Cichlid Computer Vision Project – Weekly Progress

Week ending Friday, January 31st, 2025

Time Logs

Charlie Clark

What progress did you make?

- **Attended weekly BioBoost meeting on Monday evening.**
- **Attended weekly publication seminar Tuesday evening.**
- **Attended weekly HAAG admin meeting Thursday afternoon.**
- **Attended bi-weekly computational advisor meeting with the Bird Audio team.**
- **Read through the abstracts/introductions for a set of papers which might be related to our work on BioBoost; extracted the problem statement as well as aspects that apply and don't apply to our work, for each paper.**
- **Wrote up and sent out a Meeting Recording Procedures document, outlining the roles and responsibilities of all relevant parties when it comes to the handling of meeting recordings and notes.**
- **In the process of collecting gmail accounts from all the meeting managers and higher ed project managers so I can add them as managers for the YouTube channel.**

What are you planning on working on next?

- **Continue working on BioBoost literature review.**
- **Continue helping Kailey and Eric with the BioBoost code (if needed).**
- **Ensure all required parties are added as YouTube channel managers.**
- **Start enforcing the new Meeting Recording Procedures.**
    - **Confirm my understanding about whether they apply to the Alexander groups as well.**
- **Attend all required (and some optional) meetings.**

Is anything blocking you from getting work done?

- **None.**

Kailey Quesada

What progress did you make?

- I attended the following 3 meetings:
    - Cichlid Team Working Meeting on January 25th. YOLO and SORT logistics discussed.
    - Cichlid Team Meeting on January 27th. Went over SORT and paper re-write.
    - Publication Seminar Meeting on January 28th. Went over paper re-write.
- I worked on the meeting manager role requirements for both meetings. I created the meeting tasks in Microsoft planner, created the required materials (slides, transcripts, notes, recordings, attendance), and updated the attendance sheet with meeting links in the project files section on Teams.
- I was able to take the SORT code from Bree and change the script to be able to run the modified SortFish on the new dataset. We now have the required velocity vectors for TemporalBoost. I then ran code to combine SORT csvs, filter for tracks above a certain frame threshold, check for non-consecutive tracks, and tally track information about missing frames.
- I was able to take some previous code from Bree for visualizing YOLO and SORT bounding boxes with OpenCV and modify it for the Lindenthal dataset. This took a decent amount of time, since OpenCV was pretty sensitive to file paths, etc. I generated 100 videos and shared them with the BioBoost chat for review.
- I investigated some problems with the BioBoost pipeline. I wrote a script to save off video frames and then matched my combined SORT csv with the video frames manually to inspect missing frames. It doesn't appear that YOLO missing detections is the cause of the missing frames. I identified another problem, which is that there exists at least one long track with high confidence values that does not depict any animal, just shadows. Additionally, I explored the number of tracks with missing frames with scripts and created a histogram.

What are you planning on working on next?

- I need to help make changes to the SORT code to improve SORT performance based on the direction suggested by Bree in the next meeting.
- I need to help with the paper rewrite.
- I need to fulfill my meeting manager responsibilities and attend required meetings.

Is anything blocking you from getting work done?

- Yes. I still need an email from Dr. Lytle for CS 8903. I also think that it would be helpful having more people than just 1 working on the code portion of BioBoost. This will make things more efficient.

Eric Iamarino

What progress did you make?

- Cleaned up BioBoost GitHub
    - Added docstrings to all the functions for all the scripts
    - Added type hints to the arguments of all functions
    - Organized scripts into folders with more meaningful names
    - Adding better guidance to README.md to GitHub (In Progress)
- Got access to Cichlid CV website
    - Adding meetings and reports for Spring 2025 semester (In Progress)
- Attended weekly BioBoost meeting

What are you planning on working on next?

- Attend BioBoost weekly meeting
- Attend Publication weekly meeting
- Help with BioBoost Introduction rewrite
- Help improve tracking of SORT
- Help with conversion of .bag files to .mp4 videos

Is anything blocking you from getting work done?

