Hand Gesture Tracking on an Embedded System

John David Maunder
maunder.j@northeastern.edu

Abstract

Covid 19 has re-shaped how we consciously or subconsciously interact with the world. Pre covid, interacting with devices in a public space was given little thought. This changed during the pandemic when people would go out of their way to not touch a publicly used device. Unfortunately, certain machines like transit card loaders must be used so people can travel around a city. These devices are cleaned regularly, but it only takes one infected individual to spread an infection from themselves to a commonly used device which is an issue since germs can live on a device for several days. To minimize the spread of infection, training a convolutional neural network (CNN) to recognize hand gestures can create a completely touchless device. At the time of this paper, multiple researchers and companies have implemented various convolutional neural networks on an embedded system. The CNN that has been the most effective is Mediapipe's keypoint hand tracker algorithm. In this paper, we will discuss how premade neural net frameworks like MediaPipe combined with Depthai can be expanded upon and improved to make completely touchless devices on an embedded system. In this paper we trained three different algorithms (Yolov3-Tiny, Yolov5nano, Yolov7) on a custom dataset, and deployed this on a Rasberrypi 4. By using the Yolov3Tiny and Yolov5nano algorithms and by combining the Rasberrypi with an OAK D-Lite, we were able to recognize hand gestures on an embedded system 70-110 % faster than the Mediapipe algorithm while being more than 10-18 % more accurate.

1. Introduction

Covid has re-shaped how humans interact with the world since people have minimized their contact with other humans and various commonly touched objects. Due to this re-shaping of our views, creating touchless devices by placing hand gesture recognition on embedded devices can create a safe alternative to help individuals in their everyday lives. Various empirical and research studies have been conducted into using computers to conduct hand segmentation, feature extraction and hand classification techniques[17] [11]. These techniques break down how a computer can take an image (or a series of images), outline only the hand in the image, track the wrist location to find the finger joints, then subsequently use these fingers to create a hand gesture that a computer can identify. The process of converting an image to a hand gesture is normally undertaken by detecting the hand using a skin segmentation technique which converts the input image into a black and white picture where only the hand is shown in white. Once the white hand is outlined, Kernelized correlation filters or hand markers can be used to track the palm and key points (fingertips, finger joints) of a hand. Finally, these images can be fed into a Convolutional Neural Network which will determine what hand gesture is being made by an individual. Since the field of hand gesture recognition is relatively new, prior research has primarily only focused on applications that can be used on high powered computers that contain a GPU (Graphic Processing Unit). Since embedded devices have increased in computing power, it is now feasible to run these hand gesture recognition techniques on an embedded device [1]. Therefore, this paper will aim at focusing on using these hand gesture recognition techniques and optimizing them for an embedded system so these devices can be used in a wide range of applications without having to rely on a large computer system. Since the main purpose of putting a hand gesture recognition architecture on an embedded system is for usability, the two analytical features that will be measured will be speed of the application (in FPS) and reliability (accuracy). This paper will start off by discussing the work of other research articles that helped laid the foundation for this paper. This paper will then go on to discuss the three main components of our hand gesture architecture in detail. Next, we will run speed and accuracy tests of our hand gesture architecture and compare them to other hand gesture architectures implemented by other researchers. These tests will be run on a Raspberry Pi 4 device using an OAK D-Lite camera with no attached GPU. Lastly, we will discuss what work will be done on this paper in the future.

2. Related Works

Webcams are used to take the initial image of our environment but we can also use it to track the region of interest (ROI) of a hand. This in turn leads to better hand gesture classification. This problem has applications for home appliance control or human-computer interaction fields. The authors approached this issue by building the following hand gesture recognition pipeline[5]:

1. Hand detection
   (a) Skin segmentation
   (b) Noise processing
   (c) Background subtraction (to get ROI)
2. Hand Tracking
   (a) KCF Algorithm (kernelized correlation filters)
3. Hand Recognition
   (a) CNN network-AlexNet, VGGNet (All customized)

This paper does not discuss how to use a webcam in a real-world setting, nor how to convert their hand gestures into a HCI (Human Computer Interaction) function. Lastly, this paper does not discuss the use of Depth as a useful function when determining what a hand gesture will and can do.

The authors use a novel deep learning pipeline for sign language recognition based on using a Single Shot Detector (SSD), 2D Convolutional Neural Network (2DCNN), 3D Convolutional Neural Network (3DCNN), and Long Short-Term Memory (LSTM) from RGB input videos using 3D convolutional Neural Networks in conjunction with the hand skeleton model to accurately detect and recognize sign language gestures using old and new datasets. The authors of this paper approached the issue of increasing hand gesture recognition accuracy by focusing on creating 3D key points for the hand skeleton model and feeding this into a 3D neural network. By adding the third dimension, the accuracy of their model increased since the depth dimension helped distinguish similar hand gestures [12]. This paper is different as we plan on using a 3D camera in conjunction with a smaller CNN.

