Final Project - User Drone Recognition

Austin Phillips
ECE 4850
UVU Electrical Engineering
Orem, Utah
austin.phillips@uvu.edu

Angel Rodriguez
ECE 4850
UVU Electrical Engineering
Orem, Utah
Angel.Rodriguez@uvu.edu

***Abstract*—With the use of Machine Learning and image processing, its possible to have a drone identify a user and respond to user input. This paper explores such a system.**

***Index Terms*—Drone, Machine Learning, Support Vector Machines, Neural Network, Convolutional Neural Network, Image Processing, Haar Cascade Classifiers, OpenCV**

I. Introduction

The intention of the final project is to create a software system from the the drone shown in Figure 1 that will recognize a user and respond to potential actions they would take. This process can be broken down to three major parts: recognition of the user, maintaining visual of the user, and responding to user input.

The broad picture of such system would be used in a variety of uses including entertainment, security, and aiding those with disabilities. Examples include utilizing a drone to follow a user while the user does activities that involves work of the hands, a drone that activates only upon recognition of a user similar to Face ID used for mobile cellphones, or giving new ways to control a drone to take actions for users who may be impaired or disabled.

User recognition will require supervised machine learning in order to differentiate who is a recognized user and who is not. To recognize the user, this project explores four different methods of classification: Support Vector Machines (SVM), VGGNET, ResNet, and Cascade Classifiers. All four of these methods have seen use in object classification with CNNs seeing widespread use in the current day. Each method will be trained on an extended version of the Face Research Lab London's database of peoples faces. One additional category (photos of Phillips, Rodriguez, and Shekaramiz) will be added on top with the original, serving to categorize all other individuals. Once trained, RGB photo data from a drone will be used to confirm the use by the authors.

Once the user is recognized, the drone will attempt to maintain frame of the user using RGB picture data. The plan for attempting this is to use a bounding box similar to that shown in Figure 2. The distance from the drone to the face can be calculated using trigonometry and the area of the bounding box and the result shall be used to maintain a defined range of distance between the drone and recognized user. For keeping the user within frame, the distance of the center of the bounding box to the camera frame edges will be calculated and the drone will be in a constant phase of adjusting its lateral positioning to keep the center of the bounding box within the center of the camera window.

Once an authorized user is detected, hand/kinetic tracking will be used to accept feedback from the user, performing at least two tasks such as movement based on direction or commands to lower. For example, pointing to a direction would signal the drone to move into that direction temporarily before the drone adjusts its position to keep the recognized user within the center of frame. This part should be the most difficult and time-consuming. The previous tasks will be prioritized and this aspect of the project will be done if time allows.

II. Dataset for User Verification

For taking the photos, it is important to match the photographs of the Face Research Lab London's database. See examples of facial images from the Face Research Lab London Set in Figures 3. To create training data set with the database, photos of the other category/subject being Rodriguez, Philips, and Shekaramiz shall have their images taken from multiple angles, from the collarbone up, wearing white shirts, and in front of a white background. Consequences of not doing so, such as wearing a black, shirt may result in the machine learning model having good results for distinguishing that subject from the general population only when the subject wears a black shirt. This is because in supervised machine learning, the algorithm will focus on the differences in photos for means of categorization and those differences may be unintentional. If all subjects follow the database's model in the training set, the model will more likely focus on the differences in the face (the intended goal) instead of differences in clothing and background. Using the Python script, almost 2000 images of each subject was taken. The goal is to have about 1000 images each, but closer to 2000 will prepare for getting rid of images deemed unfit to be part of the training data set.

Eventually, the database of images was slimmed down so that there are 1200 positive images split evenly between the three researchers and the negatives are built out of the Face Research Lab's photos and some images of a white wall. Examples of the new positive and negative images are in Figure 4.

