YOLO Object Detection for WPTO, using EyeSea Dataset

The purpose of this document is to explain how to run the experiments from the YOLO Object detection work to produce the results that are described in the Final Report delivered to the WPTO. There are several documents that have been uploaded to the MHKDR, including coding python files and shell scripts to execute jobs in AzureML.

Table of Contents

[Downloading and preparing the Eyesea Optic data: 1](#_Toc192507239)

[Starting Yolo models: 5](#_Toc192507240)

[How to train a model: 5](#_Toc192507241)

[How to test a model: 7](#_Toc192507242)

[Replicating specific results of the experiments: 8](#_Toc192507243)

[Description of data assets in Azure: 12](#_Toc192507244)

[Overview of other files included in upload: 16](#_Toc192507245)

## Downloading and preparing the Eyesea Optic data:

1. Go to this website and click download: <https://data.pnnl.gov/group/nodes/dataset/12978>
	1. Note that this can take some time (it was 2 hours for us) to download the ~85GB files
2. Keep unzipping the files until you get to a directory structure like this:
	1. 
3. Sub-folders you can consider deleting:
	1. Videos: we did not use them in this phase of work
	2. keras-yolo3-master folder because it is ~9GB and not data, but it does include some of their results in case we are interested
4. Files that are included in each folder:
	1. Train\_x: 54517
	2. Train\_y: 54516
	3. Wells\_test\_y: 4802
	4. Wells\_test\_x: 4801
	5. Orpc\_test\_x: 7781
	6. Orpc\_test\_y: 7781
	7. Voith\_test\_x: 3100
	8. Voith\_test\_y: 3100
5. Data cleaning required to get to the desired structure (there are sample directories of the desired file structure uploaded to the MHKDR):
	1. There are several shell scripts that work to separate the training data by source, then convert the .xml labels to .txt files to work with yolo algorithms (yolo requires .txt file labels), then to separate fish versus fishless images into separate repositories
	2. Note that some experiments, starting with Experiment 3 and later, used data that was formatted according to contiguous data streams, meaning instead of 5 different breakdowns (Orpc 1, Orpc 2, Orpc 3, Voith, WellsDam) each with their own train/ val/ test, meaning 15 total folders, the data was broken out into contiguous streams of data for each of the 3 sources and then different contiguous streams were just taken to be training or validation or testing based on how many fish images they contained (ORPC 1-5, Voith 1-4, WellsDam 1-2, so 11 total folders). The data file that creates the latter of those is the ‘create\_fish\_and\_fishless\_data.py’ file in the ‘Exp3\_and\_Exp4\_client\_deliverable’ directory, and the below instructions create the first division.
	3. Instructions for creating the 15-folder breakdown:
		1. ‘Process\_data\_like\_article.sh’ (you must personalize where your downloaded files are saved on your computer, and make sure that you have a copy of train\_x and train\_y elsewhere, if you want them, because part of the script will delete them from the specified folder)
			1. Note, the ‘like\_article’ suffix to so many of these files stem from the fact that we take 10% of the testing data and make that the validation set, and leave all the downloaded train files as train. This is the method of the scholarly article (train starts out as 80% of data and test is 20%, then val is taken as 10% of that 20%)
			2. The ‘process\_data.sh’ file processes the data differently, where we take 20% of the training data as validation, and therefore only keep the fish images of those. This results in a lot more validation and the preservation of all the testing data as testing. However, we decided against doing it this way because it was not the way the scholarly article does it and we were trying to replicate it as best as possible
		2. ‘Process\_train\_add\_few\_fishless\_img.sh’ and ‘Process\_train\_add\_half\_fishless\_img.sh’ to create alternate versions of the training data that include some fishless images (these will each require you to start with train\_x and train\_y again and will then delete them, so make sure to copy into the specified origin folder
		3. ‘Exp1\_process\_labels\_to\_run\_Exp1pretrain.sh’ is run to create a copy of all the validation labels and images, but the labels will show a 192 instead of a 0 for the ‘fish’ object detection class number
	4. Once we determined that there were duplicate images in the downloaded data, we realized we needed to remove them, we processed the data again using the following shell scripts:
		1. process\_train\_just\_fish\_images\_rmv\_dup.sh
			1. This will take in the optic image data exactly as it is downloaded (the train\_x and train\_y files) and then create a dataset that is separated by fish source (Igiugig, Wells, Voith) and includes just the fish images, while also removing duplicates (which appear only in the Orpc dataset)
		2. process\_valandtest\_and\_removing\_dups.sh
			1. This will take in the optic image data exactly as it is downloaded (the images in the test\_x and test\_y images) and then create a dataset that is separated by fish source (Igiugig, Wells, Voith) and split into validation and testing images. This will also remove any duplicates in the process
		3. process\_train\_add\_20perc\_fishless\_imgs\_delete\_duplicates.sh
			1. This will take in the optic image data exactly as it is downloaded and then create a dataset that is separated by fish source (Igiugig, Wells, Voith) and includes all fish images and 20 percent non-fish images (which is calculated so that 20% of the training images are fishless)
			2. This calls upon convert\_txt\_and\_split\_fish\_article\_incl\_fishless.py, which will split images into folders based on the optic image source and whether it is a fish or fishless image, while removing duplicates