Robots deployed in a less than optimal environment (the international space station for example), will cause hand detection to become slightly noisy. In addition, using robots as a means for hand detection will create resource capacity issues that need to be addressed. The authors solved the problem of detection and localization of an astronaut's hands by combining an Xbox Kinect V2 with a customized FF-SSD. The FF-SSD replaced the VGG-16 layers with Resnet-101 layers, since Resnet-101 is better at feature extraction than CGG-16 [6]. This paper has a similar idea to mine as we both plan on using a non computer to perform gesture recognition, but I will be using a significantly smaller device with less computing power than the customized robot discussed in this paper.

Work has been done to recognize human emotions on an embedded device. The approach to recognize these emotions was done using the following four methods [15]:

1. They used the Vila-Jones face detection technique to detect the face in each image
2. They cropped the face image and converted it to grayscale using a Sobel filter
3. They used the Active Shape Model method to extract the key facial points to detect an emotion (curved lips, narrow eyes etc.)
4. Finally they used ADA boost to classify the emotions from the above step

Hand gesture recognition is broken up into two sections and is designed to address the key issues of hand gesture problems such as reducing background noise to increase hand detection. The author proposed to break up their hand gesture architecture into two stages [10]:

1. Extracting hand regions using RetinaNet-Based Hand Detection
2. To speed up inference the authors used a lightweight CNN called MobileNet

This paper is similar to what we want to do as they tried to speed up hand recognition using smaller lightweight CNNs, but these authors only tested their data on static images and not on dynamic images(videos). In addition, these authors did not preform these tests on an embedded device.

The use of a two-channel approach to identify and classify left and right handedness in a frame can significantly increase the usefulness of any HCI hand gesture-based device. The author solved the above problem by using a two-channel approach. The first channel uses the ResNet-Inception-Single Shot MultiBox Detector to extract hand feature information. The second channel uses human body poses to estimate the positions of the left and right hand by using a forward kinetic tree [7]. Since this paper discusses the use of two hands, this paper on parallel deep neural network will have a similar implementation to our paper. The difference between this paper and our paper is we will most likely have to sacrifice accuracy for speed, and we will have to minimize our algorithm to save space on our embedded device. Also, the CNN algorithm we choose will have to be optimized for a CPU.

The individuals at google have been able to develop a hand tracking algorithm that can detect and draw 21 key points on multiple hands in an image. This algorithm is efficient and lightweight enough to work on an embedded device. This algorithm was designed by using a two-stage hand tracking pipeline. The first part of the pipeline is a palm detection machine learning model. This part of the model is a single shot detector that is optimized by using a non maximum suppression algorithm, an encoder-decoder extractor.

The second part of this pipeline optimizes the algorithm from the "Hand Keypoint Detection in single images using Multiview Bootstrapping" paper by combining it with a facial recognition algorithm that is optimized for a mobile GPU [18].

The algorithm in this paper is similar to the algorithm that we will implement on an embedded/mobile device. The difference between our paper and this paper is we plan on using this paper as a basis for my hand gesture pipeline. Since hand gesture tracking is just the first part of a hand gesture pipeline, we plan on using this output as our input for my hand gesture classifier.

The idea of using a multicamera system is useful as it allows for a reduction of occlusion when people put their hands behind items like a Cello. Occlusion is a difficult topic to deal with since it prevents people from using certain objects in hand recognition. Multiview Bootstrapping is used to take multiple pictures of a person with multiple cameras and then triangulating them into 3D Key points. If the image fails to pass the threshold (based on the user input) the image is reprojected and then passed into the training model to update the neural network. The difference between our paper and this paper is our paper will focus on avoiding hand key-point detection as it slows down detection rate.

The authors in this paper discuss how to best recognize when a hand gesture starts and ends. It also considers how an architecture should be centered around memory and a budget. The authors used a model that combines a lightweight CNN architecture in conjunction with a sliding window approach. The CNN architecture is comprised of the ResNEt-10 architecture for detection and the ResNeXt-101for classification. For continuous classification, the authors used a time weighted average strategy on the images in their pipeline to increase hand gesture detection without losing a substantial amount of accuracy [9]. This paper discusses the high-level concepts of using certain hand gesture recognition pipelines without fully implementing them using dynamic data.

3. Methodology

The two biggest problems with running a computer vision application(s) on an embedded system are configuration and computing power. Since Single Board Computers (SBC, a version of an embedded device) have very little computing power the whole device must be configured before any code can be run. Once the code is run, the code must be formatted and optimized to be run on very little computing power. Current approaches will occasionally discuss the speed of a computer vision application, but they do not specifically discuss the speed of these applications when they are implemented using a video feed on a embedded system [2].

