Department of Electronic & Telecommunication Engineering
University of Moratuwa

EN3160 - Image Processing and Machine Vision

Parasitic Egg Detection and Classification in Microscopic Images

29th October 2023

N.W.P.R.A. Perera 200462U
A.M.P.S. Samarasekera 200558U

Contents

1 Abstract

2 Introduction

3 Related Work

3.1 Multitask learning via pseudo-label generation and ensemble prediction for parasitic egg cell detection

The ICIP 2022 Challenge submission by BAD Crew (Zaw Htet Aung, Kittinan Srithaworn, Titipat Achakulvisut) from Mahidol University and Looloo Technology, Thailand [1] is discussed below.

Their paper "Multitask learning via pseudo-label generation and ensemble prediction for parasitic egg cell detection: IEEE ICIP Challenge 2022" describes a methodology for detecting parasitic egg cells in microscopic images using deep learning. The authors used a multitask learning approach, where the model was trained to perform two tasks simultaneously: detecting parasitic egg cells and segmenting the images. The following strategies were used.

3.1.1 Methodology

The authors used various data augmentation techniques to increase the size of the training set and improve the robustness of the model. These techniques included random rotations, flips, and scaling of the images.

Pseudo-Label Generation
The authors used a technique called pseudo-label generation to improve the performance of the model. This involved using the model to make predictions on unlabeled data and then using these predictions as labels for the training data.

Ensemble Prediction
An ensemble of models to improve the accuracy of the predictions. The ensemble consisted of multiple models trained with different hyperparameters and architectures.

Evaluation Metrics
The authors used per-class mean average precision (mAP) scores to evaluate the performance of the model on the validation set. They also provided examples of model predictions on the test set, including cases where the model over-predicted or predicted false positives.

Overall, the authors used a combination of data augmentation, pseudo-label generation, and ensemble prediction to improve the accuracy of the model for detecting parasitic egg cells in microscopic images. They also used per-class mAP scores to evaluate the performance of the model.

3.1.2 Results

Model	mAP	mIoU
ResNet-101-GFL	0.941	0.923
ResNet-101-TOOD	0.948	0.926
HRNetV2p-W32-CascadeRCNN	0.948	0.927
HRNetV2p-W32-HTC (Multitask)	0.940	0.928
ResNeXt-101-HTC (Multitask)	0.941	0.928
Ensembled Model	0.956	0.934

3.2 PEDCMI, TOOD Enhanced by slicing-aided-fine-tuning and inference

The ICIP 2022 Challenge submission by Alzbeta Tureckova, Tomas Turecek and Zuzana Kominkova Oplatkova from Tomas Bata University in Zlin, Czech Republic [2] is discussed below.

The submission aimed to develop a deep learning pipeline for the Parasitic Egg Detection and Classification in Microscopic Images challenge.

3.2.1 Methodology

Data preparation
The available 11,000 images were cut beforehand to train the model for sliced-aided inference. The size of image cuts was chosen empirically and was set up to 960x1280 pixels (height x weight). The cut image was valid only if at least 30% of the area of at least one bounding box was present; other cuts were discarded. Training and evaluation within the training process were made on both the cuts and the whole original images. Therefore, the number of training images was augmented to 35,850 samples. All those images were divided into five folds for cross-validation. The test set, incorporating 2,200 images, was evaluated via a leaderboard on a challenge webserver.

Model architecture
The study used two architectures, Faster R-CNN and TOOD, both utilizing the classical residual CNN (ResNet) as a backbone for image classification. In the experiments, ResNet 50 network layers deep (r50)

and 101 network layers deep (r101) were used.

Training
During the training process of the utilized models, the original implementation was followed as closely as possible using original loss functions for each architecture. All the models were initialized with weights trained on the COCO dataset and modified for a number of classes present in the dataset, i.e., ten. The models had been trained for twenty-four epochs, considering one epoch as a single pass through all training data. SGD (Stochastic gradient descent) with momentum 0.9, and weight decay of 0.0001 were utilized to optimize the model weights.

Slicing Aided Inference
To further increase the precision of model predictions, aside from predicting the whole original image, the study also applied the sliced-aided inference. There, the image is cut into patches of the same size as during the training (i.e., 960x1280 pixels). The model processes all those patches, and then the individual outputs are stitched back together. The patches overlap by a 0.3 ratio of their size to minimize the risk of missing the object cut off on the border. The NMS (Non-Maxima Suppression) algorithm with a post-process match threshold of 0.5 IoS (Intersection over a Smaller area) was applied to suppress the multiple potential predictions.

