iRecruit

Author: Mohini Banerjee, Shilika Gehlot, Jessica Meng: Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—iRecruit is an interview assistant capable of providing software engineering jobseekers with the opportunity to practice for the interview process. Students are challenged with navigating fully virtual interviews and practicing how to conduct themselves during behavioral and technical interviews. Although there exist several written guidelines about common interview practices and questions, there is a lack of opportunity to practice a simulated interview with a "real" interviewer. iRecruit aims to give users a chance to practice for interviews through facial detection for behavioral interviews and speech recognition (a combination of signal processing and machine learning) for technical interviews.**

Index Terms—Facial detection, Facial landmark detection, Fourier transform, Haar Cascades, Machine learning, Neural network, Gaussian mixture modelling, Signal processing

I. Introduction

A common task software engineering jobseekers are faced with when searching for a job is interviewing. There exist several resources for assisting jobseekers with their interview process including lists of common behavioral interview questions and platforms to practice technical problem solving. Common technical interview platforms include *HackerRank* and *LeetCode*, where users are able to solve problems in a wide range of Computer Science topics such as linked lists, dynamic programming, and strings. Additionally, common behavioral interview questions and techniques are provided in books such as *Cracking the Coding Interview* and *the Google Resume*. We wanted to improve upon current resources by creating a centralized platform where users are able to practice for both behavioral and technical interviews in a simulated environment. This way, users are able to gain an understanding for what real, virtual interviews are like.

For behavioral interviews, users are presented with three options to practice with and are asked to video record themselves answering a common behavioral interview question. During the recording, iRecruit tracks the user's eye contact and/or screen alignment, and provides real-time feedback on these factors. The goal of this facial detection algorithm is to detect the user's facial feature coordinates within five seconds and alert the user of subpar eye contact or screen alignment within five seconds with 80 percent accuracy. For technical interviews, users are asked to pick a Computer Science category from a given list provided on the platform. The list includes the following eight categories - 1. Array, 2. Binary Tree, 3. Dynamic Programming, 4. Java, 5. Linked List, 6. Python, 7. Recursion, 8. String. Their choice is then submitted via an audio recording. iRecruit uses signal processing techniques on the audio recording to generate a viable input to feed into a neural network that outputs the predicted category. A question is displayed on the screen relevant to the interviewee's chosen category. The goal of this speech recognition algorithm is to identify the category spoken by the user with 65 percent accuracy.

II. Design Requirements

We split the design requirements of our web application into three main components:

- Facial detection
- Signal processing
- Machine learning

Each component had a series of requirements that we aimed to meet. For overall qualitative requirements, we wanted to make our code consistent and well-documented. This way, while all team members had different coding styles, their code was readable and understandable for other team members. We also wanted to ensure that the iRecruit web application is usable and consistent, so that users are able to navigate the web application easily and understand how the various web pages are connected.

For the facial detection portion, there were four main requirements. The first one was that the system should be accommodating for users of various levels of experience. Users will likely come to iRecruit with different levels of exposure to behavioral interviewing, where some will be completely unfamiliar and others will be moderately or highly experienced. The system should account for these differences, so that users are able to practice in an environment that they can benefit the most from. The second requirement was that the facial detection system should be quick in detecting the user's facial features. This provides the user with an efficient experience, so that they do not have to wait for an extended period of time for the system to locate their facial features. More specifically, the system should detect the user's facial features within five seconds of the start of the video recording. The third requirement was that the system should be reliable in alerting the user of subpar eye contact or screen alignment. If the user's eyes are off-center or face is off-center, then the system should alert the user within five seconds. Additionally, the alert(s) should be noticeable by the user, so they can react accordingly. The last requirement was that the facial detection system should have a high accuracy to provide the user with a beneficial experience. We aimed to have an overall accuracy of approximately 80 percent, because we used the OpenCV library in Python. OpenCV has built-in Haar Cascades, which are pre-trained classifiers for features

such as faces, eyes, and smiles[6]. The accuracy of the Haar Cascades is around mid-90 percent[27], so we thought that aiming for 80 percent with our system that builds on top of Haar Cascades was a reasonable accuracy achievement.

For the signal processing portion, there were three main requirements. The first requirement was formulating the output of the audio signal from the user. The goal was for each audio signal to be portrayed as a feature vector in binary format consisting of an accurate and complete representation of the original signal. As a result, each input audio signal could be processed and stored as training data for the machine learning algorithm. The output was formulated through the use of different signal processing techniques that are discussed under the "System Description" section. This would allow us to manipulate and transform the input to represent each word in a way that a neural network can understand. The second requirement was that each input category as an audio file must have a distinct, unique visual representation. In order to make the output meaningful, the resulting visual representation of each audio signal would have characteristics that distinguish a specific word or phrase from another, while having similar characteristics for the same word or phrase. This means that the algorithm had to be flexible in terms of accommodating for details such as silence and background noise. The third and last requirement of this portion was in regards to testing the accuracy of our algorithm. This portion was difficult to test concretely for two reasons. First, the signal of different people recording the same word differed due to differences in pitch, loudness, and frequency. Second, the signal representing the same person saying the same word varied as well. To ensure our algorithm was behaving as expected, we required the testing to be mostly manual and did this through comparing the same words spoken by the same person and ensuring the corresponding visual representations were similar.

For the machine learning portion, there were three main requirements. First, we had to accumulate considerable training data that was generated from the signal processing algorithm described above. To build the training data, we reduced the dimension of each individual data sample in order to improve the time complexity of the machine learning algorithm. Ultimately, we created 105 samples of training data - approximately 13 samples for each of the eight possible output categories. A subset of our training data was used as validation data to prevent any overfitting. The second requirement consisted of the output being a probability distribution across the different possible categories. The word with the highest probability was the algorithm's prediction and was compared to the true word that the user spoke. Third, we expected for the machine learning algorithm to have an accuracy of 65 percent. Speech recognition is a difficult task that many talented engineers have been working on for years. Since we were three college students with three months of time, we aimed to produce a simplified version of the speech recognition algorithm. In the beginning, our goal was to reach 60 percent accuracy. However after receiving feedback from our peers to aim slightly higher, we changed our goal for our model to correctly determine the category 65 percent of the time.

