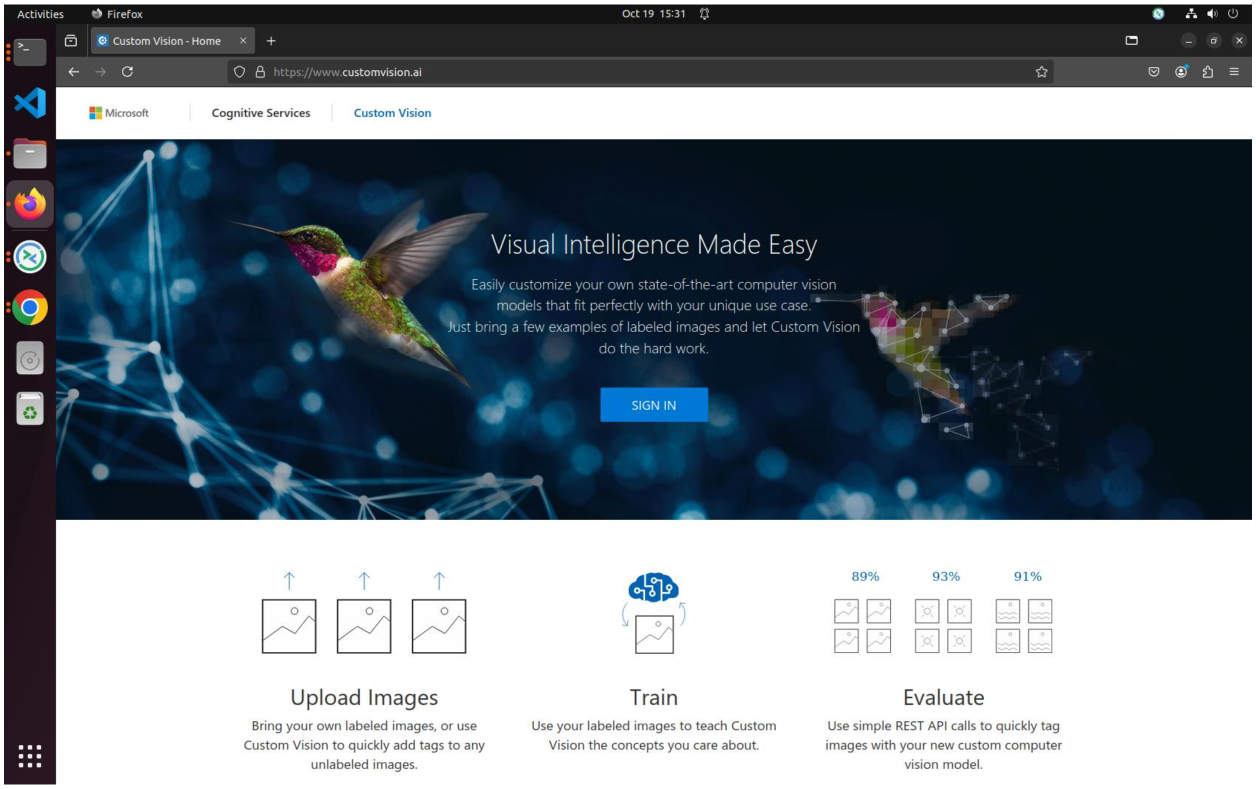
DOCUMENTATION OF AI USING AZURE CUSTOM VISION

To use the azure custom vision service, we will need to create a custom vision training and prediction resources in Azure. To do so in Azure portal , fill out the dialog window on the Create Custom Vision page to create both a Training and Prediction resource.

CREATE A NEW PROJECT

In the web browser , navigate to the https://www.customvision.ai/ and select Sign in. Sign in with the same account you used to sign in to the Azure portal.



Visual Intelligence Made Easy

Easily customize your own state-of-the-art computer vision models that fit perfectly with your unique use case. Just bring a few examples of labeled images and let Custom Vision do the hard work.

Upload Images

Bring your own labeled images, or use Custom Vision to quickly add tags to any unlabeled images.

Train

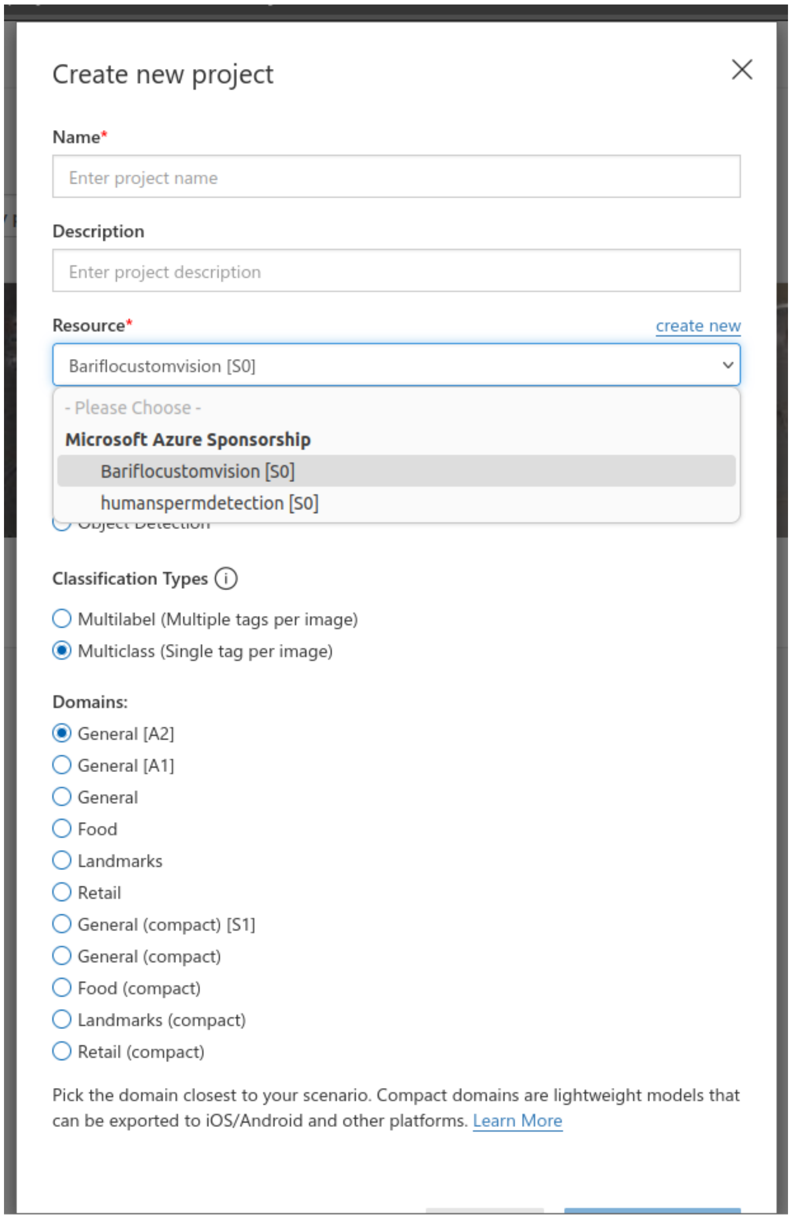
Use your labeled images to teach Custom Vision the concepts you care about.

Evaluate

Use simple REST API calls to quickly tag images with your new custom computer vision model.

