Multiple Object Tracking for Marine Science

California Polytechnic State University, San Luis Obispo, CA

Senior Project, Winter and Spring 2022

Author: Nicholas Wachter

Advisors: Franz J. Kurfess, Dongfeng Fang

Table of contents

Abstract

As drone and computer vision technology has been improving, many fields of study have been quick to utilize it to improve the accuracy and ease of data collection. The combination of the two technologies is perfect for surveying large areas and identifying features of interest. Marine science utilizes these technologies for activities such as animal tracking and population counting. I am working with the Drones for Marine Science research group at Cal Poly who want to build a fleet of drones that will fly out over the ocean to identify and track various marine animals. My role will be to build a real-time marine animal identification and tracking system that they use on the drone fleet. My overall goal of this project is to gain an understanding of various techniques for machine learning, to build skills in the different frameworks that are used in the machine learning and deep learning field, and utilize those skills and understanding in a meaningful way: to help the marine science community through this research project.

Background

The hardware component that I am working with in this project is the Nvidia's Jetson Nano[1]. This is a powerful computer with specialized hardware for neural networks that utilize parallel computing. It is small and light weight which is what the aforementioned research project needs if we are going to put it on a drone. The model that I trained and utilize in my implementation is trained off of the YOLOv5[2] real-time object detection model. YOLOv5 was originally trained on the COCO dataset and using the machine learning framework pytorch, I was able to further train the model on my own custom dataset. I use YOLOv5 because of its high accuracy and because it takes up relatively low memory space which is important for being able

to fit the model onto the Jetson Nano. Once the model was created I was able to utilize it in the tracking system which employs a technique known as deepSORT. I will explain deepSORT in more detail later but the reason I chose this system design was because it was what I determined to be the best balance of accuracy and the ability to be a real-time system.

Related Work

Within the research group itself we already have a computer vision system currently implemented. Similar to my model, the team before me performed transfer learning by using the darknet YOLOv4 pre-trained weights to create a convolutional neural network to classify multiple objects in the video stream. This is sufficient for the prerequisite of performing object detection and classification but my project added the implementation of tracking as well as improving the model that it uses. Outside of our group, NORCE (Norwegian Research Center) uses machine learning for automated fish detection using cameras that are placed underwater[3]. They soon plan to use drones like we are doing in this project. In Australia, the drone company Westpac Little Ripper has partnered up with researchers at the University of Technology Sydney to create a shark detections system[4]. They are analyzing both video and still shots from the drone camera and then alerting nearby beach goers when a shark is detected.

Features, Requirements, and Evaluation Criteria

Features include the following. The ability to take a live stream video, run it through the tracking system and output its analysis. The analysis will include object classifications, bounding boxes around the entities in the video stream, and an id number. Each entity is given one id

number and it is the goal of the system to correctly maintain which entities have which id numbers and to only assign a new id number when a new entity enters the environment.

There is a physical requirement that the model must be able to fit on a machine that can physically be carried by the drones we are using. There is a functional requirement that it must be able to do all the computations in real time as opposed to saving the video and processing it later. The performance requirement has no strict value but the model must be ‘reasonably fast and accurate’.

The following metrics will be used to evaluate the project. With regards to if it fits on a machine that goes on the drone and if it does computation in real time, that is a simple yes or no, pass or fail. Same with deciding if it has the ability to perform tracking in real time. With regards to performance, I define ‘reasonably fast and accurate’ to be ‘near human performance or better’. In terms of accuracy, for each frame will calculate the error of the frame with frame_error = 1-(number of errors / number of entities). Then calculate the error for the video as sum(frame_error) / number of frames. An error is defined as a false positive, false negative, inappropriate id switching, or incorrect classification.

System Design

Deep Learning and Image Classification

No matter what tools are used for the implementation of this project, the underlying mechanism being used will be deep learning. Deep learning is great for taking large amounts of data for pattern recognition which is exactly what I need for object recognition and tracking. Let's take a look at the inner workings of deep learning.