- **None.**

Abstracts

Charlie Clark

“YOLO-SAG: An improved wildlife object detection algorithm based on YOLOv8n”, Chen et al. (2025; Ecological Informatics0

- Abstract: “Wildlife conservation is crucial for maintaining biodiversity, ensuring ecosystem balance and stability, and fostering sustainable development. Currently, the use of infrared camera traps to monitor and capture photos of wildlife is a vital methodology in protecting and researching wildlife, and automatic detection and identification of animals within captured photographs are paramount. However, factors such as the complexity of the field environment and the varying sizes of animal targets lead to low detection accuracy, while high-precision detection models are hindered by high computational complexity and sluggish training speeds. This paper proposes a wildlife target detection algorithm based on improved YOLOv8n - YOLO-SAG, which aims to balance accuracy and speed. Training stability is enhanced by introducing the Softplus activation function, which increases detection accuracy; incorporating the AIFI enhances intra-scale feature interaction, reducing missed and false detections. Integrating the GSConv and VoV-GSCSP module lightens neck convolutions, reducing computational redundancy and balancing the computational and parametric quantities brought by the AIFI. Experimental results on a self-made wildlife dataset indicate that the YOLO-SAG achieves 94.9%, 90.9%, 96.8%, and 79.9% in Precision, Recall, mAP@0.5, and mAP@0.5–0.95, respectively, which are 3.4%, 3.3%, 3.2%, and 4.9% higher than the original YOLOv8n. Inference and post-processing times reach 1.2 ms and 0.5 ms, a speedup of 25% and 54.5%, respectively, and the computation volume is only 7.2 GFLOPs, an 11.1% decrease.”
- AI Summary by ChatGPT (4o): “This paper presents YOLO-SAG, an improved version of YOLOv8n specifically designed for wildlife object detection. It aims to balance

detection accuracy and speed, addressing challenges posed by complex environments, varying animal sizes, and high computational costs. The model incorporates Softplus activation, an Attention-based Intra-scale Feature Interaction (AIFI) module, and SlimNeck architecture to enhance feature extraction, improve detection stability, and optimize computational efficiency. Experiments on a custom wildlife dataset show that YOLOSAG outperforms YOLOv8n, achieving higher precision, recall, and mAP, while also reducing inference time and computational load."

- Key Contributions:
  - "Softplus Activation for Stability & Accuracy: Replaces SiLU with Softplus in YOLOv8n, improving training stability, reducing gradient vanishing, and increasing precision/recall."
  - "AIFI for Feature Enhancement: Introduces an Attention-based Intra-scale Feature Interaction (AIFI) module, which enhances feature interactions within the same scale, reducing false detections."
  - "Slim-Neck for Computational Efficiency: Replaces the standard YOLO Neck with a Slim-Neck design, incorporating GSConv and VoV-GSCSP to reduce computational redundancy while maintaining accuracy."
  - "State-of-the-Art Performance: YOLO-SAG achieves 94.9% precision, 90.9% recall, and 96.8% mAP@0.5, outperforming YOLOv8n while reducing inference time by 25%."
- Contributions to Knowledge:
  - "Improved Real-Time Wildlife Detection: YOLO-SAG is optimized for real-time wildlife detection, offering a better balance between speed and accuracy than existing YOLO models."
  - "Enhanced Feature Representation: The AIFI module improves feature extraction by reducing false positives and missed detections, making it more robust in occlusion and low-contrast conditions."
  - "Efficient Object Detection for Resource-Limited Environments: The Slim-Neck design reduces computational costs by 11.1%, making YOLOSAG suitable for edge computing applications like camera traps and drone-based monitoring."
- Future Research Directions:
  - "Further Lightweight Optimization: While the model reduces computational costs, additional compression techniques (e.g., knowledge distillation, pruning) could make it even more efficient."