Fig 1: Sign in page of the custom vision portal

1. To create your first project, select **New Project**. The **Create new project** dialog box appears.


Create new project

Name*

Enter project name

Description

Enter project description

Resource* create new

Bariflocustomvision [S0]

- Please Choose -

Microsoft Azure Sponsorship

Bariflocustomvision [S0]

humanspermdetection [S0]

Object Detection

Classification Types

Multilabel (Multiple tags per image)

[✓ Multiclass (Single tag per image)]

Domains:

[✓ General [A2]]

General [A1]

General

Food

Landmarks

Retail

General (compact) [S1]

General (compact)

Food (compact)

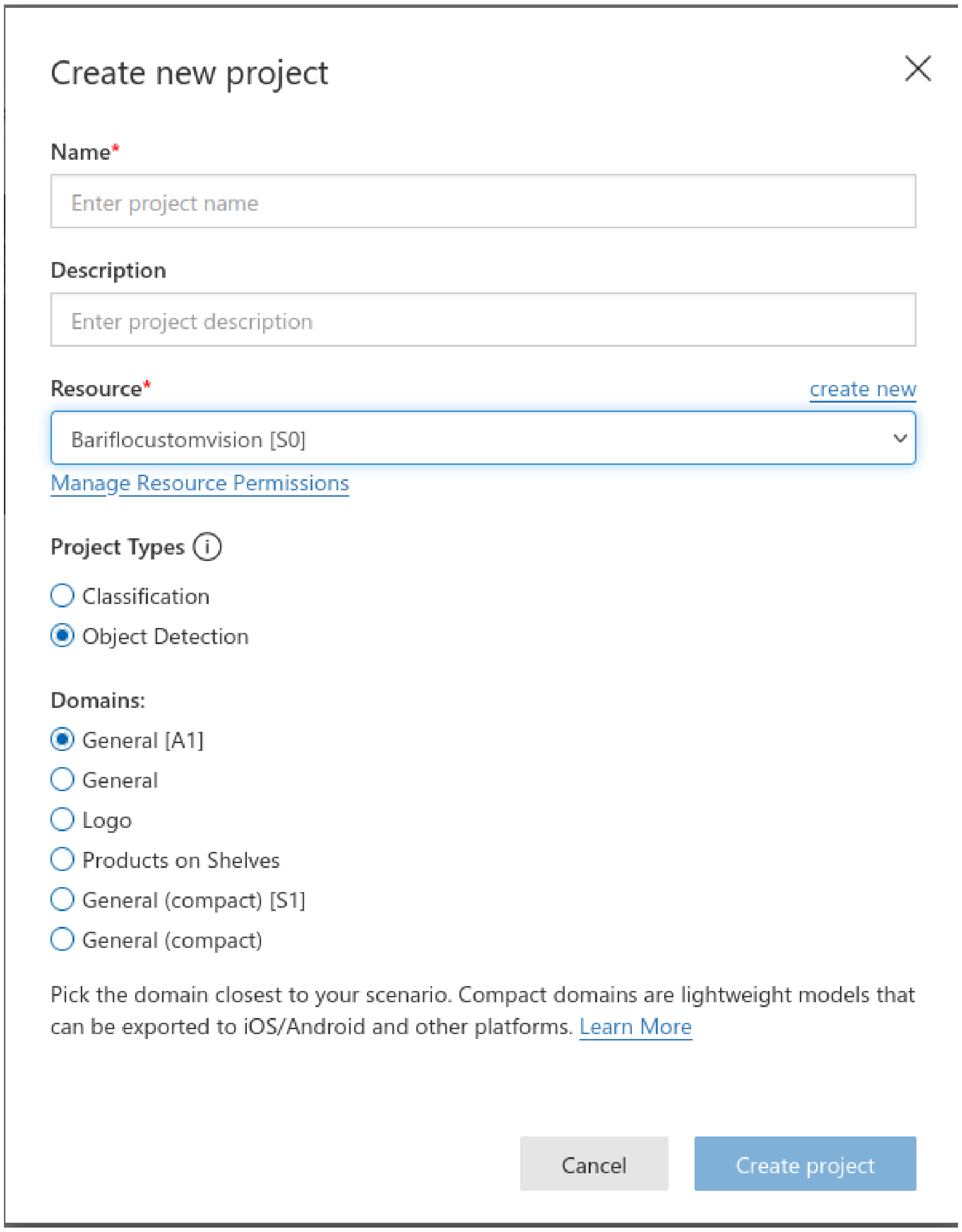
Landmarks (compact)

Retail (compact)

Pick the domain closest to your scenario. Compact domains are lightweight models that can be exported to iOS/Android and other platforms. Learn More

Fig 2: Create new project dialog box

2. Enter a **Name** and a description for the project. Then select your Custom Vision Training **Resource**. If your signed-in account is associated with an Azure account, the Resource dropdown displays all of your compatible Azure resources.


Create new project

Name*

Enter project name

Description

Enter project description

Resource* create new

Bariflocustomvision [S0]

Manage Resource Permissions

Project Types

- Classification
- [✓ Object Detection]

Domains:

- [✓ General [A1]]
- General
- Logo
- Products on Shelves
- General (compact) [S1]
- General (compact)

Pick the domain closest to your scenario. Compact domains are lightweight models that can be exported to iOS/Android and other platforms. Learn More

Fig 3: Selection of object detection

3. Select the **Object Detection** under **Project Types**.
4. In case of **Domains** select **General[A1]**.
5. Finally, select on **Create Project**.

UPLOAD THE DATASET IMAGES AND TAGGING

1. Click on the project , and then click on **Add images** and then select **Browse local files**. Select **Open** to upload the images.

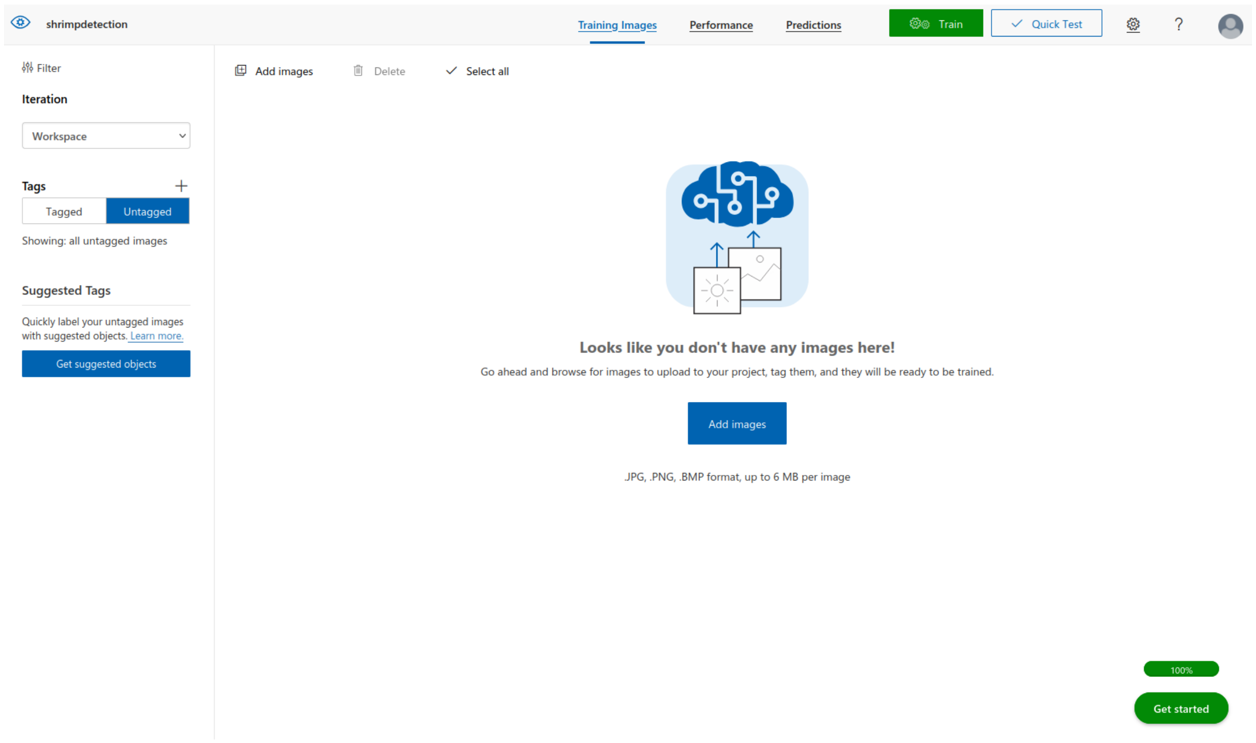
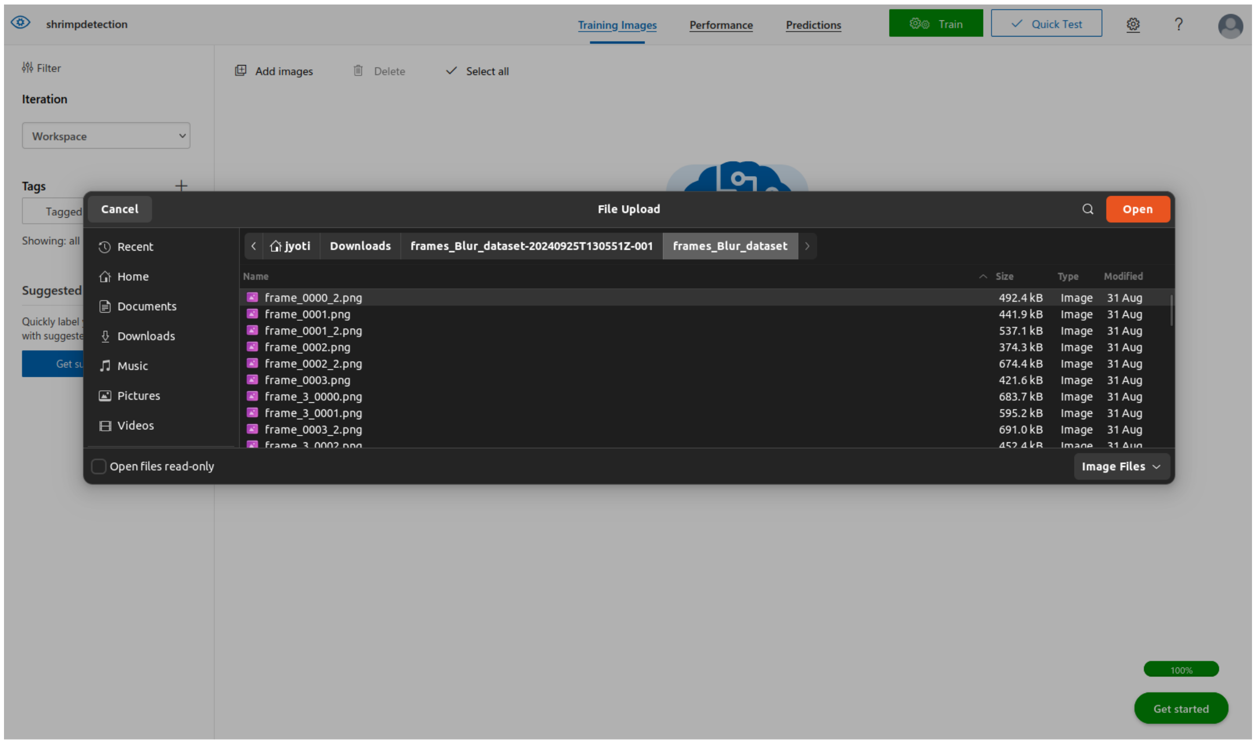