The main goal is to create a mapping such that the contents of an input layer can be identified by the labels of each neuron in an output layer. This is done using what is called an artificial neural network. An artificial neural network is a series of layers that connect to each other. Each layer contains a series of neurons which hold a number between zero and one which represents that neuron's activation level. The higher that number is, the more ‘activated’ it is considered. Each layer is meant to analyze an abstract component from the data. When data is given to the model, it ‘activates’ areas of the neural network differently depending on what the input data contains until finally it reaches the output layer. Whichever neuron in the output layer is the most ‘activated’ is considered the prediction for what the input data contains.

In order to get the neural network to capture different patterns, we assign a weight to each of the connections between each neuron of adjacent layers. These weights are multipliers then are used to calculate the value for a neuron by multiplying the weight by the previous neuron's value. Each neuron to neuron pair has its own unique weight. Weights can be both positive and negative values. For a given layer, n, we can calculate the value of each neuron using the values of the previous layer’s neurons and the weights that connect it. Each neuron value and weight are multiplied together, summed up, and then passed into a sigmoid function to compress the value to be between zero and one. The sigmoid function is simply:

$$\boldsymbol{\sigma}(x) = 1/(1 + e^{-x})$$

The sigmoid function makes the activation value a measurement of how positive the weighted sum is. For the set of neurons $\{a\}^m$ and set of weights $\{w\}^m$ neuron a_i of layer n is calculated as:

$$a_i^n = \boldsymbol{\sigma}(w_0 * a_0^{n-1} + w_1 * a_1^{n-1} + \ldots + w_m * a_m^{n-1})$$

We may only want the calculated neuron to light up when it is meaningful to do so therefore we can introduce a bias value into the equation to check if the weighted sum is greater than some bias value, b. This gives us our final formula for neuron a_i:

$$a_i^n = \boldsymbol{\sigma}(w_0 * a_0^{n-1} + w_1 * a_1^{n-1} + \ldots + w_m * a_m^{n-1} - b)$$

However, sigmoid functions aren't used as much anymore. A more common alternative is the rectified linear unit (ReLU) function. This function finds the max between zero and the weighted sum. This is how a neural network operates at a lower level. To give a quick overview, the input layer is the first layer in the chain of neural layers. These values are used with a combination of weights and biases to calculate the neuron activation levels for the next layer. The same process is done for the next layer and the next layer until it reaches the output layer. Then in the output layer we can say that the neuron with the highest activation value is the model's prediction. Let's take a look at how we can create the weights and biases needed to make accurate predictions, otherwise known as "training the model".[5]

The goal of training a model is to adjust the weights and biases so that the appropriate neuron in the final layer has a high activation value. We first initialize all of the weights and biases to be random. The cost function is a function which takes an activation value in the output and finds the squared difference from the expected output value for that neuron. The expected output value is one for the intended prediction and zero for all other values. The cost value for a training example is then the sum of all of the cost values of each output neuron. The cost function for a training example looks as follows:

$$\text{Cost} = (a_j - 1)^2 + \Sigma_0^i (a_i - 0)^2 \text{ where } a_j \text{ is the neuron intended to be predicted}$$

The cost is small when the model is closer to outputting the intended activation values and larger when it is further away. We can use the average cost across all of our training examples as a metric for performance. This is where making changes to our model comes in. We will need to change our weights and biases in the appropriate way so that we reduce our cost value. To do this we use a technique called gradient descent. For our purposes we just need to know that gradient descent finds local maxima so what we will do is take the negative gradient descent of every weight and bias to find the local minima. The idea then is to find the average negative descent for each weight and bias in every training example and adjust them accordingly. This is a great idea in theory but in practical implementation the computational complexity is impractical. What is actually used is a technique called stochastic gradient descent. In this technique the training examples are randomly shuffled, divided into smaller “batches”. The negative gradient descent is then calculated and averaged using these smaller batches. The result is that each adjustment is going to be less accurate but because it is generally in the right direction and performed many times, we get nearly the same result with a massive computational speedup. Let’s talk about how these gradients are computed, a technique known as back propagation.[6]

