparation**

**Content:**

* **Data Sources:** Images of various indoor environments (Kaggle – need source link).
* **Image Preparation:**
  + Annotation – Identify, draw bounding boxes, and label picture objects in images.
  + Creation of training, testing, and validation datasets.
  + Augmentation – Crop, rotate, vary brightness, and blur of original training images to increase number of images and variety of training dataset.
  + Tool: Roboflow
* **Dataset Size:** Total of 3415 images divided into
  + Train: 2390 images (7170 including augmented images)
  + Validation: 602 images
  + Test: 423 images

**Slide 5: Model Selection – YOLO (You Only Look Once)**

**Content:**

* **Why YOLO?**
  + Fast (compared to competitor models such as R-CNN based models). Need citation.
  + Comparable or better accuracy to slower models (https://jonathan-hui.medium.com/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn-ssd-and-yolo-5425656ae359)..
  + Different models with increasing complexity.
* **Model Architecture (should go into README but not into presentation):**
  + Overview of YOLO architecture.

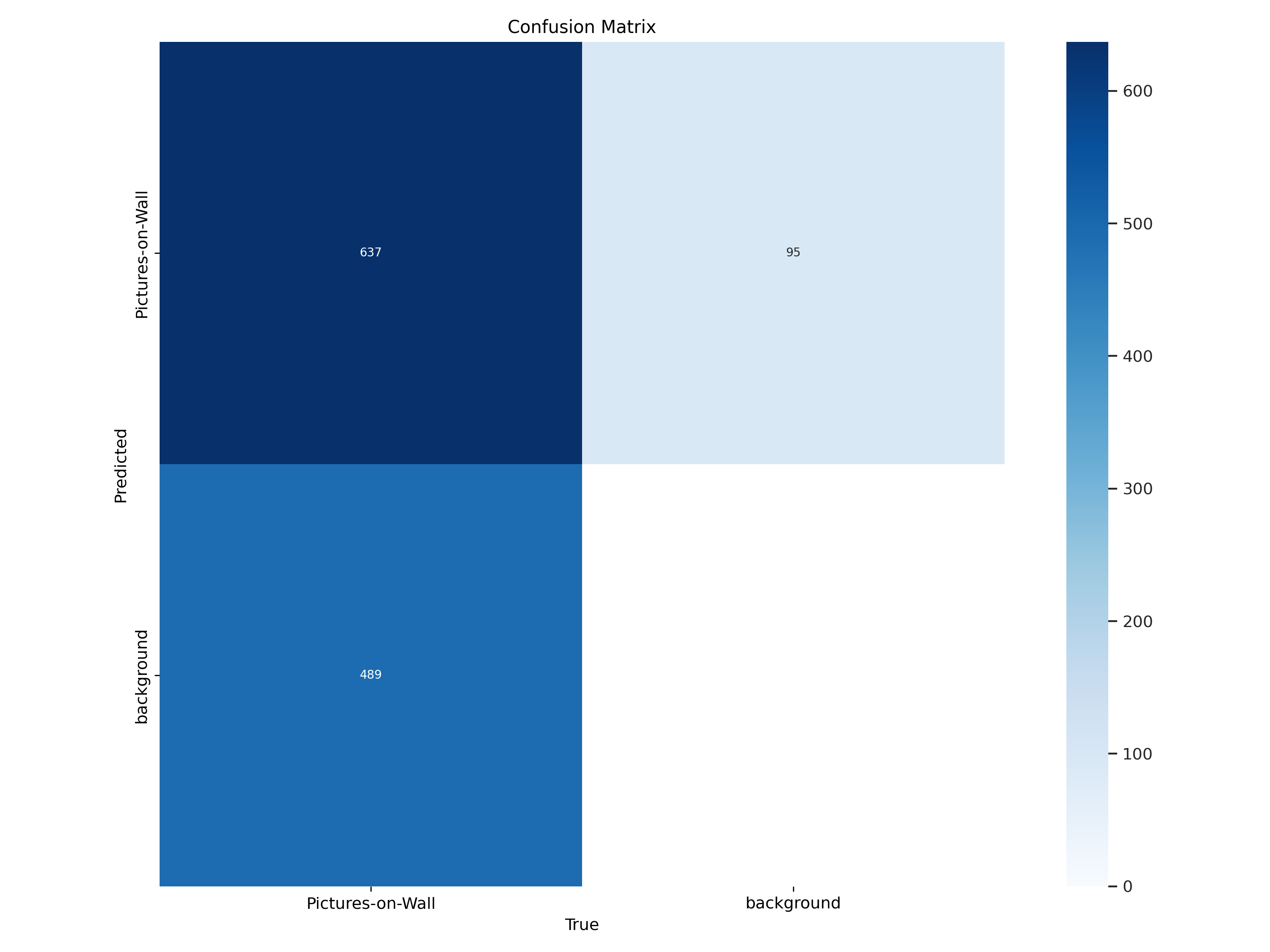
**Slide 6: Training the Model**

**Content:**

* **Setup:**
  + 1 class (Pictures-on-Wall)
  + Google Colab.
  + Use GPU
  + Use multiple processors
* **Process:**
  + Load preprocessed datasets from Roboflow.
  + Train model.
  + Evaluate performance on
    - Precision
    - Recall
    - Mean average precision
    - Confusion matrix
  + Repeat varying these parameters
    - YOLO version
    - YOLO model
    - Number of epochs
    - Batch size
    - Patience
    - Intersection over Union (IoU)

**Slide 7: Results**  
**Content:**

* **Model Chosen**
  + YOLOv10m: 369 layers, 16,451,542 parameters
* **Hyperparameters**
  + Epochs = 600, Batch Size = -1, Patience = 100, IoU = 0.5
* **Performance:**
  + Precision = 0.792, Recall = 0.552, mAP50 = 0.637, mAP50-95 = 0.425



**Slide 8: Implementing the Blurring Technique**

**Title:** Implementing the Blurring Technique  
**Content:**

* **Blurring Algorithm:**
  + Gaussian blur applied to detected regions.
  + Ensuring blurring preserves the aesthetics of the image.
* **Integration with YOLOv8:**
  + Post-processing step after detection.
  + Efficient and seamless application of the blur filter.

**Slide 8: Building the User Interface with Streamlit**

**Title:** Building the User Interface with Streamlit  
**Content:**

* **Why Streamlit?**
  + Rapid development of interactive web applications.
  + Easy integration with Python.
* **Features:**
  + Upload images and videos.
  + Display original and processed images/videos.
  + Option to download processed files.
* **UI Design:**
  + Simple and intuitive layout for user interaction.
  + Real-time feedback on processing status.

