Instruction Manual

Image Recognition for Fisheries Species

Developed by:

Tzofi Klinghoffer, Caleb Perez

Massachusetts Institute of Technology, Sea Grant College Program

27 July, 2017

Table of Contents

Overview

Motivation: Fisheries populations have a large impact on the U.S. economy. Each year the U.S. fishing industry contributes 90 billion dollars and 1.5 million jobs to the U.S. economy [1]. Each species may serve as a predator or prey for another. In this regard, fisheries populations are interconnected and dependent. While humans may depend on these populations as a source of sustenance (food, goods, etc.), humans can also negatively impact population growth. Barriers to migration, pollution, overfishing, and other forms of human-interference may impact spawning patterns of fisheries species. In 2014, 17% of U.S. fisheries were classified as overfished [1]. Therefore, it is necessary to monitor these fisheries populations to determine when policy must be changed in efforts to maintain healthy oceans.

Current Techniques: Many groups, including NOAA Fisheries, state agencies, as well as regional fisheries councils and local municipalities, deploy camera and video equipment to monitor fisheries populations. Large amounts of video and photographic data are gathered at timed intervals. However, not all photos contain aquatic life. Currently, employees at these agencies among others are responsible for manually annotating the gathered videos and photos; this means they identify and count the relevant aquatic specimens in the data. Not only is this an inefficient use of time and resources, but also it can lead to inaccurate results due to human error. NOAA Fisheries Management can make a significant improvement in time and resource use through automation of the annotation process.

Enhancing Fisheries Management: This instruction manual describes a process to use image recognition to mitigate the necessity for human labor in the process of estimating marine populations. Image recognition, also commonly referred to as object recognition, is a subset of artificial intelligence and computer vision. The basic concept is to train a computer to learn to recognize specific patterns, such what a scallop looks like, in images. Current state-of-the-art image recognition algorithms rely on Convolutional Neural Networks (CNNs) to achieve learning and recognition. In their most basic form, CNNs loosely represent biological neural networks: each layer in the CNN accomplishes a specific task,

such as detecting edges. Training an image recognition algorithm results in the generation of a weights file that can then be used as input to run detection. Training is highly computationally expensive and specialized equipment is encouraged and likely necessary. Once training is done, detection can be done at low cost. The research conducted explores the accuracy of You Only Look Once (YOLO)v2: Real-Time Object Detection software.

Using this Manual: Refer to "Running Detection on Fisheries Species" if you wish to use a pre-trained weights file. If you wish to train the algorithm with your own photos, please refer to "Image Prep & Training".

For questions regarding this instruction manual and its content, please contact authors Tzofi Klinghoffer (tzofi@mit.edu), Caleb Perez (crperez@mit.edu), or research advisor Rob Vincent, PhD (rvincent@mit.edu).

Installing YOLO v2: Real-Time Object Detection

Goal: Install YOLO v2: Real-Time Object Detection on your Linux or Windows system. Note: to run training, GPU-enabled computer is recommended.

Recommended installation YOLO v2 on Linux or Mac OS X:

To use these instructions, you must have access to the MIT Dropbox with our darknet directory. Current path is: "i/Dropbox (MIT)/Vincent/Summer 2017/Software/darknet"

Dependencies: **Python 2.7**, **gcc C/C++ compiler**, CUDA and CUDNN (if GPUs are to be used → GPUs accelerate training and real-time video detection significantly), OpenCV (needed to process video in real-time). If on Mac OS X, it may be necessary to download **XCode** from the App Store if this is your first time using Terminal commands.

1. Copy the darknet folder from Dropbox to the location you want it on your computer
2. Move into the darknet directory by typing "cd [path to darknet directory]" into the command line.
3. Edit the Makefile as necessary using your favorite text or code editor (Vim, etc)*
4. Type "make" to compile Darknet (.o files will be generated)
    a. If you are recompiling darknet, we recommend typing "make clean" in darknet directory first to remove .o files. Then type "make"

Alternative installation YOLO v2 on Linux or Mac OS X:

To use these instructions, you must have access to the MIT Dropbox with our darknet directory. Current path is: "i/Dropbox (MIT)/Vincent/Summer 2017/Software/darknet"

Dependencies: **Python 2.7**, **gcc C/C++ compiler**, CUDA and CUDNN (if GPUs are to be used → GPUs accelerate training and real-time video detection significantly), OpenCV (needed to process video in real-time)

1. Open terminal window and navigate to directory where you want to put darknet subdirectory (cd [directory])
2. Type "git clone https://github.com/pjreddie/darknet" (this installs darknet files on your computer)
3. Type "cd darknet" to move into darknet directory.
4. Edit the Makefile as necessary using your favorite text or code editor (Vim, etc)*
5. Type "make" to compile Darknet (.o files will be generated)

b. If you are recompiling darknet, we recommend typing “make clean” in darknet directory first to remove .o files. Then type “make”

* After opening Makefile, values for GPU, CUDNN, OPENCV, and DEBUG can be set to 0 or 1 to enable or disable that functionality, respectively. CUDNN is for GPU. If enabled, line 44 (COMMON+ variable for GPU) should be set to the correct include path for CUDA; always leave -DGPU -I prepended to the path as it currently is in the Makefile.