The actual python files which will be called by all the above shell scripts, which are found within the various uploaded code folders, are as follows:

* 1. Exp1\_Change\_fish\_number\_for\_exp1.py:
		1. Background: as we run our own training models, in the label files, the fish are allotted with number 0 to represent their object class. However, when we want to run an out-of-the-box pretrained model with the images (one that was trained on OpenImagesv7 data), they have more objects they can detect than just fish, so ‘fish’ is labeled as 192 in their yaml file. Therefore, we had to change all the .txt label files to say ‘192’ for each detection instead of ‘0’
		2. This file takes in all the validation data (since we won’t need the training data if the model is pre-trained) and outputs the images and labels, but label files say 192 instead of 0 as the class number for fish
	2. Convert\_txt\_and\_split\_fish\_article\_incl\_fishless.py:
		1. Takes in the large groups of train\_x and train\_y and breaks them down into sub-folders based on which video source they are from (Igiugig 1 through 3 representing different camera angles of Orpc, Voith, or WellsDam) and then creates .txt files from the .xmls only for those images that have corresponding fish. Then this script saves a specific number of images without fish (the exact number is specified as an input to the file and can be changed) to the images folder along with all the fish images
		2. We use this to create 2 different training datasets, one that is train\_files\_few\_fishless (where we only add 20 fishless images to each source file) and then train\_files\_half\_fishless (where we add enough fishless images to be equal to the number of fish images- except for WellsDam because they have more fish images than fishless so we just use all of their images)
	3. WPTO\_convert\_py\_article:
		1. Converts .xml files into .txt files, but does not separate fish and fishless (so this is used for the test data folders, since those should have fish and fishless images in them
	4. Split\_test\_val.py
		1. Splits images and labels up from being all the test set to being 10% the validation set- this is more difficult because we can’t have consecutive frames with fish go to different groups since that may skew the results and training
		2. Therefore, this script checks that we split the val and test sets on an image that does not have a fish up to 3 frames away from it (it also makes sure that there are some fish frames in both val and test, which was particularly difficult for the Igiugig 3 camera angle since there were only a handful of images with fish in them)
	5. split\_datasets\_by\_source\_Ig\_article\_rmv\_duplicates
		1. Splits the data by fish source, including splitting between all 3 Igiugig camera angles and removing duplicates within ORPC
	6. split\_datasets\_by\_source\_Ig\_article
		1. Splits the data by fish source, including splitting between all 3 Igiugig camera angles, but does not remove duplicates within ORPC
	7. split\_test\_val\_rmv\_dups
		1. Splits the test\_x and test\_y folder optic images into a validation and test set, and removes duplicates that may be in the process
		2. Note that the validation and test sets include fishless images as well as all the fish images

## Starting Yolo models:

Training a yolo model requires a starting model be used, which then gets iterated on based on the training you ask it to do. These starting models come in a variety of sizes and versions, based on what was referenced in the final report, and can be downloaded from the Ultralytics website ([version 8 model download page](https://docs.ultralytics.com/models/yolov8/#__tabbed_1_1)). From that website is where we downloaded the starting models for all our training jobs, specifically the one we ended up choosing that did best, the yolov8m.pt starting model. To download that, click on the link above and click the ‘YOLOv8m’ button under the ‘Detection (COCO)’ tab. The main starting model that we used for a majority of the experiments is the yolov8m.pt model, namely the medium version 8 yolo model, which can also be found in the uploaded ‘Yolo\_models\_downloaded’ folder.