In back propagation we start at the output layer. We know how we want the output layer to look from our cost function. We also know that the output layer is a function of the previous layer and the weights and biases that connect it. So then we can change our weights, biases, and previous layer activation values so that the output layer is adjusted appropriately. Now that we know how the layer before the output layer should look, we can use the same tactic to adjust the weights, biases, and activation values from the layer before that. We do this repeatedly until we get back to the first layer of the model. One thing to note is that since the weights and activation

values are multiplied in the calculation of the next neuron's activation value, they must be changed with the proportion of the other in mind. This is how we calculate all the changes we want to make for one training example. We can do this for all the training examples in the batch, save which changes we want to make for each example, take the average, and then apply the average change to the model. This process is then repeated for all of the batch groups that were created. The completion of all batches in the training set is called an epoch. It is a common practice to then reshuffle the training data and run many epochs in order to further reduce the cost function.[7]

So to recap, we can create a network of neurons connected by weights and biases to analyze data and make meaningful predictions. In order to train the model, we first calculate the cost value of the network to know how we want the output to change for a specific training example. We can use this value to calculate how we would need to change the weights, biases, and neuron activation levels in the previous to reduce the cost. We go backwards through the model and calculate the appropriate changes at each layer. Then find the average change for a batch and apply it. This technique is known as back propagation. It is important to note that this technique only works if we have the data and the labels to train our model. This is a technique in machine learning known as supervised learning. Other techniques can learn without having prior data with behavior incentives and by analyzing the data to discover patterns on its own. Since the tools and activities I will be engaged in with this project are using supervised learning, I wanted to explain deep learning in that context.

Now that we understand how a machine can learn, we can imagine how a neural network would be able to classify the object in a picture/video. The pixel values from the images are the

input data. As long as the data is labeled, we can use the supervised deep learning techniques to train a model to associate different patterns in the image with the appropriate output labels.

Multiple Object Tracking

The problem of multiple object tracking has two main tasks. The first is to identify objects in a video feed. This includes what the object is qualitatively and where it is spatially located within the frame(s) of the video. This task is relatively straightforward. The process of classification that we discussed in the deep learning section above is part of this recognition process here. Many tools exist to help in the process of creating these detectors on our own and they do a great job doing so. The second and not as easy part of this problem is to be able to say that an object in one frame is the same object that is in another. There are many proposed techniques to solve this problem. Some are better than others. Here I am to discuss some of the different techniques that I have considered throughout this project. While I only use one, discussing different techniques will help to explain why I chose the implementation that I did.

There are two categories that we can put multiple object tracking techniques into: batch tracking and online tracking. In batch tracking, otherwise known as offline tracking, the video being analyzed is always pre-recorded. This is really great because we are able to analyze the movement of objects in the frames relative to not only the current and past frames but also future frames as well. One of the problems that this can solve really well is recovering from occlusions. When an object is occluded it is difficult to recognize if it is the same object or not both during and after occlusion but if we are able to analyze future frames of the video then we can tell where, when, or even if the object comes back into the scene. In theory this is the best way to analyze video data but it is not always the most practical method. Sometimes we need to analyze

footage from a live video stream as is the case for my project. In this case we do not have future frames to analyze. For real time object tracking we use online tracking techniques. In online tracking we process the current frame by looking at the trajectory of objects in the previous frames to guess where they will end up. This ends up being less accurate because we can't correct mistakes and we don't have future knowledge about the video frames but that is the price we pay if we want a real time application. One of the requirements for this project is for the system to be real time so I must use online tracking methods. I wanted to bring up batch tracking methods as well for anyone reading this paper that plans to implement multiple object tracking for applications that may not have that real time requirement.[8]

Before we go into multiple object tracking, let's first discuss techniques for tracking single object tracking so that we have a better understanding of the problem of tracking itself before we have to deal with multiple agents. In a single object we are following one object through a video sequence. Object tracking is a problem of recognizing and matching an object across frames so we first must identify the object. We can do this with the many detectors available to us. Once we can detect the object in each frame we may then track it in video by detecting it throughout all of the frames. This makes it seem like object tracking is a simple task but in reality this approach doesn't work that well. There are so many other components of video such as motion blur and changes in background that throw off our model that we can't rely on this method.[8]