Debug notes:

- Error: “ . . . /usr/include/c++/6.3.1/type_traits(875): error: function returning function is not allowed, /usr/include/c++/6.3.1/type_traits(886): error: an explicit template argument list is not allowed on this declaration, Error limit reached., 100 errors detected in the compilation of "/tmp/tmpxft_000038dc_00000000-7_convolutional_kernels.cpp1.ii".” or similar, please try this:
    - If you get this error, edit the Makefile and, if not already added, add the “-std=c++98” flag to the end of the NVCC variable on line 22 within the parentheses for the “-fPIC” flag

  For other errors or debugging, please refer to the Darknet Google Group: https://groups.google.com/forum/#!forum/darknet

For more information on installing on YOLO v2, refer to: https://pjreddie.com/darknet/yolo/

Installation YOLO v2 on Windows:

Note: The pre-trained HabCam and herring weights were trained on Mac/Linux implementations of YOLO v2, which appears to use a slightly different architecture, leading to different outputs with the same weights. Therefore, the pre-trained weights fail to generate accurate predictions using this Windows port -- unfortunately, it is likely necessary to retrain with a Windows implementation to achieve comparable results, which we did not have the hardware to complete. Nevertheless, instructions on how to compile the Windows port are detailed below. Our full training data set is included in the Dropbox (with the Darknet-compatible annotations), which we hope will make the retraining process fairly painless with sufficient GPU-capable hardware. See https://timebutt.github.io/static/how-to-train-yolov2-to-detect-custom-objects/ for instructions on the training process -- you may skip to running the *process.py* script as all of the prep has already been completed, and use our modified *process.py* script saved in the Dropbox. The .data, .cfg, and .names files may also be reused. See the README in the Dropbox for file locations.

For additional information on installation and use of Windows port: https://github.com/AlexeyAB/darknet

1. Download and extract repository from the above link to your desired directory. This can be done by clicking on the green “Clone or download” button, and choosing “Download ZIP”. Alternatively, type “git clone https://github.com/AlexeyAB/darknet” into your command line.
2. On the same github page, click on the link to download Microsoft Visual Studio (MSVS) 2015. Follow the installer’s instructions to complete installation.
3. Links can also be found on this page to download and install OpenCV (we recommend version 3.x) and CUDA, if desired. Ensure that any optional dependencies installed have the paths indicated on the github page (e.g. C:\opencv_3.0\opencv\build\include); this may require creating parent additional parent directories.
4. Open darknet.sln or darknet_no_gpu.sln with MSVS, depending on whether you wish to compile with GPU support enabled (requires CUDA). If this is the first time you are opening a file of this type with MSVS, you will be asked to install some additional necessary dependencies. Follow the instructions to complete these installations.

5. If you do not have OpenCV installed, the OPENCV variable must be removed from the preprocessor definitions. Right click on the .sln filename in the "Solution Explorer" window and select properties. In the C/C++ -> Preprocessor tab, delete OPENCV from the Preprocessor Definitions.
6. Right click on the .sln filename and select properties. In the C/C++ -> Advanced tab, select "Compile as C Code (/TC)" from the "Compile As" drop down menu.
7. From the two drop down menus at the top of the screen, select Release and x64, respectively.
8. Select Build -> Build Solution.
9. If the build succeeds, a darknet.exe or darknet_no_gpu.exe file will now be located in the x64 directory near the .sln file.
10. If using built with OpenCV support, you must copy opencv_world320.dll and opencv_ffmpeg320_64.dll from C:\opencv_3.0\opencv\build\x64\vc14\bin to the same directory as the .exe file.
11. Navigate to this directory from the command line ("cd [path to directory]") to run functions, using the appropriate .exe as the first argument.

Debug notes:

- Syntax errors (e.g. missing semicolons, variables misdefined, etc.) often totaling > 600 in number: Ensure "Compile as C Code (/TC)" has been selected in properties (see step 6 above).
- OpenCV related .h files cannot be found: Ensure you have a path to the necessary OpenCV files as indicated on the linked github page. The two necessary paths are as follows: C:\opencv_3.0\opencv\build\include; C:\opencv_3.0\opencv\build\x64\vc14\lib. Ensure that the first path is defined in Properties -> C/C++ -> General -> Additional Include Directories.