shrimpdetection
Training Images
Performance
Predictions
Train
Quick Test
Filter
Iteration
Workspace
Tags
Tagged
Untagged
Showing: all untagged images
Suggested Tags
Quickly label your untagged images with suggested objects. Learn more.
Get suggested objects
Add images
Delete
Select all
Looks like you don't have any images here!
Go ahead and browse for images to upload to your project, tag them, and they will be ready to be trained.
Add images
.JPG, .PNG, .BMP format, up to 6 MB per image
100%
Get started

Fig 4: Interface to Add the image in the dataset



Name	Size	Type	Modified
frame_0000_2.png	492.4 kB	Image	31 Aug
frame_0001.png	441.9 kB	Image	31 Aug
frame_0001_2.png	537.1 kB	Image	31 Aug
frame_0002.png	374.3 kB	Image	31 Aug
frame_0002_2.png	674.4 kB	Image	31 Aug
frame_0003.png	421.6 kB	Image	31 Aug
frame_3_0000.png	683.7 kB	Image	31 Aug
frame_3_0001.png	595.2 kB	Image	31 Aug
frame_0003_2.png	691.0 kB	Image	31 Aug
frame_3_0002.png	452.4 kB	Image	31 Aug

Fig 5: On clicking on Add images

2. After clicking on **Open** then a interface will be opened , as shown in below Fig6.

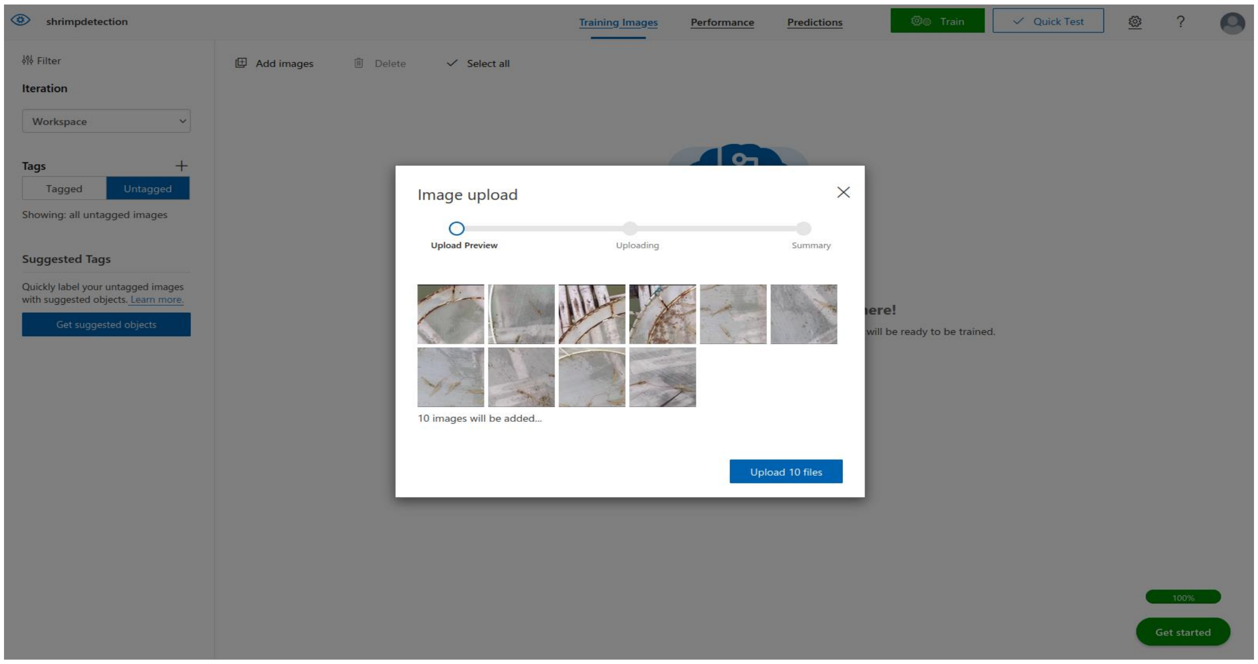
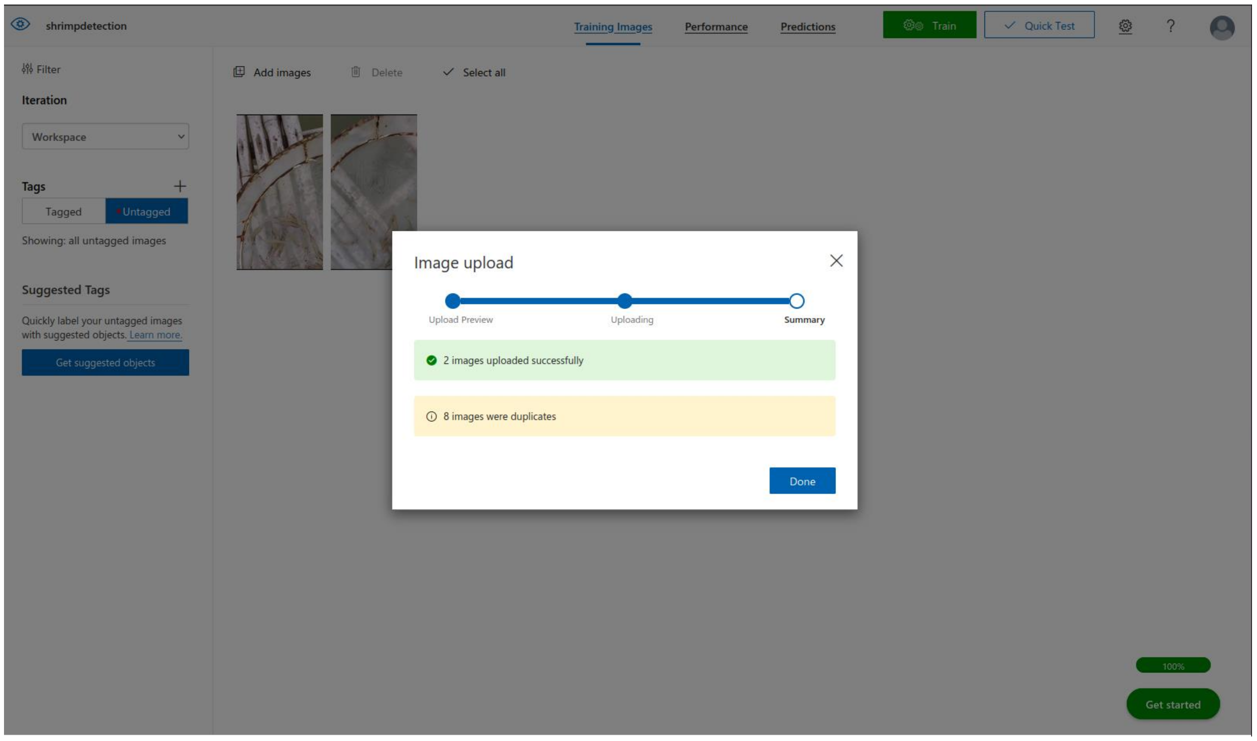