        - “Generalization to Other Ecological Domains: Applying YOLO-SAG to marine life detection, bird monitoring, or small mammal tracking could expand its ecological impact.”
        - “Integration with Multimodal Data: Future work could incorporate infrared, LiDAR, or acoustic data to improve performance in low-light or dense forest environments.”
        - “Automated Behavior Recognition: Extending YOLO-SAG for wildlife behavior analysis (e.g., feeding
- Link: https://linkinghub.elsevier.com/retrieve/pii/S1574954124003339.

Kailey Quesada

Kuswantori, A., et al. “Fish Detection and Classification for Automatic Sorting System with an Optimized YOLO Algorithm.” Applied Sciences, 2023. https://www.mdpi.com/2076-3417/13/6/3812.

This research uses deep learning and computer vision to automatically recognize and classify fish, which is important for improving efficiency in the fish industry. The researchers created a unique dataset of eight fish species, including videos of the fish moving on a conveyor belt, and applied the YOLOv4 algorithm to achieve high accuracy in detection. This work aims to address challenges in sorting fish automatically, which is currently done manually, and is particularly relevant given the increasing global food demand. The proposed method in this study is evaluated against the latest techniques in fish recognition to highlight its key contributions. Over the past few years, at least six other studies have focused on recognizing swimming fish. The findings show that this new method, which uses straightforward and effective algorithms, achieves the highest average recognition accuracy and is particularly useful for fish sorting systems in the aquaculture industry. No future work is discussed.

Eric Iamarino

Gheorghe, C., Duguleana, M., Boboc, R. G., & Postelnicu, C. C. (2024, October 31). Analyzing real-time object detection with YOLO algorithm in Automotive Applications: A Review. Tech Science Press. https://doi.org/10.32604/cmes.2024.054735

Identifying objects in real-time is a technology that is developing rapidly and has a huge potential for expansion in many technical fields. Currently, systems that use image processing to detect objects are based on the information from a single frame. A video camera positioned in the analyzed area captures the image, monitoring in detail the changes that occur between frames. The You Only Look Once (YOLO) algorithm is a model for detecting objects in images, that is currently known for the accuracy of the data obtained and the fast-working speed. This study proposes a comprehensive literature review of YOLO research, as well as a bibliometric analysis to map the trends in the automotive field from 2020 to 2024. Object detection applications using YOLO were categorized into three primary domains: road traffic, autonomous vehicle development, and industrial settings. A detailed analysis was conducted for each domain, providing quantitative insights into existing implementations. Among the various YOLO architectures evaluated (v2–v8, H, X, R, C), YOLO v8 demonstrated superior performance with a mean Average Precision (mAP) of 0.99.

Documentation of Work

Charlie Clark

- Attended weekly BioBoost meeting on Monday evening.
- Attended weekly publication seminar Tuesday evening.
- Attended weekly HAAG admin meeting Thursday afternoon.
- Attended bi-weekly computational advisor meeting with the Bird Audio team.
- Read through the abstracts/introductions for a set of papers which might be related to our work on BioBoost; extracted the problem statement as well as aspects that apply and don't apply to our work, for each paper.
- Wrote up and sent out a Meeting Recording Procedures document, outlining the roles and responsibilities of all relevant parties when it comes to the handling of meeting recordings and notes.
- In the process of collecting gmail accounts from all the meeting managers and higher ed project managers so I can add them as managers for the YouTube channel.

- Proof:
    - Literature review document: Rewrite-References
    - Meeting recording procedure: Meeting_Recording_Procedures
- Results Visualization: Nothing to visualize this week on my end (mostly reading and meetings).

Kailey Quesada

For a full list of what I did, see the time log above. The most important things I worked on this week was re-running SORT and modifying existing code for doing visualization of YOLO and SORT bounding boxes with OpenCV.