Image Prep & Training

Prep

Goal: The image recognition algorithm must be provided with training data that it can then process in order to learn the classes you want detected. Training images must be accompanied with annotations that indicate the location of relevant objects in the image.

Get the Images: It is recommended that at least 500 images are used to train for a unique class/species in order to achieve accurate results. These images should each be of the same resolution and aspect ratio. If not enough images are available, the flip-mirror.py script can be run to double your set of images (by flipping and mirroring each image, i.e. each image will be rotated about the x and y axes).

Running flip-mirror.py:

1. Move flip-mirror.py into your directory of images
2. Navigate to directory via CMD or Terminal (cd [path])
3. Run script: python flip-mirror.py

Annotate the Images: Each image must be annotated. The LabelImg tool should be used to do this; it creates VOC formated XML for each image you annotate.

Necessary Software: LabelImg - installation instructions can be found here:

https://github.com/tzutalin/labelImg

1. Create a directory to save all your annotations
2. Open LabelImg: if on Windows, navigate to the labelImg directory and double click labelImg.exe. If on Mac OS or Linux, navigate to the labelImg directory via Terminal (cd [path]) and run "python labelImg.py"
3. Click File → Change default saved Annotation dir, select the directory created in step 1
4. On the left hand pane, click Open Dir, select the directory with the training images
5. On the left hand pane, use the "Create RectBox" button to draw a box and label each relevant object in the image
6. Click "Save" and "Next Image" to proceed. Repeat Step 5 and 6 until all images are annotated

Training

This section includes instructions on training the You Only Look Once (YOLO) v2: Real-Time Object Detection software on your set of annotated images. Before starting this, you should have a directory of images and a directory of .xml annotations (created from annotating each image with LabelImg). The number of files in these two directories should be the same and each image should have a corresponding .xml file with the same name.

Troubleshooting:

If the number of files in the images and annotations directories are not equal or if you are worried not every image has a corresponding .xml, you can use the "match-xmls.py" script to move all .xml files that do not have matching image files and all image files that do not have matching .xml files to a separate directory.

1. Move match-xmls.py to parent directory of image and annotation directories.
2. Run match-xmls.py by running "python match-xmls.py [name of image directory] [name of annotations directory]"
3. Navigate to "Removed" directory to view images and .xmls that did not have corresponding .xmls and images, respectively

Train the Algorithm: Training will be done by customizing the YOLO v2: Real-Time Object Detection software to your classes and images.

In depth instructions on running training can be found here: https://timebutt.github.io/static/how-to-train-yolov2-to-detect-custom-objects/

Running Detection on Fisheries Species

Goal: Using either pre-trained weights (for HabCam or Herring) or weights you have generated, the YOLO v2 software can make predictions on the presence and counts of fish in an image or a set of images.

NOTE: All paths in this section are relative to the darknet directory.

Pre-trained HabCam and Herring Paths:

	Weights Path	Data File Path	CFG File Path
HabCam	weights/HabCam_final.weights	cfg/HabCam.data	cfg/HabCam.cfg

Herring	weights/Herring_final.weights	cfg/herring.data	cfg/herring.cfg

Running Detection on a Single Image

Set up:

1. If training with pre-trained HabCam or Herring weights, use the aforementioned weights, data, and cfg files. Image Path = path to image you want to use as input
2. If you have created your own training weights, use the weights, cfg, and data files you generated in the "Image Prep & Training" section. Image Path = path to the image you want to use as input

Running:

1. Navigate to the built darknet directory (cd [darknet path])
2. Run detector test command (-thresh is optional; default is 25%)
    a. If on Linux/Mac OS X, run:
    ./darknet detector test [Data File Path] [CFG File Path] [Weights Path] [Image Path] -thresh [0.01 - 1]
    b. If on Windows: darknet.exe detector test
    darknet.exe detector test [Data File Path] [CFG File Path] [Weights Path] [Image Path] -thresh [0.01 - 1]
3. View output predictions and predictions image: predictions.png or predictions.jpg (found in darknet directory)

Running Detection and Counting on a Set of Images

Run count-fish.py on directory of images. Use desired weights file, data file, and cfg file.

Syntax: python count-fish.py image_folder_path data_file_path cfg_file_path weights_path threshold

View predictions folder for outputted predictions images. View yolo-log.txt for detailed log of predicted classes and confidences for each image, as well as a running count of the classes of interest in the input image directory.