The final methodology utilized the TOOD model, further enhanced by slicing-aided fine-tuning and inference. The slicing helps to overcome the image size invariability of the dataset and allows the model to access all images in high resolution, and consequently helps it learn detailed features needed to distinguish different classes and find a precise object position. The results demonstrate the importance of proper data analysis and consequent pre and post-processing to improve prediction performance.

3.2.2 Results

Model	mIoU	mF1scr
FasterR-CNN-r50	0.8381	0.9381
TOOD-r50	0.8393	0.9362
TOOD-r101	0.8770	0.9575
5-fold TOOD-r101	0.8781	0.9594
5-fold TOOD-r101+sahi	0.8857	0.9646

3.3 Parasitic Egg Detection and Classification by Utilizing the YOLO Algorithm

The ICIP 2022 Challenge submission by Yohanssen Pratama, Yuki Fujimura, Takuya Funatomi and Yasuhiro Mukaigawa from Graduate School of Science and Technology, Nara Institute of Science and Technology, Japan [4] is discussed below.

The methodology followed in the report titled "Parasitic Egg Detection and Classification by Utilizing the YOLO Algorithm with Deep Latent Space Image Restoration and GrabCut Augmentation" by involved the following steps.

3.3.1 Methodology

Dataset Preparation
The authors used a dataset of 11,000 images that were already enhanced and added a synthetic image for the training dataset. They used four scenarios that used 11,000 original images and four scenarios that used 11,000 restoration images. Additionally, they used one scenario that used 78,000 images, which was a combination of an original image (11,000 images), an image that had already undergone the restoration process (11,000 images), average blurring (11,000 images), vertical flip (11,000 images), horizontal flip (11,000 images), rotation (11,000 images), a raise hue (11,000 images), and a synthetic image (1,000 images).

Scenarios using the YOLO Algorithm
The authors used four different models of the YOLO algorithm with different resolutions and epochs. They used Yolo v5s and Yolo v5x models with a dataset of 11,000 images and resolutions of 416x416 and 640x640. Each model was trained for 100 epochs.

Synthetic Image Creation
The authors created a synthetic image by performing geometrical transformation, image processing, and augmentation. They used this synthetic image in the 9th scenario.

Experimental Result and Discussion
The authors aimed to get the best strategy and configuration to optimize the YOLO v5 Algorithm. They used 10 different scenarios and combined all detected images in all scenarios to get the best result.

In summary, the paper used a combination of enhanced and synthetic images to train the YOLO algorithm with different resolutions and epochs. They also created a synthetic image by performing geomet-

rical transformation, image processing, and augmentation. Finally, they combined all detected images in all scenarios to get the best result.

3.3.2 Results

Scenario	mIoU	mF1
1.YOLO v5s 416x416	0.647	0.821
2.YOLO v5x 416x416	0.687	0.845
3.YOLO v5s 640x640	0.695	0.845
4.YOLO v5x 640x640	0.708	0.860
5.YOLO v5s 416x416 (r)	0.593	0.765
6.YOLO v5x 416x416 (r)	0.639	0.811
7.YOLO v5s 640x640 (r)	0.661	0.819
8.YOLO v5x 640x640 (r)	0.684	0.837
9.YOLO v5x 640x640 78000	0.715	0.864
10.Combination of scenarios	0.717	0.864

Where r = restoration.

4 Method

4.1 Data Pre-Processing

Dataframe formation
The dataset used for this project is the Chula Parasite Dataset, which comprises annotated images of parasite eggs on faecal smears. The dataset labels were loaded from a JSON file, providing essential information about each image and its corresponding annotations. This data was organized into two separate Pandas dataframes, one for image metadata and another for annotations. Duplicate values within the annotations dataframe were identified and then addressed to ensure data integrity.

Dataframe combination
The two dataframes were then combined based on a unique identifier, specifically the 'id' column of the image dataframe and the 'image_id' column of the annotations dataframe. This integration process unified the metadata and annotations, creating a comprehensive dataset ready for further processing.

Bounding Box computation
The bounding box coordinates were computed using the original bounding box data. This transformation involved converting coordinates from COCO format $(x\_min, y\_min, \text{width}, \text{height})$ to YOLO format $(x\_center, y\_center, \text{width}, \text{height})$, with x_center and y_center representing normalized coordinates.