This paper focuses on integrating deep learning techniques, more specifically hand gesture recognition techniques on a live video feed on an embedded device (RasberryPi 4 device). The live video feed will come from a person standing in a room who will in turn perform various hand gestures (hand actions) (see image 1). These hand actions will be detected by our camera and will then be converted by our embedded device into real time computer actions. Since the focal point of this paper is converting hand gestures into human usable computer actions, the combination of speed (Frames Per Second) and accuracy (Mean Average Precision) will be the key valuation metrics we base the performance of our algorithm on. Since the speed and accuracy of any algorithm is highly correlated to the underlying hardware the algorithmic tests will be run on the following setup:

1. RasberryPi 4 which includes 4 GB of RAM SDRAM and a Quad core Cortex-A72 (ARM v8) processor

2. 128 GB Micro SDXC configured to the Fat32 File system

3. OAK-D-Lite Camera (60 FPS Colour and Stereo Camera) (Image 1)

4. Bullseye 64 bit OS

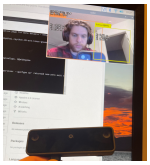

Figure 1. OAK D Lite Camera

Based on the above setup, to test the speed of the algorithms, the average frames per second over a two minute video will be used to determine the speed of the algorithm. The two minute video will be the best approach to mirror a real world scenario. The frames per second score will be taken every second and the total amount will be averaged by 120. Examples of these hand gestures and the FPS score can be seen in image 2.

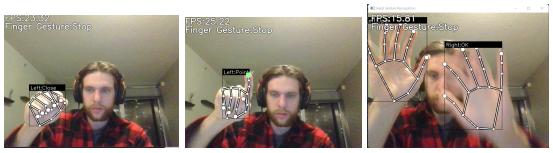

Figure 2. Frames Per Second Examples

Speed is important but the hand gesture recognition algorithm has to be accurate as well for the algorithm to be practical. We will measure accuracy using Mean Average Precision (mAP). mAP is based on the average of all correct and incorrect detection for each class [17]. Thus, for each hand gesture in our dataset, each hand gesture will be classified as correctly detected(TP), incorrectly detected(FP) and incorrectly undetected(FN) [17]. Utilizing these outputs, we can formulate the following two ratios:

$$Precision = \frac{\text{TP}}{\text{TP+FP}} \tag{1}$$

$$Recall = \frac{\text{TP}}{\text{TP+FN}} \tag{2}$$

Recall measures how accurate our model detects all people in our given data set based on the number of hand gestures. Precision focuses on how many hand gestures were annotated in the dataset and compares this to the total number of hand gestures the model correctly and incorrectly detected. Comparatively, recall focuses on how many people our model correctly detects in the dataset, compared to the correct total number of hand gestures that were annotated in the dataset.

The PR curve is then given by plotting these recall-and-precision scores on various Intersection Over Union (IOU) thresholds (image 3)[14] . The area under the interpolated PR curve is the AP value for the class (Image 4)[16].

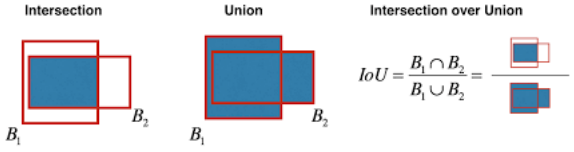

Figure 3. Intersection Over Union

Based on the above two formulas, we want to maximize the speed of the algorithm on the RasberryPi while making sure that our algorithm can still accurately detect or not detect a hand gesture. The general process of our experiment can be seen in image 5.

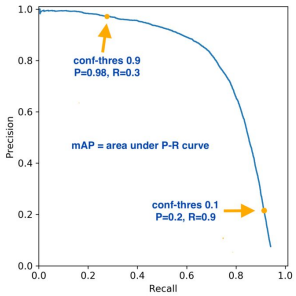

Figure 4. PR Curve

Figure 5. Experiment Overview

4. Algorithm Description

4.1. Convolutional Neural Networks

Convolutional neural networks can be generally broken down into two stage and one stage detectors [3]. Both detectors have four key stages that transition an image into an output classification for all images in the image. These four key stages are the input, backbone, neck, and head stages. The input stage usually transitions an input image or video into a vector that can be fed into the backbone of the convolutional network. The backbone of the convolutional network is a customized network with a finite amount of convolutional and pooling (average and max) stages. The backbone stage will produce a vector feature map containing the semantic and spatial image of the input image. This feature vector map is converted into a single row vector by the neck stage of the convolutional neural network. The neck stage of the neural network "flattens" (converts) the 2D feature maps into a 1D vector while minimizing the loss of the semantic and spatial information produced from the backbone stage [3]. The output from the neck stage will be a large 1D vector that will be passed to the "Fully Connected" layer (the head stage).