Fig 6: Interface on clicking on Open

3. Click on **Upload** button.

4. After clicking on **Upload** button then click on **Done.** Then the interface is appeared.



2 images uploaded successfully

8 images were duplicates

Fig 7: Interface on clicking on upload button

5. After clicking on **Done** then it is in the section of **untagged** image.

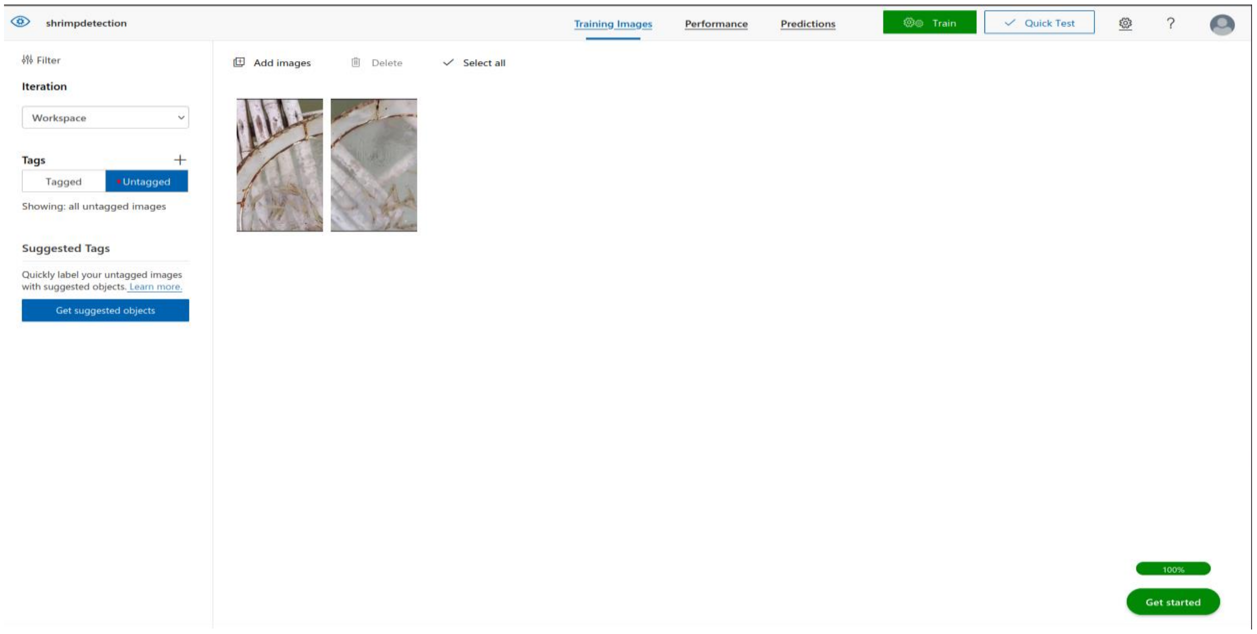


Fig 8: Interface of the untagged image

6. Then click on each image and tag the object with the name. As shown in Fig 9.

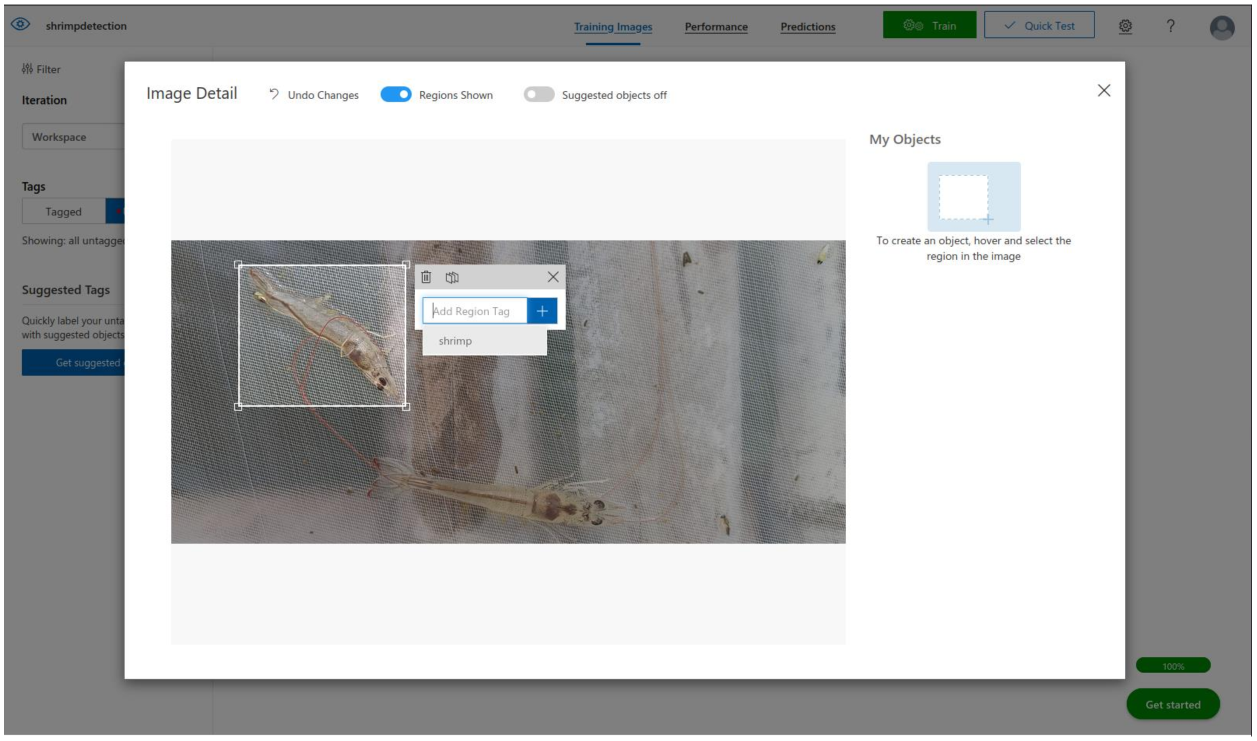


Fig 9: Images tagged with the suitable name

7. After tagging all the images then all the images will be shift to **Tagged** and make sure that there is no images at **Untagged**. Images are shift to tagged as shown in figure Fig 10.