Gathering Relevant Statistics on Accuracy

Built-in Darknet Functions, Valid and Recall:

1. Ensure all images in the test set are in .jpg format, as necessary for these functions. On Linux/Mac, convert images to .jpg by running "mogrify -format jpg *.png" from the image directory, replacing .png with the appropriate image type. On Windows, an image conversion software must be used -- find a link here: https://timebutt.github.io/static/how-to-train-yolov2-to-detect-custom-objects/.
2. Ensure there are .txt files for each image in the test set containing Darknet-compatible annotations, as well as a test.txt file containing the path of each image in the test set, as detailed here in the above link. Convert-voc.py can be run to generate .txt files given VOC XML annotations, and process.py can be run to generate the test.txt file. See scripts for detailed usage instructions.

3. Ensure the appropriate .data file contains the path to the test.txt file under the “valid” field.
4. Change the string in line 506 of the detector.c code, located in darknet/examples/detector.c, to the path of to the test.txt file (can be relative to the darknet application file, or absolute). Re-compile the function by running “make” from the darknet directory.
5. Run desired functions from the darknet directory:
    a. Valid: “./darknet detector valid [.data path] [.cfg path] [.weights path]”
        i. Generates text files for each class containing all potential detections and their associated confidence values in a results folder. These files can then be read to calculate precision and recall values by class (see below).
    b. Recall: “./darknet detector recall [.data path] [.cfg path] [.weights path]”
        . Calculates overall average IOU and recall values for the test set, incorporating all classes.

Generating Precision-Recall (PR) Curves:

1. Generate result text files with the valid function using instructions above.
2. Move the calculate-recalls.py script from the Scripts directory to the results folder containing the text files. Run the script, including an argument with the path to the Darknet .txt annotation files.
3. The script will generate a .csv file for each class, containing recall and precision values as the threshold is varied from 0.01 to 1.
4. See the PRCurves.m Matlab script to generate PR curves.

Experimental Results

Applying Image Recognition to Enhance Fisheries Management Capabilities

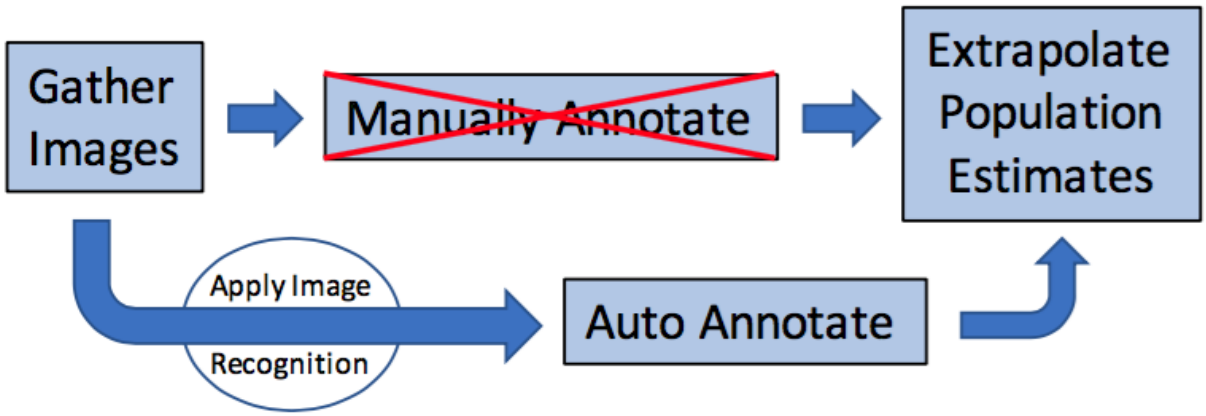
Abstract

Marine fisheries populations have a large impact on the U.S. economy – from commercial fishing to coastal communities. Overfishing, barriers to migration, and other forms of human activity may impact spawning patterns of these species. Therefore, it is necessary to monitor these populations to maintain sustainable resources, healthy oceans, and marine life. Federal and state agencies deploy camera equipment to monitor fisheries populations; employees then manually count the number of specimens in the gathered videos and images. Not only is this an inefficient use of resources and employee time, but also it can lead to inaccurate results due to human error. We propose an alternative approach. Through the application of deep learning-based image recognition, identification of target species in video and image data can be automated. Current state-of-the-art image recognition relies on Convolutional Neural Networks (CNNs) to achieve learning and recognition. CNNs loosely represent biological neural networks: each neuron, or layer, accomplishes a specific task, such as edge detection. Such algorithms can be adopted to enhance the capabilities of Fisheries Management in monitoring fisheries populations. This research introduces a new set of training data and explores the accuracy of several image recognition algorithms, such as You Only Look Once (YOLO) 2 and Faster R-CNN, in detecting target species, including alewife herring (*Alosa pseudoharengus*), blueback herring (*Alosa aestivalis*), Atlantic sea scallop (*Placopecten magellanicus*), flatfish such as flounder (Pleuronectiformes), skates (Rajidae), and various round fish species. Results suggest this approach is viable; 93% average recall is obtained in detecting Atlantic sea scallops, skates, flatfish, and round fish species.