The main difference between the one stage and two stage detectors occurs at the head stage. In one stage detectors, bounding boxes and classification is done in one stage using a Dense Prediction approach. In two stage detectors, the detectors first group the output from the neck into a "region proposal network" before classifying images and placing bounding boxes on these images (see image 6)[4].

4.2. Yolo

The first iteration of the Yolo algorithm was developed by Joseph Reymond in 2015, but the algorithm did not become widespread until the third iteration which was released in 2018. The third iteration of Yolo (Yolov3) be-

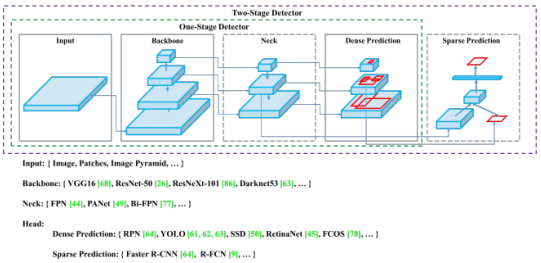

Figure 6. One Stage vs. Two Stage Detectors

came more widely used since it was able to achieve a similar mAP-50 score while running at twice the speed of similar CNN's like RetinaNet-50 and RetinaNet-101[13]. The increase in speed and accuracy was a result of the implementation of the Darknet-53 backbone structure and implementation of the new Yolo Head[13].

Feature Extractor: The Darknet-53 architecture is a neural network with 53 convolutional layers. The significant performance increase in this architecture and prior versions of Yolo, is that this 53-layer backbone uses the below architecture at three various scales (image 7)[4]. By combining the 3X3 with a 1X1 convolution layer at three different scales this algorithm can down sample the input image without losing semantic and spatial details allowing the Yolo algorithm to track large, medium and small objects without losing speed[13].

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 7. DarkNet-53 Architecture

Head/Detection: For each image passed into the Head, the image will be divided into an S * S grid where S denotes grid size. Each grid cell predicts B bounding boxes. Each bounding box will predict whether there is an object in that bounding box (confidence score), the four coordinates of that bounding box and the probability score for each class. Confidence is defined as Pr(Object) * IOU(truth, pred). The confidence score equals the Intersection Over Union (IOU) between the predicted box and the ground truth. IOU is the area of overlap of bounding boxes divided by the union area of the same bounding boxes. If no object exists in that cell, the confidence score is zero[13].

$$\Pr(Object) * IoU_{pred}^{truth} \tag{3}$$

Each bounding box will only contain at most one object. Therefore, the class (C) with the highest probability will be bounded by that bounding box. The class probability is conditional since it is dependent on a bounding box containing an object (confidence score of 1). The confidence score formula can be seen below. The probability of each class is calculated by using independent logistic classifiers, where these losses are trained using binary cross entropy. If the same or multiple grid cells have overlapping bounding boxes when they are predicting the same object, non maximum suppression will be used to make the final bounding box. Non max suppression takes the bounding box with the largest Intersection over Union (IoU) with the ground truth bounding box.

$$\begin{aligned}\Pr(Class_i|Object) * \Pr(Object) * IoU_{pred}^{truth} = \\ \Pr(Class_i) * IoU_{pred}^{truth}\end{aligned} \tag{4}$$

The bounding boxes are trained using anchor boxes where the shape of the anchor boxes are modified during training using a K-means clustering algorithm. The output prediction for Yolo can be summarized as S * S * 3 * (B * (5 + C)) [13]

Yolo Loss Function:

$$\begin{aligned}&\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}1_{ij}^{obj}\left[(x_i-\widehat{x_i})^2+(y_i-\widehat{y_i})^2\right]\\ &+\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}1_{ij}^{obj}[(\sqrt{w_i}-\sqrt{\widehat{w_i}})^2+(\sqrt{h_i}-\sqrt{\widehat{h_i}})^2]\\ &+\sum_{i=0}^{S^2}\sum_{j=0}^{B}1_{ij}^{obj}(C_i-(C_i))^2\\ &+\lambda_{noob}\sum_{i=0}^{S^2}\sum_{j=0}^{B}1_{ij}^{obj}\ (C_i-\widehat{C_i})^2\\ &+\sum_{i=0}^{S^2}1_{ij}^{obj}\sum_{c\epsilon classes}(p_i(c)-\widehat{p_i}(c))^2\end{aligned} \tag{5}$$

4.3. Yolov4/5

Yolov3 and Yolov5 will be used to train our dataset for our hand gesture recognition algorithm. As of August 18th,

2022. Yolov5 does not have a paper released on the algorithm. The Yolov5 algorithm is very similar to the yolov4 algorithm based on the ultralytics github repo, but for the purpose of this paper, we will discuss the architecture of the Yolov4 algorithm since there is a published paper on this algorithm whereas there is no paper published for the Yolov5 algorithm.