## How to train a model:

For this experiment, we used Visual Studio to write coded files in python which we then called upon through the command prompt to be executed and run in Azure. We followed several tutorials to assist getting our command prompts and working laptops to our instance of AzureML, and here are a few general guides that we used to figure out the process:

* Ultralytics Yolo website, guide to executing with Azure ML: [here](https://docs.ultralytics.com/guides/azureml-quickstart/#can-i-use-both-the-ultralytics-cli-and-python-interface-on-azureml)
* Medium.com article on executing Yolo training through the command prompt into Azure: [here](https://medium.com/%40ouphi/how-to-train-the-yolov8-model-with-azureml-and-the-az-cli-73d3c870ba8e)

Please find below a general walk through of some files and how to execute the training of a model imitating the way we did it for this experiment:

* + - 1. To train one model, you need a job.yaml file and a data.yaml file
			2. A job yaml file example is ‘job\_exp1.6\_v8m\_fewfishless\_50ep.yaml’ and a data yaml file example is ‘dataset\_all3train\_all3val\_fewfishless.yaml’
			3. Within the job yaml file, customize the following parameters:
				1. Experiment name: larger group of model runs that can be grouped together under this one name for easier identification in Azure
				2. Command: this is the actual code that is being run to do the Yolo model object detection. This is the line that could be run on a local computer to do the training, but we have extracted several parameters and made them variables so we could change them as we liked, and configured it so it was run on Azure and not locally
				3. Inputs: Each of these will need to be configured, one of which is the experiment name which should be the same as the one defined above, and the name of the dataset yaml file
			4. Within the dataset.yaml file, you are defining where the data is located that you will use to train/ test the model. All the files on GitHub will have the data configured according to how we have it saved within Azure. However, this will need to be re-configured so the paths and filenames match where the data is stored on the computer of whomever is performing the experiment
			5. Once you have both the job and data .yaml files written, and an environment built into your Azure workspace as well as the data uploaded to Azure, you can use the command prompt to execute the job in Azure through the template “az ml job create -f <name\_of\_job\_file>.yaml --name <name\_to\_be\_given\_to\_job\_in\_azure>”
			6. Another way to launch a job in Azure is to have shell scripts that automatically create the job.yaml and data.yaml files for you for several jobs you’d like to run at one time, which is what we ended up doing towards the end of the experimentation. This would need to be customized based on your Azure and computer file setup. Looking at an example file such as “Exp4\_orpc\_preprocessing\_comparison\_train.sh”, these shell scripts have various variable inputs set equal to given values, some of which stay the same for each job in the shell script, some of which change. Then the script creates the job and data yaml files and launches each of the jobs in the command line using those files.
			7. Once a job is running in Azure, it is visible in the Azure webpage interface, with the status (e.g., queued, running, complete, failed).
			8. Whenever ‘training mAP’ was referenced in the final report, that refers to the maximum mAP50 of the training job, or mean average precision with an Intersection over Union (IoU) set to 0.5, found on the completed job page. This represents the best a model did during the training process, when it was determining weights for the neural network within the model, then testing it on the validation set, and then capturing feedback to then choose different layer weights. This is different than the testing mAP because that would indicate that the pre-trained model was used on new data, and no feedback could be received by the model on how well it did with that object detection in order to do better in testing- it just has to use the weights it has to identify objects within the given test data.
			9. In this way, the training mAP can be seen after a training job is Complete (which is when its status in Azure is ‘complete’) and then clicking on the job and scrolling to its max mAP50.

## How to test a model:

Testing a model involves launching a job in Azure the same way as described in the previous section, namely by having a job and data yaml created. However, the biggest difference is that the line of executable code in the job yaml that was previously set to ‘yolo task= detect train…’ will now say ‘yolo task= detect val…’, changing the model to be a valuation instead of a training. Additionally, the dataset yaml will have the train and val section, but it will not matter what you put in the training section, since the model specified to be used in the job yaml will be a pre-trained model that you are trying to evaluate, as opposed to a model that was just downloaded from the Ultralytics website and requires training.