The important snippets of my code this week are below. First of all, we want to modify the SortFish script created by Bree and Tucker to be able to run it on the new dataset in Jupyter Notebook. We want to change the width and height constants, remove the scripting args, and add Python code to run from a notebook. It took me a couple hours to understand SORT and to get this to run. I also realized that a lot of the code that Charlie and I had written was redundant with the code Tucker and Bree had already written. We should take note of this when assembling the GitHub for other people to use it.

```
...
IMG_W = 1280
IMG_H = 720
...
# Custom Class
class SortFish:
    def __init__(self, infile_dir, detections_file, tracks_file, base_name):
        '''Start Update'''
        self.infile_list = [os.path.join(infile_dir, x) for x in os.listdir(infile_dir)]
        '''End Update'''
        self.detections_file = detections_file
        self.tracks_file = tracks_file
        self.detections_fp = open(detections_file, 'w')
        self.tracks_fp = open(tracks_file, 'w')
        self.base_name = base_name # added in v2
        self.tracks_fp.write('base_name,track_id,frame,xc,yc,w,h,class_id,u_dot,v_dot,s_dot,p_value\n')
        self.detections_fp.write('base_name,frame,x1,x2,y1,y2,p-value,class,tracked\n')

    # Converts YOLO-formatted Object Detections to SORT-formatted Bounding Boxes
    def yolodet_to_sortdet(self, det):
        """converts a detection of the form [class, x_center, y_center, width, height, score] to the form
        [x1, x2, y1, y2, score, class]
        """
        det = [float(d) for d in det]
        scaled_det = [det[0], det[1] * IMG_W, det[2] * IMG_H, det[3] * IMG_W, det[4] * IMG_H, det[5]]
        return np.array([scaled_det[1] - scaled_det[3] / 2,
                         scaled_det[2] - scaled_det[4] / 2,
                         scaled_det[1] + scaled_det[3] / 2,
                         scaled_det[2] + scaled_det[4] / 2,
                         scaled_det[5], scaled_det[0]])

    # Writes Tracker Data to Output File
    def update_outfile(self, trackers, frame):
        """
        adds new rows to the output file
```

```
        :param trackers: nx5 array, where n is equal to the number of active tracks. This object is returned by Sort.update
        :type trackers: np.ndarray
        :param yolodets: list of the original detections, in yolo format (class, xc, yc, w, h, score)
        :type yolodets: list(list(float | int))
        :param frame: current frame number
        :type frame: int
        :param file_obj: file object that we are writing to
        :type file_obj: TextIOWrapper or equivalent open stream
        """
        for t in trackers:
            xc = ((t[0] + t[2]) / 2) / IMG_W
            yc = ((t[1] + t[3]) / 2) / IMG_H
            w = (t[2] - t[0]) / IMG_W
            h = (t[3] - t[1]) / IMG_H
            class_id = t[4]
            u_dot, v_dot, s_dot = t[5], t[6], t[7]
            p_value = t[8]
            track_id = t[9]
            print(f'{self.base_name}, {track_id}, {frame}, {xc}, {yc}, {w}, {h}, {class_id}, {u_dot}, {v_dot}, {s_dot}, {p_value}', file=self.tracks_fp)

    # Run Sort and Process Frames, Saving Tracking and Detection Data
    def run_sort(self, min_track_len=0, max_age=5, min_hits=3):

        tracker = Sort(max_age=max_age, min_hits=min_hits)
        curr_frame = 0

        for detection in self.infile_list:
            frame = int(detection.split('_')[-1].split('.')[0])
            with open(detection) as f:

                while frame > curr_frame:
                    trackers = tracker.update()
                    self.update_outfile(trackers, curr_frame)
                    curr_frame += 1

                yolodets = [[float(val) for val in l.strip().split(' ')] for l in f.readlines()]
                sortdets = np.array([self.yolodet_to_sortdet(yd) for yd in yolodets])
                trackers = tracker.update(sortdets)

                for i,sdet in enumerate(sortdets):
                    if i in tracker.matched:
                        print(','.join([self.base_name] + [str(frame)] + [str(x) for x in sdet] + ['1']), file = self.detections_fp)
                    elif i in tracker.unmatched:
                        print(','.join([self.base_name] + [str(frame)] + [str(x) for x in sdet] + ['0']), file = self.detections_fp)
                    else:
                        pdb.set_trace()
                if trackers.any():
                    pass
                    #print(trackers)
                self.update_outfile(trackers, curr_frame)
                if not curr_frame % 100:
                    pass
                    #print(f'frame {curr_frame} processed')
                curr_frame += 1

        self.detections_fp.close()
        self.tracks_fp.close()

        df = pd.read_csv(open(self.tracks_file), index_col=1)
        track_lengths = df.groupby('track_id').count()['frame']
        t = track_lengths.groupby(track_lengths.values).count()

