Final Report for PoolPal
Drowning Detection Model

In this report, the Springboard Data Science team hopes to share the results of an exploration into generating an object detection model which will differentiate between those who are drowning and not drowning. This is a complex problem which requires the usage of complicated neural networks. We will discuss our approach to begin building a proof of concept approach to the problem and offer some recommendations for future development.

Problem: Given footage of a public pool, can we design an object detection model which can identify people who are drowning vs those who are not drowning.

Data: Public video data was given by PoolPal through a Github repository (here). The categories of video were “not drowning” (left) and “drowning” (right). Below are example screenshots of what the video sources looked like.

Methodology

Environment Set-Up

We started by creating a new environment designed to utilize the TensorFlow Object Detection API. The exact components can be found in a requirements.txt file within the ANTLER github repository. However, the following versions of critical components were used: Python 3.9, Tensorflow 2.5.0, CUDA 11.1, and CuDNN 8.1.0. We also made sure to install LabelImg for future labeling. After the environment was created, we collected images of various frames within the videos. The images (as seen above) are what we used to create labels.

Labeling and Tensorflow Object Detection API

We utilized LabelImg to begin labelling our pictures. We created tight bounding boxes focused on the actions being performed by the targets within each image. For the not drowning group, we created small bounding boxes on targets which were clearly not drowning and focusing on keeping the box as close to the human target as possible. For the drowning target, the bounding box was somewhat more nebulous. Due to the potentially large area that could be covered by the target, we sometimes had boxes outside the “human” but encapsulated important information such as splashes.

After labeling, we split the image groups into both a “training” and “testing” group. The goal was to use the testing group as our validation data set. In order to perform the training of models, we must finish the set-up of the TensorFlow Detection API. We downloaded the TensorFlow model garden and compiled the protobufs for each model. This allows our local environment to utilize the model garden which contains a wide variety of pre-trained models which can be re-trained from scratch to focus on our target variables.

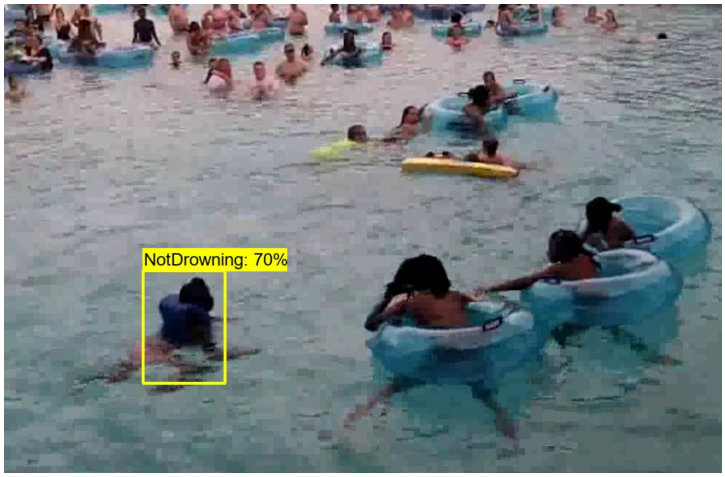
Following the installation of the Tensorflow Detection API, we must prepare our image labels. We created a label_map.pbtxt file which contains the labels for the classes. This matches up with the class labels utilized in LabelImg while annotating. In each “training” and “testing” group we ran a script which converts the XML labels into the appropriate tensorflow record file. Through this process, we have finished the initial set-up for the model and are now ready to begin training.

Model Training

We trained and tested multiple models. Due to the lack of a powerful GPU, some models were limited in their batch size. With a stronger GPU the training sizes may improve and thus so will performance. The following models were trained:

- EfficientDet d0 COCO17
- SSD MobileNet V2 320x320 FPNLite
- SSD MobileNet V2 640x640 FPNLite
- SSD Resnet50 FPN
- SSD Resnet101 FPN

Of these models, we found that the EfficientDet d0 & MobilenetV2 640x640 FPNLite were the best at providing meaningful results. EfficientDet is quite stringent and Mobilenet V2 640x640 FPNLite would be better for recall. I believe that MobileNet was somewhat more sensitive to “drowning” targets but could reasonably detect “not drowning” targets. You can see some of the results below of the object detection. Some issues we found was the over-generalization of what is considered “drowning” with large areas being covered as “drowning” and the lack of “not drowning” boundaries being drawn. These are early development proof of concepts and below, we explore some future developments.



Final Thoughts and Future Development

Overall, we feel that there is promise with the model development that we have had so far. The training files can be found within the GitHub though due to file constraints, the models would need to be retrained in order to receive the checkpoint data for future training. There are a few future improvements that we believe could greatly improve the model.

Using 3D CNNs

Change approach to involve 3D CNNs. Utilizing 3D CNN would allow the model to look at frames as ‘actions’ instead of frame-by-frame analysis. This may prove to be a more powerful methodology but requires great computing power. Tensorflow offers the ability to utilize 3D CNNs and can be used to train the model. The exact design and finer details would take further study and development for these tasks. It may also be challenging to deploy and productionize such models due to their size and complexity.

Improved Training Pipeline

Due to the stark difference in “not drowning” vs “drowning” images, there may be some confounding occurring within the training process. We see that the not drowning videos are wide shot, with a large group of people whereas the drowning videos are tightly cropped. Thus, we propose an additional step to our training pipeline. First, we train a more generalized “human” detector for the “not drowning” videos. Using that trained detector, we can crop out images of people and make sure to generate those at a similar resolution to the “drowning” images. Then we can re-annotate the images that are now on a similar scale of resolution then retrain the models.

Final Thoughts

I think there is great promise in using object detection and we have made good steps towards building a model which may be functional. With some more people hours and a dedicated team, I believe that this can truly be something that serves a great purpose.