Fig. 1. DJI Tello Drone

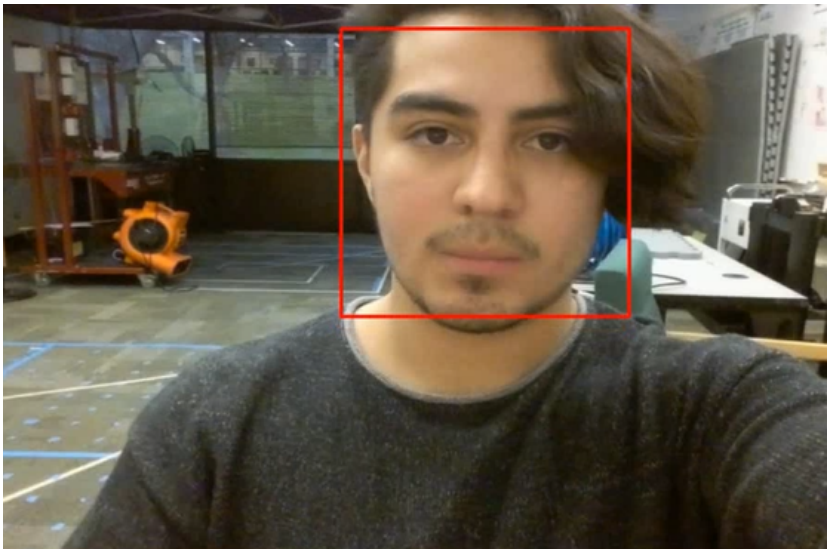

Fig. 2. Bounding box of recognized face taken from drone camera

III. Using Python Script to Operate Drone's Camera

To train the machine learning model, massive amounts of training data must be created or the learning process. A Python script was written to connect to the drone, record video using the drone's front RGB camera, and extract photos from the video to save into a local directory. While this would be adequate on its own, an output window depicting live feed of the drone camera was also desired for the purpose of checking image conditions during the program's run-time. However, an issue arose since both extracting images from the video and outputting a live video feed are done within infinite while loops (while True loops). The solution was to use Python's multi-threading library so that multiple processes could be done at once. In the Python script, a thread was created with the task of running the function which extracted images from the recording video while other parts of the script would be executed within an infinite while loop whose task was to show the constant live feed of the video on an new window that would pop-up on the device running the Python script. With the new code, it was possible to place the drone in front of one's face to take photographs while at the same time seeing the drone's point of view on a screen. This allows for the user to see if their facial position on the drone's camera view while the drone is taking video and pictures.

IV. User Verification Through Machine Learning

To perform user verification with the drone, three ML techniques were explored: ResNet50V2, VGG19, and SVM. Initially, a fourth option, Haar Cascade Classifiers, was con-

Fig. 3. Facial image examples with varying angles from Face Research Lab London

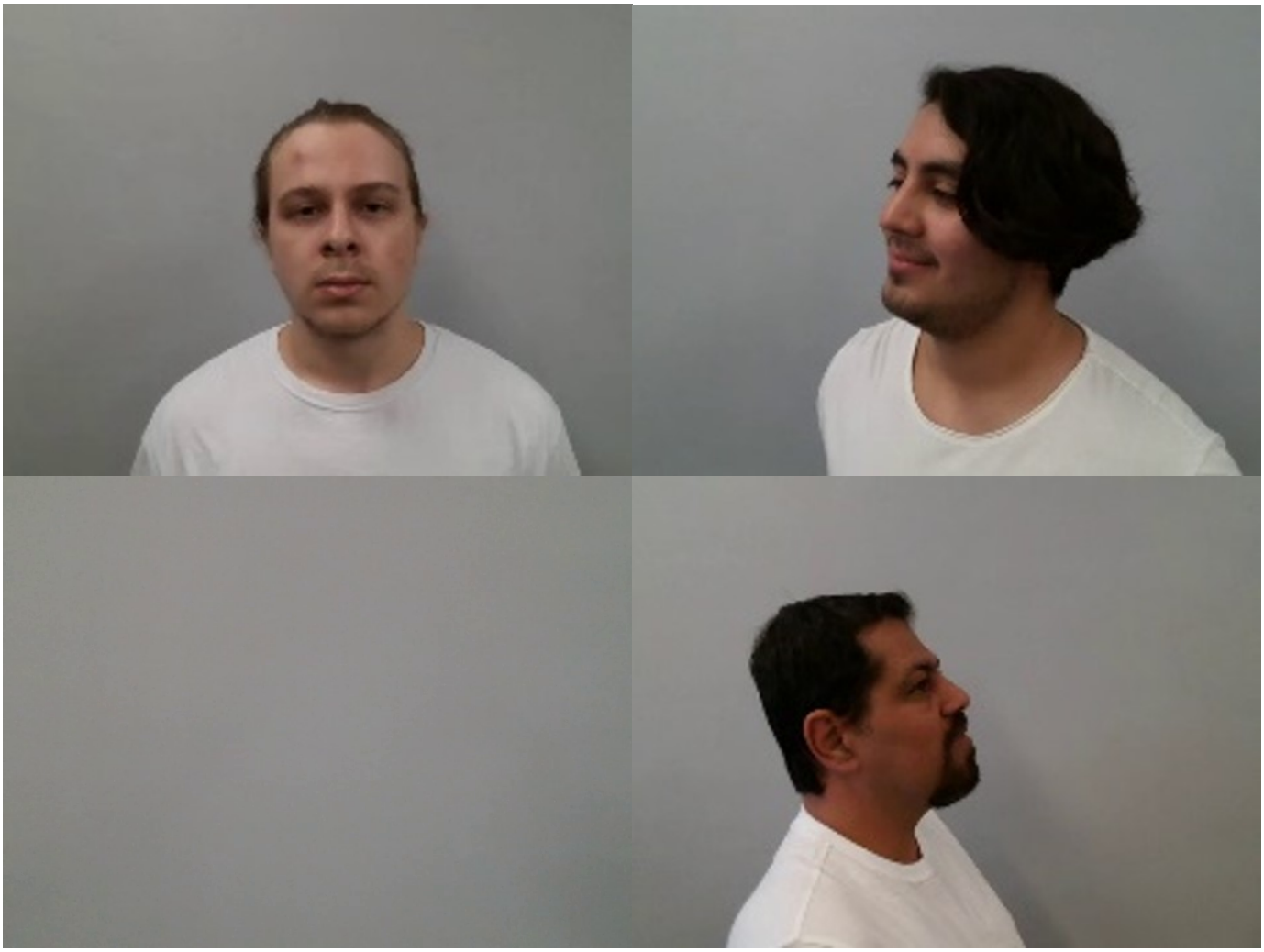

Fig. 4. New facial image examples with varying angles taken to be similar to Face Research Lab London set as well as a negative wall image for ML training

sidered but time restraints and software difficulties make explorations with the technique difficult leading to a focus on the previous three methods.