**Slide 9: Deployment and Scalability**

**Title:** Deployment and Scalability  
**Content:**

* **Deployment Options:**
  + Hosting on cloud platforms (e.g., AWS, GCP).
  + Containerization using Docker for consistent deployment.
* **Scalability Considerations:**
  + Handling large volumes of images and videos.
  + Ensuring real-time processing capabilities.
* **Future Enhancements:**
  + Extending detection to other privacy-sensitive objects.
  + Continuous model improvement with more data.

**Slide 10: Business Impact and Conclusion**

**Title:** Business Impact and Conclusion  
**Content:**

* **Privacy Protection:**
  + Enhancing privacy in shared and public spaces.
  + Application in sectors like real estate, interior design, and social media.
* **Market Potential:**
  + Growing demand for privacy-preserving technologies.
  + Competitive advantage with state-of-the-art AI solutions.
* **Conclusion:**
  + Summary of the project benefits.
  + Call to action for stakeholders to support and invest in the project.

**Introduction**

Object detection is a groundbreaking computer vision task that has a ton of applications across various industries. It goes beyond traditional image classification, where a model assigns a single label to an entire image, to identify and locate multiple objects within an image, often accompanied by bounding boxes outlining their positions.

When working on custom models for object detection or other machine learning tasks, one of the challenges that researchers and developers may encounter is the lack of suitable databases or datasets. Overcoming these challenges often requires creativity and resourcefulness so this post will focus on how to create your custom database.

(INSERT IMAGE SHOWING LABELED DOCUMENT)

**Part 1: Introduction and Setup for Roboflow**

Welcome to Part 1 of our three-part tutorial series on Building Your Own Real-Time Object Detection App: Roboflow(YOLOv8) and Streamlit. In this series, we will walk you through the process of building an end-to-end object detection app that can identify objects from a photo. This web app was built only for images because we are using [share.streamlit.io](http://share.streamlit.io/) this is the Streamlit project hub where you can post your Streamlit projects free and it has a limit of 1 GB memory space for the app, there is a few libraries that cover a lot of that space so in another post or series I’ll add more about video and webcam functions to complement this app.

In Part 1, we will introduce the project, give you a demo of the app in action, and explain why I chose Roboflow and Streamlit for this project. We will also guide you through the setup process, including installing dependencies and creating the necessary files and directories.

By the end of this series, you will have the skills to build your own object detection app. So, let’s dive in!

**Demo of the Object Detection App**

This is the [web app](https://objectdetection-eduardo.streamlit.app/) demo from the project that we are going to create and build together in the Streamlit share cloud. The app Object Detection will Upload an image on the WebApp and show detected objects.

**Object Detection**

Object detection is a computer vision solution that identifies instances of objects in visual media. Object detection scripts draw a bounding box around an instance of a detected object, paired with a label to represent the contents of the box. For example, a person in an image might be labeled “person” and a car might be labeled “vehicle”.

**What is YOLOv8?**

YOLOv8 is the newest state-of-the-art YOLO model that can be used for object detection, image classification, and instance segmentation tasks. YOLOv8 was developed by [Ultralytics](https://ultralytics.com/?ref=blog.roboflow.com), this model is used in Roboflow.

**Why Should I Use YOLOv8?**

Here are a few main reasons why you should consider using YOLOv8 for your next computer vision project:

1. YOLOv8 has a high rate of accuracy measured by COCO and Roboflow 100.
2. YOLOv8 comes with a lot of developer-convenience features,an a well-structured Python package.
3. The labeling tool is easy to use and you don’t need to install a tool for that.
4. And last but not least is not difficult to run it also is faster than use a notebook with TensorFlow. In my case it takes 3 hours to train the model in Google Colab but with Roboflow it took me a few minutes.

**Why Streamlit is a Good Choice for Building a ML App**

[Streamlit](https://docs.streamlit.io/) makes it easy to build web-based user interfaces for machine learning applications, enabling data scientists and developers to share their work with non-technical stakeholders.

Streamlit is an open-source framework that simplifies the process of building web applications in Python. And it has it’s own project cloud that makes really easy deploy your project.

**Project Setup: Installing Dependencies and Creating Required Files and Directories**

Before diving into the project, make sure you have the following dependencies installed on your system. In my case I’m a Windows user so everything in this tutorial is working for July 2023 in Windows 11.

For this project I have Python 3.11 but in Streamlit cloud only has the version 3.8 to 3.11 so I recommend using that range of versions and the Python packages that we will use will be PyTorch, Ultralytics and Streamlit. We can install these packages using pip into a separate virtual environment.

**Creating Virtual Environment**

When working on a Python project, it’s important to keep your dependencies separate from your global Python environment to prevent conflicts between different projects, especially with Pytorch.

Make sure you already have installed Python, VS code(or other IDE) and Git. Follow the next steps:

Create a new virtual environment by running the following command in the terminal after venv you can name as you wish your environment:

python -m venv env

Then activate the enviroment:

env\Scripts\activate

The first step is getting our data set (Images folder). In this case I recommend having at least 200 images. While the more pictures you have, the better your model becomes but don’t use pictures nearly identicals. I’m using 4 different sign hand posture so taking 50 photos with any device can take a lot of time so let’s create an environment only for the script that will take photos with our web cam. In this environment we only need to install OpenCV. So run in your terminal:

pip install opencv-python

Now you can run the following script, basically you can modify the labels, these labels will be used to create folders and will take the number of images that you declared. After finishing with the first label it will continue with the next one until it finishes the labels list. And will display a window that shows what is capturing. Also you can modify the time between each shot and time between the labels capture. Start taking pictures:

import cv2   
import uuid  
import os  
import time  
  
labels = ['thumbsup', 'hi', 'loveyou', 'livelong'] #modify the labels as you need  
number\_imgs = 20#number of images that will take  
  
IMAGES\_PATH = os.path.join('images')  
  
if not os.path.exists(IMAGES\_PATH):  
 os.makedirs(IMAGES\_PATH)  
  
for label in labels:#Loop that creates folders for the labels  
 path = os.path.join(IMAGES\_PATH, label)  
 if not os.path.exists(path):  
 os.makedirs(path)  
  
for label in labels:#Loop that takes the pictures for each label  
 cap = cv2.VideoCapture(0)  
 print('Collecting images for {}'.format(label))  
 time.sleep(10)#Time before start taking pictures  
 for imgnum in range(number\_imgs):  
 print('Collecting image {}'.format(imgnum))  
 ret, frame = cap.read()  
 imgname = os.path.join(IMAGES\_PATH, label, label + '.' + '{}.jpg'.format(str(uuid.uuid1())))  
 cv2.imwrite(imgname, frame)  
 cv2.imshow('frame', frame)  
 time.sleep(2)#Time between each camera shot  
  
 if cv2.waitKey(1) & 0xFF == ord('q'):  
 break  
cap.release()  
cv2.destroyAllWindows()