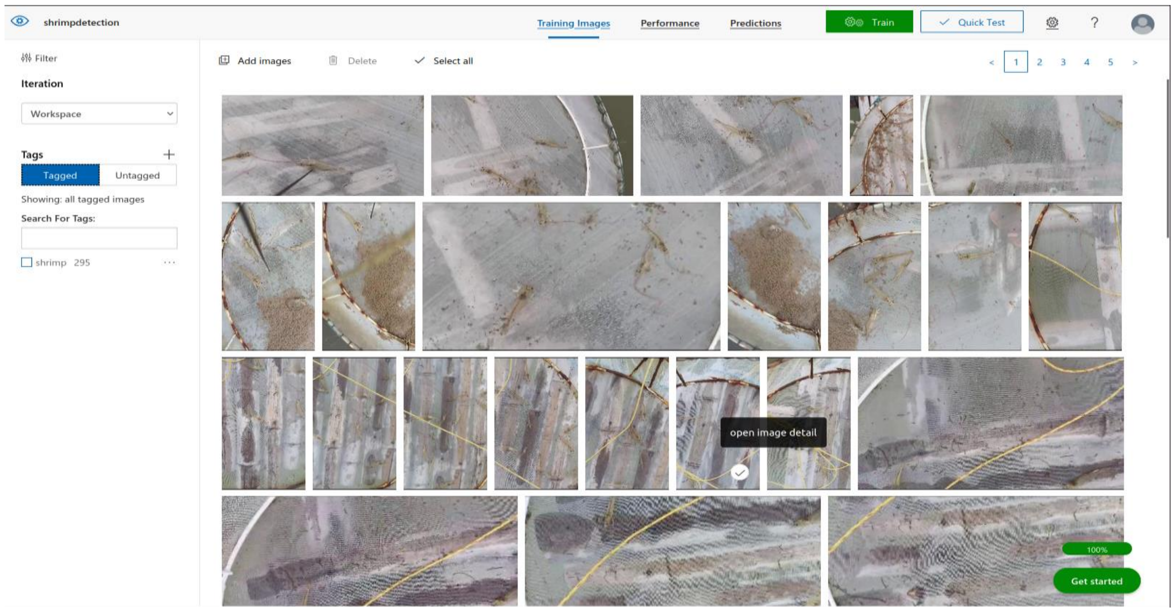
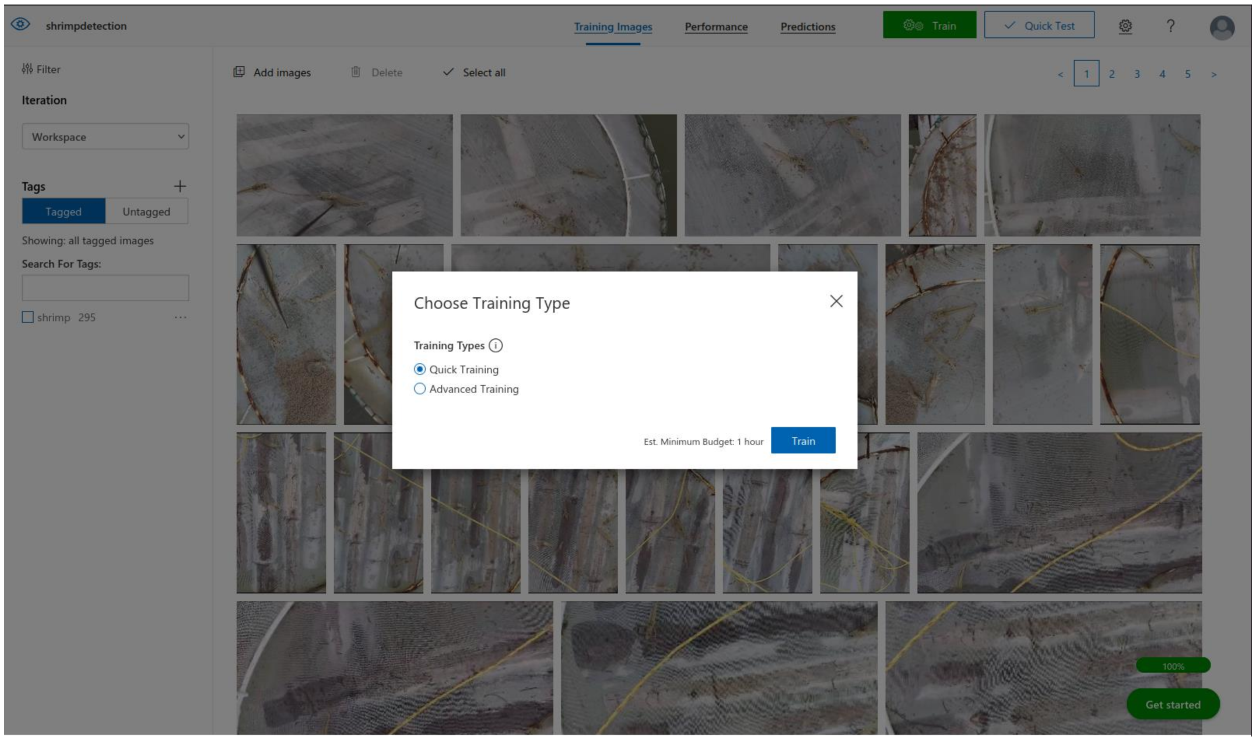


Fig 10: Images stored in tagged

TRAINING OF THE MODEL

After ensuring that all the images are tagged then click on **Train** button. Then select the training types i.e. **Quick Training** and **Advanced Training .** As shown in figure Fig 11:



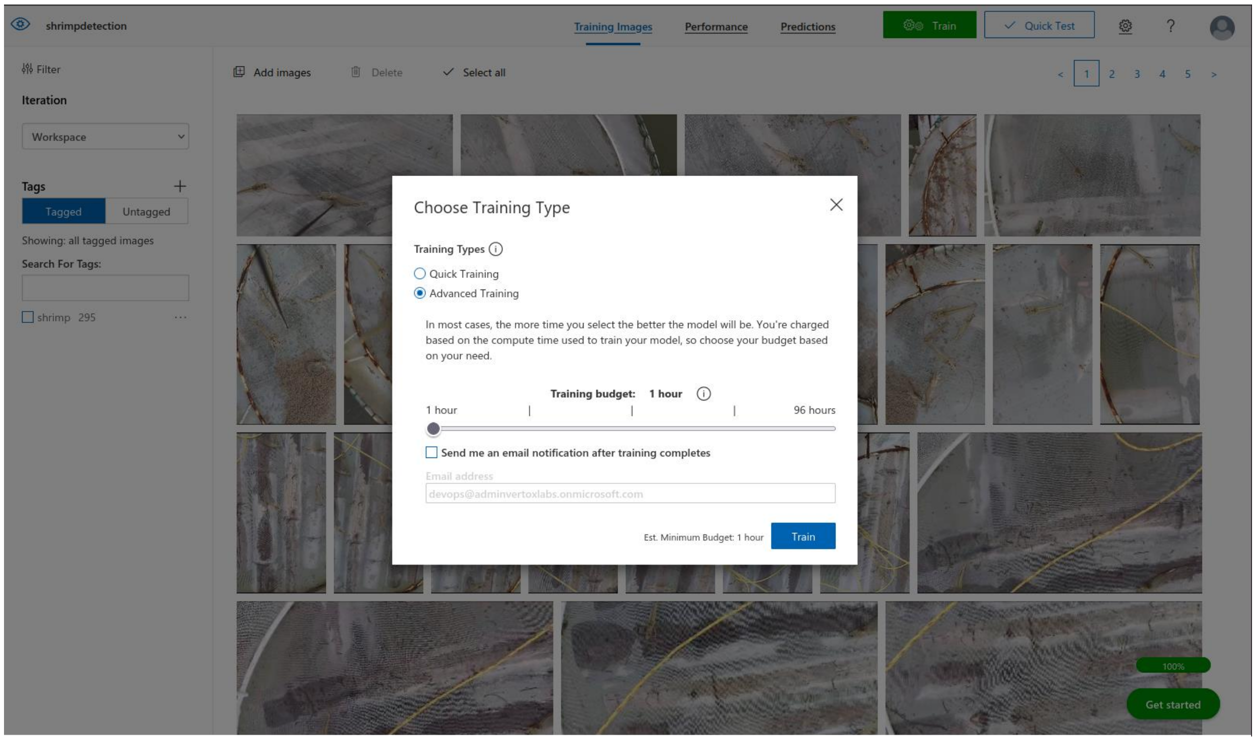
Choose Training Type

Training Types
● Quick Training
○ Advanced Training

Est. Minimum Budget: 1 hour
Train

Fig 11: Interface on clicking on Train button

On clicking on **Quick Training ,** then click on **Train** button **,** It will start training the model. But In case of **Advanced Training ,** we have to set the training hour . And then click on **Train.** As shown in figure Fig 12.