Background

Fisheries species have a large impact on the U.S. economy. Each year, the U.S. fishing industry contributes approximately 90 billion dollars and 1.5 million jobs to the U.S. economy [4]. In 2014, 17% of fisheries were classified as overfished [4]. This can lead to depressions in the fishing industry that have a noticeable impact on the overall economy. Therefore, NOAA Fisheries Management is interested in monitoring fisheries populations.

Current technique for monitoring fisheries populations are time and resource inefficient. First, photographic images must be gathered from the ocean floor. The HabCam (Habitat Mapping Camera System), developed by the Woods Hole Oceanographic Institute (WHOI) is a popular method for gathering these photographic data. Among others, it is used by the NOAA Northeast Fisheries Science Center in their fisheries stock assessments. The HabCam can be dragged behind a ocean vessel and will continuously take photographs of the ocean floor. After these photographs have been gathered, they must be annotated. Currently, human annotations are done; employees and volunteers review each of the millions of images to identify 1) which species, if any, are present, 2) how many relevant species are present in each image, and 3) the pixel location of the relevant species in each photograph. Then, counts can be extracted and extrapolated to estimate overall fisheries species population levels.



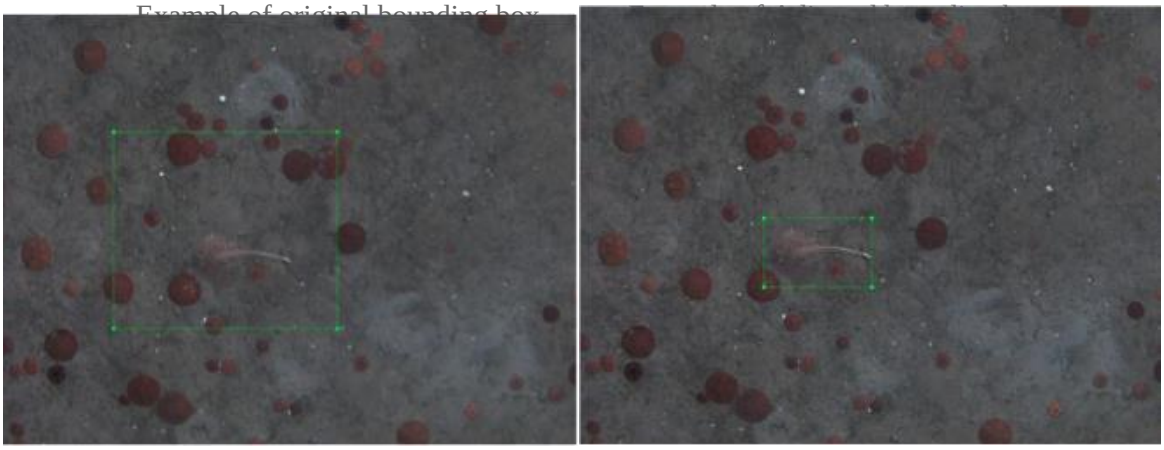
The goal of this research is to replace the human annotation currently used with automatic annotation or automatic annotation in conjunction with human annotation. This is done through the application of image recognition. This research addresses the application of image recognition to 4 relevant species: Atlantic sea scallops, skates, flatfish, and various round fish species. Herring is an item for Future Work. Three research questions are addressed in this research:

1. Can image recognition be used to accurately detect and count fisheries species?
2. How many iterations of training are needed to yield accurate results?

a. This is relevant for those without significant computing resources who may not know how long training must be run to gain accurate results

3. How does the quality of annotations used in training impact accuracy?

. During the analysis of human annotations that were extracted into VOC XML format for image recognition training, it was noticed that bounding boxes were always not drawn accurately or precisely around the objects of interest. A second training set was then created with adjusted annotations that had accurate and precise bounding boxes. The results of training with each training set are then analyzed.