One technique for single object tracking is a technique named "GOTURN". In GOTURN we utilize a siamese neural network (SNN) which runs both the current and previous frame through the same neural network. The layers that process each frame have the same weights and so we extract the same features. We can pass these features to a series of fully connected layers

to compare both the current and previous frames to create a more informed prediction about where the object in the current frame may be located. Using the placement of the object in the previous frame we can crop our search space to a smaller search. Limiting the search has a great benefit of computational speedup but the downside is that we restring the motion of the object. If the object moves too fast or goes out of our search window then we are not able to recover.[9]

Another technique, coined “ROLO’’, separates the spatial and temporal domains. YOLO is the detector model that was used in the development of this technique and it was used in conjunction with a recurrent network hence the term ROLO. It utilizes a convolutional neural network (CNN) to extract features from frames and then passes those features to a long short-term memory network to predict the location of the object in the next frame. It looks for the object in the predicted region, detects it, updates its location, and does it all over again. This is a very similar idea to GOTURN except instead of looking at the the same place for the object to be in, it makes a prediction of where it is going to be based off of past frames which helps mitigate some of the problems that GOTURN was having with objects moving outside of the search area.[9]

Now let’s discuss the problem of multiple object tracking. The biggest problem in multiple object tracking is the correspondence of object identities across frames. If there are multiple objects of the same type, which is often the case, keeping the identities of each object separate becomes a very difficult task. When we have multiple moving objects in frame, the number of occlusions tends to increase which adds to the complexity of the problem. Combining these two problems, when two objects of the same type cross paths and one occludes the other, keeping the identities of the two objects separate is almost impossible. Most of the time re-identifying the objects is required.[8]

One method for object tracking is known as the tracktor method. The main idea is to take a detector and give it tracking capabilities. This method takes in the image, detects the objects within it, and uses regression to make slight adjustments to the bounding boxes from the previous frame. The issue with this implementation, as was alluded earlier, is that there is a problem with being able to claim correspondence of object identities across frames. This is especially a problem in crowded spaces and noisy backgrounds. Occlusions will kill the track of an object. Large camera motion or low frame rates will also cause issues because the regression function is only meant to shift the boxes by a small amount. There are many potential problems with this model but if we have the right circumstances to avoid these problems, this is a great simple solution for online tracking.[9]

Another method I would like to discuss changes the problem statement of multiple object tracking. Instead of keeping track of the movement of an object from frame to frame, this method aims to look at frames as separate instances and match the objects within each frame with a known reference. As discussed previously, losing the identities of objects easily occurs. Being able to re-identify objects as they come in and out of view would be useful and that is what this technique aims to accomplish. We start by training a model to recognize specific objects. This is of course a limitation. We need to know what objects to look for before the video starts but for some applications this is perfectly feasible. Then each frame is analyzed to identify objects from the list of known objects. This helps eliminate the problem of losing identification because we never actually lose objects if we are always looking to identify specific ones. This is the same idea that we see in technologies such as face scanning biometrics. The idea is to give a pass/fail value for if the object in question is known.[9]

One of the most common ways to keep identities separate is by tracking the trajectories of each object separately. There are several ways to do this but the overall idea is similar to that of which we discussed in single object tracking. For each object on the screen, identify it in the previous frame and use the last two frames to predict its location for the current frame then compare the predictions to the real detections actually made and make an educated guess which objects are the same across frames. To make an educated guess on which objects correspond across frames we can use the Hungarian algorithm. The Hungarian algorithm takes a two dimensional list of values, a bipartite graph, and minimizes the effective cost of those values by finding the lowest value in each unique row/column combination.[10] We can use this as an object matching algorithm for matching the identities of our predicted object locations with our actual detected object locations. The bipartite graph of values contains the distances between each predicted and detected object. Then the most likely object correspondence will be ones with the lowest distance which can be computed with our Hungarian algorithm. There are two common scenarios that should be addressed: the situation where we are missing a prediction and the situation where there is no prediction suitable for the match. We then need to add dummy nodes to our bipartite graph which hold some threshold cost to determine if the options presented to us are worthy of matching to previously detected objects. This helps us avoid type 1 errors. We then are forced to accept that if no threshold value is met then we don't include the object as detected for the subsequent frame.[9]