At this point we will have the amount of images that we need but the name of each picture is random so we have to rename it to make it easier to identify each image. The next code will rename each image in just one folder so run the code for each folder in your project.

import os  
import glob  
  
def rename\_images(folder\_path):  
 # Change the current working directory to the folder with images  
 os.chdir(folder\_path)  
  
 # Get a list of all image files in the folder  
 image\_files = glob.glob("\*.jpg") + glob.glob("\*.jpeg") + glob.glob("\*.png") + glob.glob("\*.gif")  
  
 # Sort the list of image files alphabetically  
 image\_files.sort()  
  
 # Initialize a counter to create sequential numbers  
 counter = 1  
  
 # Rename each image file  
 for old\_name in image\_files:  
 # Get the file extension  
 extension = os.path.splitext(old\_name)[1]  
  
 # Create the new name with the desired format (e.g., "title\_1.jpg", "title\_2.jpg", etc.)  
 new\_name = f"thumbsup\_{counter}{extension}" #change the word before the \_ for the name  
  
 # Rename the file  
 os.rename(old\_name, new\_name)  
  
 # Increment the counter for the next image  
 counter += 1  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 folder\_path = "images\hi"#change path for every folder  
 rename\_images(folder\_path)

**Create a project with Roboflow**

Building a custom dataset can be a painful process. It might take dozens or even hundreds of hours to collect images, label them, and export them in the proper format. Fortunately, Roboflow makes this process straightforward. If you only have images, you can label them in [Roboflow Annotate](https://docs.roboflow.com/annotate?ref=blog.roboflow.com). (When starting from scratch, consider [annotating large batches of images via API](https://docs.roboflow.com/annotate/annotate-api?ref=blog.roboflow.com) or use the [model-assisted labeling](https://blog.roboflow.com/announcing-label-assist/) tool to speed things up.)

Before you start, you need to create a Roboflow [account](https://app.roboflow.com/login?ref=blog.roboflow.com). Once you do that, you can create a new project in the Roboflow dashboard.

A screenshot of a computer

Description automatically generated

Keep in mind to choose the right project type. In this case choose, *Object Detection*.

A screenshot of a project

Description automatically generated

**Upload your images**

Add data to your newly created project. You can do it through the [web interface](https://docs.roboflow.com/adding-data/object-detection?ref=blog.roboflow.com). If you don’t have a dataset, you can grab one from [Roboflow Universe](https://universe.roboflow.com/?ref=blog.roboflow.com).

If you drag and drop a directory with a data set in a supported format, the Roboflow dashboard will automatically read the images and annotations together. To create a data set with annotations locally in Windows check [this post](https://medium.com/@lalodatos/label-your-images-with-labelimg-in-windows-for-object-detection-models-1b0a66f00a7b).

A screenshot of a browser

Description automatically generated

A screenshot of a computer

Description automatically generated

After all images uploaded you can click Save and Continue.

A screenshot of a computer

Description automatically generated

Then it will appear the pop-up window and you can Click only in *Assing Images*, in this part if you are working with a Team you can invite them to add images or labeling.

A screenshot of a computer

Description automatically generated

Then we need to click Start Annotating in case you upload images only to use the label tool from Roboflow.

A screenshot of a computer

Description automatically generated

**Label your images**

Use the tool to select the element with the classes that you are going to use in your model. And repeat the same process for all the images.

A screenshot of a computer

Description automatically generated

After you finish labeling all the images click the back button highlighted in red in the image below.

A screenshot of a computer

Description automatically generated

Now we can add all the images to the Dataset with the button Add *n* Image to the Dataset.

A screenshot of a computer

Description automatically generated

Noe will appear the option to Add Images you can choose different options I recommend using the default option.