Choose Training Type

Training Types

Quick Training

[✓ Advanced Training]

In most cases, the more time you select the better the model will be. You're charged based on the compute time used to train your model, so choose your budget based on your need.

Training budget: 1 hour

1 hour — 96 hours

Send me an email notification after training completes

Email address: devops@adminvertoxlabs.onmicrosoft.com

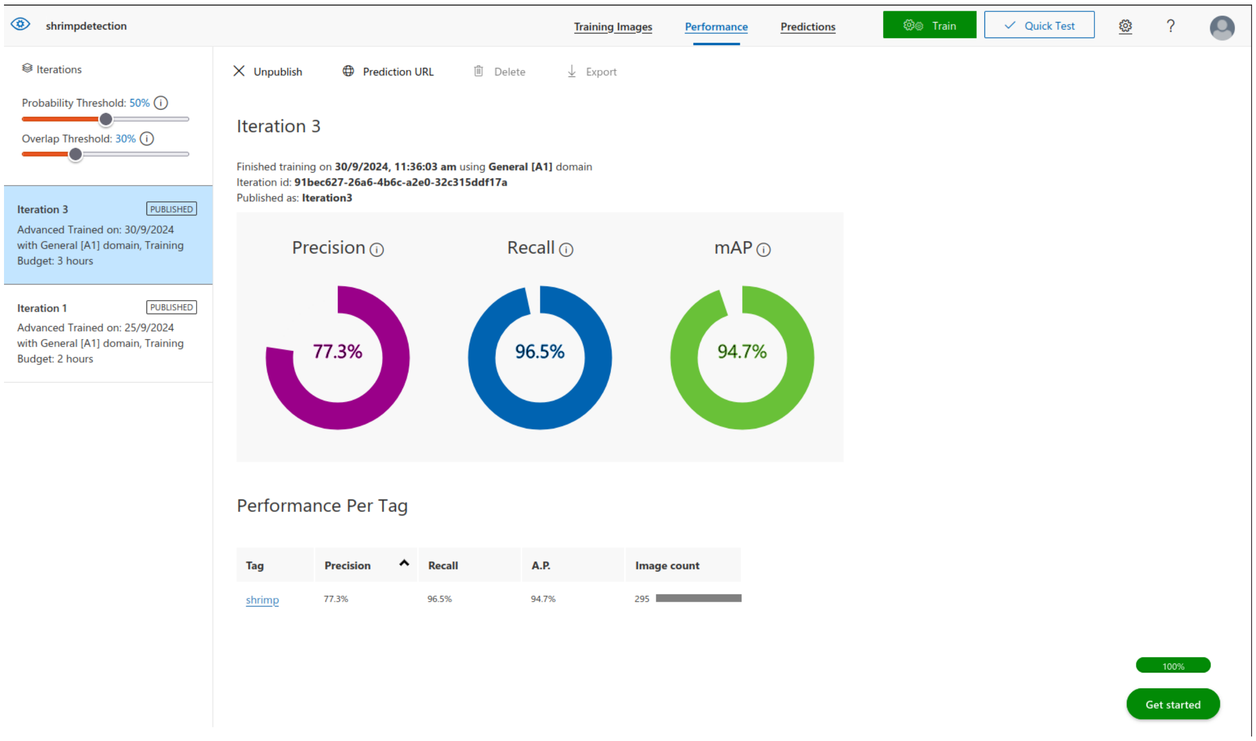
Est. Minimum Budget: 1 hour

Train

Fig 12: Interface of Advance Training

COMPLETION OF TRAINING

After the completion of training then click on **Performance** to have the report how the model is trained. As shown in Fig 13.



Probability Threshold: 50%

Overlap Threshold: 30%

Iteration 3 PUBLISHED
Advanced Trained on: 30/9/2024 with General [A1] domain, Training Budget: 3 hours

Iteration 1 PUBLISHED
Advanced Trained on: 25/9/2024 with General [A1] domain, Training Budget: 2 hours

Iteration 3

Finished training on **30/9/2024, 11:36:03 am** using **General [A1]** domain
Iteration id: **91bec627-26a6-4b6c-a2e0-32c315ddf17a**
Published as: **Iteration3**

Performance Per Tag

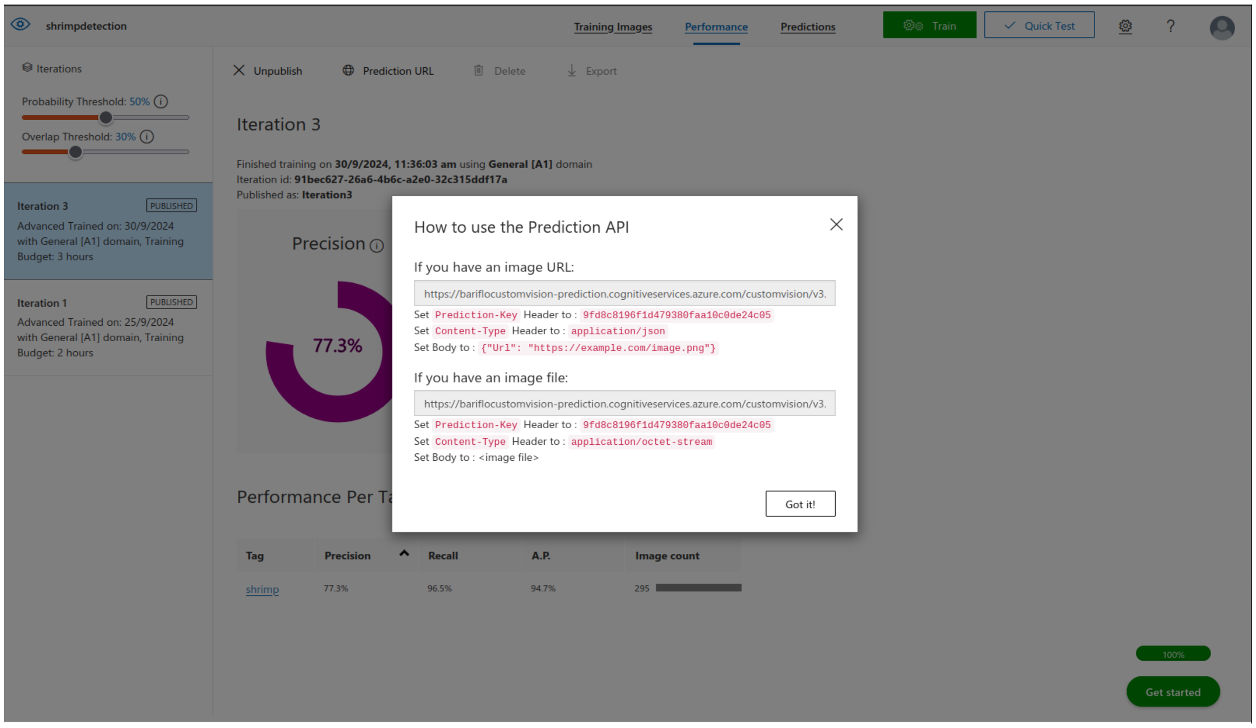
Tag	Precision	Recall	A.P.	Image count
shrimp	77.3%	96.5%	94.7%	295

Fig 13: Report of the model trained

PREDICTION

For the prediction we have to required the prediction URL and the Key. For the Prediction URL , click on the **Publish** button at the top right corner of the

Report , after that click on the **Prediction URL ,** we will be getting the **Prediction URL** and the **Key.** As shown in Fig 14.



