Sean Coughlin

CS 445 - Final Project - 5/6/2021

Swimming Stroke Detector

Summary

An object detector that can recognize what swimming stroke is being performed from an image. The detector was trained on a dataset that I collected.

Motivation

Computer vision is being used to augment many aspects of life. Object detection and identification is a key part of computer vision. Swimming was the sport I did growing up, so I thought it would be fun to tie together learning a new skill with a subject matter that I enjoy.

A possible use of a swimming stroke detector would be as part of a camera that automatically moves, zooms, and focuses on swimmers without need for human intervention. This would remove the need for a camera crew and make filming swimming events, like for the Olympics, a cheaper process.

Approach

First, I collected 357 images of myself swimming the four strokes. These images were collected at one pool during one photography session. To increase the diversity of images in the dataset, the angle of the photographs was varied, and I changed swimming suits and caps halfway. The collected dataset is comprised of 173 freestyle images, 60 backstroke images, 58 butterfly images, and 56 breaststroke images. Since freestyle is the most common stroke extra emphasis was placed on collecting freestyle images with a roughly equal mixture of the three other strokes.

The images were downsized to 600x400 resolution using the transform_image_resolution python script.[1] Downsizing the images speeds up the training process. Then the images were classified and labelled with bounding boxes using the open-source software labelImage.[2] LabelImage produces labels in

[1] https://github.com/TannerGilbert/Tensorflow-Object-Detection-API-Train-Model/blob/master/resize_images.py
[2] https://github.com/tzutalin/labelImg

XML format. The XML files were converted to CSV format using the xml_to_csv python script.[3] CSV formatting is required for TensorFlow training. The dataset was divided into 66 testing images and 291 training images for a roughly twenty-eighty split. The dataset along with CSVs that break the dataset into training and testing portions can be found on Kaggle.[4]

After collection and labelling, TensorFlow[5] was used for object detection. Training of the model followed the tutorial and code provided by the TensorFlow blog.[6] The tutorial used the pretrained SSD MobileNet v2 320x320[7] because of the balance of good accuracy and high speed. I decided to use the same model in order to speed up the training process. After configuration the model was left to train for 7,000 steps with a batch size of 16. At this point I measured performance to check training was working properly. To squeeze out some more performance the model was trained for another 3,000 steps. In total, the model was trained 10,000 steps which is equivalent to about four hours of training in Google Colab on a GPU.