A screenshot of a computer

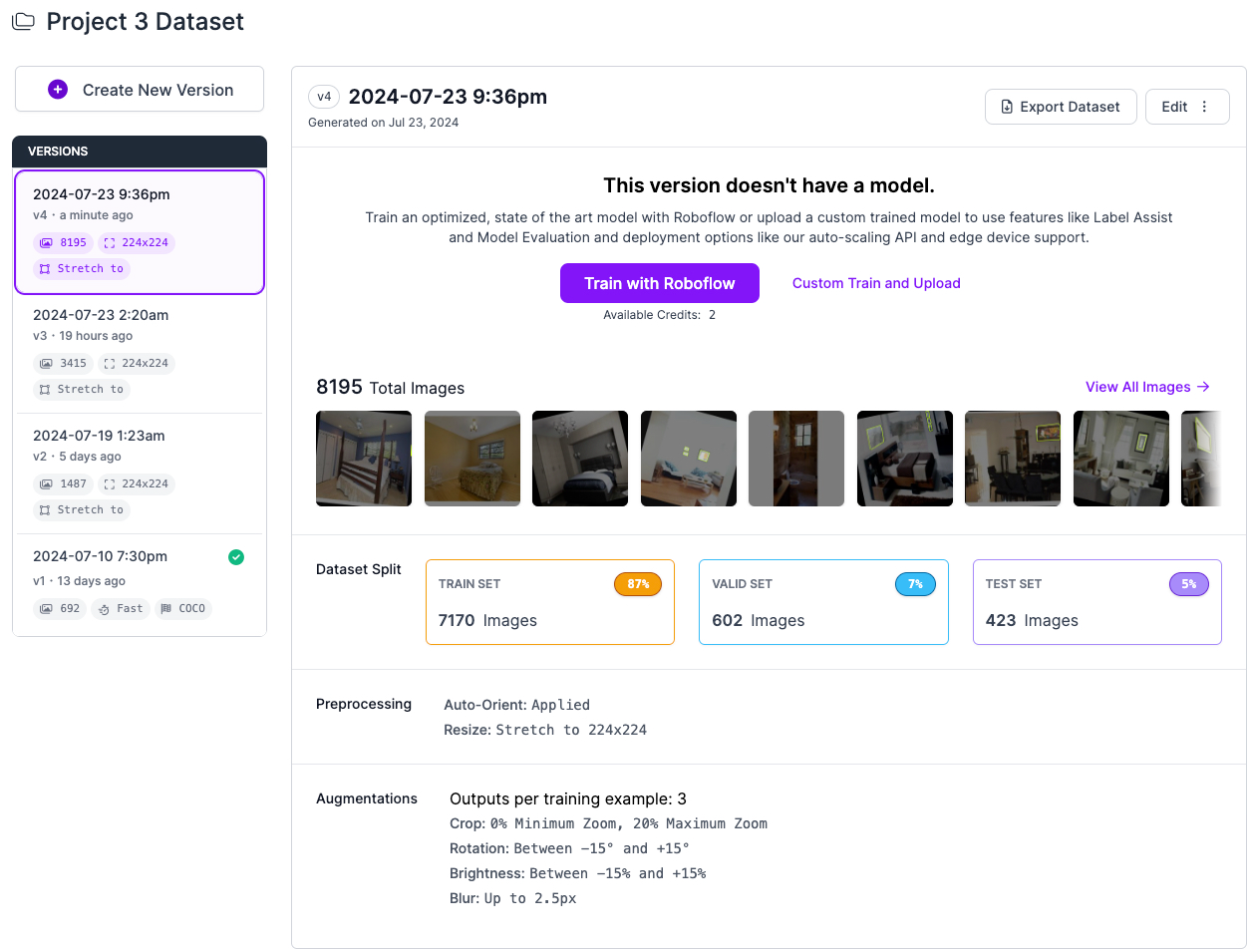
Description automatically generated

After loading our images to the database another window will appear. You need to make sure that there are no *UNASSIGNED* images and the Dataset is ready, once you have it similar as the image below you can Click Generate New Version.

A screenshot of a computer

Description automatically generated

When we *Generate a New Version* we can use some tools to prepare the data and experiment with them. Go to option 3.



In this option we can apply transformations in all the images, so make sure to configure this depending on your project. Maybe you are using a camera in Raspberry Pi or maybe you want to use images with a specific format. For my project this configuration is perfect.

A screenshot of a computer

Description automatically generated

Option 4 is an amazing tool because you can generate extra versions from your images that can duplicate or triplicate in the free version of the dataset. Let’s see the options.

A screenshot of a computer

Description automatically generated

For this project I’ll use flip horizontal, try to experiment with it, and depending on your project you can choose the options that you need.

A screenshot of a video game

Description automatically generated

After you choose an *Augmentation* you will see extra options. For my project I only need the Horizontal. Try to check what is best for your custom project. After that click *Apply*

A screenshot of a computer

Description automatically generated

Then click continue to step 5 and last.

A screenshot of a computer

Description automatically generated

Select the Maximun Version and then click *Generate* and is ready to go.

A screenshot of a computer

Description automatically generated

After this will appear the next page:

A screenshot of a computer

Description automatically generated

Congratulations now you have an Image Dataset ready to train a model.

**Conclusion**

In this first part of our tutorial series, we have introduced you to the Image Data Set in Roboflow. This post gives you a script to capture images from your webcam and create the necessary files and directories. Also a full guide of Roboflow tools in action.

In [Part 2](https://lalodatos.medium.com/building-your-own-real-time-object-detection-app-yolov8-and-streamlit-part-2-52cac252b766), we will focus on Training using Roboflow and a notebook in Google Colab and get the necessary file for the Streamlit code. We will explain the concept of object detection and tracking, and guide you through the process of setting up your computer vision environment and integrating YOLOv8 with OpenCV.

[Roboflow](https://medium.com/tag/roboflow?source=post_page-----f577cf0aa6e5---------------roboflow-----------------)

[Streamlit](https://medium.com/tag/streamlit?source=post_page-----f577cf0aa6e5---------------streamlit-----------------)

[Python](https://medium.com/tag/python?source=post_page-----f577cf0aa6e5---------------python-----------------)

[Object Detection](https://medium.com/tag/object-detection?source=post_page-----f577cf0aa6e5---------------object_detection-----------------)

**Train YOLOv8 on a custom dataset**

In this section, we will dive deeper into the YOLOv8 object detection model and explore how to train it .

There are a wide range of open-source object detection models available. A popular choice is models in the YOLO (You Only Look Once) family, which continue to represent the state-of-the-art in object detection tasks.

Once you have a labeled dataset, and you have made your augmentations, it is time to start training an object detection model. Training involves showing instances of your labeled data to a model in batches and iteratively improving the way the model is mapping images to predictions.

As with labeling, you can take two approaches to training and inferring with object detection models train and deploy yourself, or use training and inference services like Roboflow Train and Roboflow Deploy. Both of which are free for Public plans.

A screenshot of a computer

Description automatically generated

In [Part 1](https://lalodatos.medium.com/building-your-own-real-time-object-detection-app-roboflow-yolov8-and-streamlit-part-1-f577cf0aa6e5) we finished the *Versions*tool from our Roboflow project now is time to train the model. We have to choose the option Custom Train using YOLOv5 and then *Get Snippet*.

A screenshot of a computer

Description automatically generated

A pop up copy the lines or save it we need the api\_key to modify the notebook, will open a notebook in Google Colab after clicking Copy Snippet. Is a repository make sure to create a copy to save the changes first.

A screenshot of a computer

Description automatically generated

When you open the notebook it is necessary to run all to set up the Colab session. There are a few cells that you can avoid but check it first.

A screenshot of a computer

Description automatically generated

If we remember we have the api\_key and extra information about our data set we will use it in the *Step 5: Exporting dataset* from the Notebook we will find a code cell and we need to replace with the copied lines from Roboflow after that we can run everything without modifying anything else.

A screen shot of a computer

Description automatically generated

**Deploy model on Roboflow**

Once you have finished training the YOLOv8 model, you’ll have a set of trained weights ready for use. These weights will be in the /runs/detect/train/weights/best.pt folder of your project. You need to download thefilebest.pt to use it in the Streamlit app.

A screenshot of a computer

Description automatically generated

**Conclusion**

In this second part of our tutorial series, we have introduced you to the training . This post is short but necessary. This gives us an idea about how important the first step is. Roboflow gives us a notebook that has everything you need with small but necessary changes of how to train the model with our data set.

**Creating a Streamlit WebApp for Image Object Detection with a Roboflow model**

Streamlitis an open-source app framework for Machine Learning and Data Science teams. Create beautiful web apps in minutes. Streamlit apps are Python scripts that run from top to bottom. Every time a user opens a browser tab pointing to your app, the script is re-executed. As the script executes, Streamlit draws its output live in a browser.

[**Create an app**](https://docs.streamlit.io/library/get-started/create-an-app) using Streamlit’s core features to fetch and cache data, draw charts, plot information on a map, and use interactive widgets, like a slider, to filter results.