Probability Threshold: 50%
Overlap Threshold: 30%

Iteration 3 PUBLISHED
Advanced Trained on: 30/9/2024 with General [A1] domain, Training Budget: 3 hours

Iteration 1 PUBLISHED
Advanced Trained on: 25/9/2024 with General [A1] domain, Training Budget: 2 hours

Iteration 3

Finished training on **30/9/2024, 11:36:03 am** using **General [A1]** domain
Iteration id: **91bec627-26a6-4b6c-a2e0-32c315ddf17a**
Published as: **Iteration3**

How to use the Prediction API

If you have an image URL:

https://bariflocustomvision-prediction.cognitiveservices.azure.com/customvision/v3.
Set `Prediction-Key` Header to : `9fd8c8196f1d479380faa10c0de24c05`
Set `Content-Type` Header to : `application/json`
Set Body to : `{"Url": "https://example.com/image.png"}`

If you have an image file:

https://bariflocustomvision-prediction.cognitiveservices.azure.com/customvision/v3.
Set `Prediction-Key` Header to : `9fd8c8196f1d479380faa10c0de24c05`
Set `Content-Type` Header to : `application/octet-stream`
Set Body to : <image file>

Got it!

Tag	Precision	Recall	A.P.	Image count
shrimp	77.3%	96.5%	94.7%	295

Fig 14: Prediction URL and Key

Take the Prediction URL and Key and use it in the python code for the prediction . If we have the image file then choose the API for **image file** , If we are having the **image url** then using the API for image URL.

PYTHON CODE DOCUMENTATION FOR PREDICTION

Use the Prediction URL and the key in the python code. This script processes a video to detect shrimp using Azure Custom Vision. It extracts frames, enhances them, detects shrimp, and calculates their sizes. The results, including detected shrimp and their sizes, are saved as images and summarized in tables based on their size relative to the average.And the Documentation of the python code is written as below:

Imports and Initial Configurations

```
import cv2
import requests
import os
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from PIL import Image, ImageFilter, ImageEnhance
import math
import random
from prettytable import PrettyTable
from azure.storage.blob import BlobServiceClient
```

- **cv2**: OpenCV library for video and image processing.
- **requests**: For making HTTP requests, such as sending images to the Azure Custom Vision API.
- **os**: Provides functions to interact with the file system (e.g., check directories or file paths).
- **matplotlib.pyplot**: Used for visualizing and saving annotated images.
- **matplotlib.patches**: Enables adding shapes (like rectangles) to images for annotation.
- **PIL (Pillow)**: Provides tools for image manipulation (sharpening, enhancing, etc.).
- **math**: Used for mathematical operations, such as calculating diagonals.
- **random**: Generates random values for bounding box colors.
- **PrettyTable**: Creates formatted tables for displaying shrimp data in the console.

- **BlobServiceClient**: Azure SDK for interacting with Azure Blob Storage.

Azure Configuration

PREDICTION_URL = "https://bariflocustomvision-prediction.cognitiveservices.azure.com/customvision/v3.0/Prediction/300ad940-da23-4789-864d-ee6f8dd69e2d/detect/iterations/Iteration3/image"
KEY = "9fd8c8196f1d479380faa10c0de24c05"

- **PREDICTION_URL**: Azure Custom Vision endpoint to send images for shrimp detection.
- **KEY**: API key required for authentication.
  PIXEL_OF_REFERENCE = 238.65
  REFERENCE_LENGTH_CM = 5.858
  PIXELS_PER_CM = PIXEL_OF_REFERENCE / REFERENCE_LENGTH_CM

- **PIXEL_OF_REFERENCE**: Pixel size of a reference object (known size) in an image.
- **REFERENCE_LENGTH_CM**: Actual length of the reference object in centimeters.
- **PIXELS_PER_CM**: Conversion factor to calculate physical dimensions of objects from pixel values.

Headers for Azure Requests

```
headers = {
    "Content-Type": "application/octet-stream",
    "Prediction-Key": KEY
}
```

- Sets headers for the API request, specifying the content type and including the prediction key for authorization.

Azure Blob Storage Configuration

```
BLOB_SAS_URL =
"https://checktray.blob.core.windows.net/shrimpdata?sp=racwli&st=2024-10-
01T04:51:15Z&se=2024-12-31T12:51:15Z&sv=2022-11-
02&sr=c&sig=Uug2oLDr05I7WAgmzBE91ymnguoadWw6zH8tyv%2BZDy4%3D"
BLOB_SAS_TOKEN = "sp=racwli&st=2024-10-01T04:51:15Z&se=2024-12-
31T12:51:15Z&sv=2022-11-
02&sr=c&sig=Uug2oLDr05I7WAgmzBE91ymnguoadWw6zH8tyv%2BZDy4%3D"
CONTAINER_NAME = "shrimpdata"
```

- **BLOB_SAS_URL**: URL for accessing the Azure Blob Storage container with a shared access signature (SAS).
- **BLOB_SAS_TOKEN**: Token allowing limited access to the container.
- **CONTAINER_NAME**: Name of the Azure Blob Storage container where data is stored.

Function: upload_to_blob

Uploads a file to Azure Blob Storage.

```
def upload_to_blob(file_path, blob_name):
    """Upload a file to Azure Blob Storage."""
    try:
        blob_service_client = BlobServiceClient(account_url=BLOB_SAS_URL,
credential=BLOB_SAS_TOKEN)
        blob_client =
blob_service_client.get_blob_client(container=CONTAINER_NAME,
blob=blob_name)

        with open(file_path, "rb") as data:
            blob_client.upload_blob(data, overwrite=True)

        print(f"File {blob_name} uploaded successfully to Blob Storage.")
    except Exception as e:
        print(f"Error uploading {blob_name} to Blob Storage: {e}")
```

- **file_path**: Path of the file to be uploaded.
- **blob_name**: Name of the file in the blob container.

Function: random_deep_color

Generates random deep colors for bounding box annotations.

```
def random_deep_color():
    """Generate a random deep color in RGB format."""
    return (random.uniform(0, 0.5), random.uniform(0, 0.5), random.uniform(0,
0.5))  # Values closer to 0 for deeper colors
```

- Returns an RGB tuple with values closer to zero (darker colors).

Function: sharpen_image

Enhances the sharpness of an image.

```
def sharpen_image(image):
    """Sharpen the image to reduce blur."""
    return image.filter(ImageFilter.SHARPEN)
```

- Uses ImageFilter.SHARPEN from Pillow.

Function: enhance_image

Adjusts image brightness, contrast, and sharpness.

```
def enhance_image(image, brightness_factor=1.2, contrast_factor=1.5,
sharpness_factor=2.0, detail_filter=True):
    """
    Enhance the input image by adjusting brightness, contrast, sharpness, and
applying a detail filter.
    """
    enhancer = ImageEnhance.Brightness(image)
    image_enhanced = enhancer.enhance(brightness_factor)

    enhancer = ImageEnhance.Contrast(image_enhanced)
    image_enhanced = enhancer.enhance(contrast_factor)

    enhancer = ImageEnhance.Sharpness(image_enhanced)
    image_enhanced = enhancer.enhance(sharpness_factor)

    if detail_filter:
        image_enhanced = image_enhanced.filter(ImageFilter.DETAIL)

    return image_enhanced
```

- Applies multiple enhancements for better object detection.