Once those changes are enacted, then you can launch a job in the same way, namely through the command prompt directly using the template command ‘az model create -f …’ or a shell script, and then the job will appear in Azure again. This will take a lot less time to test a model than to train one, because it is essentially only doing one iteration of work as opposed to several iterations for training.

Once the job is complete, you can click on the job, navigate to the ‘Outputs + logs’ section of the results, and look under the mAP50 category of the std\_log.txt file to see the numerical mAP50 which represents the validation. In our experiments, we used this approach to test the model on the test set of images, so the number found here would be the testing mAP and would be the ultimate sign of how well a model performed.

## Replicating specific results of the experiments:

1. Experiment 1, out of the box model:
	1. Starting model: for this experiment, we did not train a model, so the starting model was directly downloaded using [this link](https://docs.ultralytics.com/datasets/detect/open-images-v7/), scrolling down to choose one of the models under the ‘Open Images V7 Pretrained models’ table (I chose the nano and the large sized models to experiment with). The one that did the best that we used to create the numbers and photos in the final report was the large pre-trained model, and can be found in the Experiment 1 uploaded folder in MHKDR
	2. Given the downloaded model, you would reconfigure the Exp1\_val\_orpc\_v8loiv7\_pretrained.py (same naming convention for other 2 sources, wells and Voith) so the pre-trained model is set up for validation on the given source’s image data. You also would need to modify the dataset yaml to include your images wherever they are located.
	3. To see some example images of exactly how the pre-trained model works on the images, you can use the Exp1\_val\_orpc\_v8loiv7\_pretrain\_results\_vis.py (same naming convention for other 2 sources, wells and Voith) which is the code to produce the tagged images which helps display how the pre-trained model is identifying (or not identifying) fish from our dataset
		* 1. The actual photos that the results\_vis files produce are found in the ‘Inference\_results\_v8loiv7pretrained’ folder within the Exp1 folder uploaded to MHKDR
2. Best Experiment 2 model:
	1. The various sub-experiments that led us to find the best performing model for experiment 2, namely the method and Yolo parameters that resulted in the highest mAP, can be found in the Exp2 uploaded folder. To run the experiments that correspond to each one, you would run the .yaml files that are associated with the certain jobs, based on naming.
		1. For example, the file ‘job\_exp1\_v8m\_all3train\_all3val\_80ep.yaml’ when run, would give the trained model and mAP from running a Yolov8m model on all 3 fish sources (namely all 3 of ORPC, Wells, and Voith, which is what Experiment 2 does), for 80 epochs. By running the job from that file as well as the job from the v8n, v8l, v8x versions of that same file name, we can graph the mAPs across all the 80 epochs to determine which size model results in the highest mAP, and also how many epochs we need until it seems like the gains from epochs are not materializing into a higher mAP anymore.
			1. Note that Exp2 used to be called Exp 1 since our experiments 1-5 used to be Experiment 0-4, and we just shifted the numbering up 1
	2. By launching several different sub-experiments, we landed on one model that got us to the highest test mAP (0.661). Note, this is just how to replicate the best results from experiment 2, as we had improved these with the later experiments:
		1. Train the best model:
			1. Run the ‘job\_exp1.6\_v8m\_20percfishless\_50ep.yaml’ file to launch the job in Azure
			2. The training will get to a max mAP of ~0.71
			3. Save that model as ‘Exp1-6\_v8m\_all3trainandval\_20perc\_fishless\_bestof50eps’, which is also found in the ‘Yolo\_models\_downloaded folder uploaded to MHKDR as well, in case you want to jump directly to testing
		2. Validate with the best model:
			1. We validated on all 3 fish sources and then on each source individually, by running each of these scripts:
				1. ‘job\_exp1.6best\_test\_all3\_20percfishless.yaml’ (replace ‘all3’ with Voith, wells, or Orpc for the other 3 scripts)
				2. Once these test scripts are run, go to the Code section of the output to see the mAP under the mAP50 columns
				3. This will give us the mAPs that we see in the final report
		3. Note, these all use the all-fish-incl-fishless version 2 data set which contains the version of the data that has 20% of the training images be without fish (hence the name 20percentfishless)
3. Experiment 3, best models from results table
	1. We had improved our model by determining that the ORPC dataset needed to be de-duplicated, which resulted in a slightly higher mAP. The way to replicate and get that result (0.67 testing mAP) is:
		1. Train the model:
			1. Use 20 percent fishless and all fish data in training
			2. Run the ‘job\_exp1.6\_v8m\_20percfishless\_50ep\_duplicatesremoved.yaml’ file to launch the job in Azure
			3. The training will get to a max mAP of ~0.67
			4. Save that model as ‘exp1-6-20percfishless-de-dup-v8m-best’
				1. This model can also be found in the ‘Yolo\_models\_downloaded’ folder uploaded to MHKDR, in case you want to skip the training and go straight to testing