Let’s prepare the virtual environment for the Streamlit app. First let’s create a virtual environment and once created then activate it (Windows).

python -m venv env  
env\Scripts\activate

Then we have to install PyTorch, Ultralytics and Streamlit. Try to install in the next order.

pip install torch  
pip install ultralytics  
pip install streamlit

After this we are ready to try the hello world in Streamlit to check that everything is installed correctly. Create a file called app.py and put the next code lines using your favorite IDE:

import streamlit as st  
st.write("Hello, World!")

Then run it from the terminal in cmd and if everything works fine will open the browser.

streamlit run app.py

Then to create a tool to upload our pictures and use the model we need to open the code editor and let’s get started by replacing the previous file and creating a new one named app.py. But we also need a folder called weights and for the moment is everything. Now let’s go to the next step.

**Create a Uploading an Image On Streamlit WebApp**

We’ll use Streamlit to allow users to upload an image. After successfully uploading an image, is ready to run object detection on the uploaded image using YOLOv8. This step involves loading the YOLOv8 model and passing the uploaded image through the model to identify the objects present in the image.

We will also visualize the output of the model with the identified objects highlighted in the image. Let’s go into the code.

In [Part 2](https://lalodatos.medium.com/building-your-own-real-time-object-detection-app-yolov8-and-streamlit-part-2-52cac252b766) of this series, we have discussed how to download a pre-trained weight file of the Yolov8 model. downloaded the best.pt file and saved it inside our weights directory. We will use the same weight file. In the created file with the name app.pywrite the following lines of code:

# Import required libraries  
import PIL  
  
import streamlit as st  
from ultralytics import YOLO  
  
# Replace the relative path to your weight file  
model\_path = 'weights/best.pt'  
  
# Setting page layout  
st.set\_page\_config(  
 page\_title="Object Detection", # Setting page title  
 page\_icon="🤖", # Setting page icon  
 layout="wide", # Setting layout to wide  
 initial\_sidebar\_state="expanded", # Expanding sidebar by default  
   
)  
  
# Creating sidebar  
with st.sidebar:  
 st.header("Image Config") # Adding header to sidebar  
 # Adding file uploader to sidebar for selecting images  
 source\_img = st.file\_uploader(  
 "Upload an image...", type=("jpg", "jpeg", "png", 'bmp', 'webp'))  
  
 # Model Options  
 confidence = float(st.slider(  
 "Select Model Confidence", 25, 100, 40)) / 100  
  
# Creating main page heading  
st.title("Object Detection")  
st.caption('Updload a photo with this :blue[hand signals]: :+1:, :hand:, :i\_love\_you\_hand\_sign:, and :spock-hand:.')  
st.caption('Then click the :blue[Detect Objects] button and check the result.')  
# Creating two columns on the main page  
col1, col2 = st.columns(2)  
  
# Adding image to the first column if image is uploaded  
with col1:  
 if source\_img:  
 # Opening the uploaded image  
 uploaded\_image = PIL.Image.open(source\_img)  
 # Adding the uploaded image to the page with a caption  
 st.image(source\_img,  
 caption="Uploaded Image",  
 use\_column\_width=True  
 )  
  
try:  
 model = YOLO(model\_path)  
except Exception as ex:  
 st.error(  
 f"Unable to load model. Check the specified path: {model\_path}")  
 st.error(ex)  
  
if st.sidebar.button('Detect Objects'):  
 res = model.predict(uploaded\_image,  
 conf=confidence  
 )  
 boxes = res[0].boxes  
 res\_plotted = res[0].plot()[:, :, ::-1]  
 with col2:  
 st.image(res\_plotted,  
 caption='Detected Image',  
 use\_column\_width=True  
 )  
 try:  
 with st.expander("Detection Results"):  
 for box in boxes:  
 st.write(box.xywh)  
 except Exception as ex:  
 st.write("No image is uploaded yet!")

You can modify the app text in the st.caption line codes as you prefer for your project now let’s run the app with in the terminal:

streamlit run app.py

This will deploy our app in the web browser that we are currently using, upload an image an check that identifies the objects:

A screenshot of a computer

Description automatically generated

If everything run properly please run the next command to get the requirements:

pip freeze > requirements.txt

Also we need to create a file called packages.txt in the code folder that and put this line in it:

libgl1

Now we can create a repo in Github to put our app in the streamlit.io but before that make sure erase everything in the requierements except for 3 things:

torch==2.0.1  
ultralytics==8.0.142  
streamlit==1.25.0

The Streamlit.io only allows uploading 1GB. The installations use the most of the space, so to avoid that we leave the three mandatory libraries for our app. Check my [repository](https://github.com/fullmakeralchemist/teststream/tree/master) and how it needs to uploaded.

A screenshot of a computer

Description automatically generated

From here we are ready to go to[streamlit.io](https://share.streamlit.io/) and deploy our app. Create an account and then will appear the next window click in the New app.

A white background with black text

Description automatically generated

Connect the Streamlit account with Github and then select the repository where your app is located. then select the branch and change the Main file path:

A screenshot of a application

Description automatically generated

Now select the Python version and save the changes after this we can click the deploy button.

A screenshot of a computer

Description automatically generated

Then will appear the next window showing all the installations. Check if there are errors.

A close-up of a peanut

Description automatically generated

Then you can try your app and check if it works properly.

A hand gesture with fingers

Description automatically generated with medium confidence

**Conclusion**

I hope this guide is useful for your projects. The combination of Roboflow and Streamlit enables the development of applications with a user-friendly interface. This approach makes it easier to detect and track objects in real time, allowing for a wide range of use cases not only for Object detection models for other data science and ML projects.

(MISSING PART 5)

**So, what is the solution for this?..**You need to use the tool named[streamlit-webrtc](https://pypi.org/project/streamlit-webrtc/) .[streamlit-webrtc](https://github.com/whitphx/streamlit-webrtc?ref=blog.streamlit.io), is a component that enables Streamlit to handle real-time media streams over a network to solve this problem. In this part , I’ll also briefly introduce you to WebRTC (check out the article from the creator of the library[article here](https://dev.to/whitphx/python-webrtc-basics-with-aiortc-48id?ref=blog.streamlit.io) for more in-depth info on WebRTC). If you want to jump right to playing with the component[here is a sample](https://share.streamlit.io/whitphx/streamlit-webrtc-example/main/app.py?ref=blog.streamlit.io) app.

Ready? Let’s dive in.

A yellow folder and a cloud

Description automatically generated

**The problem with existing approaches**

Streamlit is actively used by many developers and researchers to prototype apps backed with computer vision and machine learning models, but it can’t yet natively support real-time video processing.

One existing approach to achieve real-time video processing with Streamlit is to use OpenCV to capture video streams. However, this only works when the Python process can access the video source — in other words, only when the camera is connected to the same host the app is running on.