A more advanced technique is the deepSORT algorithm. This system actually utilizes the Hungarian algorithm discussed above. There are three parts that make up deepSORT: Kalman filtering, the Hungarian algorithm, and a neural network for appearance detection. DeepSORT is an extension of the Simple Online Real Time (SORT) tracking algorithm where SORT is

composed of Kalman filtering and the Hungarian algorithm so adding the neural network is what makes it “deep”. Since we already discussed the Hungarian algorithm I am only going to discuss Kalman filtering and the appearance tracker here. Kalman filtering is an algorithm that takes a series of temporal measurements and utilizes joint probability distribution to create an estimate for future time frames.[12] For Kalman filtering, in the context of object tracking, we have 8 properties that we keep track of per object detected. There is the center point of the box composed of 2 dimensional cartesian coordinates points, the aspect ratio, the height of the image, and then the constant velocity of each one of these components that are calculated by the change from the previous frames. The Kalman filtering algorithm is applied using these 8 properties as inputs and this results in an estimated location in the next frame. In deepSORT, this is the trajectory prediction algorithm for the Hungarian that was discussed above. Together, Kalman filtering and the Hungarian algorithm provide a way to make a strong educated guess which objects detected in one frame correspond to objects detected in the next. Alone, this is already a great technique for simple applications. Problems arise, however, when objects are occluded, velocities are not constant, the viewpoint angle of the camera changes, detections fail, and a multitude of other factors which all inhibit the ability to keep track of each individual object. This is where deep learning is able to help. The component that makes deepSORT such an effective algorithm for keeping track of objects is the addition of a neural network that keeps track of objects by appearance. The general idea is to use a convolutional neural network like the one used for the detector to identify unique features for each object detected. This can be done by removing the output layer and taking the outputs of the feature layers instead as a unique identifier for each object. Then we can compare the estimated corresponding IDs from this neural network to the ones estimated by Kalman filtering and the Hungarian algorithm to

improve the accuracy of ID tracking between frames. These appearance identifying neural networks have actually shown to be highly accurate. So much so that in some instances, IDs were able to be kept track of when no correspondence thresholds were met in the Hungarian algorithm. The original creators of the deepSORT algorithm[13] report that adding this appearance tracking methodology increased the performance of multiple object tracking by 45% in comparison to the basic SORT method. The only downside they found was a slight decrease in processing speeds but they were still fast at processing relative to other multiple object tracking techniques. The deepsort method of course was not as accurate as some of the batch processing techniques but it was the best combination of accuracy and processing speeds that I could find for online techniques. [11][13]

There are many ways to perform multiple object tracking and each one has its benefits and drawbacks depending on the implementation that it is being utilized in. For my purpose, I have to have an online system, but I do have a clean background and I don't need to worry about quick and sudden movements. Therefore I can choose one of the simple online approaches. That is why I chose the deepSORT method. It has the best accuracy for an online system and does not take up so much space such that I couldn't put in on the drone's hardware.

Implementation

Initial implementation

In my first implementation I did the deepSORT technique as described in the system design section. I took the weights file that was created by the Drones for Marine Science machine learning team before me, created a model from the weights, and then used the deepsort algorithm created by github user 'theAIGuysCode' and modified it to work with the model I

