Abstract—This study is aimed at developing an online service by implementing object detection remotely to surveillance cameras. We propose both real-time and none real-time implementation method of services. Open source code was analyzed, modify and implemented in order to analyze what type of complications we can run into when providing the fastest and most accurate service possible. Three convolutions neural networks architectures implementations were analyzed using only a CPU and Python3.

Index Terms—component, formatting, style, styling, insert

I. INTRODUCTION

There have been many advancements in security systems within the past decade. A major contribution to these advancements are recent developments in various subfields of Machine Learning. Object detection is one of these subfields and is also the focus of our project. Security surveillance systems can use object detection to detect suspicious activity, vehicles involved in a crime, dangerous objects, and other security concerning matters.

Some difficulties arise when trying to select the right architecture for the job. Deep Learning based surveillance systems use both real-time and non-real-time object detection. Our project explores different methods of Object Detection along with their potential uses and downsides.

When it comes to real-time object detection surveillance systems, we want speed. If our network architecture is too complex this increases the amount of time it takes for the network to process a frame in a video. With a low fps (frames per second) rate, the system risks missing crucial parts of a crime or activity. Unfortunately, the downside to using a less complex architecture to increase the fps, is that it decreases the accuracy of detecting objects. While a complex architecture is detrimental to real-time object detection, it serves as a great option for post processing of security footage. For non-real-time object detection applications, high accuracy is most desirable.

Our goal for this project is to find an optimal method for real-time object detection. The method we want is one that can achieve decent fps while maintaining high accuracy. In addition, we also aim to create an online application able to run high accuracy object detection.

II. OBJECT DETECTION

Deep Learning is a branch of Machine Learning which is based on the layers used in artificial neural networks. Deep learning architectures are used in a variety of fields, including computer vision. [9] Object detection is a subfield of computer vision that deals with detecting semantic objects in images or videos and dividing them into their respective classes (human, vehicle, building, etc.). [14] Various types of network architectures are used in object detection and have their own advantages and disadvantages. Popular architectures include, YOLO (You Only Look Once), Faster R-CNN (Region-based Convolutional Neural Network), Mask R-CNN, and various other RPN (Regional Proposal Network) based networks.

Object detection works by running image classification alongside object localization. Image classification simply classifies the image by extracting features and running it through a ConvNet (Convolutional Network). Object localization deals with locating where an object is within a given image. Localization is responsible for generating multiple regions to run image classification on. The differences amongst the architectures we’ve mentioned lies mostly with the method they use for region proposals.

A. R-CNN, Faster R-CNN and Mask R-CNN

Choosing when and how to implement region proposals largely affects the speed of object detection. R-CNN, a predecessor to Faster R-CNN, uses selective search to generate region proposals before running feature extraction. In doing this, R-CNN is left to run feature extraction and image classification on an excessive number of regions, increasing the time it takes to run both inference and training on the model. [8] Faster R-CNN drastically reduces the run time by changing when the regions are proposed. Feature extraction is done before the RPN proposes its regions. This lets the network run region proposals on a high-level feature map. By doing this, the RPN is able to use the high-level features to more accurately generate the ROIs (Regions Of Interest). Less regions are proposed due to the semantic rich feature map functioning to filter out irrelevant features. [11]
Mask R-CNN is an extension of Faster R-CNN. It adds a branch to generate a mask for an object in parallel with the branch for bounding box recognition. It also uses the ROIAlign method instead of ROI Pooling to more accurately preserve spatial locations in an area. Preserving the spatial location is important when using a FPN (Feature Pyramid Network) because the upscaling and downscaling of an image can cause an ROI to encase the wrong region. Mask R-CNN can be split into two stages. The 1st stage is a light-weight RPN which scans the FPN top-bottom pathway and proposes regions within the image where objects may reside. The 2nd stage is another neural network that takes the proposed regions and assigns them to specific areas of a feature map level. This is done by using the ROIAlign technique to locate the relevant areas in the feature map. After assigning the regions, it scans those same areas and then generates the object class, bounding box, and mask.

B. YOLO and Tiny-YOLO

YOLOv3 (You Only Look Once) has an architecture similar to that of a FCNN (Fully Connected Neural Network). It also uses an FPN for high-level feature extraction. RPNs run detection and prediction on multiple regions. While it does yield exceptional accuracy, it adds to the time it takes to run inference on the image. YOLO tackles this issue by using a different technique. It passes the image once through the FCNN which splits the image \((n \times n)\) into several \((S \times S)\) grid cells. YOLO then generates a class probability map along with bounding boxes and confidence scores. Unfortunately, this model only predicts one set of class probabilities per grid cell. For that reason, YOLO runs into an issue with instance segmentation on smaller objects as well as clusters of objects. Tiny-YOLO is a descendant of YOLO dedicated to run faster at the sacrifice of its accuracy. This is achieved by scaling down the size of its tensors. Due to the decrease in semantics as a result of a lower resolution, it takes less time to run detection on an image.

III. None Real-time Object Detection Website Implementation

The website design we propose is aimed at providing object detection service by allowing users to upload images and videos to the server, processing it and making the results available for the user to access. However, to simplify our implementation we used only images. Two of the of the problem we will initially be considering are what language or framework to use, how to processes requests from multiple consumers, and provide asynchronous communication to show progress?