The two main differences between Yolov4 and the normal Yolov3 algorithm is that Yolov4 uses CSPDarknet (Cross Stage Partial Darknet) as its backbone and Spatial Pyramid Pooling (SPP) as well as modified Path Aggregation Network (Pan).

The difference between CSP Darknet and Darknet is CSPDarknet contains "Dense Blocks". Dense blocks divide the input into two halves where one half is sent through the dense block, while the other half will be routed directly to the next step without any processing. CSP preserves fine-grained features for more efficient forwarding, stimulates the network to reuse features and decreases the number of network parameters. The addition of dense blocks will help minimize the gradient descent problem making the model more robust. The downside of adding dense layers is it can make the model slower, which is why the Yolov4 algorithm minimizes the use of dense layers[3].

Yolov4 uses a spatial pyramid pooling approach and a Path Aggregation Network to transition the feature maps from the CSP Backbone to the Yolo Head. The SPP block is used to preserve spatial dimension for the feature map by applying a 1x1 convolution from the feature map and then duplicating and pooling the different scales so the 3 feature maps will retain the sizes of sizefmap x sizefmap x 512. The SPP process is combined with the PAN process to produce a vector output that will be passed to the Yolo Head[8].

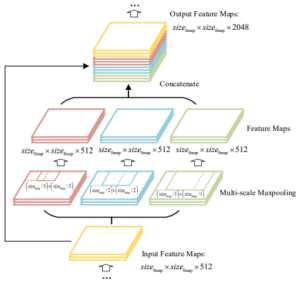

Figure 8. Spatial Pyramid Pooling

[8]

The PAN process is like the feature pyramid pooling approach discussed above except it uses a bottom-up approach to pool semantic features together. This bottom-up approach allows the Yolov4 algorithm to create a shortcut to directly connect fine-grained features from low-level layers to the top one[8].

[8]

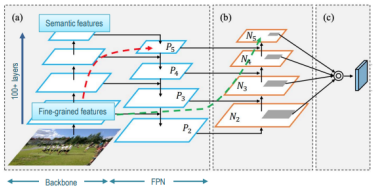

Figure 9. Path Aggregation Network

4.4. Optimizations for the Yolo Algorithm

Since the goal of this paper is to implement a speedy algorithm that can still accurately detect a hand gesture, we will modify the algorithm in the following away:

1. Minimize the depth of the algorithm
2. Hyper Tune parameters
   (a) Minimize Bounding box amounts
   (b) Fine tune anchor boxes
   (c) Apply batch normalization
   (d) Fine tune activation functions
3. Perform data augmentation to make the data set more robust

Since our hand gestures will be performed in front of a camera that will be attached to a monitor, it is highly likely that the individual will be in a moderately well-lit room, in the middle of the focal point of the camera. Since we can somewhat control how and where the user performs the hand gesture we can make modifications to the Yolo algorithm that will speed up the algorithm without sacrificing our mAP score.

5. Methodology

Image 10 provides an illustrative guide of how the hand gesture detection experiment was performed. This experiment starts with image collection and labelling and eventually ends with comparing a custom trained model to other pre-existing hand gesture models. This paper discusses the object detection models on a customized dataset. Therefore, the first step in creating the model pipeline was data collection.

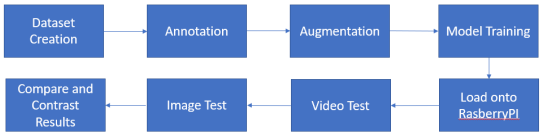

Figure 10. Methodology Overview

The goal of this paper is to discover which data model works best for hand gesture recognition on an embedded system so individuals or companies can make touchless devices. Since the main goal is to make interactive touchless devices, the average use case will involve a person standing about 1-3 meters away from the device, making various hand gestures. The data set was created to replicate this scenario. Most of the images used in this scenario were taken inside a building or a house, where the background contained other individuals and various objects (please see image 11).

Figure 11. Data-set Examples

A couple of the data set images were taken in an outdoor setting, as a touchless device could also be used in an outdoor setting. The varying background and lighting were used to replicate a wide range of use cases. Each image contains 1-2 different hand gestures per image. There are 10 classes in total. The classes were custom created and were implemented with the intention of being distinct hand gestures. This distinction gesture will help increase the mAP score and overall usability of the model in a real-world setting. Once the 1532 number of images were taken (3673 images after augmentations), the images were transferred into Roboflow where they were annotated and broken up into training, validating, and testing groups in an 87-8-4 split (please see image 12). From Roboflow, the images were imported into Google Collab where the Yolov5-nano algorithm was trained on a Tesla T4.