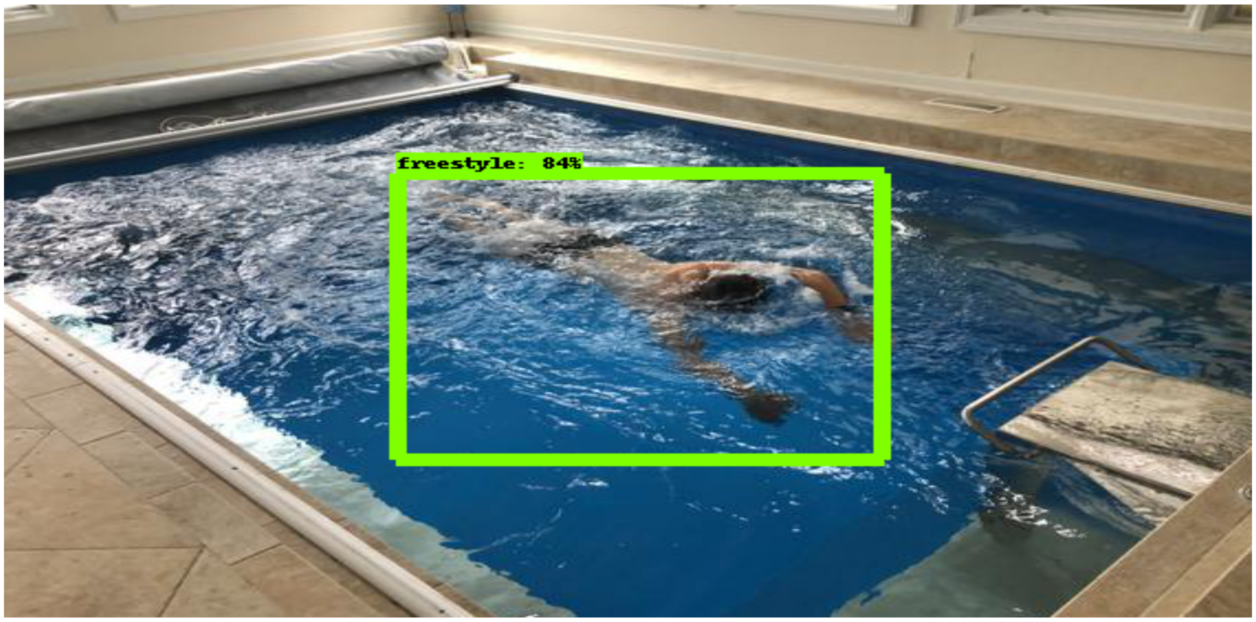
Results

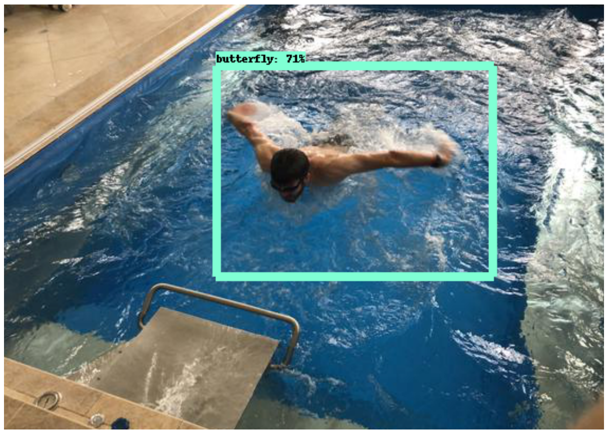
Performance of the model was measured using loss[8] and COCO[9] evaluation metrics. The model has a total loss of 0.570864. This can be subdivided into localization loss, classification loss, and regularization loss. Localization loss comes from differences in predicted and expected bounding boxes. Classification loss comes differences in predicted and expected classes. Regularization loss comes from the network's regularization function. The model has a localization loss of 0.183671, a classification loss of 0.208722, and a regularization loss 0.178470. With an intersection over union equal to 0.5:0.95 the model has an average precision of 0.539 and an average recall of 0.567. Full readouts of the model's loss and accuracy are contained in screenshots within the results folder.

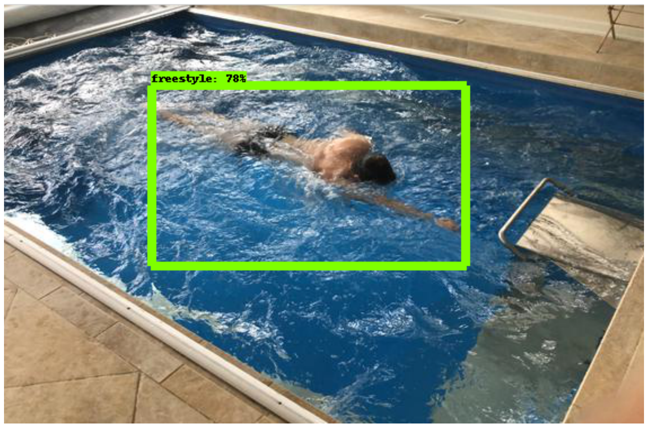
[3] https://github.com/TannerGilbert/Tensorflow-Object-Detection-API-Train-Model/blob/master/xml_to_csv.py
[4]www.kaggle.com/dataset/3122f8d0287f6094b28d66e36b216b3a278eae858b6c7cb10f821010825397a2
[5] https://www.tensorflow.org
[6] https://blog.tensorflow.org/2021/01/custom-object-detection-in-browser.html
[7] https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md
[8] https://developers.google.com/machine-learning/crash-course/descending-into-ml/training-and-loss
[9] https://cocodataset.org/#detection-eval

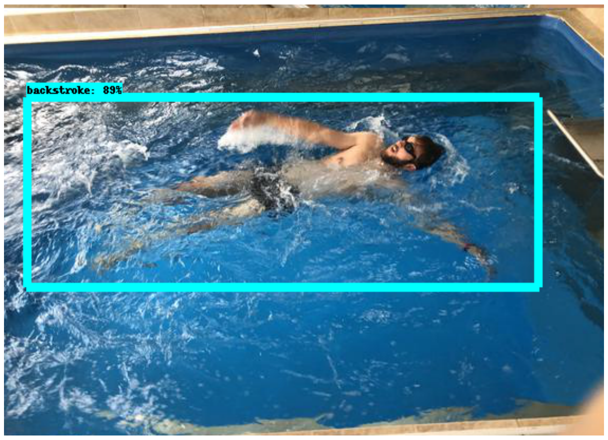
Sample of Correct results

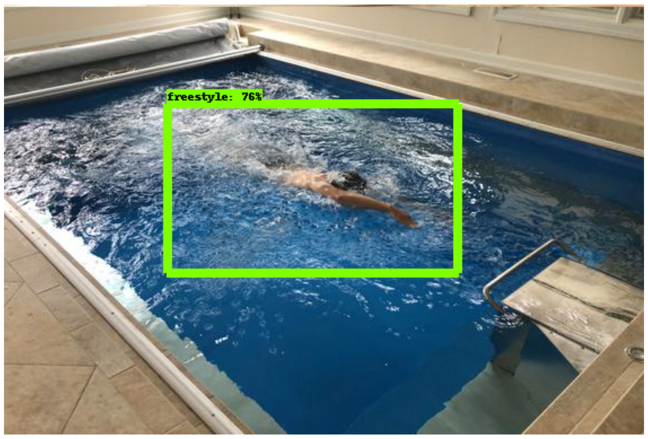
Testing consisted of a variety swimming strokes and angles. Below are a sample of correct results to illustrate the model in action.