A. Django, Celery and RabbitMQ

Since the code we are working with is in python, we will be using Django. Django is a web framework that is written in python, most of all it is very secure and scalable. Django also allows developers to start developing quickly, by generating Django projects and apps quickly command.

Celery is an asynchronous task queue/job queue based on distributed message passing. It is focused on real-time operation, but supports scheduling as well [1]. RabbitMQ is an open source message broker [2].
By combing Django, Celery and RabbitMQ, we can use RabbitMQ to communicate between Django and Celery. A user on the website will send a task request, RabbitMQ forwards the message to Celery’s queue. Once the task is in queue it waits for its turn while other tasks are finishing. Celery has work nodes that processes a task at a time, thus the number of task we can process at a time is limited to the number of workers. But increasing the number of workers means more CPU usage, thus there must be a balance between CPU and number of workers. In our implementation we used only one worker. The workers then update their progress to a database, allowing users to check their task progress by checking the database.

Fig. 5. The diagram shows how Django, RabbitMQ and Celery communicate. Django requests a task to RabbitMQ, RabbitMQ passes down the requests to the Celery workers and performs the task when ready.

IV. REAL-TIME OBJECT DETECTION IMPLEMENTATION

Our implementation of real-time object detection was implemented as simple as possible. The ideal device for this implementation would be a camera, that can connect to a network and allowing us to process the captured frames and displaying the processed images live. To simplify our implementation, we don’t consider concurrent users or multiple cameras. And, analyze the accuracy and average time to process each frame.

We developed a simple Android application, using Unity, that uses the camera on the phone to capture a frame and updates an image on some machine through http by overwriting at some time interval. A Google Cloud virtual was used in this case, and using a VNC (virtual network computing) viewer displayed the results in real-time.

V. RESULTS

For the web application, we decided to go with Mask R-CNN due to its high accuracy. The figure below is a snapshot of the website. Here, we can see the form that is used to upload an image to the site. Once the image is uploaded, the server uses the pre-trained Mask R-CNN model to run object detection and generate a new image. When finished, it saves the new image to the database and allows users to view the results. MaskR-CNN took an average of 7.5 seconds to run inference on the given image and display back to the user.

For our real-time implementation, we worked with YOLOv3 and Tiny-YOLOv3. YOLOv3 took an average of 0.75 seconds per image. Tiny-YOLOv3 took an average of 0.65 seconds per image, but was far less accurate than the YOLOv3.

VI. FUTURE WORK

The work we did gave us a lot of ideas for software products, the problems that we would be running into and ideas on how to solve them. One of these problems was how to deal with the multiple users, if many users are it can make are servers very slow. In the case of the website that we implemented, finding a balance in the amount of celery workers and how many pool process is the key for making the website efficient.

For real time object detection, we need in to insure to process images faster at the expense of less accuracy. In real time object detection we find that it’s not too bad if the object is not detected at exactly every frame. A miss per frame rate could be evaluated to satisfy some guaranteed value. In the case of providing real time object detection to multiple users, GPU would have to be utilized in order to process the images faster. For large number of users we have consider to use a
file distribution system, like Hadoop, in order to process many frames from many users as evenly as possible among many servers and GPUs.

A. Surveillance

One of the many motivations for this project was security by applying object detection to surveillance cameras in real time. Surveillance cameras are useful for keeping watch and recording criminal activity. However, not all cameras are monitored by people, and motion detectors can be used to alert if anything moves, but does not specify what it could be. By applying YOLOv3 with a GPU, we could implement object detection to surveillance cameras in real time, and it would be fast and accurate. Number of people and facial detection could also be implemented in order to find out who and how many people are present at the moment. This would be useful not only to notify someone remotely if someone is or isn’t in a the area, but also to detect if someone that is not allowed in the area is present.

B. Search In a Vast or Overcrowded Areas

We find that object detection is very useful for finding objects in vast or overcrowded areas, because it could be overwhelming for any person to have to perform such search, especially for objects that are small or far away. What it could take a person hours to search through, it would take a machine only the time it takes to process the image, which could be minutes or seconds. However, object detection is still not perfect at detecting everything, it depends a lot on the images as well. Objects that are small or/and far away could be hard to detect because it would have a very low resolution, and features of the object might be to vague for the machine to detect.

A solution problem had been proposed by [3], using a Generative Adversarial Network to super-resolved small objects, called Perceptual Generative Adversarial Network. This is done by feeding the low resolution representation of the object, and creating a super-resolved representation of the object and testing it against the discriminator.

Fig. 8. Top is using YOLOv3 and the bottom is using Tiny-YOLOv3.

Fig. 9. Object detection can be used in surveillance cameras to detect who is or who isn’t in an indicated area, as well as, how many people are in the area.

Fig. 10. A diagram depicting how a Perceptual Generative Adversarial Network works. As you can see the Perceptual GAN can give a close is it estimation of a high resolution representation of the object.

By implementing this method to super-resolved objects that are too small or too far a way to detect, we can develop software for assisting people to search for objects. An Android app can be used to, to assist people in searching for objects or people in crowded areas. Implementing this into a drone or satellite images can be utilized for searching and pin pointing where people lost in large areas are located.

REFERENCES