		2. Validate the model:
			1. Run the ‘Exp1.9\_preprocessing\_methods\_full\_exp\_test.sh’ file and comment out anything having to do with the non-baseline job run (anything required by lines 55-69), you can see that this job uses the model as specified in the last section above
			2. The job will start in Azure and get to a mAP output of 0.67
	2. Then we had determined that pre-processing might work if we changed the BS methods, so we changed the way that background subtraction was applied, and included the CLAHE method which got to a 0.69 testing mAP, which was our highest one achieved on optic images. The way to get to this model is:
		1. Create CLAHE and BS optic images:
			1. In the Exp3 and 4 folder, there are several ‘Exp3.0.1-create…’ related python files which create CLAHE and BS train and test sets for both just fish and also fishless images. For input, those require the original source data that has been processed into the subfolder structure as discussed previously (so the data in different folders for orpc1-5, voith1-4, etc.)
			2. The ‘CL 2.0’ and ‘15by15’ in the title refers to what clip limit and grid size parameters are in effect when the dataset is created, since those are the 2 input parameters for CLAHE. We chose a grid size of 15x15 and clip limit of 2.0 since those worked out the best in training when we experimented just with CLAHE
		2. Train the model:
			1. Use training data that includes all fish images and 20 percent fishless (meaning 20% of the training set is fishless images)
			2. Run the ‘Exp3\_preprocessing\_comp\_separate\_train.sh’ file to launch the job in Azure, along with others for pre-processing
			3. The training will yield a max mAP of ~0.25
			4. Save that model as ‘exp1-9-1\_clahe\_separated-try2-best’, which found in the ‘Yolo\_models\_downloaded’ folder uploaded to MHKDR
		3. Validate the model:
			1. Run the ‘Exp3\_preprocessing\_comp\_separate\_val.sh’ file and comment out anything not having to do with the CLAHE validation job launch (you only require anything necessary for rows 57-70)
			2. The job will start in Azure and get to a mAP output of ~0.69
4. Experiment 4, and testing on just the Voith domain:
	1. There were several different attempts at training a model then testing on a completely different test set. We discussed the ORPC and Voith attempts at this, so to replicate those, do the following:
		1. Voith:
			1. The best results came with the no pre-processing, but to run all the pre-processing training jobs, you would run the ‘Exp4\_voith\_preprocessing\_comparison\_train.sh’ file
			2. Then you save the output of each of those training jobs and run the ‘Exp4\_voith\_preprocessing\_comparison\_test.sh’ file, replacing all the model names with what you have named the saved models from the previous step
			3. The best test mAP will be the ‘base, no-pre-processed’ file name, which results in a 0.67 mAP
			4. That model is named exp3-0-3\_base\_no\_preproc-best.pt and is found in the ‘Yolo\_models\_downloaded’ folder uploaded to MHKDR
		2. ORPC:
			1. The best result for Orpc was the Background-subtracted method. To launch all the training jobs, you would run the ‘Exp4\_orpc\_preprocessing\_comparison.sh’ file which launches 5 jobs
			2. Then after those run, save the best weights from each training job and run the ‘Exp4\_orpc\_preprocessing\_comp\_val.sh’ file with the model names replaced with what the saved models from the previous steps are named, and then that launches all 5 testing validation jobs
			3. Once those are done running, you will see the testing mAP, which should be very low for all 5 jobs, although the background subtracted one is the highest
			4. The best model is named exp3-0-1\_bs\_best.pt and is found in the ‘Yolo\_models\_downloaded’ folder uploaded to MHKDR
5. Experiment 5: Applying Yolo models to sonar data
	1. This experiment requires sonar images to be downloaded and prepared into various forms, including channel separation and hyper-image formation. All of the links for download and image preparation can be found in the ‘Hyper-Image-main folder within the Exp5 folder uploaded to MHKDR.
	2. Then, 6 training jobs were launched using the 6 different versions of the sonar data (combined channels, channel 0, channel 1, channel 2, hyper image diff, hyper image stack), by running the ‘Exp5\_sonar\_data\_diff\_channels\_train.sh’ and the ‘Exp5\_sonar\_data\_diff\_channels\_train\_part2.sh’.
		1. Note that 2 shell script files were required because the first one does not run the channel 0 or channel 1 versions of the sonar data, and the part2 file runs those 2 but not the others that ran in the original
	3. Once those 6 jobs are done training, you would save all the best model weights and then ensure the validation shell script files have those newly saved model names in them before launching them.
		1. The files to run for testing those models would be the ‘Exp5\_sonar\_data\_diff\_channels\_val.sh’ and ‘Exp5\_sonar\_data\_diff\_channels\_val\_part2.sh’, and when those are complete, you will have the testing mAPs as referenced in the final report, namely that the hyper-image diff yielded a 0.65 mAP
	4. The best model weights are titled exp4-hyperver-diff-train-last.pt and are found in the ‘Yolo\_models\_downloaded’ folder uploaded to MHKDR