III. Architecture and/or Principle of Operation

We split the architecture of iRecruit's system into four main parts:

- Web application
- Facial detection
- Signal processing
- Machine learning

The facial detection, signal processing, and machine learning components exist within the main web application. The facial detection portion exists relatively independently, while the signal processing and machine learning portions were integrated with each other.

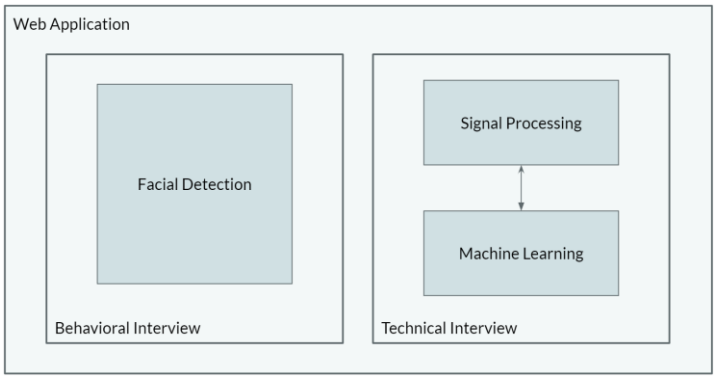

Fig. 1. High-level diagram of the 4 main parts of iRecruit's system.

The web application is the baseline for the system, as this is where the user will be able to access the various components. The web application is split into three main components:

- Behavioral
- Technical
- Profile

When a user registers or logs into the system, they will be led to the dashboard, where they are able to navigate to these three pages. Additionally, there is a permanent sidebar menu where users are able to go to the "Behavioral" and "Technical" pages. If they choose to go to the "Behavioral" page, they will be presented with three possible options to practice with. These three options are to account for various levels of experience with behavioral interviewing and are elaborated upon in the "System Description" section. For each option, the user will be presented with a randomly generated common behavioral interview question that they will video record themselves responding to. iRecruit expects that the user is sitting in front of the camera and that their face will be visible in the video screen. When the user is recording, the facial detection portion will attempt to detect their facial features using Python's OpenCV library, specifically Haar Cascades and/or facial landmark detection. This may require the user to position themselves accordingly and adjust their angle if necessary, as the Haar Cascades and facial landmark detection are sensitive to the angle of the face. The facial detection algorithm allocates five seconds to an initial setup phase, to allow the user to position themselves and depending on the option, for the system to determine the frame(s) of reference of the user's appropriate facial feature coordinates. Once the system has the frame(s) of reference, it will refer back to this for each video frame. If the user's facial feature coordinates are not within a range of the

frame(s) of reference coordinates (also known as off-center), the system will alert the user within five seconds. This alert is a visual pop-up message box, so it is clear to the user that they are off-center and to reposition.

If a user chooses to go to the "Technical" page, they will be presented with a screen that lists the eight categories of questions that iRecruit offers. Users are then able to audio record their category of choice, and iRecruit runs its speech recognition algorithm to provide the user with a randomly generated technical question from our database. Our backend database includes questions for each of the following possible categories - arrays, binary trees, dynamic programming, Java, linked lists, Python, recursion, and strings which we either created based on prior knowledge or retrieved from the LeetCode website[28]. Users are expected to determine and submit the output of the technical question into the designated answer form, at which point iRecruit will display the correct answer on the screen. Each time the user answers a question, it is stored in a database so that an up-to-date list of questions that a given user has answered is kept for their convenience.

The user's past behavioral and technical interview results are saved in the "Profile" section of the web application. On the "Profile" page, the user is able to choose to go to either the "Completed Behavioral Interviews" or "Completed Technical Interviews" pages. In the "Completed Behavioral Interview" page, they can review past behavioral interview practices to look at a summary of the results for each practice. The summary will consist of the video recording number with the option practiced with, timestamp of when the practice took place, subpar eye contact count, and subpar screen alignment count. In the "Completed Technical Interview" page, users can review past technical interview practices, which contains a record of all the technical questions answered so far, as well as the user's answer and the correct answer for each question.

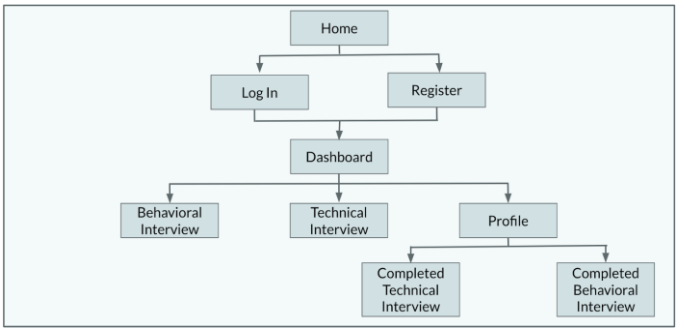

Fig. 2. High-level diagram of the web application architecture.

Fig. 2. is a high-level diagram of the web application architecture that shows the different web pages we have as well as how the pages connect to each other. The behavioral interview, technical interview, and profile pages are where most of the backend processes (facial detection, signal processing, and machine learning) happen. On the other hand, the home, log in, register, and dashboard pages are more for sake of completeness of our web application and to provide an intuitive flow between different pages.

IV. Design Trade Studies

We considered various approaches to each of the components of the system and performed an initial research phase to weigh these approaches. There were a handful of options and algorithms that we took into account, but ultimately went with the ones that we believed had the most available documentation and provided us with a high accuracy.

A. *Web Application*

For the web application, we chose to use a Python web framework known as Django to build it. Django is one of the more popular existing web frameworks. There are four main reasons we chose to use Django. First, it is based on the Python programming language, there exists many tutorials and documentation for us to refer to, and many of the web components are already developed, allowing us to focus on the features we are trying to implement. Second, Django is fast and simple as making changes in the frontend or backend code does not require the whole system to restart. This allows for separate testing of backend and frontend components. Third, it is secure as Django security protects against clickjacking, cross-site scripting, and SQL injection[13]. Lastly, it is suitable for any web application project of any size and capacity. Django can handle large amounts of data, be used on any operating system including Mac, Windows, and Linux, and incorporate multiple databases into the project to store information. Due to these reasons, we did not consider alternate web frameworks to host our application and believed Django to be the best approach from the start.