Test-Train Splitting
The resultant datafrane was subsequently divided into training and validation sets utilizing an 80-20 split ratio. Images corresponding to these sets were then copied to their respective directories, providing distinct datasets for subsequent training and evaluation processes.

4.2 Model Training

For this study, the YOLOv8n (nano) model, a lightweight variant of the You Only Look Once (YOLO) architecture, was chosen for object detection tasks. YOLOv8n is optimized for efficiency without compromising on accuracy. In comparison with other YOLOv8 models such as YOLOv8s (small) and YOLOv8m (medium), YOLOv8n (nano) was chosen due to its exceptionally high speed and low processing time with a sufficient amount of accuracy which can be further improved by increasing the number of epochs. We carried out the training using **30 epochs** which gave a good balance between computation time and model accuracy. The model was initialized using the configuration file (`yolov8n.yaml`), that contains the paths to image data and labels. Each training cycle consists of two phases: a training phase and a validation phase.

During the training phase, the train method does the following,

- Extracts the random batch of images from the training dataset.
- Passes these images through the model and receives the resulting bounding boxes of all detected objects and their classes.
- Passes the result to the loss function that's used to compare the received output with correct result from annotation files for these images. The loss function calculates the amount of error.
- The result of the loss function is passed to the optimizer to adjust the model weights based on the amount of error in the correct direction. This reduces the errors in the next cycle. By default, the SGD (Stochastic Gradient Descent) optimizer is used.

During the validation phase, train does the following:

- Extracts the images from the validation dataset.

- Passes them through the model and receives the detected bounding boxes for these images.
- Compares the received result with true values for these images from annotation text files.
- Calculates the precision of the model based on the difference between actual and expected results.

The following metrics are measured during each training cycle.

- **box_loss** : Measures the error in predicting the coordinates of bounding boxes. It encourages the model to adjust the predicted bounding boxes to align with the ground truth boxes.
- **cls_loss** : Quantifies the error in predicting the object class for each bounding box. It ensures that the model accurately identifies the object's category.
- **dfl_loss** : Defocus loss is a specialized loss component that helps improve object detection in scenarios with defocused or blurry images. It encourages the model to focus on improving the detection in such challenging conditions.
- **Precision** : Calculated as the ratio between the number of Positive samples correctly classified to the total number of samples classified as Positive (either correctly or incorrectly). $Precision = TP/(TP + FP)$
- **Recall** : Calculated as the ratio between the number of Positive samples correctly classified as Positive to the total number of Positive samples. $Recall = TP/(TP + FN)$

Mean Average Precision (mAP) is a crucial metric in object detection that evaluates model performance by considering both precision and recall across multiple object classes.

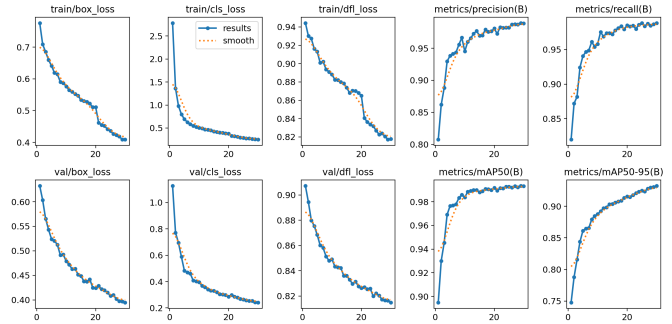
- Specifically, **mAP50** focuses on an IoU threshold of 0.5, measuring how well a model identifies objects with reasonable overlap. Higher mAP50 scores indicate superior overall performance.
- To provide a more comprehensive assessment, **mAP50-95** extends the evaluation to a range of IoU thresholds from 0.5 to 0.95. This metric is especially valuable for tasks requiring precise localization and fine-grained object detection.

In practice, mAP50 and mAP50-95 help assess model performance across different classes and conditions, offering insights into object detection accuracy while considering the precision-recall trade-off.