```

```
        df = df[df.index.value_counts(sort=False) > min_track_len]
        idx = df.index.to_numpy().astype(int)
        for i, j in enumerate(np.unique(idx)):
            idx[idx == j] = i
        df.index = idx
        df.index.rename('track_id', inplace=True)
        df = df.sort_values(by=['track_id', 'frame'])
        dt = pd.read_csv(self.detections_file)

        df.to_csv(self.tracks_file)
...
for file_folder_name in folder_names:
    base_name = file_folder_name
    infile_dir = rf"charlie-dropbox-data\lindenthal-yolo-and-sort-predictions\{
    file_folder_name}\detections\labels"
    detections_file = rf"charlie-dropbox-data\lindenthal-yolo-and-sort-predictions\tracks-
    sortfish\{file_folder_name}_detects.csv"
    tracks_file = rf"charlie-dropbox-data\lindenthal-yolo-and-sort-predictions\tracks-
    sortfish\{file_folder_name}_tracks.csv"
    sort_obj = SortFish(infile_dir, detections_file, tracks_file, base_name)
    sort_obj.run_sort()
```

This code gives us the required velocity vectors for TemporalBoost. Next, I ran code to combine SORT csvs, filter for tracks above a certain frame threshold, check for non-consecutive tracks, and tally track information about missing frames. The code for combining SORT csvs and filtering tracks was similar to the code that I presented last week.

Next, I modified existing code from Bree for doing visualization of YOLO and SORT bounding boxes with OpenCV. The important sections are below. The overall idea is that we add a column for both border thickness and color and then use OpenCV to draw the boxes. Pretty simple, but the code can be pretty confusing at first.

```
...
# This class is a simplified version adapted from Bree's code. We want YOLO boxes to be
    black and SORT boxes to alternate colors based on Track_ID.
class ColorDF:
    COLORS = {
        'YOLO': (0, 0, 0),  # Black for YOLO
    }

    rgb_colors = [
        (255, 0, 0),   # Red
        (0, 255, 0),   # Green
        (0, 0, 255),   # Blue
        (255, 255, 0), # Yellow
        (255, 0, 255), # Magenta
        (0, 255, 255), # Cyan
        (128, 0, 0),   # Maroon
        (0, 128, 0),   # Green (dark)
        (0, 0, 128),   # Navy
        (128, 128, 128) # Gray
    ]

    def r_class(self, df, detection_type):
        """
        Assign colors based on the detection type (e.g., YOLO or SORT).
        """
        df['color'] = [self.COLORS[detection_type]] * len(df.index)
        return df

    def r_unique(self, df, name):
        """
        Assign colors based on unique column elements like Track_ID.
        """
        num_groups = df[name].nunique()
```

```
        color_indices = np.tile(np.arange(len(self.rgb_colors)), int(np.ceil(num_groups / len(self.rgb_colors))))[:num_groups]
        df['color'] = df[name].map(dict(zip(df[name].unique(), np.array(self.rgb_colors)[color_indices])))
        df['color']= df['color'].apply(lambda x: tuple(int(y) for y in x))
        return df
...
# We need to define class names for the labels.
class_names = {
    0: "deer",
    1: "goat",
    2: "donkey",
    3: "goose"
}
...
# We filter each file's detections using file name.
yolo_df = yolo_df[yolo_df['File_Name'] == video_name]
sort_df = sort_df[sort_df['File_Name'] == video_name]
...
# We draw the bounding boxes found by SORT.
# Frame numbers in the data start at 1. This assumes frame_index = 0 at the start.
sort_frame_data = sort_df[sort_df['Frame'] == frame_index + 1]
        for _, row in sort_frame_data.iterrows():
            top_left_x = int(row['X_Center'] * frame_width - row['Width'] * frame_width / 2)
            top_left_y = int(row['Y_Center'] * frame_height - row['Height'] * frame_height / 2)
            bottom_right_x = int(row['X_Center'] * frame_width + row['Width'] * frame_width / 2)
            bottom_right_y = int(row['Y_Center'] * frame_height + row['Height'] * frame_height / 2)

            # Draw SORT bounding boxes.
            color = tuple(map(int, row['color']))
            thickness = int(row['thickness'])
            cv2.rectangle(frame, (top_left_x, top_left_y), (bottom_right_x, bottom_right_y), color, thickness)
...
# Do the same for YOLO.
...
# If you want class labels from YOLO, use this. You'll have to define the label using the class ID column, and can style elements like font or thickness.
cv2.putText(frame, label, (top_left_x, top_left_y - 2),
                font, font_scale, (255, 255, 255), font_thickness, lineType=cv2.LINE_AA)
...
```