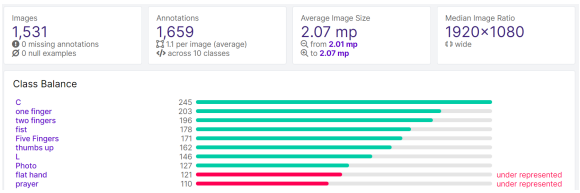

Figure 12. Data-set Class Breakdown

[8]

The Yolov5nano algorithm was trained on a cloud device instead of an embedded system since it sped up training time and minimized the wear and tear on the rasberrypi. Once the model was trained, the ".pt" (weights file) was uploaded to google collab so the weights file could be downloaded onto the raseberrypi for testing. Once on the raspberry pi, a python document was setup to run two different tests to detect the mAP and the frame per second rate score for the various object recognition algorithms. The four object recognition algorithms (listed below) were put through two different tests.

Video Test:
The video test is comprised of a two-minute video. The video is fed into three object detection algorithms below. The output from this video test is an average frame per second score.

1. Yolov5nano
2. Yolov3Tiny
3. Mediapipe/Depth AI custom Object Recognition algorithm

Image Test:
The image test is comprised by running inference on 495 images (not from the original dataset) and then calculating the mAP score from these images. All four algorithms were tested in this section. The results of the two tests are discussed in the next section. Three object recognition algorithms:

1. Yolov5nano
2. Yolov3Tiny
3. Yolov7
4. Mediapipe/Depth AI custom object recognition algorithm

6. Results

The goal of these two simulations is to create an algorithm that can accurately (Precision and Recall) and quickly (FPS) detect or not detect a hand gesture during a video feed. Since a video feed is a compilation of individual frames, the accuracy of these neural networks will be tested on the individual frames (Image Test) and the speed will be tested on the video feed.

6.1. Image Test

For the image test, we passed 459 images (36 with multiple hands) into the three various neural network algorithms at various confidence levels (0.4-0.9). The results of the test are as follows:

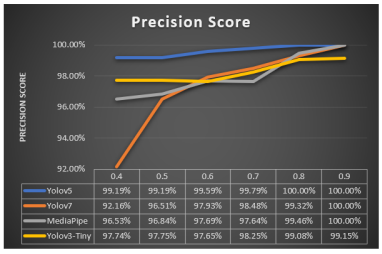

Figure 13. Precision Score

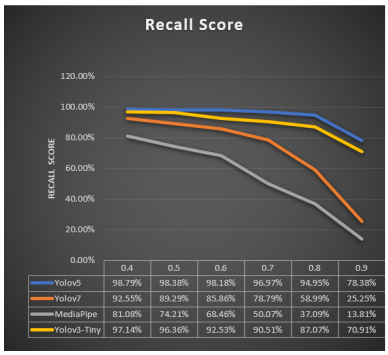

Figure 14. Recall Score

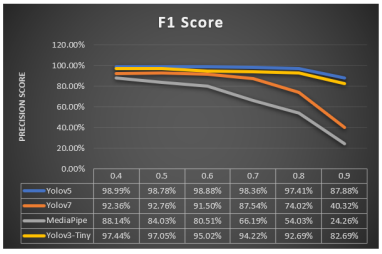

Figure 15. F1 Score

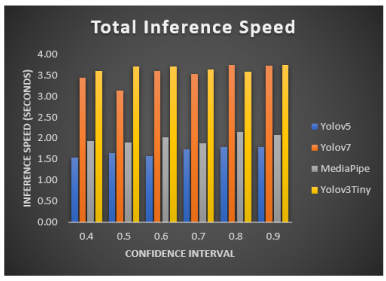

Figure 16. CPU Inference Speed

As can be seen above, Yolov5 preformed the best, yolov3-Tiny the second best, the MediaPipe algorithm the third best, and the new Yolov7 performed the worst. This is as expected as Yolov7 was recently released and has not been optimized for this use case. MediaPipe's keypoint tracking is useful as it helps to track hand movements, but occlusion in the key-points when the hand is turned at an inoptimal angle can create false positives or false negatives. Lastly, Yolov5nano's neck feature allows more detail to be passed to the Yolo head which gives Yolov3 the slight edge over Yolov3Tiny.

Inference outputs for Yolov3Tiny can be seen in image 17 and inference outputs for Yolov5nano can be seen in image 18.