The progress and results of each phase for each epoch were displayed as outputs and plotted as graphs. Hence, we can see how the model learns and improves from epoch to epoch.

```
Starting training for 30 epochs...

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       1/30      2.4G     0.7762      2.783     0.9444          3        640: 100%|
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%
                   all       2202       2202      0.808       0.82      0.895      0.748

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       2/30     2.42G     0.7087      1.364     0.9302          6        640: 100%|
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%
                   all       2202       2202      0.862       0.872      0.93      0.788

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       3/30      2.4G      0.685     0.9813     0.9273          3        640: 100%|
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%
                   all       2202       2202      0.888       0.882      0.945      0.816

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       4/30      2.4G     0.6599     0.7979     0.9162          6        640: 100%|
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%
                   all       2202       2202      0.929       0.924      0.969      0.844
```

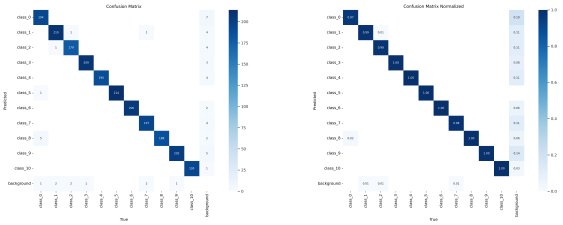
This code snippet displays how the losses decrease and metrics improve as number of epochs increase.



The reduction in losses and gradual improvement of all metrics as the number of epochs increase can be observed.

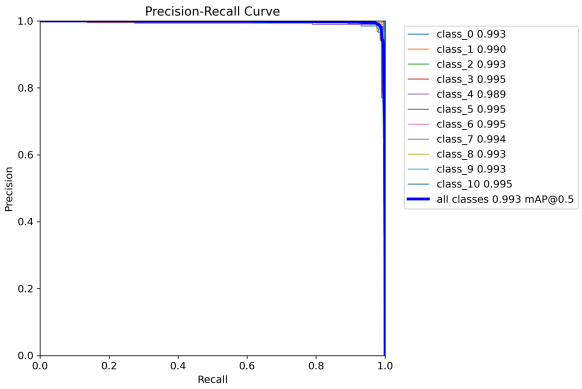
4.3 Model Evaluation

Confusion matrices of the have almost all of its entries in its diagonal and this in an indication that the model is performing extremely well.

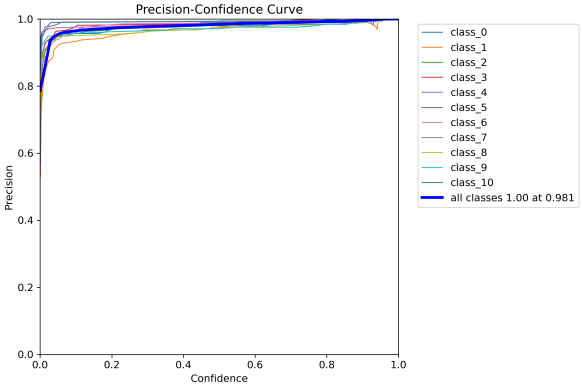


The Precision-Recall curve is a graphical representation of the trade-off between precision and recall for different thresholds. A high area under the curve represents both high recall and high precision, where high precision relates to a low false positive rate,

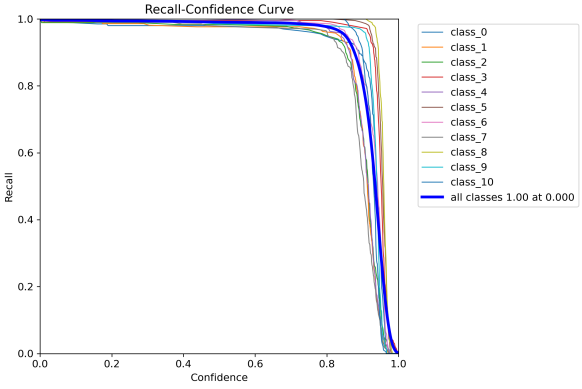
and high recall relates to a low false negative rate.



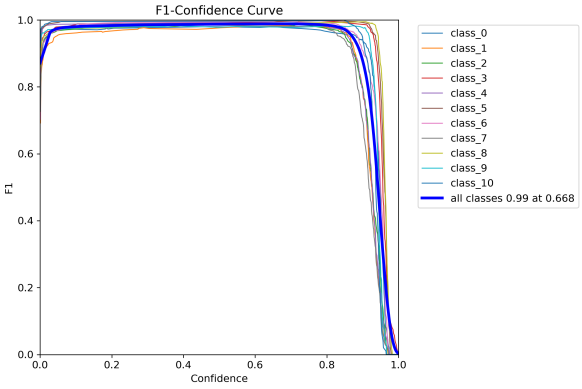
The Precision-Confidence curve shows how precision changes as a function of confidence threshold. It demonstrates the relationship between the confidence level of predictions and the precision of those predictions. A high area under the curve in the Precision-Confidence curve indicates that the model is able to make highly confident predictions that are also very precise. This means that when the model is confident about a detection, it is usually correct.