Due to this limitation, there have always been problems with deploying the app to remote hosts and using it with video streams from local webcams. cv2.VideoCapture(0) consumes a video stream from the first (indexed as 0) locally connected device, and when the app is hosted on a remote server, the video source is a camera device connected to the *server*- not a local webcam. In a Raspberry Pi we can confirm this running this command in the command linels/dev/video\* .

**How WebRTC resolves this issue**

WebRTC (Web Real-Time Communication) enables web servers and clients, including web browsers, to send and receive video, audio, and arbitrary data streams over the network with low latency.

It is now supported by major browsers like Chrome, Firefox, and Safari, and its specs are open and standardized. Browser-based real-time video chat apps like Google Meet are common examples of WebRTC usage.

WebRTC extends Streamlit’s powerful capabilities to transmit video, audio, and arbitrary data streams between frontend and backend processes, like browser JavaScript and server-side Python.

If you want to know more about these WebRTC concepts, read [this article](https://dev.to/whitphx/python-webrtc-basics-with-aiortc-48id?ref=blog.streamlit.io).

This tutorial will use Streamlit-WebRTC and Pytube.

**Install dependencies**

Install the necessary packages. Note that this tutorial work with the latest version of all libraries mentioned before but now that we have our project create a copy and in the requirements.txt file we will add the next lines:

git+https://github.com/fullmakeralchemist/pytube/  
git+https://github.com/gatagat/lap@new-packaging  
streamlit==1.2.0  
streamlit-webrtc==0.35.1  
torch==2.0.1  
ultralytics==8.0.150

We will focus on the first line because in part 4 we made changes in one file from the library but how we can do that in Streamlit deployment, unfortunately Streamlit share doesn’t have a terminal to work so how can we make changes in the library? I forked the repository of the library

A screenshot of a computer

Description automatically generated

Once I had the fork in my github profile I look for the file Cipher.py and then cliked the edit file button to modify the line number 30 for:var\_regex = re.compile(r"^\w+\W") to var\_regex = re.compile(r"^$\*\w+\W"). With this change we can save the file and add the url to the first line in the requirements.txt file.

A screenshot of a computer

Description automatically generated

**The changes in the code of OpenCV to**streamlit-webrtc

**The advantages of web-based apps**

We have been typically using OpenCV to build real-time demo apps of image or video processing. Some of you may have seen the following code or similar many times.

import cv2  
  
cap = cv2.VideoCapture(0)  
while True:  
 ret, frame = cap.read()  
 img = cv2.Canny(frame, 100, 200) # Some image processing  
 cv2.imshow('frame', img)  
 if cv2.waitKey(1) & 0xFF == ord('q'):  
 break  
cap.release()  
cv2.destroyAllWindows()

Compared to the GUI apps like above using cv2.VideoCapture and cv2.imshow that run on local environments, web-based apps have some advantages like below.

Easy to share and run:

* If we deploy the apps on the cloud, we can share the apps with our users simply by sending the URLs.
* The users can use the apps only by accessing them through web browsers. It does not require any set-ups or external dependencies.

Usable on smartphones:

* Because all the users need is web browsers, the users can use the apps on their smartphones. It’s convenient if we can show demos on such portable devices.

**Install necessary packages**

Next, we have to install the packages necessary for this guide. Is not necessary, only if you are going to run it locally but this guide is to run it in the Streamlit share so with having the correct name in the requirements.txt file is only we need.

$ pip install streamlit-webrtc

* streamlit-webrtc: A custom component of Streamlit which deals with real-time video and audio streams.

**Introduce the real-time video/audio streaming component**

Create a file app.py as below.

import streamlit as st  
from streamlit\_webrtc import webrtc\_streamer  
  
st.title("My first Streamlit app")  
st.write("Hello, world")  
webrtc\_streamer(key="example")

We have added a single line with webrtc\_streamer(). The web app would be like the screenshot below.

A screenshot of a black screen

Description automatically generated

At the first trial, it may take some time to compile the package so that the page keeps showing the “running” message for a while after clicking the “Rerun” button. In such a case, wait for the process to finish.

Click the “START” button to start the video and audio streaming. You may be asked for permission to access the webcam and microphone at the first trial. Allow permission in that case.

The webrtc\_streamer(key="example") above is a Streamlit component which deals with video and audio real-time I/O through web browsers. The key argument is a unique ID in the script to identify the component instance. We have set it as "example" here, but you can use any string for it. The component in this example only receives video and audio from the client-side webcam and microphone and outputs the raw streams. It's the most basic version of the component. We are going to enhance its functionality by adding other options in the following sections.

**Development of a real-time video processing application**

We will use the code from part 4 and we only have to update the helper.py as follows.

from ultralytics import YOLO  
import streamlit as st  
from streamlit\_webrtc import webrtc\_streamer, VideoTransformerBase  
import numpy as np  
from PIL import Image  
import av  
import cv2  
from pytube import YouTube  
  
import settings  
  
  
def load\_model(model\_path):  
 """  
 Loads a YOLO object detection model from the specified model\_path.  
  
 Parameters:  
 model\_path (str): The path to the YOLO model file.  
  
 Returns:  
 A YOLO object detection model.  
 """  
 model = YOLO(model\_path)  
 return model  
  
  
def display\_tracker\_options():  
 display\_tracker = st.radio("Display Tracker", ('Yes', 'No'))  
 is\_display\_tracker = True if display\_tracker == 'Yes' else False  
 if is\_display\_tracker:  
 tracker\_type = st.radio("Tracker", ("bytetrack.yaml", "botsort.yaml"))  
 return is\_display\_tracker, tracker\_type  
 return is\_display\_tracker, None  
  
  
def \_display\_detected\_frames(conf, model, st\_frame, image, is\_display\_tracking=None, tracker=None):  
 """  
 Display the detected objects on a video frame using the YOLOv8 model.  
  
 Args:  
 - conf (float): Confidence threshold for object detection.  
 - model (YoloV8): A YOLOv8 object detection model.  
 - st\_frame (Streamlit object): A Streamlit object to display the detected video.  
 - image (numpy array): A numpy array representing the video frame.  
 - is\_display\_tracking (bool): A flag indicating whether to display object tracking (default=None).  
  