## Description of data assets in Azure:

Even though the future users of this code may not work in AzureML, it will be helpful to review what the datasets look like that are used for various experiments as described above. You will also see these dataset names mentioned in job yaml files above, since launching a job in AzureML requires the job scripts to explicitly list which dataset it will be using. Also note, this was a privately-owned AzureML instance and therefore access to these exact datasets could not be granted, which is why instructions and scripts are included so they can be replicated.

1. all-fish-incl-fishless: this was created to include fishless images in the training sets
	1. Description of sub-files:
		1. train\_files\_few\_fishless has training data with just 20 added fishless images per camera angle/ source (so each of the 5 sub-folders has 20 images that are fishless along with all the fish images)
		2. train\_files\_half\_fishless has training data with added fishless images equal to the number of fish images from each camera angle/ source (except for WellsDam which had more fish images than non-fish, so that sub-folder just includes all the fish and non-fish images that exist). Namely, half of the training images will be fishless.
		3. val\_for\_pretrained is all the validation images and labels but the labels say 192 for the fish detections instead of our usual 0, so that it is compatible with the out-of-the-box model that was trained on OpenImagesv7
		4. val\_and\_test\_files are all normal validation and test images and labels, which includes fishless images. All labels have a 0 to represent the detected fish class
2. all-fish-data-justfish: Classic train/val/test data set, no fishless images, with 10% of test being defined as val, all training data remains training data
	1. 
	2. 
	3. Description of sub-files:
		1. train\_files has all training data of just images with fish in them and the associated txt label files
		2. val\_and\_test\_files are all normal validation and test images and labels, which includes fishless images and all labels have a 0 to represent the detected fish class
3. exp3-BS-CLAHE-combos:
	1. This includes subfolders for all the various camera angles for all the different fish sources, within each sub-folder of a different pre-processing method
	2. 
	3. 
	4. These were formed by first experimenting on what set of parameters worked best within each given method, namely which BS method (MOG2 mean of gaussians 2 or KNN k nearest neighbor) worked best, and what clip limit and grid size worked best for CLAHE. Once we determined that MOG2 worked best for Background subtraction and GS 15x15 and CL 2.0 worked best for each, we could create the datasets
	5. This new division of 11 folders instead of 15 (5 different sources/camera angles, each with train/val/test) came about after we knew we weren’t getting great testing results when we tried to apply Yolo to a new environment. A deep dive of the data showed that the datasets we were using for training, validation, and testing had varied numbers of actual fish images which could have negatively affected the model mAP results. We therefore needed to distribute the data in this way to be able to cherry-pick which exact streams of fish optic image data to use for each of the training, validation, and testing so we could have enough fish images in each group.