The Recall-Confidence curve shows how recall changes as a function of confidence threshold. A high area under the curve in the Recall-Confidence curve means that the model is able to achieve high recall even when it's confident about its predictions. This suggests that even when the model is highly certain about a detection, it is still able to capture a large portion of the true positives.



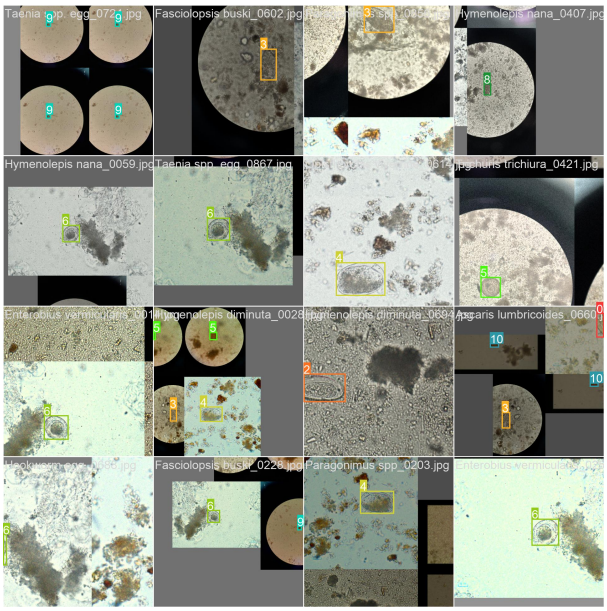
F1 score is the harmonic mean of precision and recall. It provides a balance between precision and recall. The F1-Confidence curve demonstrates how the F1 score changes as a function of confidence threshold. A high area under the curve in the F1-Confidence curve signifies that the model is able to maintain a high F1 score across various confidence thresholds. This means it is consistently balancing precision and recall effectively.

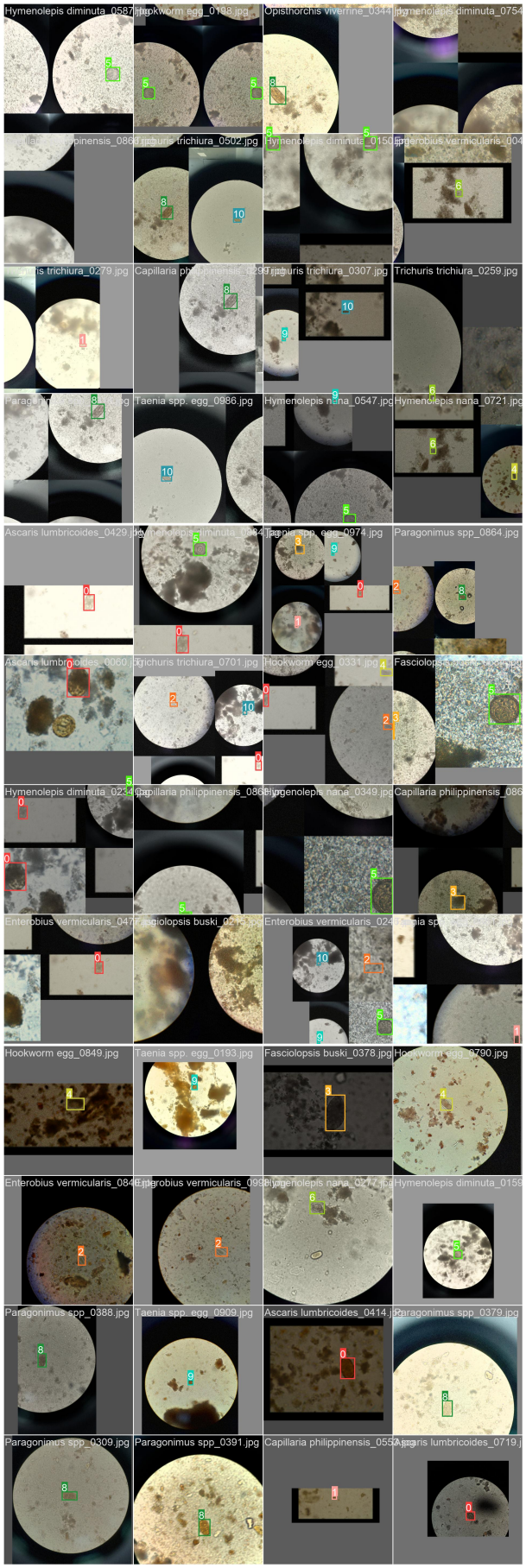


The above graphs and the confusion matrices obtained are a good indication that the trained model has good performance and accuracy.