A. ResNet50V2

ResNet50V2 is a specific network design of a Convolutional Neural Network (CNN). The ResNet family of networks introduced the concept of residual learning to the CNN toolbox [1]. Residual learning, often seen as the shortcut connections seen in Fig. 5, prevents network degradation that is caused by the vanishing gradient problem. ResNet50 is a fifty-layer network that integrates residual learning. ResNet50V2 is an evolution of ResNet50 that improves on the latter by leading the convolution step of ResNets with the batch normalization and activation function as seen in Fig. 6 [2]. Both networks can be visualized as Fig. 7.

As seen in Fig. 8 and Fig. 9, the accuracy became 1.0 and the loss became very small.

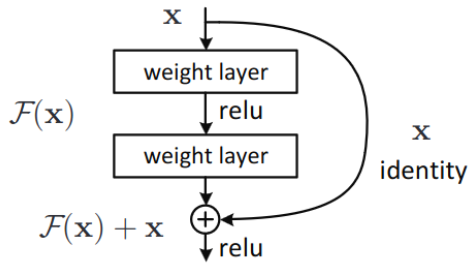

Fig. 5. Shortcut connection introduced by the ResNet family [1]

B. VGG19

VGG19 is also a specific CNN design; it is an efficient and effective nineteen-layer CNN following the design seen in Fig. 10 [3].

As seen in Fig. 11 and Fig. 12, the accuracy became 1.0 and the loss became very small.

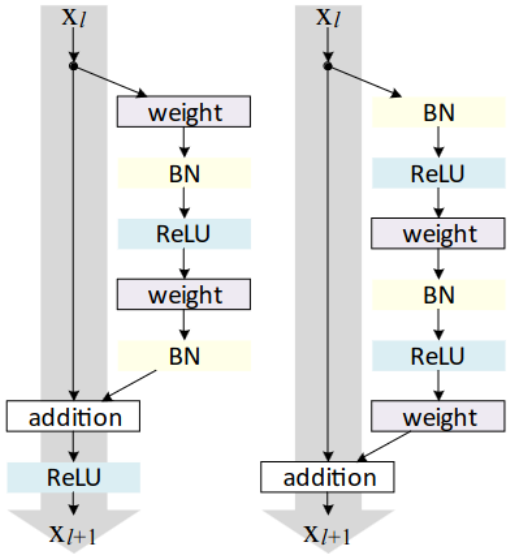

Fig. 6. Original ordering of ResNet, pre-activation ordering of ResNetV2 [2]

layer name	output size	50-layer
conv1	112×112	7×7, 64, stride 2
conv2_x	56×56	3×3 max pool, stride 2
		$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times3$
conv3_x	28×28	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times4$
conv4_x	14×14	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times6$
conv5_x	7×7	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times3$
	1×1	average pool, 1000-d fc, softmax
FLOPs		3.8×10^9

Fig. 7. ResNet50V2 model map

C. SVM

Support Vector Machine (SVM) is an established ML algorithm for classification. The SVM algorithm attempts to find a hyperplane that maximizes the separation between the classes. To create a separation in some cases, a "kernel trick" is required where a non-linear function is used to create an additional input to separate the data [4].

Three different kernels were tested in this procedure: linear, polynomial, and RBF. As seen in Fig. 13, the accuracy on the network quickly jumped to an accuracy of 1.0 in both training and testing accuracy. This would suggest an underlying issue with the dataset.

D. Verification Results

With all of our methods achieving an accuracy of 1.0, it became immediately suspicious of underlining issues. As a sanity check in-the-field, photos were taken to test the networks in the field. Table I suggests that the networks did not generalize well, even with efforts such as image augmentation and dropout. Even ResNet50V2 failed as it consistently assigned all photos a positive label.

TABLE I
ACCURACY ON IN-THE-FIELD IMAGES

	Accuracy
ResNet50V2	0.775
VGG19	0.300
SVM Linear	0.250
SVM Poly	0.500
SVM RBF	0.250

V. FACE DETECTION USING HAAR CASCADES

Object Detection using Haar feature-based cascade classifiers is an effective method proposed by Paul Viola and Michael Jones in the 2001 paper, "Rapid Object Detection using a Boosted Cascade of Simple Features". It is a machine learning based approach in which a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.

It will be used here for detecting faces. Initially, the algorithm needs a lot of positive images (images of faces) and negative images (images without faces) to train the classifier. Then features are needed to be extracted from it. For this, Haar features shown in Figure 14 are used. They are similar to a convolutional kernel where each feature is a single value obtained by subtracting the sum of pixels under the white rectangle from the sum of pixels under the black rectangle.

Now all possible sizes and locations of each kernel are used to calculate plenty of features. For each feature calculation, finding the sum of the pixels under the white and black rectangles must be done. To solve this, integral images are introduced. It simplifies calculation of the sum of the pixels, how large may be the number of pixels, to an operation involving just four pixels.