 Returns:  
 None  
 """  
  
 # Resize the image to a standard size  
 image = cv2.resize(image, (720, int(720\*(9/16))))  
  
 # Display object tracking, if specified  
 if is\_display\_tracking:  
 res = model.track(image, conf=conf, persist=True, tracker=tracker)  
 else:  
 # Predict the objects in the image using the YOLOv8 model  
 res = model.predict(image, conf=conf)  
  
 # # Plot the detected objects on the video frame  
 res\_plotted = res[0].plot()  
 st\_frame.image(res\_plotted,  
 caption='Detected Video',  
 channels="BGR",  
 use\_column\_width=True  
 )  
  
  
def play\_youtube\_video(conf, model):  
 """  
 Plays a webcam stream. Detects Objects in real-time using the YOLOv8 object detection model.  
  
 Parameters:  
 conf: Confidence of YOLOv8 model.  
 model: An instance of the `YOLOv8` class containing the YOLOv8 model.  
  
 Returns:  
 None  
  
 Raises:  
 None  
 """  
 source\_youtube = st.sidebar.text\_input("YouTube Video url")  
  
 is\_display\_tracker, tracker = display\_tracker\_options()  
  
 if st.sidebar.button('Detect Objects'):  
 try:  
 yt = YouTube(source\_youtube)  
 stream = yt.streams.filter(file\_extension="mp4", res=720).first()  
 vid\_cap = cv2.VideoCapture(stream.url)  
  
 st\_frame = st.empty()  
 while (vid\_cap.isOpened()):  
 success, image = vid\_cap.read()  
 if success:  
 \_display\_detected\_frames(conf,  
 model,  
 st\_frame,  
 image,  
 is\_display\_tracker,  
 tracker  
 )  
 else:  
 vid\_cap.release()  
 break  
 except Exception as e:  
 st.sidebar.error("Error loading video: " + str(e))  
  
  
def play\_rtsp\_stream(conf, model):  
 """  
 Plays an rtsp stream. Detects Objects in real-time using the YOLOv8 object detection model.  
  
 Parameters:  
 conf: Confidence of YOLOv8 model.  
 model: An instance of the `YOLOv8` class containing the YOLOv8 model.  
  
 Returns:  
 None  
  
 Raises:  
 None  
 """  
 source\_rtsp = st.sidebar.text\_input("rtsp stream url")  
 is\_display\_tracker, tracker = display\_tracker\_options()  
 if st.sidebar.button('Detect Objects'):  
 try:  
 vid\_cap = cv2.VideoCapture(source\_rtsp)  
 st\_frame = st.empty()  
 while (vid\_cap.isOpened()):  
 success, image = vid\_cap.read()  
 if success:  
 \_display\_detected\_frames(conf,  
 model,  
 st\_frame,  
 image,  
 is\_display\_tracker,  
 tracker  
 )  
 else:  
 vid\_cap.release()  
 break  
 except Exception as e:  
 st.sidebar.error("Error loading RTSP stream: " + str(e))  
  
  
  
  
def play\_webcam(conf, model):  
 """  
 Plays a webcam stream. Detects Objects in real-time using the YOLO object detection model.  
  
 Returns:  
 None  
  
 Raises:  
 None  
 """  
 st.sidebar.title("Webcam Object Detection")  
  
 webrtc\_streamer(  
 key="example",  
 video\_transformer\_factory=lambda: MyVideoTransformer(conf, model),  
 rtc\_configuration={"iceServers": [{"urls": ["stun:stun.l.google.com:19302"]}]},  
 media\_stream\_constraints={"video": True, "audio": False},  
 )  
  
class MyVideoTransformer(VideoTransformerBase):  
 def \_\_init\_\_(self, conf, model):  
 self.conf = conf  
 self.model = model  
  
 def recv(self, frame):  
 image = frame.to\_ndarray(format="bgr24")  
 processed\_image = self.\_display\_detected\_frames(image)  
 st.image(processed\_image, caption='Detected Video', channels="BGR", use\_column\_width=True)  
  
 def \_display\_detected\_frames(self, image):  
 orig\_h, orig\_w = image.shape[0:2]  
 width = 720 # Set the desired width for processing  
  
 # cv2.resize used in a forked thread may cause memory leaks  
 input = np.asarray(Image.fromarray(image).resize((width, int(width \* orig\_h / orig\_w))))  
  
 if self.model is not None:  
 # Perform object detection using YOLO model  
 res = self.model.predict(input, conf=self.conf)  
  
 # Plot the detected objects on the video frame  
 res\_plotted = res[0].plot()  
 return res\_plotted  
  
 return input  
  
  
  
def play\_stored\_video(conf, model):  
 """  
 Plays a stored video file. Tracks and detects objects in real-time using the YOLOv8 object detection model.  
  
 Parameters:  
 conf: Confidence of YOLOv8 model.  
 model: An instance of the `YOLOv8` class containing the YOLOv8 model.  
  
 Returns:  
 None  
  
 Raises:  
 None  
 """  
 source\_vid = st.sidebar.selectbox(  
 "Choose a video...", settings.VIDEOS\_DICT.keys())  
  
 is\_display\_tracker, tracker = display\_tracker\_options()  
  
 with open(settings.VIDEOS\_DICT.get(source\_vid), 'rb') as video\_file:  
 video\_bytes = video\_file.read()  
 if video\_bytes:  
 st.video(video\_bytes)  
  
 if st.sidebar.button('Detect Video Objects'):  
 try:  
 vid\_cap = cv2.VideoCapture(  
 str(settings.VIDEOS\_DICT.get(source\_vid)))  
 st\_frame = st.empty()  
 while (vid\_cap.isOpened()):  
 success, image = vid\_cap.read()  
 if success:  
 \_display\_detected\_frames(conf,  
 model,  
 st\_frame,  
 image,  
 is\_display\_tracker,  
 tracker  
 )  
 else:  
 vid\_cap.release()  
 break  
 except Exception as e:  
 st.sidebar.error("Error loading video: " + str(e))

We have to define a callback that receives an input frame and returns an output frame. We also need to put image processing code inside the callback. As a result, we have injected the image processing code into the real-time video app through the callback.

Detailed explanations about the code follow.

1. play\_webcam(conf, model): This function plays a webcam video stream. It uses the webrtc\_streamer function from the streamlit\_webrtc library to capture the webcam feed and process frames in real-time using the MyVideoTransformer class.
2. MyVideoTransformer(VideoTransformerBase): This class is a subclass of VideoTransformerBase provided by streamlit\_webrtc. It initializes with the confidence threshold and YOLOv8 model. The recv method processes each frame received from the webcam feed and displays the detected objects.