4.4 Model Utilization

A few examples of detections done by the model are shown below. Predicted bounding boxes and predicted classes are annotated.

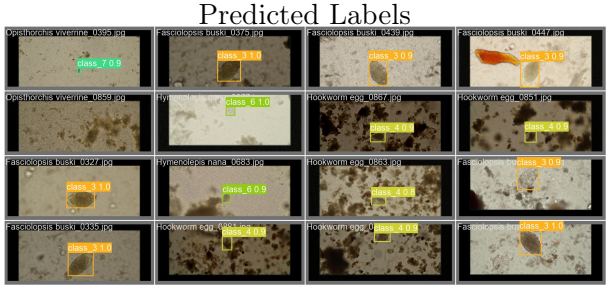




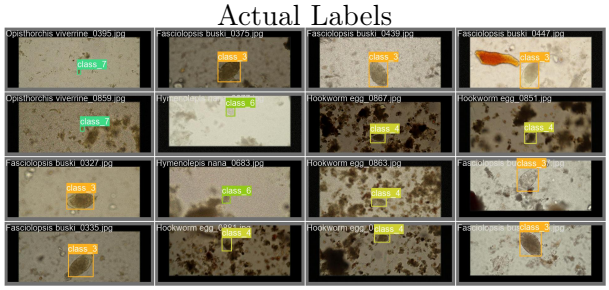
4.5 Comparisons between Predicted & Actual Labels

4.5.1 Example 1

Predicted Labels



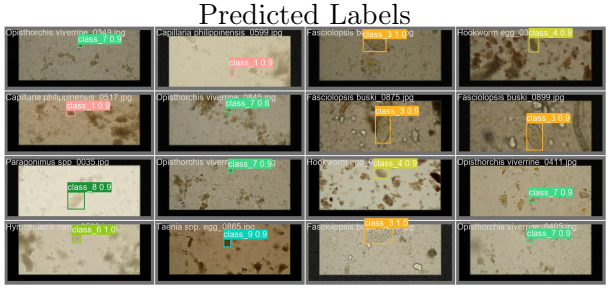
Actual Labels



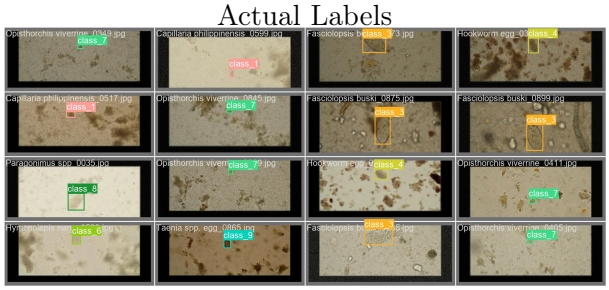
The model correctly identified 15 out of the 16 instances with high confidence, and missed only a single instance `Opisthorchis viverrine_0859.jpg`

4.5.2 Example 2

Predicted Labels



Actual Labels



The model accurately predicted all 16 instances with high confidence.

5 Acknowledgements

6 Conclusion

7 References

[1] "Dataset – ICIP 2022 Challenge." https://icip2022challenge.piclab.ai/dataset/ (accessed Oct. 17, 2023).

[2] Z. H. Aung, K. Srithaworn, and T.Achakulvisut, "Multitask learning via pseudo-label generation and ensemble prediction for parasitic egg cell detection: IEEE ICIP Challenge 2022," ieeexplore.ieee.org. https://ieeexplore.ieee.org/abstract/document/9897464 (accessed Oct. 20, 2023).

[3] A. Tureckova, T. Turecek and Z. K. Oplatkova, "ICIP 2022 Challenge: PEDCMI, TOOD Enhanced by Slicing-Aided Fine-Tuning and Inference," ieeexplore.ieee.org. https://ieeexplore.ieee.org/document/9897826/ (accessed Oct. 18, 2023).