created.[14] I began testing it on some of the videos that were received from CSU Long Beach and got some decent results. The program was able to identify various objects in the video and keep track of their IDs with decent accuracy. However, there is a lot of room for improvement. There were a couple cases where objects were not identified when they should have been. Sometimes this was when they were a dark object and they were in areas of the ocean that were also dark. There were times when objects were too deep into the water and just simply weren't defined enough to be recognized. The most disappointing part was that there were times when the objects appeared to be in reasonable conditions to be recognized and they just simply were not recognized. Another type of problem that was occurring was the misclassification of objects. There were instances when sharks were being identified as persons, persons were being identified as seals and sealions, and seals were being identified as sharks. I would say that for the most part the classifications were correct but there were definitely instances where it was not which I plan to fix. Another error that I observed was that IDs were switching, even when the objects were not being occluded or leaving the frame. What I noticed was that this was happening the most when the drone was changing position or the camera was moving thus changing the angle of the shot. In the deepSORT algorithm one of the ways that it keeps track of IDs is through appearance as discussed in the system design section above. So when the angle of view changes, the appearance may change as well. Overall, I think the best thing I can do to improve the system is to train a better initial detector. This will help in reducing the amount of missed detections and incorrect classifications which I am hoping will result in a reduction of ID switching.

Final implementation

I decided that the deepSORT method was sufficient for what I was trying to accomplish so I am using it for my final implementation. To best improve the system I decided to train my own detector. I took 10 videos, split all of the frames into jpgs. Then I randomly chose 30 images from each video and compiled them into one set. I went through all 300 images and labeled them by hand. Then I used the website “roboflow.com” to add additional images to the set by blurring or rotating them which gave me a total of 633 images for training, 60 for validation, and 30 for testing. Then I trained a detector off of the YOLOv5l.pt weights using pytorch. The result was a detector that worked really well! I had access to a dataset from the previous group that had over 3500 images and I trained a detector on those as well but there were problems with the detector. For one, I couldn’t fully train it with pytorch because it took too long and my google colab instance kept running out. So then I trained it for as long as possible and took the best weights that it had produced so far but they were terrible. Originally I built this model using Tensorflow and had also tried training based on the large data set but that model was terrible as well. I had to convert the text files that labeled the data to the xml files that my program was expecting so I might have done that incorrectly. I did go back and check that the bounding boxes were correct so I am not sure what the problem was. Being able to train on that dataset would have vastly improved the performance of the model. After the model was completed, I utilized part of the track system created by github user “mikel-brostrom”[15]. I used this program over the other one because it utilized the YOLOv5 weights for the appearance detector instead of YOLOv4. With some slight adjustments, and adding in my model, I was able to successfully perform multiple object tracking using my new detector.

Evaluation

Now I will discuss how my program performed to meet the criteria that I had set out at the beginning of this project. With regards to the memory requirement, the Jetson Nano's that the program will be running on have 2 Gigabytes of storage and my program is 1.18 Gigabytes on disk so I have successfully met the memory requirement. While for demonstration and evaluation purposes I have been running the system on videos, the system has the capability to work on a video stream. Therefore I have met the online requirement. Finally, I would argue that the system is reasonably accurate. In the videos that I was testing on, my system never scored below 80% accuracy and since the intended activity of shark spotting doesn't require near perfect levels of accuracy, the system is more than accurate enough. In comparison to the previous model, my model almost always had better performance. In the videos where there were many entities, my model did a better job at consistently identifying and tracking them. The cases where my model did not perform as well were when there was just one shark in view, my model tended to have a few more false positives than the other model. Considering that the intended use case is going to be in an environment with multiple entities, I believe that my model would be the preferable option for deployment. Overall I am pleasantly surprised to see how my project is performing.

Future Work and Conclusions

As far as the activity of shark tracking itself goes, I don't believe there are additional aspects that need to be implemented. Additionally, while improving the system is most certainly possible, I don't believe that efforts to do so are necessary. There are, however, extensions of this project that would be interesting. Another student and I were discussing the possibility of

performing behavior analysis based on the swimming patterns of sharks. The basic idea would be to analyze the video stream after flight and feed the change in shark positions as a feature vector to a recurrent neural network and classify behavior that way. Another application would be to do population counting. I'm not sure how to solve the problem of entities leaving and coming back into view but it would be really interesting to see a project where a drone could fly over an area and return an accurate count of the population of a species. Lastly, while the purpose of this project is to track the movement of sharks, the same idea could be applied to other animals such as dolphins, sea lions, and sea otters.