Figure 17. Yolov3Tiny Inference Outputs

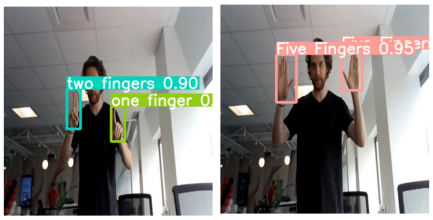

Figure 18. Yolov5nano Inference Outputs

6.2. Video Test

For the video test, Yolov5 and Yolov3 were converted to a blob file and were run on an OAK D Lite TPU (an Intel Tensor Processing Unit). Mediapipe is already in a blob format, therefore this algorithm could be run out of the box. Yolov7 is currently too new to run on an OAK D Lite so no video speed was run for this algorithm as it cannot be converted to a blob file just yet.

The video test entailed running all three algorithms for two minutes. During the video feed the high medium and low FPS rate was tracked. The results are as follows:

	High	Average	Low
Yolo5nano	40	35	32
YoloV3-Tiny	54	49	42
MediPipe	30	28	20

Figure 19. Oak D Lite Video Test Results

Yolov3Tiny and Yolov5nano did not slow down in speed when more hands were added to the screen. This is significant as it allows for a combination of hand gestures to be used in a single frame. It also allows for more versatility in actions that a Single Board Computer can understand without compromising speed. Lastly, Yolov3Tiny was able to detect four hands in a given frame without reducing speed as can be seen in image 20.

Yolov5 did not deviate much with its speed, while Mediapipe deviated a lot when the hand was removed from

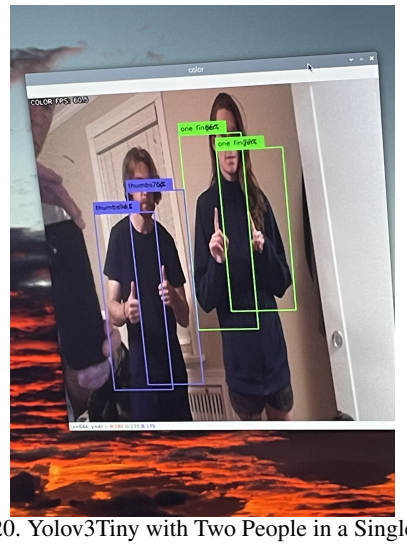

Figure 20. Yolov3Tiny with Two People in a Single Frame

the screen. Both Yolov3 and Yolov5 out-performed Mediapipe as they are single neural networks vs Mediapipe which uses two neural networks. In comparison, between Yolov3Tiny and Yolov5nano, Yolov3Tiny performs faster than Yolov5nano. The photo below shows the comparison between the two CNN's. As can be seen in the photo below, Yolov3-Tiny is smaller than Yolov5nano in all categories except for floating point operations. This minimization of a CNN design is the reason for Yolov3Tiny's speed. These results are in line with the GPU speed tests that were run on Google Collab. It took roughly 6.1ms vs 9ms when running inference tests on Yolov3 tiny vs Yolov5nano respectively.

	Yolov3-Tiny	Yolov5nano
Anchors	12	18
Layers	23	62
Slides	6	9
Parameters	1,144,500	1,867,405
BFLOPS	5.462	4.5

Figure 21. Yolov3Tiny vs Yolov5Nano Structure

7. Conclusion and Future Work

7.1. Conclusion

This paper examined the use of four algorithms (Yolov3Tiny, Yolov5Nano, Yolov7 and MediaPipe's Hand Tracking algorithm) in the context of hand gesture recognition using a custom dataset. The custom dataset was trained on google collab and was then deployed and tested on a RaspberryPi 4 and an Oak D Lite. There were three tests that were run on the four algorithms. The tests were calculating F1, Precision and Recall scores on 495 hand gestures using the CPU, calculating inference speed using the CPU, and calculating Frame Per Second Speed on a two-minute video using the Oak D Lite. The purpose of these tests was to compare the Yolo algorithms to the well established MediaPipe hand tracking algorithm on a custom dataset. The results of these tests showed that Yolov5 was about 12-22% more accurate than MediaPipes hand gesture algorithm on the CPU test and was about 7-20% faster on the CPU inference test. Lastly, Yolvo3Tiny preformed 75-110% faster than MediaPipe's algorithm on the 2-minute video test using the Oak D Lite.

The Yolo algorithm tended to preform better than the MediaPipe algorithm since it is using one neural network instead of two. The first neural network created the keypoint tracker while the second CNN converts this key point tracker into a hand gesture output. By having the Yolo algorithms generalize distinct hand gestures, we can create a Yolo algorithm that only needs one CNN instead of two. This will allow the OAK D Lite to detect more hand gestures per minute.