But among all these features calculated, most of them are irrelevant. For example, consider Figure 15. Top row shows two good features where the first feature selected seems to focus on the property that the region of the eyes is often darker than the region of the nose and cheeks. The second feature selected relies on the property that the eyes are darker than the bridge of the nose. But the same windows applying on cheeks or any other place is irrelevant.

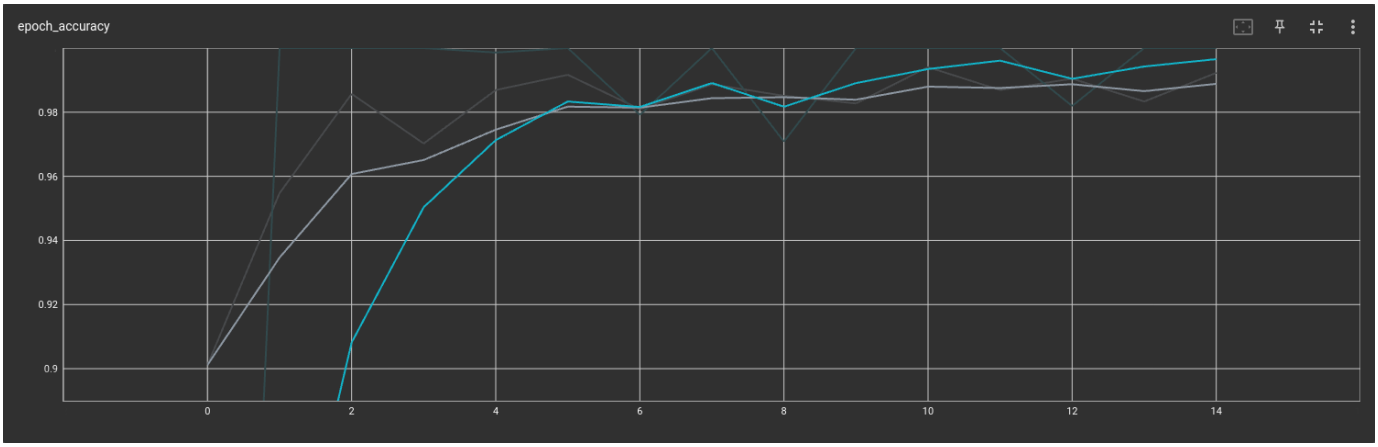

Fig. 8. ResNet50V2 accuracy: gray is training, blue is testing

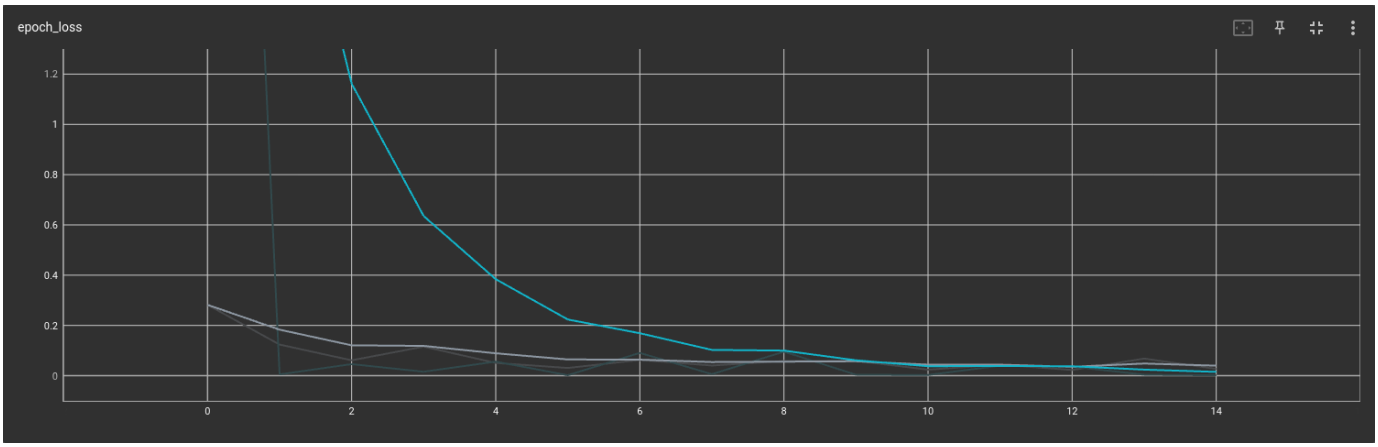

Fig. 9. ResNet50V2 loss: gray is training, blue is testing

VGG19 19 weight layers
input
conv3-64
conv3-64
maxpool
conv3-128
conv3-128
maxpool
conv3-256
conv3-256
conv3-256
conv3-256
maxpool
conv3-512
conv3-512
conv3-512
conv3-512
maxpool
conv3-512
conv3-512
conv3-512
conv3-512
maxpool
FC-4096
FC-4096
FC-1000
soft-max

Fig. 10. VGG19 model map

Simplified, this Haar cascade models focus on groups of properties that are likely to be a face. Training cascade models are time consuming as they require manually drawing rectangles over each of the objects of interest in the training data. Due to the custom ML models mentioned earlier not being adequate in detecting authorized users in non-training environments in addition to time constraints, an already trained cascade classifier model will be used for the drone to track faces to follow.

