Documentation for Crab Detection and Video Processing Script

Overview

This Python script processes video files to detect crabs (crablets) in individual frames using Azure Custom Vision. The script enhances frames, detects objects, and annotates them with bounding boxes and probabilities. Additionally, it generates a summary table of the detection results.

Imports

```python
"""
 Author : Tapash Ranjan Nandi
"""
import cv2
import requests
import os
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from PIL import Image, ImageFilter
import random
from prettytable import PrettyTable

```

- **Libraries Used:**
  - cv2: For video frame extraction and manipulation.
  - requests: To send API requests to Azure Custom Vision.
  - os: To handle file and directory operations.

- matplotlib.pyplot and matplotlib.patches: For visualizing and annotating images.
- PIL.Image and ImageFilter: To enhance image quality.
- random: To generate random colors for bounding boxes.
- prettytable: To create a formatted summary table.

Global Constants

```
# Azure Custom Vision Prediction URL and Key
PREDICTION_URL = "https://mudcrab-prediction.cognitiveservices.azure.com/customvision/v3.0/Prediction/35bcb58a-1479-4ece-9559-b9064b17903a/detect/iterations/Iteration1/image"
KEY = "GFKIgNWeRHV1eG7bCM4wnXHNHOc8xzYTzAxIqcV6xAjy6ymaZIV5JQQJ99AKACGhs1BXJ3w3AAAIACOGY695"

```

```
# Headers for the request
headers = {
    "Content-Type": "application/octet-stream",
    "Prediction-Key": KEY
}

```

- **Purpose:**
  - PREDICTION_URL: The endpoint for Azure Custom Vision prediction API.
  - KEY: The API key for authentication.

- headers: Defines the headers for the POST request, including the API key and content type.

Functions Used

random_deep_color()

```python
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))

```

- Generates random deep colors for bounding boxes.
- Ensures distinct and easily visible annotations.

(b) sharpen_image(image)

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

```

- Enhances image clarity by applying a sharpening filter

detect_crab (image_path, output_image_path)

```python
def detect_crab(image_path, output_image_path):
    """Detect crab 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)

    crab_count = 0  # Initialize the count of crablets
    if response.status_code == 200:
        predictions = response.json()["predictions"]

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

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

            if probability > 0.90:
                crab_count += 1  # Increment the count of crablets
                bounding_box = prediction['boundingBox']
                left = bounding_box['left'] * image.width
                top = bounding_box['top'] * image.height
                width = bounding_box['width'] * image.width
                height = bounding_box['height'] * image.height

                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"Prob: {probability:.2%}",
                         color=color, fontsize=12, weight='bold')

        plt.axis("off")
        plt.savefig(output_image_path, bbox_inches='tight', pad_inches=0)
        plt.close()
    else:
        print(f"Failed to make prediction: {response.status_code}")
        print(response.json())

    return crab_count  # Return the count of crablets
```

- **Inputs:**
  - image_path: Path to the input image.
  - output_image_path: Path to save the annotated image.
- **Outputs:**
  - Returns the number of crablets detected.

- **Key Operations:**
  - Reads the image and sends it to the Azure Custom Vision API.
  - Parses the API response to identify bounding boxes for crablets with a probability > 90%.
  - Draws bounding boxes and annotations on the image.
  - Saves the annotated image.

process_video (video_path, output_folder)

```python
def process_video(video_path, output_folder):
    """Convert video to frames, enhance them, and detect crab."""
    # Check if the output folder exists; if not, create it
    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

    # Create a table to store frame names and number of crablets
    table = PrettyTable()
    table.field_names = ["Frame Name", "Number of Crablets"]

    while success:
        # Set the frame position to extract one frame every 2 seconds
        cap.set(cv2.CAP_PROP_POS_FRAMES, frame_count * frame_interval)
        success, frame = cap.read()

        if not success:
            break

        # Convert the frame to a PIL Image for enhancement
        image = Image.fromarray(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
        enhanced_image = sharpen_image(image)
        # Save the enhanced frame
        frame_file_path = os.path.join(output_folder, f"frame_{frame_count:04d}.jpeg")
        enhanced_image.save(frame_file_path)

        # Detect crab in the enhanced image and annotate it
        detected_image_path = frame_file_path.replace(".jpeg", "_detected.jpeg")
        crab_count = detect_crab(frame_file_path, detected_image_path)

        # Add the frame name and crab count to the table
        table.add_row([os.path.basename(frame_file_path), crab_count])

        frame_count += 1

    cap.release()
    print(f"Processed {frame_count} frames from the video.")
    print("\nSummary of Crablet Detection:")
    print(table)  # Print the table

```

- **Inputs:**
  - video_path: Path to the input video file.
  - output_folder: Directory to save extracted frames and annotated images.
- **Outputs:**
  - Saves frames and annotated images in the output folder.
  - Prints a summary table of crablet detection.
- **Key Operations:**

- Extracts frames at 2-second intervals using OpenCV.
- Enhances frames using the sharpen_image function.
- Detects crablets in each frame using the detect_crab function.
- Saves the results and logs them in a formatted table.

Main Execution Block

```python
if __name__ == "__main__":
    # Replace with the path to your video
    video_path = "/home/jyoti/Documents/mudcrab_detection/crablet_10.mp4"
    output_folder = "/home/jyoti/Documents/mudcrab_detection/Detected_crab_test_1"

    process_video(video_path, output_folder)
```

- **Purpose:**
  - Specifies the input video and output directory.
  - Calls the process_video function to start the detection process.

Expected Outputs

- Annotated frames are saved in the specified output_folder.
- A summary table is displayed in the console, showing the number of crablets detected in each frame.