## Overview of other files included in upload:

The uploaded code for the Yolo experiments contains various shell and python scripts that were run throughout this experiment. Here is an overview of what each folder contains:

* + - 1. Exp3 and 4 folder:
				1. Automated\_jobs

Contains the necessary scripts template for when the shell scripts are run that automatically launch jobs in Azure by creating job and dataset yaml files

* + - * 1. Src

Python files with defined functions for repeatable actions required for processing optic images, such as drawing the object detection labels on the images and showing a loading bar when performing a time-consuming training so progress can be monitored

* + - * 1. Various other python files are in this folder and are used for creating other pre-processed images, such as VS scripts which stand for velocity subtraction
			1. Templates
				1. Used for the automated shell scripts that launch several jobs at a time
				2. This contains the base job and data yaml files that the automated job shell scripts customize to run jobs in Azure
			2. Exp1 folder
				1. Inference\_results\_v8loiv7pretrained

Optic images with labels representing the object detections from Experiment 1 which was run locally

* + - 1. Yolo\_models\_downloaded
				1. All Yolo models that were either downloaded directly from the Ultralytics site, or were downloaded after training if they were the best model, and then were called upon again during testing of those models on testing data
				2. The naming conventions here indicate if the models were a direct download, since the models named ‘yolov<NumberLetter>’.pt were directly downloaded, while the models directly named in the sections above, or ones with longer and more custom names, are the ones that came out of training on the EyeSea dataset
			2. Exp2 folder
				1. azureml\_create\_entities

The sub-folders here have yaml or docker files corresponding to what had to be run to launch an environment, create a compute instance, upload a local dataset to Azure so that jobs could be run in Azure using those objects

* + - * 1. The individual scripts are named according to which job they were launching.

The naming convention of 1.X where X is the experiment number is implemented so that each individual value of X was testing a new parameter for Experiment 1 (which we have been calling experiment 2 because we had an experiment 0 that is being changed to an experiment 1), and the title of the yaml files will show the values for that parameter being tested

For example, Exp1.1 was testing pixel size, and the 4 yaml files with that as the name show that they are launching jobs with the v8m sized model, using all 3 fish sources (Voith, wells, Orpc), using fewfish (adding 20 fishless images to each data source for training), run for 50 epochs, and what pixel size the images are.

The yaml files starting with ‘job\_val’ or ‘job\_test’ are launching a validation job in Azure to get the validation or testing mAP for a given fish source, by using the validation or testing data and a model that has already undergone training, and the ones starting with ‘dataset’ are used to define what data is being used, data that must be already loaded into AzureML

## Content and Technical Notes

* mAP: mean average precision, so average of the maximum precision values at different recall values of the model
* YOLO: You Only Look Once models, widely used neural network-based algorithms for video object detection, segmentation, and tracking
	+ Ultralytics is the company behind its release and implementation, who have extensive documentation available publicly on their website
* Article of inspiration and comparison:
	+ Previous work in this realm applied YOLO v3 to the Eyesea dataset (Xu and Matzner 2018), while this project applied YOLO v8 and estimated performance improvements. We were able to compare our experiments 1-3 to results from similar experiments from this previous work
	+ Formal citation is:
		- Xu, X., & Matzner, S. (2018). Underwater Fish Detection Using Deep Learning for Water Power Applications. In 2018 International Conference on Computational Science and Computational Intelligence (CSCI) (pp. 313-318). Las Vegas, NV, USA. doi:10.1109/CSCI46756.2018.00067.
* Wholistic Experiment Goals:
	+ Experiment 1. Baseline, out-of-the-box test
		- Assessed a pre-trained, non-fish-specific YOLO model to determine whether a custom YOLO model was necessary for our fish application, or if existing models from the YOLO developer website would be adequate
	+ Experiment 2. Mixed-source model- parameter experimentation
		- Replicated and improved upon Xu and Matzner (2018) experiment 1. The YOLO object detection model was trained on all three fish sources' training and validation sets, then saved the best model to apply on the test images
	+ Experiment 3. Pre-processing optimization
		- Enhanced Experiment 2 results by using the same model configurations on improved images and selecting optimal preprocessing for the Eyesea dataset
	+ Experiment 4. Single-source, applying models to unfamiliar domains
		- Trained a model on one source and tested it on the same source, but from a different camera angle
	+ Experiment 5. Sonar data source models
		- Used the best-performing YOLO model, trained on optical images, to assess its performance on sonar images