VI. Face Tracking

To implement face tracking, an already trained Haar cascade model was used to detect the front side of faces and Python code was implemented to draw bounding boxes on faces which would work for the general population. The necessary file ”haarcascade_frontalface_default.xml” was loaded using the load function from openCV.

The drone was programmed to take off when a face is detected for 10 camera frames. The reason being was to avoid mistakes in which a face is ”detected” where one does not exist. Once the drone is in the air the bounding boxes resulting from the detected face in frame would be used to calculate the distance from the drone to the face. Larger bounding boxes imply that the detected face is further whereas smaller bounding box implies a closer distance. The Python program is also able to give the x and y coordinates of the bounding box

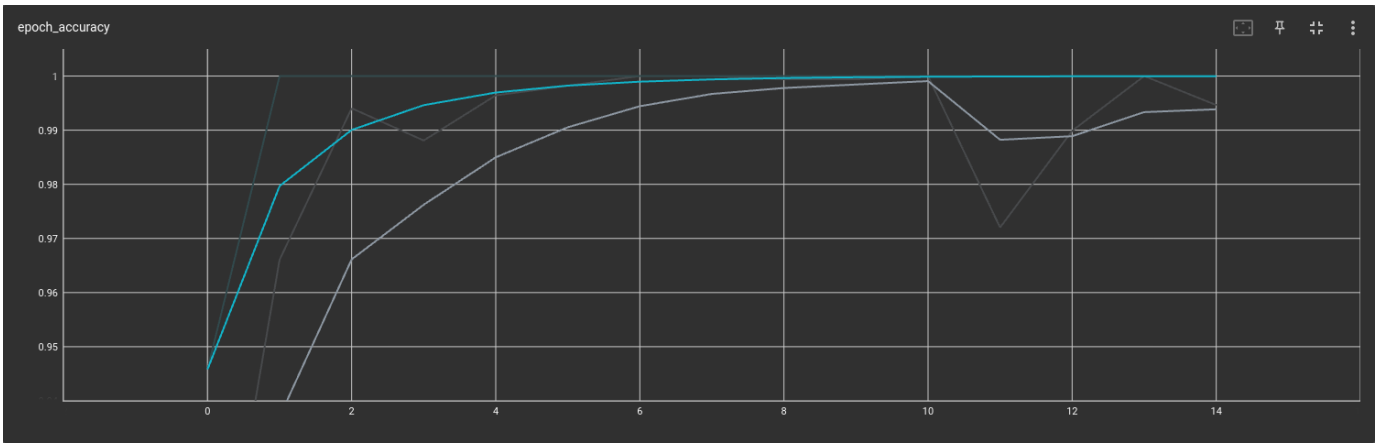

Fig. 11. VGG19 accuracy: gray is training, blue is testing

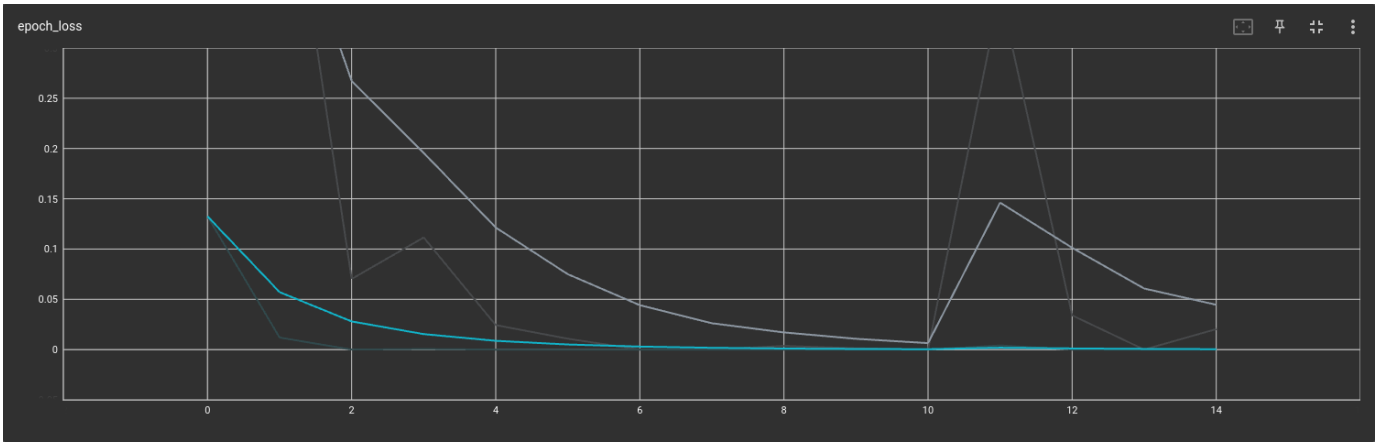

Fig. 12. VGG19 loss: gray is training, blue is testing

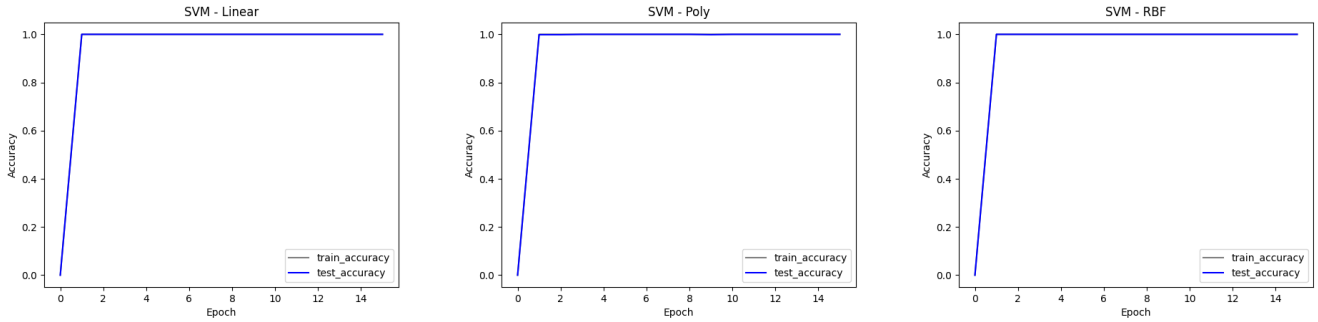

Fig. 13. SVM Linear, Poly, and RBF

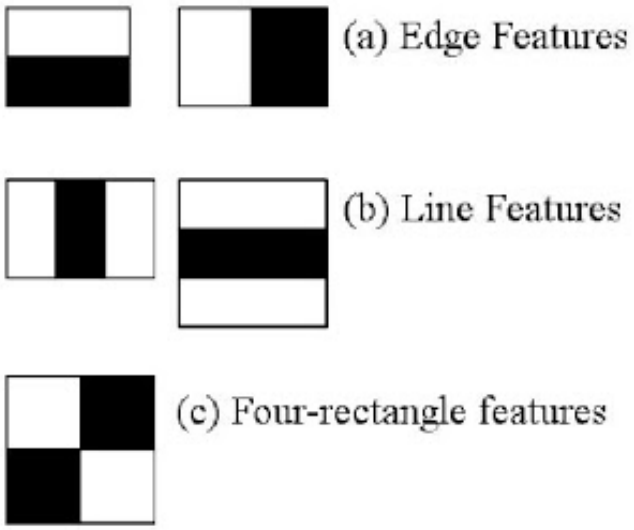

Fig. 14. Haar features used where each feature is a single value obtained by subtracting the sum of pixels under the white rectangle from the sum of pixels under the black rectangle.

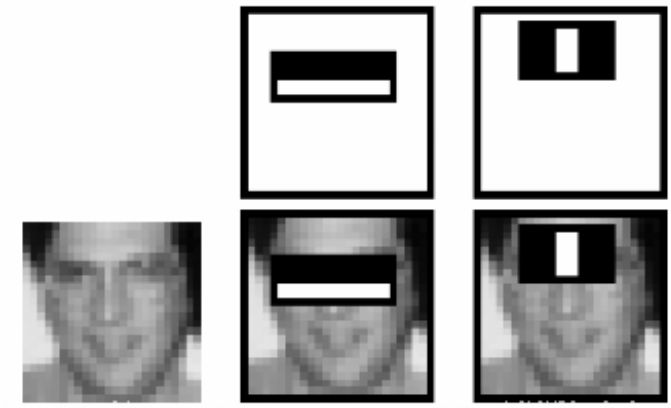

Fig. 15. Good features used for determining if a face is in the image.

center with respect to the dimensions of the frame (where the frame size is adjustable). The coordinates are used to calculate whether the drone should shift left, right, up, or down to center the person's face in frame. Video frame from airborne drone can be found in Figure 16.

Distance is calculated to determine how much the drone should move forward or back. The distance is calculated by focal length of lens * real-world width of object / width of bounding box in pixels. The units used are in centimeters since that is the unit used for drone movement commands. A couple of issues with this is that the face-width of different different people vary and the bounding box may encompass an area whose width is slightly greater than the face-width seen on camera. So the real-world face-width value within the Python file was tweaked to be a little larger than actual face width of users so that the program would output more accurate distance values for both Angel and Austin. The goal was to have the distance measurement within 10 cm which was successful based on test runs. Results of testing are depicted in Figure 17 where the user was about 84 cm away from the drone.

The algorithm for keeping the detected face in center of the drone camera frame is depicted in Algorithm 1. For every instance that the function for centering the user's face and adjusting distance is finished running, the drone does a forward flip for easy validation and acknowledgement. For several test runs, the drone showed promising results of being able to track the face of an individual and follow, appropriately shifting laterally, adjusting height, and moving forward or backwards to keep defined distance in front of user's face. Issues arise when the user suddenly moved behind the drone as the drone programming at the moment plans for the user being somewhere in front of the drone. The drone will move back if the user is found to be too close in frame, but that