B. *Facial Detection*

For the facial detection portion, we decided to use the OpenCV library in Python, particularly for Haar Cascades. There were two main Python libraries that we were considering for facial detection, which were dlib and OpenCV. dlib is a library that contains machine learning algorithms and tools, and although it is principally a C++ library, it can also be used with Python[5]. It has a Histogram of Oriented Gradients (HOG) feature descriptor, which is very powerful and actually more accurate than OpenCV Haar Cascades[12]. However, we ultimately chose OpenCV, because OpenCV is much more commonly used in Python. It also has more documentation and tutorials available, which we thought would be helpful because we were not familiar with facial detection before iRecruit. Haar Cascades also have an accuracy of approximately 95 percent[27], which we believed was a sufficient baseline accuracy for us to build on top of for our aim of 80 percent accuracy. We were originally going to measure three things in the facial detection portion: eye contact, posture, and screen alignment. However, we decided to forgo posture, as there was no sufficient way to measure it. The first measurement we thought of was to attempt to detect the mouth, and if the mouth disappeared (e.g. user's head is down), that constituted subpar posture. However, if there is no mouth detected, there is also no face detected. Another measurement we thought of was to attempt to detect the shoulders and measure the distance between the shoulders and the center of the face, and if that distance was less than that of the frame of reference, that constituted subpar posture. However, if a user has long hair and the hair is covering the shoulders, shoulder detection would not be possible. Forgoing this measurement allows us to focus on implementing the eye contact and screen alignment portions, and making them more

robust to meet accuracy demands. For the facial landmark portion of the facial detection system, we decided to find and only use the coordinates of the nose and mouth, so that we would have definitive coordinates instead of a plethora of coordinates. This allowed us to find the frame of reference for subpar screen alignment detection, as elaborated upon in the "System Description" section. Another decision we made was to have three possible options to practice with. We wanted the system to be accommodating for users of different levels of experience, so that users can select which option suits them best. While this required separating and combining implementation on our end, we thought this would be beneficial from a user perspective. More information on the three options is provided in the "System Description" section. Facial detection accuracy was calculated by the following equation[19], where TP stands for the number of true positive tests, TN stands for the number of true negative tests, FP stands for the number of false positive tests, and FN stands for the number of false negative tests:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (1)$$

Positive here is defined as detecting subpar eye contact and/or screen alignment. Therefore, true positive tests occur when the user has subpar eye contact or screen alignment, and the system alerts them correctly within 5 seconds. True negative tests occur when the user does not have subpar eye contact or screen alignment, and the system does not alert them. False positive tests are when the user does not have subpar eye contact or screen alignment, and the system alerts them. False negative tests occur when the user has subpar eye contact or screen alignment, and the system does not alert them.

C. Signal Processing

For the signal processing portion, we hit many roadblocks as we experimented with the best approach to reach the desired output. There are 4 main decisions we had to make. First, our initial design process involved letter-by-letter classification. However, early into the design process, we decided to reduce the scope of the speech recognition algorithm by implementing word classification on a predetermined list of eight categories. Although letter-by-letter classification would give the user the freedom to choose a category of their choice, instead of from a predetermined list, we decided that we did not have the capacity to build such a system in the given time. Another advantage with word classification is that the user can speak the word rather than spelling it out, which is impractical and what letter-by-letter classification would require.

Second, we faced confusion on whether to use the time domain or frequency domain to represent the signal. The time domain signal seemed to provide us with more information as the signal on the graph was raw and not manipulated. However, the amplitudes of different signals representing the same word were different. We recognized this was likely due to differences in pitch, frequency, and loudness. Thus, we tried representing the signal in the frequency domain through the use of the Fourier transform. This also caused complications as every signal, no matter the word, looked similar and each representation had a peak at the lower frequencies and another peak at the higher frequencies. The Fourier transform tells us what frequencies are present in our signal. It also tells us how much power the time domain signal has at each frequency. By breaking a signal into its frequency components, it allows us to block out certain frequencies[22]. We realized that although the information that the frequency domain gives is less intuitive than the time domain, it would ultimately help differentiate between words.

The next decision involved using a windowing function to analyze the Fourier transform. We decided to split the audio signal into 20 millisecond (ms) chunks. In theory, this is equivalent to multiplying the signal by a rectangular window to extract the 20 ms chunks. Based on research of why windowing is necessary, we found that taking the Fourier transform of a signal that is not perfectly periodic results in some discrepancies. The Fourier transform assumes the time domain signal is a finite, periodic signal. When this is not the case, the endpoints of the signal are not continuous, and this discontinuity is present in the frequency domain as high frequency components that do not exist in the original signal. Therefore, the Fourier transform is not accurate, as the different energy components are leaking onto each other[26]. We found that windowing can help provide a solution for this problem. It can reduce the impact of the discontinuity present in the time signal and thus provide a more accurate representation of the signal in the frequency domain. It consists of multiplying the time domain signal by a window of finite size that has a sinusoidal amplitude that approaches zero at the discontinuous parts. Out of all the possible windows, we saw that the Hamming window function best emulates this sinusoidal pattern, as it has a wide peak but low side lobes[26].

Lastly, we had to decide on the optimal representation of the training data. The tradeoff was between using the original spectrogram representation that was the result of the windowing applied to the Fourier transform (as described above) or reducing the dimension of the former representation through the use of the Mel Filter Coefficient Bank approach. We decided to use the latter, because although the former is a better representation of the data, it drastically slowed down our system when the neural network tried to process the data due to the large size of the data. Additionally, the Mel Filter Coefficient Bank approach applies frequency scaling that simulates how the human ear works[24]. Once this decision was made, we had to decide whether to store the raw data provided from the Mel Filter Coefficient Bank approach as the training data or further modify it. We decided to plot the raw data as a spectrogram representation, save it as an image and use the pixel values of the image as our training data. This was because we saw visible differences in the spectrogram between different words and extreme similarities between different people speaking the same word.