[4] Y. Pratama, Y. Fujimura, T. Funatomi and Y. Mukaigawa"Parasitic Egg Detection and Classification by Utilizing the YOLO Algorithm with Deep Latent Space Image Restoration and GrabCut Augmentation," ieeexplore.ieee.org. https://ieeexplore.ieee.org/document/9897645 (accessed Oct. 28, 2023).

[5] G. Jocher, A. Chaurasia, and J. Qiu, "YOLO by Ultralytics," GitHub, Jan. 01, 2023. https://github.com/ultralytics/ultralytics

8 Appendix

For a detailed exploration of the source code and experimental results of our study, refer to our **GitHub Repository**.

Major parts of our source code implemented have been appended below.

```
### Installing Ultralytics &  WANDB library
! pip install ultralytics
! pip install wandb
```

```
### Loading the dataset labels from a JSON file.

file = open('/kaggle/input/chula-parasite-dataset/Chula-ParasiteEgg-11/Chula-ParasiteEgg-11/Chula-ParasiteEgg-11/labels.json')
data = json.load(file)
print()
print('SUCCESSFULLY LOADED THE DATASET LABELS!')
print()
```

```
### Converting the JSON file into a pandas DataFrame.

# Convert parts of the JSON file into 2 pandas DataFrames for easier manipulation
image_dataframe = pd.DataFrame.from_dict(pd.json_normalize(data['images']), orient='columns')
print()
print('Image DataFrame')
display(image_dataframe)
annotations_dataframe = pd.DataFrame.from_dict(pd.json_normalize(data['annotations']), orient='columns')
print()
print('Annotations DataFrame')
display(annotations_dataframe)
duplicate_values = annotations_dataframe['image_id'].duplicated()
print()
print('Duplicate Values in Annotations DataFrame')
display(duplicate_values)

# Merges the 2 DataFrames based on the 'id' column of image_dataframe and 'image_id' column of annotations_dataframe
merged_dataframe = pd.merge(image_dataframe, annotations_dataframe, left_on='id', right_on='image_id', how='inner')
# Drop the extra 'image_id' column as it's now redundant
merged_dataframe.drop(columns=['image_id'], inplace=True)
print()
print('Merged DataFrame')
display(merged_dataframe)
```

```
### Calculating the YOLO bounding box values for each image.

"""
In COCO, a bounding box is defined by four values in pixels [x_min, y_min, width, height].

These are coordinates of the top-left corner along with the width and height of the bounding box.
```

```
    In YOLO, a bounding box is represented by four values [x_center, y_center,
        width, height].

    x_center and y_center are the normalized coordinates of the center of the
        bounding box.
    To make coordinates normalized, we take pixel values of x and y, which marks
        the center
    of the bounding box on the x-axis and y-axis.
    Then we divide the value of x by the width of the image and value of y by
        the height of the image.

    width and height represent the width and the height of the bounding box.
    They are normalized as well.
    """

    # Computes YOLO-style bounding box coordinates and dimensions based on
        original bounding box data and merges them into a new column
    merged_dataframe['bbox_yolo'] = merged_dataframe.apply(lambda row: [
        ( ( row['bbox'][0] + row['bbox'][2]/2 ) / row['width']  ),
        ( ( row['bbox'][1] + row['bbox'][3]/2 ) / row['height'] ),
        ( row['bbox'][2] / row['width']  ),
        ( row['bbox'][3] / row['height'] )
    ], axis=1)

    # Display the new DataFrame with the bbox_yolo field
    print()
    print('New DataFrame with bbox_yolo field')
    display(merged_dataframe)
    print('SUCCESSFULLY CONVERTED THE JSON FILE INTO A PANDAS DATAFRAME WITH
        YOLO BOUNDING BOX VALUES!')
    print()
```

```
    ### Splitting the merged DataFrame into training and validation sets.

    # Splits the merged DataFrame into training and validation sets using a
        80-20 split
    training_dataframe, validation_dataframe =
        train_test_split(merged_dataframe, test_size=0.2, random_state=42)
    print()
    print('Training DataFrame')
    display(training_dataframe)
    print()
    print('Validation DataFrame')
    display(validation_dataframe)
    print('SUCCESSFULLY SPLIT THE MERGED DATAFRAME INTO TRAINING AND VALIDATION
        SETS!')
    print()
```

```
    ### Copying the training and validation images to their respective
        directories.

    # Specify the source and destination paths
    source_path =
        r"/kaggle/input/chula-parasite-dataset/Chula-ParasiteEgg-11/Chula-