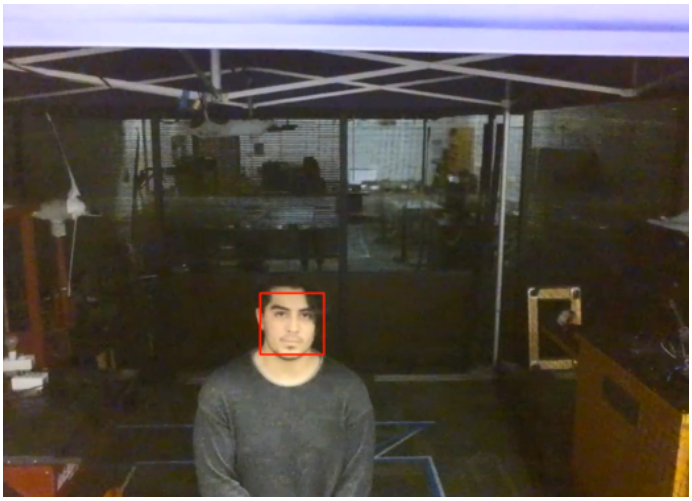

Fig. 16. Frame from drone camera while it is airborne shortly after takeoff initialization by facial detection.

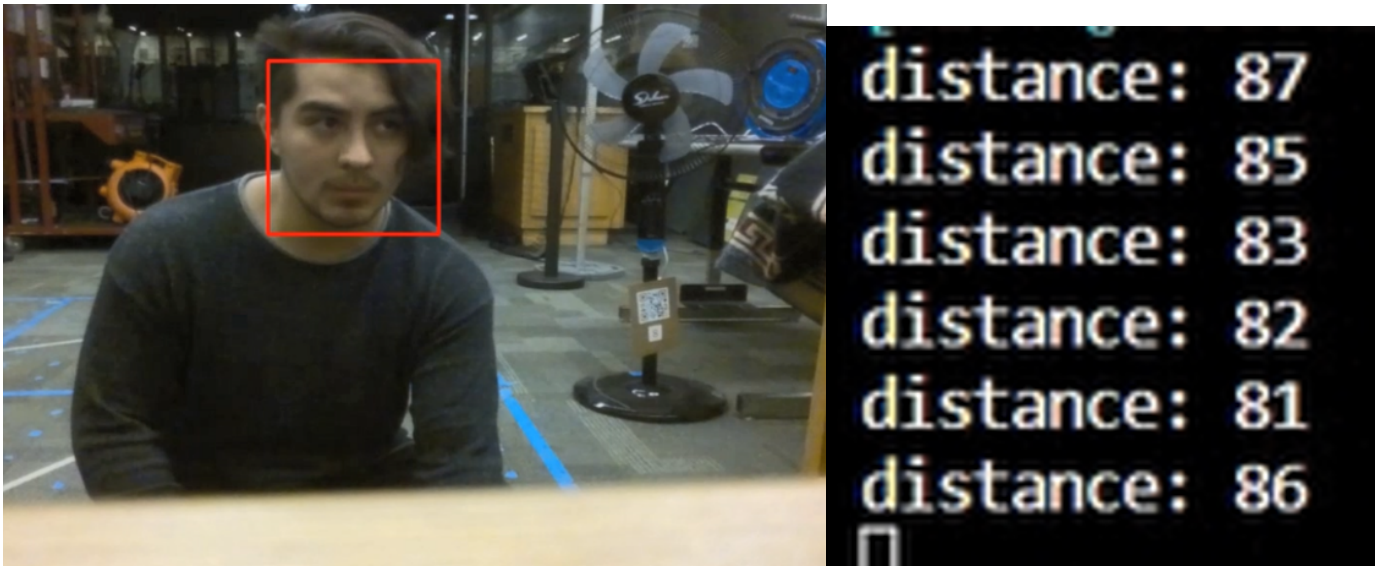

Fig. 17. Live video feed from the drone with calculated distances outputted to the console where each measurement is from a different frame while the user was still.

can only happen when the drone has a bounding box to work with. Future work will enable the drone to search for a face to continue following. Photos of these test runs are depicted in Figure 18. Due to time constraints, kinetic tracking was not yet implemented.

VII. Conclusion and Future Work

The rates of success for the custom ML models are not as high as expected. It is possible that there are issue with our drone-captured data set, but it cannot be concluded for sure. Room for improvements are though to be possible by having the unauthorized user (negative) data set be taken from the drone as well in the case that there were issues in resizing images and editing.

Future work will include improving the ML models so that only specific users will be recognized and activate the drone. A possibility can include implementing kinetic tracking of hands to command specific maneuvers of the drone such as different direction flipping. Another future ability should entail the drone to find a user when the user is lost from camera frame.

It can be concluded that ML models do as they are trained to do and thus, are only as good as their training sets. The difficulty comes in what determines a good training set and how to improve one without knowing for sure what causes shortcomings in testing. Training sets may be improved by having all photos come from the same source and/or taken within the same environment. In this project, it was learned how to do real-time image processing to dictate the tasks of the drone using Python as well as how to train varying ML models.