D. Machine Learning

To implement the machine learning algorithm for the speech recognition portion, we were debating two main approaches, neural network and Gaussian mixture modelling. The goal of a neural network was to program a simulation of connected human brain cells so eventually it could recognize patterns, point out similarities and differences, and behave like a human brain[11]. As discussed in more detail under the "System

Description" section, a complete neural network consists of the input and output layer, and one or more hidden layers. Varying details such as the number of hidden layers, hidden units, or epochs change the amount of time it takes to train the algorithm. These details were taken into account as we began the implementation process. The other approach we considered, the Gaussian mixture model, is a probabilistic model that represents subpopulations within an overall population that have a normal distribution[9]. By gathering user input of people speaking different words, we could create a model that would determine the probability of likelihood of a specific word being spoken. This model would predict unknown utterances when the user is speaking one of the eight possible categories. The Gaussian mixture model would not require as many training samples as a neural network would because creating a probabilistic model is less complex than trying to simulate the human brain.

When deciding which approach to pursue, we considered the pros and cons of both the neural network and Gaussian mixture model. Our final decision was most influenced by the fact that none of us had experience with Gaussian mixture modelling while two of us had experience with neural networks. We believed that learning a new algorithm from scratch may not have been the best use of our time. Therefore, our team decided to first try the neural network algorithm as we are most familiar with it. The machine learning algorithm was adapted from the neural network algorithm Mohini wrote in 10-301: Introduction to Machine Learning. If the classification was completely incorrect and time permitting, we would have looked into using Gaussian mixture modelling. After deciding to go with the neural network, we discussed exactly what we are trying to classify within speech recognition. The most advanced algorithms such as those used to implement Siri, Alexa, and Google Assistant are able to identify full sentences. Due to our time constraint, we decided to narrow our scope. Thus, we decided to make our speech recognition model identify the eight categories our product offers for technical interview practices. This way, we are only trying to distinguish between eight possibilities, and not the plethora of words and phrases that exist in the English language.

1. *Number of Hidden Units and Epochs Trade-offs*

With our algorithm and classification decided, we made decisions on variable factors that would change the accuracy of our model, such as the number of hidden layers, hidden units, and epochs. To train our algorithm, we used 105 training data samples and to validate (or test), we used 50 testing data samples. We automated the testing process by calculating and displaying the percent of training and testing data that was classified incorrectly given each dataset. The first set of parameters used to test the algorithm were one hidden layer, four hidden units, and 100 epochs. All the mentioned parameter values, excluding the number of hidden layers, were easy to update as we designed our code to allow us to input a specified value for each of these variables. These parameter values resulted in an error rate of 73.3 percent for predicting the output of the training data and 82.6 percent for predicting the output of the testing data.

As we experimented with different values for the number of hidden units, we saw that more hidden units led to a larger accuracy. At a value of 10 hidden units, the error dropped to approximately 0 percent for the training data and 60.9 percent for the testing data. Past 10 hidden units, the error for both training and testing data stayed consistent as shown in the figure below. When we changed the number of total epochs, we saw a very similar trend, as the testing accuracy increased until 100 epochs and plateaued for values above that. Therefore, we decided to use 10 hidden units and 100 epochs to maximize the probability of our algorithm predicting the correct category and to minimize the total time it takes to run the algorithm. This is because with each additional hidden unit or epoch, the amount of time the neural network takes to learn increases.

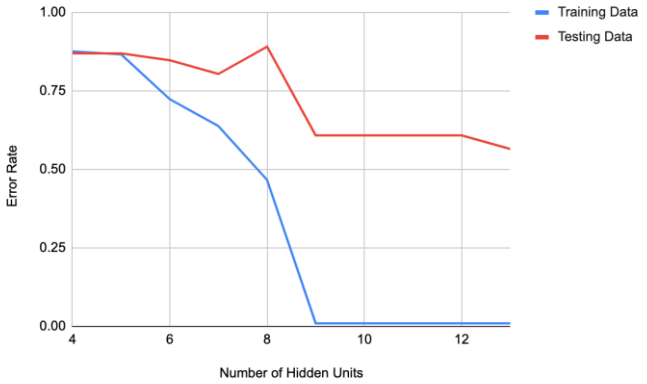

Fig. 3. Error rate of training and testing data of number of hidden units from 4 to 13.

2. *Number of Hidden Layers Trade-offs*

After finding optimal values for the number of hidden units and number of epochs, the remaining parameter to alter was the number of hidden layers. We added an additional hidden layer to the algorithm, resulting in two hidden layers in between the input layer and the output layer, each with 10 hidden units. This process involved restructuring the baseline algorithm. Currently, the algorithm consists of the input, hidden, and output layers, and the weight matrices, alpha and beta, between each layer to compute the probability distribution over the eight classes. With an additional layer, we added another weight matrix α_2 between the first and second hidden layer and modified the back propagation algorithm so it takes into account α_2 when computing the updated weight matrices. With this modification, the error increased to 87.3 percent for our training data and 84.8 percent for our testing data using the original 105 training samples and 50 testing samples. These results were surprising, because an additional hidden layer typically increases training accuracy by overfitting the training data[2].To improve the accuracy, we tried increasing the number of training data samples as more data typically results in a higher accuracy since the algorithm has more information to learn from . However, the error with 130 training samples and 50 testing samples resulted in a training error of 86.4 percent and testing error of 87.5 percent. We believe these conflicting results may have been caused by the fact that with every additional hidden layer, the input feature vector is further manipulated. Due to the complexity of the audio inputs that were processed in the signal processing component of our

project, these significant transformations may have had a negative effect on learning trends and patterns, as the values in the output layer were confusing to learn from. Thus, we ultimately reverted back to our algorithm with a single hidden layer as it had higher accuracy measures.