**Deploy the app to the cloud**

We are going to make the web app available to everyone by deploying it to the cloud. To deploy the app to the cloud, we have to add rtc\_configuration parameter to the webrtc\_streamer(). And also add media\_stream\_constraints to avoid using audio with the webcam.

webrtc\_streamer(  
 key="example",  
 video\_frame\_callback=callback,  
 rtc\_configuration={ # Add this line  
 "iceServers": [{"urls": ["stun:stun.l.google.com:19302"]}]  
 },  
 media\_stream\_constraints={"video": True, "audio": False}  
)

This configuration is necessary to establish the media streaming connection when the server is on a remote host.

streamlit\_webrtc uses WebRTC for its video and audio streaming. It has to access a "STUN server" in the global network for the remote peers (precisely, peers over the NATs) to establish WebRTC connections. While we don't look at the details about STUN servers in this article, please google it with keywords such as STUN, TURN, or NAT traversal if interested.

We configured the code to use a free STUN server provided by Google in the example above. You can also use any other available STUN servers. After this changes we are ready to run the app in Streamlit Share following the same steps in [part 3](https://lalodatos.medium.com/building-your-own-real-time-object-detection-app-yolov8-and-streamlit-part-3-3f69a2a05f3c) to create a repo and then deploy it.

A screenshot of a computer

Description automatically generated

We can see in the image above that in the manage app function from Streamlit is running but unfutonately at this moment is not showing in the webcam the object frames I need to do a deep investigation about Streamlit-WebRTC to solve this but it detects the objects as showen below.

A screen shot of a computer

Description automatically generated

I’ll update this post when I fix it, for the moment is everything and with this you should be able to run the app without problems in Streamlit Share. Check my [demo](https://objectdetectionwebcam.streamlit.app/) and check my [github repository](https://github.com/fullmakeralchemist/gitstreamlit) you can clone it to make the necessary changes for your project.

Keep an eye I added one extra part to upgrade your model I recommend you [part 6 to Enhancing Active Learning and improve your model with new data](https://medium.com/@lalodatos/enhancing-active-learning-uploading-data-to-roboflow-from-windows-or-google-colab-using-the-api-e841ee4c9a4f). In case you want to use the app in Raspberry Pi check [parts 7](https://lalodatos.medium.com/enhancing-active-learning-uploading-data-to-roboflow-from-raspberry-pi-using-the-api-part-4-7ae1f15a8f7d), [part 8](https://lalodatos.medium.com/how-to-deploy-a-roboflow-yolov8-model-to-a-raspberry-pi-part-6-a06ba3b298c9) and [part 9](https://lalodatos.medium.com/building-your-own-real-time-object-detection-app-roboflow-yolov8-and-streamlit-part-9-e204480fa99).If you find errors following this or feedback about this guide let me know in the comments, thank you for following this post. Good luck with your projects.

**Enhancing Active Learning: Uploading Data to Roboflow from Windows or Google Colab using the API (Part 6)**

**Introduction**

Active learning aims to select the most informative samples from a large pool of unlabeled data to be labeled by usually a human annotator to improve the model’s performance.

In the context of Windows, active learning in this project will be collecting data and uploading this new data in Roboflow to increase model performance. It can be particularly useful when working with small datasets, which is common to improve models.

To Achieve this I’ll use Roboflow API to upload images to our existing dataset on the Roboflow platform. You can upload images to Roboflow projects using the web interface, Python SDK, REST API, and CLI.

First we have to have create our Notebook in Google Colab or create a Virtual envioroment in Conda or venv. If you are using Windows check this first. If you are using Colab jump this step. and go to the code

python -m venv env  
env\Scripts\activate

Then we can install notebook to open the notebooks.

pip install notebook

To set up our noteook we have to run the next commands:

import os  
HOME = os.getcwd()  
print(HOME)

# Pip install method (recommended)  
  
!pip install ultralytics==8.0.20  
  
from IPython import display  
display.clear\_output()  
  
import ultralytics  
ultralytics.checks()  
  
from ultralytics import YOLO  
  
from IPython.display import display, Image

!mkdir {HOME}/datasets  
%cd {HOME}/datasets  
  
!pip install roboflow --quiet

After this our notebook is ready to run the scrip. Before diving in to the code I need to remember that we need new data to upload. So make sure you have run the code from [Part 1](https://lalodatos.medium.com/building-your-own-real-time-object-detection-app-roboflow-yolov8-and-streamlit-part-1-f577cf0aa6e5).

After that we now have a new data set that now we can upload our data to Roboflow using the API.

**Upload images to Roboflow using the API and Python.**

First we need to check if we run the pip install roboflow. Then we need to run in the terminal. If we had success installing the library, create a new cell add the code below and click Run with the next code to ulpload the images in the folder you create recently (Just change the corresponding values of your project). The next code is to just upload one image and test:

from roboflow import Roboflow  
  
# Initialize the Roboflow object with your API key  
rf = Roboflow(api\_key="xxxxxxxxxxxxxxxxxxxx")  
  
# Retrieve your current workspace and project name  
print(rf.workspace())  
  
# Specify the project for upload  
project = rf.workspace("xxxxxxxxxx").project("xxxxxxxx")  
  
# Upload the image to your project  
project.upload("/content/two.jpg")  
  
"""  
Optional Parameters:  
- num\_retry\_uploads: Number of retries for uploading the image in case of failure.  
- batch\_name: Upload the image to a specific batch.  
- split: Upload the image to a specific split.  
"""

To upload a entire folder use the next code:

import os  
from roboflow import Roboflow  
  
# Initialize the Roboflow object with your API key  
rf = Roboflow(api\_key="xxxxxxxxxxxxxxxxxxxx")  
  
# Retrieve your current workspace and project name  
print(rf.workspace())  
  
# Specify the project for upload  
project = rf.workspace("xxxxxxxxs").project("xxxxxxxxx")  
  
# Folder path containing all the images  
folder\_path = "/content/images" # Update this to your folder path  
  
# Get a list of all image files in the folder  
image\_files = [os.path.join(folder\_path, file) for file in os.listdir(folder\_path) if file.lower().endswith(('.jpg', '.jpeg', '.png'))]  
  
# Upload each image to your project  
for image\_file in image\_files:  
 project.upload(image\_file)

In the Shell will appear all the information and if is running add some print to warning you when it’s over. When is running you will see that it takes a while, but the images will be uploaded.

A screenshot of a computer

Description automatically generated

After it finishes you can wait a few minutes to see the images in the assign section from your Rboflow project as we see in the image below.

A screenshot of a computer

Description automatically generated

Congratulations now you have a new data set of images to label manually and then retrain your model! By harnessing the potential of Active Learning with Windows or Google Colab and Roboflow API, you’ve created a streamlined process for capturing images and utilizing the Roboflow label them seamlessly. So, go ahead, dive into the labeling process, and let the possibilities of Active Learning and Roboflow API propel your machine learning journey forward.