And that's it! The finished scripts will be uploaded to BioBoost upon completion: https://github.c om/Human-Augment-Analytics/Bio-Boost. To see the videos generated with OpenCV, please see the 100 videos that I added to the BioBoost channel in a zip file.

Eric Iamarino

This week, my main task was to clean up the BioBoost GitHub for the reviewers and add guidance to the README. I started off by adding docstrings to all functions, as well as type hints to all of the arguments/returns of the functions. From here, I cleaned up scripts that had standalone code, and I am now working to finalize the README and file structure of the Github. Additionally, I got permissions for editing the Cichlid CV website and am finalizing the Spring 2025 additions to the page.

Proof below:

Local changes made to BioBoost GitHub; Code review will be finalized by this weekend.

```
no changes added to commit (use "git add" and/or "git commit -a")
(base) ericiamarino@Erics-MBP scripts % git diff --stat
 scripts/extract_videos.py                           |  13
 scripts/manual_annotation_variance_analysis.py      |  73
 scripts/results_class_metrics.py                    |   3
 scripts/results_entropy.py                          |   4
 scripts/run_sort_cluster_track_association_preparer.py |  43
 scripts/run_sort_fish_detections.py                 | 168
 scripts/run_sort_fish_tracking_preparer.py          |  10
 scripts/run_temporal_model.py                       | 245
 scripts/run_yolo.py                                 | 178
 9 files changed, 486 insertions(+), 251 deletions(-)
```

Example of docstring and type hints added to functions

```
def train_class(self, data_yaml_path: str, batch_size: int = 800, imgsz: int = 75, device: int = 0, model: str = "yolov5s-cls.pt") -> str:
    """
    Trains Model for Image Classification

    Parameters:
        data_yaml_path (str): Path to the dataset's YAML configuration file.
        batch_size (int, optional): Batch size for training.
        imgsz (int, optional): Image size for training.
        device (int, optional): Device ID for training.
        model (str), optional: Path to pre-trained weights for classification.

    Returns:
        str: Path to the model after training
    """
```

Addition to Cichlid CV Website

Weekly Meetings

Spring 2025

Week 4 – Monday, January 27, 2025

Summary

Kailey: Modified Bree's SORT code to be able to run the modified SortFish on the new dataset. Modified Bree's YOLO/SORT visualization code to work with the Lindenthal dataset. Investigated and fixed some problems in BioBoost pipeline.

Charlie: Worked on re-running SORT and cropping tracks into videos. Continued reviewing manuscript submission guidelines for BioBoost project.

Eric: Reviewed current version of BioBoost rewrite, and reviewed manuscript submission guidelines. Cleaned up the BioBoost GitHub by adding docstrings, type hints, and more guidance to README.

Recording: Cichlid CV Weekly Meeting-20250127_144424-Meeting Recording – View-only

Slides: Cichlid CV Week 4 Meeting Slides

Fall 2024