3. *Number of Training and Testing Data Trade-offs*

As mentioned above, the number of training and testing data samples we used varied our accuracy results. Once we finalized the parameter values of one hidden layer, 10 hidden units, and 100 epochs, we decided to add additional training data. With 105 training data samples and 50 testing samples, the training error and testing error was approximately 0.01 percent and 60.9 percent respectively. This model overfit the data because it classified the training data almost to perfection and was not able to successfully generalize the unseen testing data. We predicted that adding additional training data would reduce overfitting, since the training data would have more diversity and our model would not try to classify every single training data sample perfectly[2]. This would allow the algorithm to generalize unseen data, in turn increasing the testing accuracy. However, this was surprisingly not the case. With 180 training data samples and 50 testing data samples, the error rate for training and testing data were 86.2 percent and 84.8 percent respectively. Although the training error increased and avoided overfitting, the margin by which it increased was significantly higher than expected. Additionally, the testing error increased by approximately 24 percent which rendered the accuracy of our model to a mere 15.2 percent. We reverted back to our original model by only training with 105 training data samples. We discuss cross-checking the accuracy of this model in the "System Description" section.

V. System Description

A. *Facial Detection*

The facial detection portion of iRecruit was implemented utilizing the OpenCV library in Python. The face and eye detection and facial feature detection parts of it were based off of two tutorials for eye tracking and facial landmark detection[8, 17]. When a user records themselves answering a practice behavioral question, the system calls the OpenCV VideoCapture class to begin a new capture of video frames at a rate of 30 frames per second[16]. The practice behavioral question presented to the user comes from a question bank that is an array of common behavioral interview questions from Top Echelon[18]. The random library in Python is used to randomly select one of the questions from the question bank. The question is displayed to the user through OpenCV's putText method at the top of the video screen. The word "RECORD" will appear in red on the bottom right of the screen when the initial setup phase is complete, which indicates to the user that they may begin answering the question. There is a yellow circle in the middle of the screen to serve as a guideline for users to center their face. See Fig. 4 for an example of what the behavioral interview practice platform looks like.

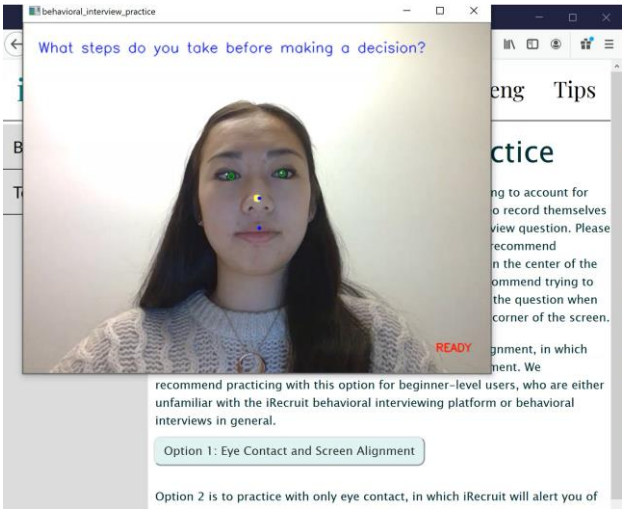

Fig. 4. Example of the video screen presented when the user is practicing.

For the alerts to the user, this was done through the Python ctypes library MessageBox function to notify the user visually with a pop-up message box. iRecruit provides three options for users to practice with to account for various levels of experience. The first option is for beginner-level users, who are unfamiliar with behavioral interviews in general or the iRecruit behavioral interviewing platform and want as much feedback as possible. It allows for users to practice with both eye contact and screen alignment, where iRecruit provides real-time feedback for both behavioral interviewing tactics. The second and third options are for intermediate-level to advanced-level users, who are familiar with behavioral interviewing and understand what they would like to improve upon. The second option allows for users to practice with only eye contact, where iRecruit provides real-time feedback on subpar eye contact. The third option allows for users to practice with only screen alignment, where iRecruit provides real-time feedback on subpar screen alignment. This is useful if a user knows their strengths and weaknesses with behavioral interviewing and only desires to practice with feedback on one of the tactics.

For option 1, there will be 2 parts that exist within the facial detection portion, one for eye contact and one for screen alignment.

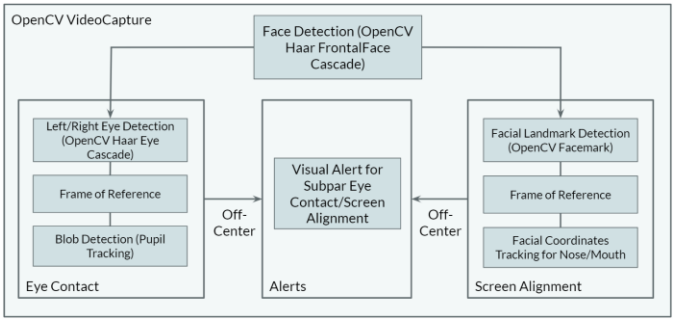

Fig. 5. Block diagram for facial detection option 1 architecture.

For option 2, there will be one part that exists within the facial detection portion for eye contact.

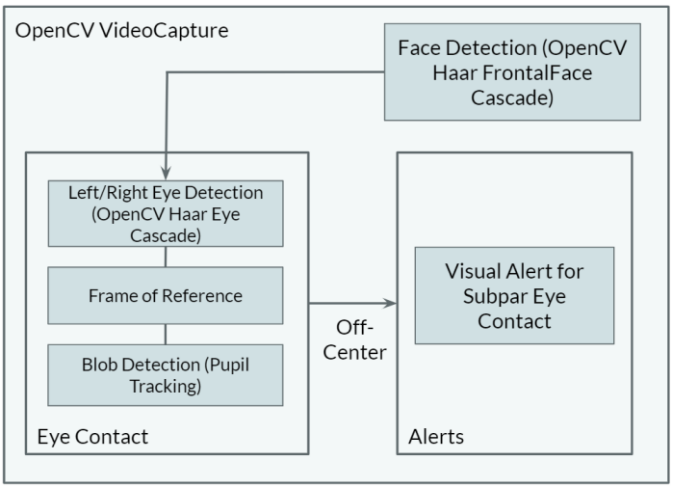

Fig. 6. Block diagram for facial detection option 2 architecture.

For option 3, there will be one part that exists within the facial detection portion for screen alignment.