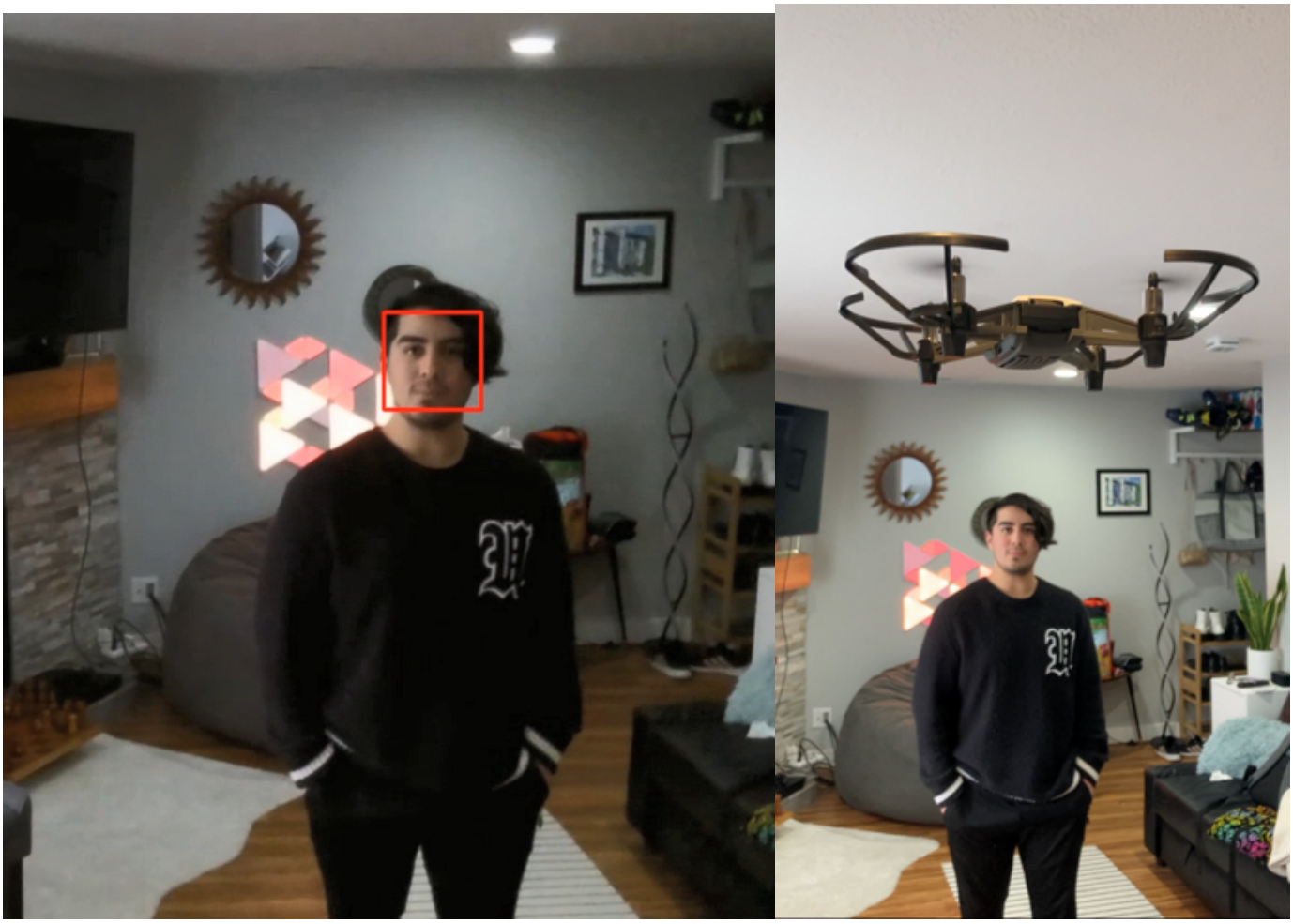

Fig. 18. Demonstrations of the drone tracking a face and following from the view of the drone and the view of an external source where both photos are from approximately the same time.

REFERENCES

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[2] ——, "Identity mappings in deep residual networks," 2016.

[3] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.

[4] M. Hearst, S. Dumais, E. Osuna, J. Platt, and B. Scholkopf, "Support vector machines," *IEEE Intelligent Systems and their Applications*, vol. 13, no. 4, pp. 18–28, 1998.

Algorithm 1 Centering Face in Front Camera Frame

Function name: trackObject()

Require: dimensions of bounding box: $length, width$
Location of bounding box within camera frame: x, y

Ensure: $340 <= x <= 380$, $240 <= y <= 340$, z = height of user
$x\_distance\_cutoff \leftarrow 150$ cm ▷ How close to the drone are you comfortable with?
if object detected **then**
$distance \leftarrow (650 * 18.5/width)$
▷ distance is approximately focal length of camera lense * Real-world width of object) / Width of object in pixels
if $distance <= x\_distance\_cutoff$ **then** move drone back $x\_distance\_cutoff - distance$
end if
if $0 < y <= 240$ **then** move drone down
else if $y >= 340$ **then** move drone up
end if
if $0 < x <= 340$ **then**
$angle \leftarrow ((360 - x)/360) * 18.5$
$shift \leftarrow abs(distance * sin(angle * \pi/180))$
if $shift < 20$ **then** move drone counter-clockwise by $angle$
else move drone left by $shift$
end if
trackObject()
else if $x >= 380$ **then**
$angle \leftarrow ((x - 360)/360) * 18.5$
$shift \leftarrow abs(distance * sin(angle * \pi/180))$
if $shift < 20$ **then** move drone clockwise by $angle$
else move drone right by $shift$
end if
trackObject()
flip drone forward
end if
else
pass
end if