Materials

Hardware	Software
1 x Intel Xeon CPU E5-2623 v4, 2.60GHz	LabelImg Graphical Image Annotation Tool
4 x NVIDIA Titan X GPU	You Only Look Once (YOLO) v2 Real-Time Object Detection [5]
1 x Toshiba 1 TB Canvio Slim II Hard Drive	Fedora 25 Linux Operating System

Methodology

1. Gather and Annotate Images
2. Train using the YOLO v2: Real-Time Object Detection software

a. Original Training Set = 5,063 images
b. Adjusted Training Set = 5,063 images

3. Test on set of 300 images with 489 objects

Results

Three metrics are used to analyze the results of the image recognition software as it is applied to detecting fisheries species: Intersection Over Union (IOU, %), recall (%), and precision (%). Intersection Over Union measures the accuracy of each bounding box that is created by the image recognition software:

$$IOU = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

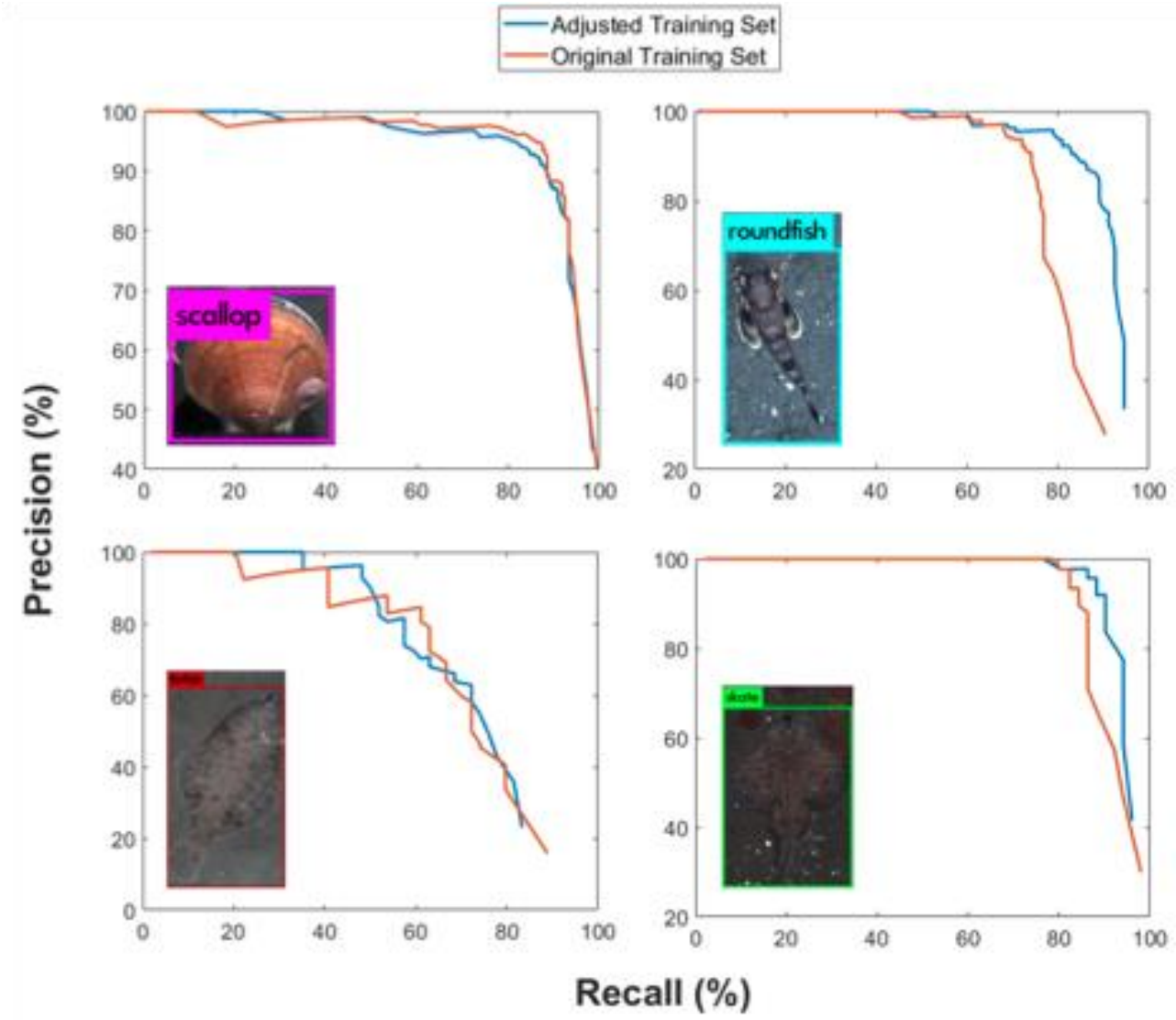
Recall measures the percent of objects correctly identified over the true number of objects in the image. A high recall indicates the presence of less false negatives.