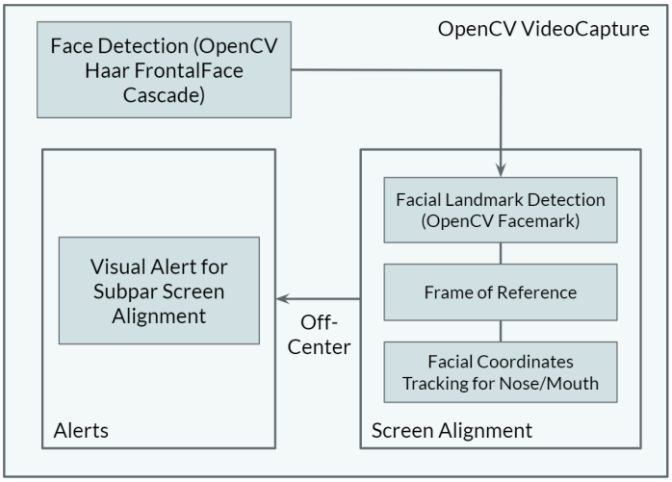

Fig. 7. Block diagram for facial detection option 3 architecture.

The eye contact and screen alignment parts are further described in detail, as well as the testing procedure and results. The implementation of each part is the same regardless of the option — the main variation is the combination of which parts exist in each option.

1. *Eye Contact*

The eye contact portion utilizes OpenCV Haar Cascades as well as the numpy library. For each frame of the video, the system will use the frontal face Haar Cascade to detect the user's face. The base image (video frame) is transformed into grayscale in order to detect the face, and the face is found on the original base image. After the face is retrieved, the system will use the eye Haar Cascade to detect the user's eyes. The base face image is transformed into grayscale in order to detect the eyes. The height and width of the original face image is calculated using numpy to make the eye detection easier. This is because the eyes will always exist on the upper half of the face, and the left eye will be on the left half of the face, while the right eye will be on the right half of the face. After the eyes are retrieved, the system will track the irises/pupils within the eyes using a blob detector in OpenCV[14]. The eyes are then transformed into grayscale and a threshold is set to determine the cutoff of which parts of the eye become black and white, which allows for the detection of the irises/pupils. When the irises/pupils are found, OpenCV moments are used to calculate the centroid[25]. The center is with respect to a specific origin, which is the left edge of each iris (in alternative wording, 0 is the left edge of each iris, not the left edge of the screen). Each eye usually has the same center if the user is facing straight and forward, because of this origin reference. The center is calculated every frame, and the X and Y coordinates of the center are added to arrays over the period of the 5 second initial setup phase. The average of these coordinates is taken to find the frame of reference, which is what the system will use as the baseline of what constitutes as centered for the eyes. In every video frame following, if the coordinates of the user's eyes are not within range of the frame of reference for up to 5 seconds, iRecruit alerts the user visually of subpar eye contact and to adjust accordingly. The user's eye movement must be severe (e.g. looking completely off screen) for longer than simply an instant to be detected.

2. *Screen Alignment*

The screen alignment utilizes OpenCV Haar Cascades and the Facemark API to perform facial landmark detection. Similar to the eye contact portion, for each frame of the video, the system will use the frontal face Haar Cascade to detect the user's face. The base image (video frame) is transformed into grayscale in order to detect the face, and the face is found on the original base image. After the face is retrieved, the system will use the Facemark API Local Binary Features (LBF) to determine the locations of all landmark facial features[15]. This includes the eyebrows, eyes, nose, mouth, and edges of the face. We decided to find and pinpoint the coordinates of the nose and mouth, as this allows for definitive coordinates to use for the frame of reference. If the nose coordinates are not centered, then neither are the mouth coordinates, and vice versa. We wanted to have both the nose and mouth coordinates for points of reference in case the facial landmark detection for one of them fails unexpectedly. The X and Y coordinates of the nose and mouth are calculated every frame and added to arrays over the period of the 5 second initial setup phase. The average of the coordinates is taken to find the frame of reference, which is what the system will use as the baseline of what constitutes centered for the nose and mouth. In every video frame following, if the coordinates of the user's nose (or mouth if the nose detection fails) are not within range of the frame of reference for up to 5 seconds, iRecruit alerts the user visually of subpar screen alignment and to adjust accordingly. The user's movement off screen is detected once they pass 150 pixels to the left or right of the frame of reference coordinates.

3. *Testing and Results*

The accuracy for each of the three options exceeded our goal for the overall facial detection system accuracy of 80 percent. We decided to test and calculate the accuracy for each option individually, as all of the options are distinct in what part(s) they are incorporating. The testing was manually performed, as the facial detection system depends on user interaction and provides real-time feedback. We looked into automated testing, and found that it has been and could be done through a robotic

arm that is able to rotate to recognize faces and perform tests[1]. However, this was out of scope for our project, as we did not have the resources or experience needed to create this automated testing setup. Therefore, we decided to do manual testing, where Jessica attempted to test all possible scenarios users would run into.

For option 1, both eye contact and screen alignment were tested. A positive test meant providing the user with the correct alert within 5 seconds if they had subpar eye contact or screen alignment (True Positive (TP)), or not providing the user with an alert if they did not have subpar eye contact or screen alignment (True Negative (TN)). A negative test meant providing the user with no alert if they had subpar eye contact or screen alignment (False Negative (FN)), or providing the user with an alert if they did not have subpar eye contact or screen alignment (False Positive (FP)). All of these test cases were covered, where Jessica would have subpar eye contact, subpar screen alignment, or neither. The most common negative tests included the system alerting the user of subpar eye contact when the user did not actually move their eyes (FP tests), and the system alerting the user of subpar eye contact when the user actually had subpar screen alignment (FP and FN tests — FP for eye contact and FN for screen alignment). Each test constituted for two separate test cases, one for the result of eye contact and one for the result of screen alignment. For example, if the user has subpar screen alignment, the system should have an alert for subpar screen alignment, but no alert for subpar eye contact. Both behavioral interview tactics account for one test case each. We kept track of the centered eye coordinates, whether the eye coordinates of the current frame were off-center, the eye contact test result (TP, TN, FP, FN), the centered facial landmark coordinates, whether the facial landmark coordinates were for the nose or mouth, whether the nose/mouth coordinates of the current frame were off-center, and the screen alignment test result (TP, TN, FP, FN). Part of the test table is shown in Table I. There was a total of 106 test cases, of which there were 27 TP results, 63 TN results, 13 FP results, and 3 FN results. Following equation (1) from the "Design Trade Studies" section, this results in an accuracy of 84.91 percent.