    ParasiteEgg-11/Chula-ParasiteEgg-11/data"
    training_set_destination_path    =
        r"/kaggle/working/Chula-ParasiteEgg-11/images/training_set"
```

```
    validation_set_destination_path =
        r"/kaggle/working/Chula-ParasiteEgg-11/images/validation_set"

    # Create destination directories if they don't exist
    os.makedirs(training_set_destination_path,   exist_ok=True)
    os.makedirs(validation_set_destination_path, exist_ok=True)

    # Copy files for the training set
    for index, row in training_dataframe.iterrows():
        filename = row['file_name']
        src_file = os.path.join(source_path, filename)
        dst_file = os.path.join(training_set_destination_path, filename)
        if os.path.exists(src_file):
            shutil.copy(src_file, dst_file)
        else:
            print(f"Source file {src_file} does not exist.")

    # Copy files for the validation set
    for index, row in validation_dataframe.iterrows():
        filename = row['file_name']
        src_file = os.path.join(source_path, filename)
        dst_file = os.path.join(validation_set_destination_path, filename)
        shutil.copy(src_file, dst_file)

    print('SUCCESSFULLY COPIED THE TRAINING SET AND VALIDATION SET IMAGES TO
        KAGGLE WORKING DIRECTORY!')
    print()
```

```
    ### Creating text files for the training and validation labels.
    ### These text files will be used by the yolov8n model for training and
        validation.

    # Define the output directories
    training_set_labels_dir   =
        '/kaggle/working/Chula-ParasiteEgg-11/labels/training_set'
    validation_set_labels_dir =
        '/kaggle/working/Chula-ParasiteEgg-11/labels/validation_set'

    # Create output directories if they don't exist
    os.makedirs(training_set_labels_dir,   exist_ok=True)
    os.makedirs(validation_set_labels_dir, exist_ok=True)

    # Creating text files for the training set
    for index, row in training_dataframe.iterrows():
        category_id = row['category_id']
        bbox = row['bbox_yolo']
        filename = os.path.splitext(row['file_name'])[0]    # Remove the '.jpg'
            extension
        text_content = f"{category_id} {' '.join(map(str, bbox))}"   # Create
            the text content
        output_filename = os.path.join(training_set_labels_dir,
            f"{filename}.txt")
        with open(output_filename, 'w') as text_file:
            text_file.write(text_content)   # Write the text content to a file
                in the training output directory

    # Creating text files for the validation set
    for index, row in validation_dataframe.iterrows():
        category_id = row['category_id']
```

```
        bbox = row['bbox_yolo']
        filename = os.path.splitext(row['file_name'])[0]    # Remove the '.jpg' extension
        text_content = f"{category_id} {' '.join(map(str, bbox))}"    # Create the text content
        output_filename = os.path.join(validation_set_labels_dir, f"{filename}.txt")
        with open(output_filename, 'w') as text_file:
            text_file.write(text_content)    # Write the text content to a file in the validation output directory
```

```
### Training the Model

# Load and initializes the yolov8n model using the configuration file 'yolov8n.yaml'
# yolov8n model is used for object detection
# This configuration file specifies the architecture, hyperparameters, and other settings of the YOLO model
model = YOLO('yolov8s.yaml').load('yolov8n.pt')

# Specify the number of epochs for training
n_epochs = 30

# Training the model
results  = model.train(data="/kaggle/input/chulaparasitey/parasite_configure.yaml", epochs=n_epochs)
print()
print('SUCCESSFULLY TRAINED THE MODEL!')
```

```
### Evaluating the Model

# Evaluate the model's performance on the validation set
metrics = model.val()
print()
print('SUCCESSFULLY EVALUATED THE MODEL ON THE VALIDATION SET!')
```

```
### Packaging the results as ZIP archives for ease of downloading

source_directory = '/kaggle/working/wandb'
zip_file_path    = '/kaggle/working/wandb.zip'
shutil.make_archive(zip_file_path.split(".")[0], 'zip', source_directory)

source_directory = '/kaggle/working/runs'
zip_file_path    = '/kaggle/working/runs.zip'
shutil.make_archive(zip_file_path.split(".")[0], 'zip', source_directory)

print('SUCCESSFULLY CREATED THE ZIP FILES!')
```