$$\text{recall} = \frac{tp}{tp + fn}$$

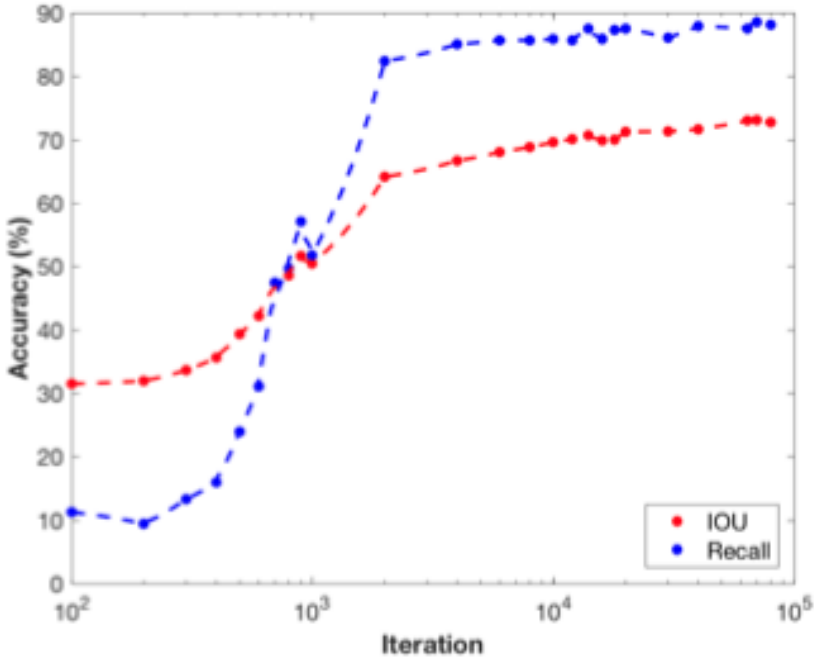
Precision measures the percent of objects correctly identified over the number of objects identified (both correctly and incorrectly). A high precision indicates the presence of less false positives.

$$\text{precision} = \frac{tp}{tp + fp} = \frac{tp}{n}$$

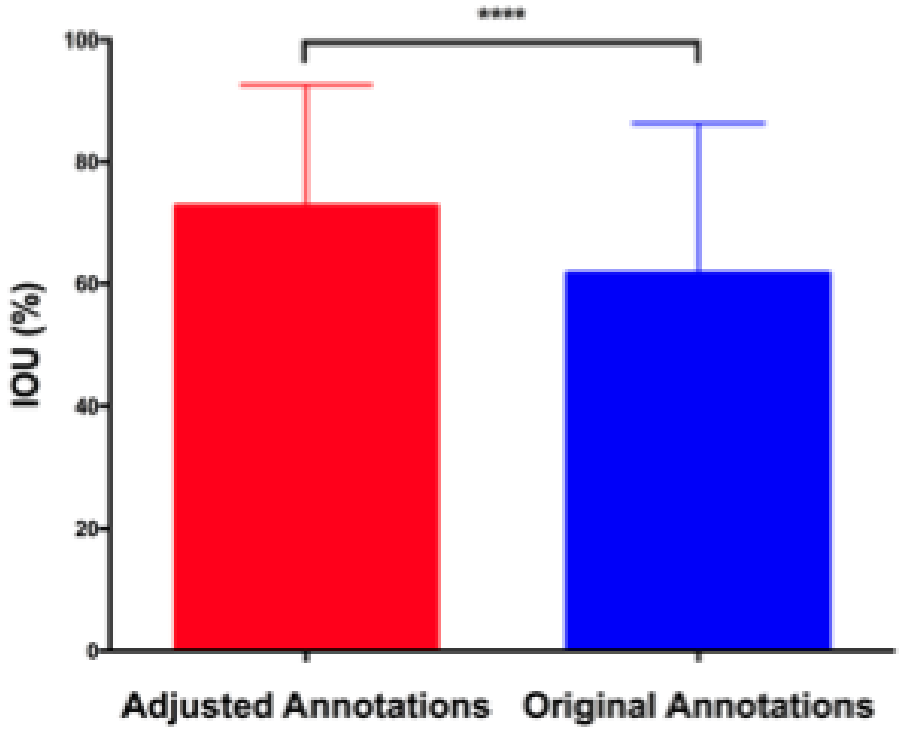
tp = # true positives, *fn* = # false negatives, *fp* = # false positives, *n* = # total detections