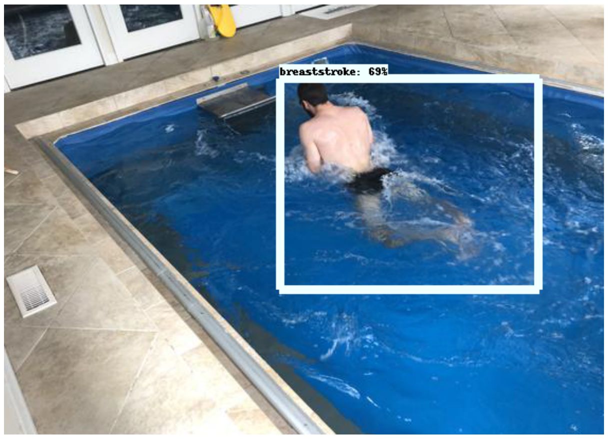


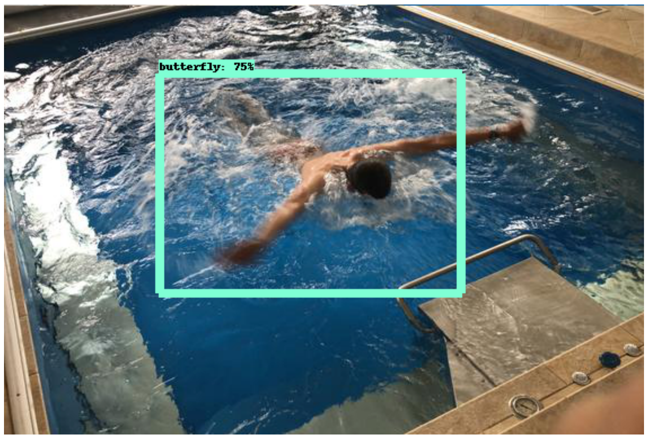






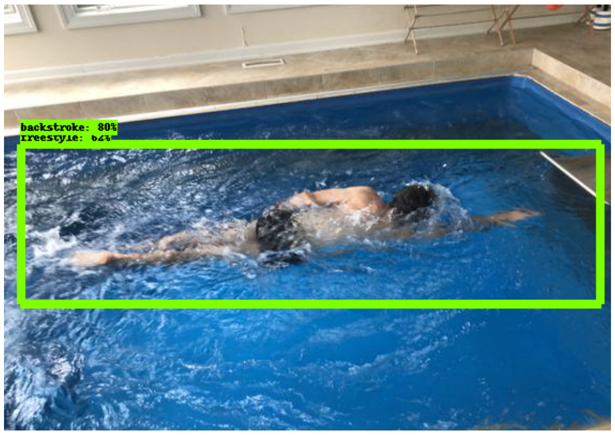


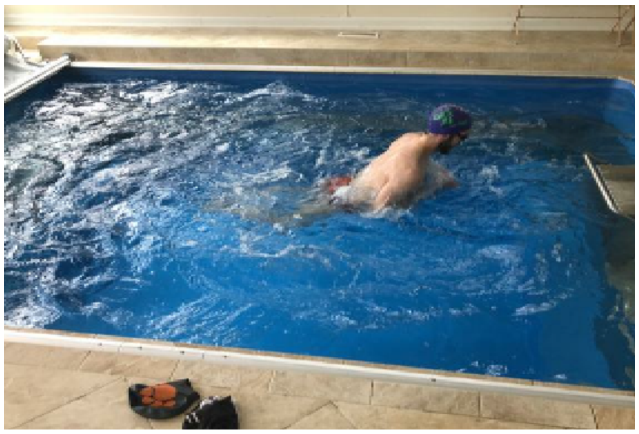




Sample of Incorrect results

Anecdotal evidence from experimentation with the model showed a few key weaknesses. The model sometimes struggled to differentiate between backstroke and freestyle from a side angle. The example below shows a case where the model guessed both backstroke and freestyle. The model sometimes failed to recognize any swimmer in images of breaststrokers. I believe these issues could be fixed if more images of the non-freestyle strokes were added to the dataset.





Implementation Details and References

The code was written using Python and the model trained using TensorFlow. I followed tutorials and used code from Hugo Zanini on the TensorFlow blog[10] and Gilbert Tanner's "Creating your own objected detector" posted on *towards data science*.[11] Image labeling and bounding boxes was completed

[10] https://blog.tensorflow.org/2021/01/custom-object-detection-in-browser.html
[11] https://towardsdatascience.com/creating-your-own-object-detector-ad69dda69c85

using labelImg.[12] The uploaded files contain a readme with more information about training and executing the model.

[12] https://github.com/tzutalin/labelImg
☺