Overall, while the main goal of this project was to create a multiple object tracking system, my underlying goal was to learn how to leverage machine learning for computer vision and to simply learn more about the field of computer vision in general. I can proudly say I was able to accomplish both of these goals. From self-driving cars to marine science, computer vision has many applications to improve existing systems and to help create new ones. The applications that this field has are going to help improve the lives of many. There was a project that I recently learned about where the world bank is developing computer vision and other AI technologies for risk management of national disasters in impoverished nations[16]. Technology and technology like this make the world a better place. If you are reading this, I hope that this field and the potential difference you can make in it excites you. Thank you for reading.

Sources

[1]

"Jetson Nano Developer Kit," *NVIDIA Developer*, 14-Apr-2021. [Online]. Available: https://developer.nvidia.com/embedded/jetson-nano-developer-kit. [Accessed: 2022].

[2]

F. C. Akyon, "Large scale object detection & tracking with Yolov5 package," *Medium*, 15-Oct-2021. [Online]. Available: https://medium.com/geekculture/large-scale-object-detection-tracking-with-yolov5-package-c8eca66b80b6. [Accessed: 2022].

[3]

W. L. Michaels, N. O. Handegard, K. Malde, and H. Hammersland-White, Eds., "Machine Learning to Improve Marine Science for the Sustainability of Living Ocean Resources," *NOAA Technical Memorandum* , Nov-2019. [Online]. Available: https://spo.nmfs.noaa.gov/sites/default/files/TMSPO199_0.pdf. [Accessed: 2022].

[4]

"Sharkspotter: A world first in shark detection," *University of Technology Sydney*, 24-Jan-2020. [Online]. Available: https://www.uts.edu.au/news/tech-design/sharkspotter-world-first-shark-detection. [Accessed: 2022].

[5]

But what is a neural network? | *Chapter 1, Deep learning*. 3Blue1Brown, 2017.

[6]

Gradient descent, how neural networks learn | *Chapter 2, Deep learning*. 3Blue1Brown, 2017.

[7]

What is backpropagation really doing? | *Chapter 3, Deep learning*. 3Blue1Brown, 2017.

[8]

V. Meel, "What is object tracking? - an introduction," *viso.ai*, 04-Mar-2022. [Online]. Available: https://viso.ai/deep-learning/object-tracking/. [Accessed: 2022].

[9]

CV3DST - Object tracking. Dynamic Vision and Learning Group, 2020.

[10]

"Hungarian algorithm," *Wikipedia*, 07-Mar-2022. [Online]. Available: https://en.wikipedia.org/wiki/Hungarian_algorithm. [Accessed: 2022].

[11]

S. R. Maiya, “DeepSORT: Deep learning to track custom objects in a video,” *AI & Machine Learning Blog*, 24-Apr-2020. [Online]. Available: https://nanonets.com/blog/object-tracking-deepsort/. [Accessed: 2022].

[12]

“Kalman filter,” *Wikipedia*, 11-Mar-2022. [Online]. Available: https://en.wikipedia.org/wiki/Kalman_filter#Underlying_dynamical_system_model. [Accessed: 2022].

[13]

“SIMPLE ONLINE AND REALTIME TRACKING WITH A DEEP ASSOCIATION METRIC” N. Wojke, A. Bewley, and D. Paulus, rep., Mar. 2017. Available: https://arxiv.org/pdf/1703.07402.pdf. [Accessed: 2022]

[14]

theAIGuysCode, “TheAIGuysCode/yolov4-deepsort: Object Tracking implemented with Yolov4, DeepSort, and tensorflow.,” *GitHub*. [Online]. Available: https://github.com/theAIGuysCode/yolov4-deepsort. [Accessed: 2022].

[15]

mikel-brostrom, “mikel-brostrom/Yolov5_DeepSort_OSNet: Yolov5 + Deep Sort with OSNet,” *GitHub*. [Online]. Available: https://github.com/mikel-brostrom/Yolov5_DeepSort_OSNet. [Accessed: 2022].

[16]

“Machine Learning for Disaster Risk Management.” *Worldbank.org*, World Bank, 2018, https://documents1.worldbank.org/curated/en/503591547666118137/pdf/133787-WorldBank-DisasterRiskManagement-Ebook-D6.pdf.