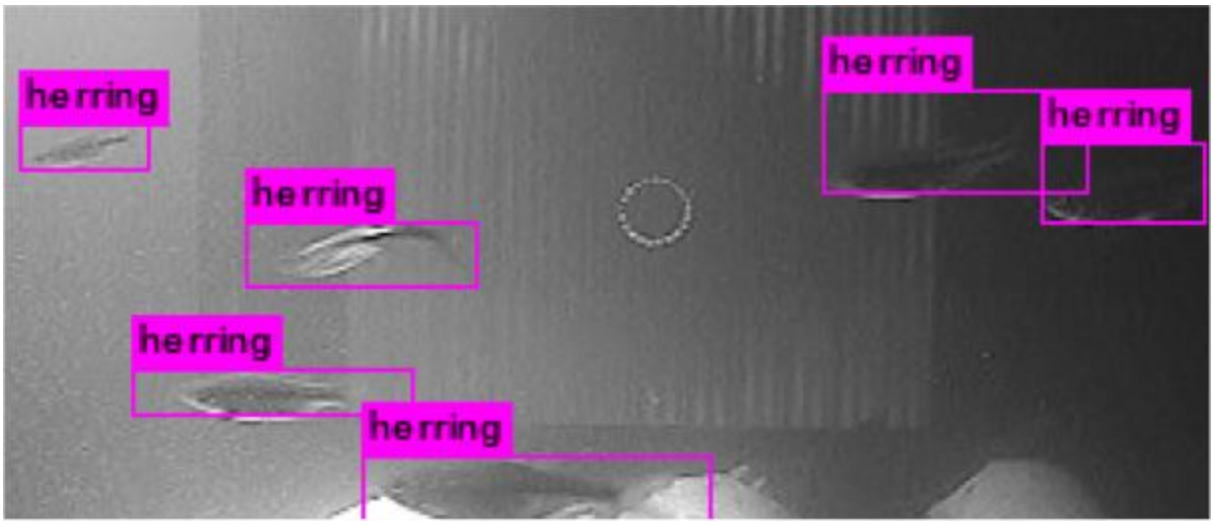
These graphs show the Precision - Recall (PR) curves for each relevant species examined in this research. An ideal PR curve would maintain a perfect precision even until perfect recall is obtained.



This graph indicates that 2000 iterations of training are needed before accuracy increases stabilize and there are no more significant increases in accuracy with additional iterations.



IOU values averaged across all objects (N = 489) in both the adjusted and original training sets.



Example of output from YOLO v2: Real-Time Object Detection trained on herring species

Conclusions

Image recognition is a viable solution to detecting and counting fisheries species in photographic data. You Only Look Once (YOLO) v2: Real-Time Object Detection software can obtain as high as 93% average recall in detecting Atlantic sea scallops, skates, flatfish, and various roundfish species. According to [2] Chang et al. 2016, imperfect automated annotation can be combined with human annotation. In other words, by using the high recall achieved in this research, photographic data can be filtered to only images with a high probability of having a relevant species in it. Humans can then cross-examine these images to remove any false positives. Additionally, based on the significant difference in IOUs achieved when testing on the original dataset vs the adjusted training set, we recommend annotation guidelines be strictly followed.

Deliverables for this research include: training sets, trained weights, programs for counting fisheries species.

Implications for this research are that NOAA Fisheries can use these techniques to create large savings in time and resource allocation.

Future Work

Future work includes continuing to apply image recognition to herring. This is of interest to: NOAA Fisheries, state agencies, as well as regional fisheries councils and local municipalities. Image recognition is a novel approach to the problem of detecting and counting herring populations. In addition, a graphical user interface for end users must be developed. Finally, other image recognition architectures used in implementations such as Faster R-CNN and Mask R-CNN may achieve even more accurate results. Therefore these implementations should also be trained and tested with fisheries photographic data.

References

[1] Chang, Jui-Han, Burton V. Shank, and Deborah R. Hart. "A comparison of methods to estimate abundance and biomass from belt transect surveys." Limnology and Oceanography: Methods 15.5 (2017): 480-494.

[2] Chang, Jui-Han, et al. "Combining imperfect automated annotations of underwater images with human annotations to obtain precise and unbiased population estimates." Methods in Oceanography 17 (2016): 169-186.

[3] Karpathy A. Convolutional Neural Networks (CNNs / ConvNets). In: Stanford University [Internet]. [cited 21 Jul 2017]. Available: http://cs231n.github.io/convolutional-networks/

[4] Kearney, Melissa S., Benjamin H. Harris, and Brad Hershbein. "Economic Contributions of the U.S. Fishing Industry." *Brookings*. Brookings, 28 July 2016. Web. 25 July 2017.

[5] Redmon, Joseph, and Ali Farhadi. "YOLO9000: better, faster, stronger." *arXiv preprint arXiv:1612.08242 (2016)*. APA

BIOELECTRIC FIELD MEASUREMENTS: AUGMENTING IMAGE RECOGNITION FOR FISHERIES MANAGEMENT

Please see Caleb Perez's paper:

/Dropbox (MIT)/Vincent/Summer 2017/Documentation/Perez_Final_Poster_Report.pdf