TABLE I. FACIAL DETECTION OPTION 1 TEST TABLE (PARTIAL)

Centered Eye Coordinates (X, Y)	Eyes Off-center?	Eye Contact Test Result (TP, TN, FP, FN)	Centered Facial Landmark Coordinates (X, Y)	Nose or Mouth?	Off-center of Screen?	Screen Alignment Test Result (TP, TN, FP, FN)
(24, 25)	No	TN	(305, 239)	Nose	Yes	TP
(24, 25)	No	TN	(305, 239)	Nose	No	TN
(25, 25)	Yes	TP	(345, 263)	Nose	No	TN
(25, 25)	No	FP	(345, 263)	Nose	Yes	FN
(25, 25)	No	TN	(345, 263)	Nose	No	TN
(22, 22)	No	TN	(303, 260)	Nose	Yes	TP
(22, 22)	No	FP	(303, 260)	Nose	Yes	FN
(22, 22)	Yes	TP	(303, 260)	Nose	No	TN
(22, 22)	No	FP	(303, 260)	Nose	No	TN
(21, 21)	No	TN	(328, 282)	Nose	Yes	TP
(21, 21)	Yes	TP	(328, 282)	Nose	No	TN
(21, 21)	No	FP	(328, 282)	Nose	No	TN

For option 2, only eye contact was tested. A positive test meant providing the user with the correct alert within 5 seconds if they had subpar eye contact (True Positive (TP)), or not providing the user with an alert if they did not have subpar eye contact (True Negative (TN)). A negative test meant providing the user with no alert if they had subpar eye contact (False Negative (FN)), or providing the user with an alert if they did not have subpar eye contact (False Positive (FP)). All of these test cases were covered, where Jessica would have subpar eye contact or not. The most common negative tests included the system alerting the user of subpar eye contact when the user did not actually move their eyes (FP tests), and the system not alerting the user of subpar eye contact within 5 seconds (FN tests). We kept track of the centered eye coordinates, whether the eye coordinates of the current frame were off-center, and the eye contact test result (TP, TN, FP, FN). Part of the test table is shown in Table II. There was a total of 54 test cases, of which there were 35 TP results, 12 TN results, 5 FP results, and 2 FN results. Following equation (1) from the "Design Trade Studies" section, this results in an accuracy of 87.04 percent.

TABLE II. FACIAL DETECTION OPTION 2 TEST TABLE (PARTIAL)

Centered Eye Coordinates (X, Y)	Eyes Off-center?	Eye Contact Test Result (TP, TN, FP, FN)
(23, 23)	Yes	TP
(23, 23)	No	TN
(23, 23)	Yes	FN
(21, 21)	Yes	TP
(21, 21)	Yes	TP
(24, 24)	No	FP
(24, 24)	No	TN
(24, 24)	Yes	TP
(26, 26)	Yes	TP
(26, 26)	Yes	TP
(25, 25)	No	FP

For option 3, only screen alignment was tested. A positive test meant providing the user with the correct alert within 5 seconds if they had subpar screen alignment (True Positive (TP)), or not providing the user with an alert if they did not have subpar screen alignment (True Negative (TN)). A negative test meant providing the user with no alert if they had subpar screen alignment (False Negative (FN)), or providing the user with an alert if they did not have subpar screen alignment (False Positive (FP)). All of these test cases were covered, where Jessica would have subpar screen alignment or not. We kept track of the centered facial landmark coordinates, whether the facial landmark coordinates were for the nose or mouth, whether the nose/mouth coordinates of the current frame were off-center, and the screen alignment test result (TP, TN, FP, FN). Part of the test table is shown in Table III. There was a total of 51 test cases, of which there was only one negative test

case, where the system did not alert the user of subpar screen alignment (FN test). There were 38 TP results, 12 TN results, 0 FP results, and 1 FN results. Following equation (1) from the "Design Trade Studies" section, this results in an accuracy of 98.04 percent.

TABLE III. FACIAL DETECTION OPTION 3 TEST TABLE (PARTIAL)

Centered Facial Landmark Coordinates (X, Y)	Nose or Mouth?	Off-center of Screen?	Screen Alignment Test Result (TP, TN, FP, FN)
(332, 287)	Nose	Yes	TP
(332, 287)	Nose	Yes	TP
(332, 287)	Nose	No	TN
(332, 287)	Nose	No	TN
(298, 266)	Nose	No	TN
(298, 266)	Nose	Yes	TP
(335, 316)	Nose	Yes	TP
(335, 316)	Nose	Yes	TP
(335, 316)	Nose	Yes	FN
(315, 240)	Nose	Yes	TP
(315, 240)	Nose	No	TN

B. *Speech Recognition*

1. *Signal Processing*

The signal processing algorithm is implemented using a Python script. The first step is to record the user speaking a word from a predetermined list of eight possible categories and save the recording as an audio file. This is recorded using the built in sound device module and saved to an output path on our local desktop. Then, the wav module from the scipy library is used to translate the saved audio file to a float array which we can then manipulate and modify. Once we store the audio input, we apply a pre-emphasis filter to emphasize the higher frequency components of the signal in order to improve the signal to noise ratio[7]. This is done by following a simple equation:

$$y(t) = x(t) - \alpha x(t-1) \quad (2)$$

where x(t) represents the original audio input and α is a filter coefficient with a value of 0.97.