Function: detect_shrimp

Processes an image for shrimp detection and annotations.

```
def detect_shrimp(image_path, output_image_path):
    """Detect shrimp in the image and annotate it."""
    with open(image_path, "rb") as image_file:
        image_data = image_file.read()


    response = requests.post(PREDICTION_URL, headers=headers,
data=image_data)


    if response.status_code == 200:
        predictions = response.json()["predictions"]
        print("Detected objects:")


        image = Image.open(image_path)
        plt.figure(figsize=(10, 10))
        plt.imshow(image)
        ax = plt.gca()


        shrimp_data = []
        unique_id_counter = 1  # Initialize a counter for unique IDs


        for prediction in predictions:
            probability = prediction['probability']


            if probability > 0.96:
```

```
            bounding_box = prediction['boundingBox']

            print(f"Probability: {probability:.2%}")



            left = bounding_box['left'] * image.width

            top = bounding_box['top'] * image.height

            width = bounding_box['width'] * image.width

            height = bounding_box['height'] * image.height



            diagonal_pixels = math.sqrt(width**2 + height**2)

            diagonal_cm = diagonal_pixels / PIXELS_PER_CM

            print(f"Bounding Box Diagonal: {diagonal_pixels:.2f} pixels,
{diagonal_cm:.2f} cm")



            shrimp_data.append((unique_id_counter, diagonal_cm))

            unique_id_counter += 1



            color = random_deep_color()

            rect = patches.Rectangle((left, top), width, height, linewidth=2,
edgecolor=color, facecolor='none')

            ax.add_patch(rect)



            plt.text(left, top - 10, f"ID: {unique_id_counter - 1}\nProb:
{probability:.2%}\nSize: {diagonal_cm:.2f} cm",

                    color=color, fontsize=12, weight='bold')
```

```
        plt.axis("off")
        plt.savefig(output_image_path, bbox_inches='tight', pad_inches=0)
        print(f"Annotated image saved to: {output_image_path}")
        plt.show()

        print("\nShrimp Sizes:")
        table = PrettyTable()
        table.field_names = ["Unique ID", "Size (cm)"]
        for shrimp in shrimp_data:
            table.add_row(shrimp)
        print(table)

        return shrimp_data  # Return the shrimp data for further processing

    else:
        print(f"Failed to make prediction: {response.status_code}")
        print(response.json())
        return []
```

1. Reads the image and sends it to the Azure Custom Vision endpoint.
2. Parses the response to extract predictions.
3. Annotates detected shrimp with bounding boxes.
4. Calculates shrimp size in centimeters using the PIXELS_PER_CM factor.

5. Saves the annotated image and prints a formatted table of detected shrimp.

Function: process_video

Processes a video, converting it to frames, detecting shrimp, and uploading annotated images.

```
def process_video(video_path, output_folder):
    """Convert video to frames, enhance them, detect shrimp, and upload images to
Azure Blob Storage."""
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)
    else:
        print(f"The directory '{output_folder}' already exists. Proceeding with the
existing directory.")

    cap = cv2.VideoCapture(video_path)
    frame_rate = cap.get(cv2.CAP_PROP_FPS)  # Get the frame rate of the video
    frame_interval = int(2 * frame_rate)    # Set the interval to 2 seconds
    frame_count = 0
    success = True

    all_shrimp_data = []  # List to store all shrimp data across frames

    while success:
        cap.set(cv2.CAP_PROP_POS_FRAMES, frame_count * frame_interval)
        success, frame = cap.read()

        if not success:
            break

        image = Image.fromarray(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
        enhanced_image = sharpen_image(image)
        frame_file_path = os.path.join(output_folder,
f"frame_{frame_count:04d}_6.jpeg")
        enhanced_image.save(frame_file_path)

        shrimp_data = detect_shrimp(frame_file_path,
frame_file_path.replace(".jpeg", "_detected.jpeg"))
        all_shrimp_data.extend(shrimp_data)

        # Upload detected images to Blob Storage
        blob_name = f"frame_{frame_count:04d}_detected.jpeg"
        upload_to_blob(frame_file_path.replace(".jpeg", "_detected.jpeg"),
blob_name)

        frame_count += 1

    cap.release()
    print(f"Processed {frame_count} frames from the video.")
```

1. Extracts frames from the video at intervals based on the frame rate.
2. Enhances each frame and saves it.
3. Detects shrimp in each frame using detect_shrimp.
4. Uploads annotated frames to Azure Blob Storage.

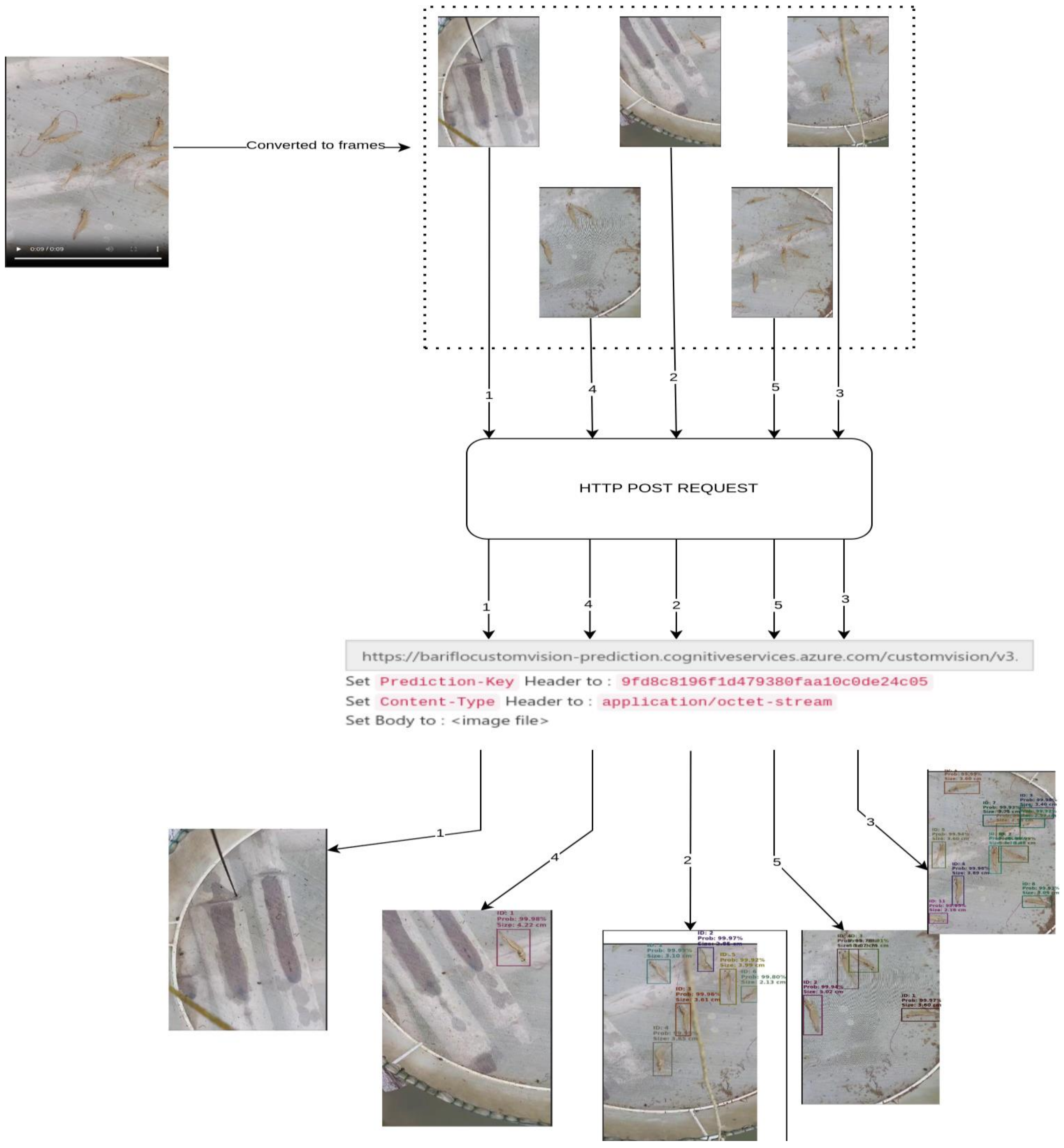
Main Execution

Processes a sample video for shrimp detection.

```
if __name__ == "__main__":
    video_path = "/home/jyoti/Documents/shrimp_vdo_60.mp4"
    output_folder = "/home/jyoti/Documents/Detected_shrimp_test_66"
    process_video(video_path, output_folder)
```

- **video_path**: Path to the input video.
- **output_folder**: Directory to store processed frames and detected images.

WORKING OF THE ALGORITHM



https://bariflocustomvision-prediction.cognitiveservices.azure.com/customvision/v3.

Set `Prediction-Key` Header to : `9fd8c8196f1d479380faa10c0de24c05`

Set `Content-Type` Header to : `application/octet-stream`

Set Body to : <image file>