A custom dataset allows convolutional neural networks to train on data that it is likely going to see during real time usage. Lastly, having more distinct hand gestures will allow the user more flexibility when these hand gestures ultimately become applications.

7.2. Future Work

This paper has shown that multiple different CNN's can be used to detect hand gestures on a CPU and a TPU with strong accuracy. The next step to make this paper more applicable to real world scenarios would be to convert these hand gestures into computer actions. This can be accomplished using pythons' "keyboard" and "mouse" modules or by using C++'s "event handler class". These modules and classes can convert the 10 given hand gestures into any keyboard and mouse function. Once this event link has been established, the possible use cases for this project are endless, since any person can take these actions and convert them into any useable program they desire. To make a hand gesture-based program run more effectively, a bigger dataset is needed to detect a wide variety of skin tones in different environments. The custom dataset made has different backgrounds in it, but the number of backgrounds is small compared to how many different backgrounds this application could be used in. Lastly, all images in the dataset only contain photos of me in short sleeve clothing. People with a darker skin tone, people who want to use this application with gloves, or people who want to wear longer sleeve clothing will have to be added to make these two Yolo algorithms run more efficiently on a larger user base.

8. Acknowledgements

I would like to thank Saurav Shah for helping me build my data-set and I would like to thank Larah Maunder for helping me take photos of my experiments. Lastly, I would

also like to thank Dr. Michal Aibin for his guidance and my teaching assistants Tianyi Chen and Yuanxi Li for their input on my report.

References

[1] A. A. Abed and S. A. Rahman. Python-based raspberry pi for hand gesture recognition. *International Journal of Computer Applications*, 975:8887, 2017.

[2] R. Benenson, M. Mathias, R. Timofte, and L. Van Gool. Pedestrian detection at 100 frames per second. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2903–2910, June 2012.

[3] B. Benjdira, T. Khursheed, A. Koubaa, A. Ammar, and K. Ouni. Car detection using unmanned aerial vehicles: Comparison between faster r-cnn and yolov3. In *2019 1st International Conference on Unmanned Vehicle Systems-Oman (UVS)*, pages 1–6, 2019.

[4] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020.

[5] H.-Y. Chung, Y.-L. Chung, and W.-F. Tsai. An efficient hand gesture recognition system based on deep cnn. In *2019 IEEE International Conference on Industrial Technology (ICIT)*, pages 853–858, 2019.

[6] Q. Gao, J. Liu, and Z. Ju. Robust real-time hand detection and localization for space human–robot interaction based on deep learning. *Neurocomputing*, 390:198–206, 2020.

[7] Q. Gao, J. Liu, Z. Ju, and X. Zhang. Dual-hand detection for human–robot interaction by a parallel network based on hand detection and body pose estimation. *IEEE Transactions on Industrial Electronics*, 66(12):9663–9672, 2019.

[8] Z. Huang, J. Wang, X. Fu, T. Yu, Y. Guo, and R. Wang. Dc-spp-yolo: Dense connection and spatial pyramid pooling based yolo for object detection. *Information Sciences*, 522:241–258, 2020.

[9] O. Köpüklü, A. Gunduz, N. Kose, and G. Rigoll. Real-time hand gesture detection and classification using convolutional neural networks. In *2019 14th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2019)*, pages 1–8. IEEE, 2019.

[10] A. A. Q. Mohammed, J. Lv, and M. Islam. A deep learning-based end-to-end composite system for hand detection and gesture recognition. *Sensors*, 19(23):5282, 2019.

[11] P. Parvathy, K. Subramaniam, G. Prasanna Venkatesan, P. Karthikaikumar, J. Varghese, and T. Jayasankar. Development of hand gesture recognition system using machine learning. *Journal of Ambient Intelligence and Humanized Computing*, 12(6):6793–6800, 2021.

[12] R. Rastgoo, K. Kiani, and S. Escalera. Hand sign language recognition using multi-view hand skeleton. *Expert Systems with Applications*, 150:113336, 2020.

[13] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[14] A. Rosebrock. Intersection over union for object detection, 2016.

[15] Suchitra, S. P., and S. Tripathi. Real-time emotion recognition from facial images using raspberry pi ii. In *2016 3rd International Conference on Signal Processing and Integrated Networks (SPIN)*, pages 666–670, 2016.

[16] V. Wadawadagi. Metrics to use to evaluate deep learning object detectors, 2020.

[17] M. Yasen and S. Jusoh. A systematic review on hand gesture recognition techniques, challenges and applications. *PeerJ Computer Science*, 5:e218, 2019.

[18] F. Zhang, V. Bazarevsky, A. Vakunov, A. Tkachenka, G. Sung, C.-L. Chang, and M. Grundmann. Mediapipe hands: On-device real-time hand tracking, 2020.