Next, we split the data into 20 ms chunks, or frames, with a 10 ms overlap between frames. This is done in order to account for the fact that frequencies vary greatly over time and to ensure that the Fourier transform is taken over stationary frequencies. The original input signal is padded with zeros to ensure that the length of each frame is the same. Afterwards, as described in the "Design Trade Studies" section, we apply a Hamming window to each frame to prevent leakage of the different energy components when the Fourier Transform is taken[7]. This consists of performing a dot product between each frame and the following equation that represents the Hamming window:

$$w[n] = 0.54 - 0.46\cos\left(\frac{2\pi n}{N-1}\right) \quad (3)$$

where N is the window length and in our case was 320 samples.

In our case, we have 198 frames, each of length 320 samples. Following this, the 512-point Fourier transform is applied to each frame using the built in fft function that the scipy library provides, and the result is stored in a 198 by 257 matrix. Because the Fourier transform is a symmetric signal, we only need to store the first half of the signal, hence explaining why the length of each row in the resultant matrix is half of 512. When we visualized this resultant 198 by 257 matrix in a spectrogram, the result is as we expected - similar for similar words and different for different words.

While we could end the signal processing here and pass in the 198 by 512 matrix into the neural network, we decided to further modify the output. In order to visualize the spectrogram with time on the x axis and frequency power on the y axis, we apply the Mel Filter Bank coefficient approach. This is done by applying triangular filters using a Mel scale to extract the different frequency components[24]. We create 40 triangular filters, equally spaced out between the lowest and highest frequencies and perform matrix multiplication between the filters and the resultant 198 by 257 matrix. This leads to a 40 by 257 matrix, which we then visualize on the spectrogram (as shown in Fig. 9.).

The last step involves creating the training data. As mentioned in "Design Trade Studies", we save the spectrogram as an image and use the pixel values of the image to create the training data. This is done because the 40 by 257 matrix described above does not seem to have any apparent differences between different words. While it is possible that the neural network can learn patterns that the human eye cannot, we thought that the neural network would learn better on the images.

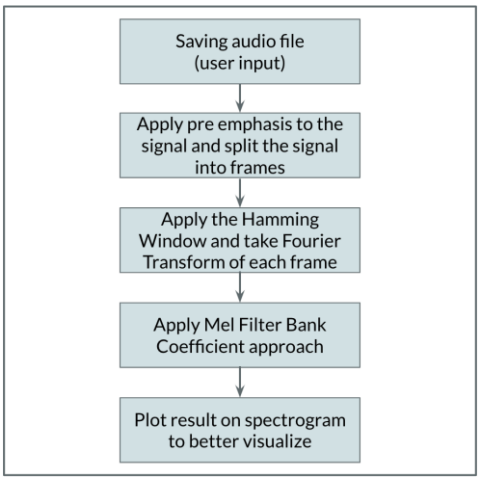

Fig. 8. Block diagram for signal processing.

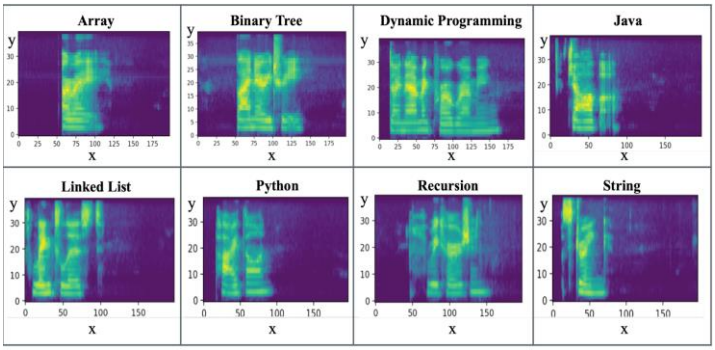

Fig. 9. Spectrogram representations for the eight different categories; x-axis - time (ms), y-axis - frequency (kHz).

2. *Machine Learning*

The machine learning algorithm that we implement is a convolutional neural network, adapted from Mohini's neural network assignment from 10-301: Introduction to Machine Learning. As described in the signal processing section, the input is the image of the saved spectrogram representation of the word spoken by the user. As discussed in "Design Trade Studies", we discussed experimenting with 2 hidden layers and how we found the error was significantly higher. Therefore, the final model used has one hidden layer to perform the transformation of the input vector, and an output layer that represents the probability distribution across the eight possible categories. The hidden layer consists of 10 hidden units. The feature vector in the hidden layer is calculated by performing a weighted linear combination of the input feature vector and the weights in a matrix α. The result of the linear combination is passed through a sigmoid function in order to normalize the vector. Similarly, the probability distribution in the output layer is calculated by performing a weighted linear combination between the feature vector in the hidden layer and the weights in a matrix β. The result of the linear combination is once again passed through a sigmoid function in order to normalize the probability distribution[11].

There are two sets of parameters that the algorithm optimizes. The α matrix has dimension length of input vector by the number of hidden units in the hidden layer. In our case, α has dimensions 1500 by 10. It represents the weights that connect each element in the input vector to each hidden unit in the hidden layer. The β matrix has dimension number of hidden units in the hidden layer by number of classes in the output. In our case, β has dimensions 10 by 8. It represents the weights that connect each element in the hidden layer to each class in the output.

To determine the weight matrices α and β, we use a process called Stochastic Gradient Descent (SGD), which updates the parameter values for each training example. This means the model is able to learn and generalize to each sample in the training data set. The objective of SGD is to minimize the objective function, or the Mean Squared Error (MSE), by using a technique called backpropagation to compute the gradient[21]. This requires the use of the Chain Rule as computing the gradient includes many intermediary variables, and backpropagation provides an efficient way to keep track of the intermediary derivatives in order to prevent recomputing them in a future iteration. Through this information, SGD updates the parameter values in the opposite direction of the gradient.

Training the model involves finding the optimal parameter matrices that minimize the Mean Squared Error given by the following equation:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2 \quad (4)$$

Y_hat, the predicted classification for each sample in the dataset, is found through the feedforward propagation described above with the current weight matrices α and β, which are found through a process called Stochastic Gradient Descent. Y is the true classification for each sample in the dataset. N is the total number of samples. This process updates each of the weights in α and β by choosing one sample from the dataset and adding the current weight to the partial derivative of the mean squared error with respect to each of the weights. This is repeated until the MSE is successfully minimized.

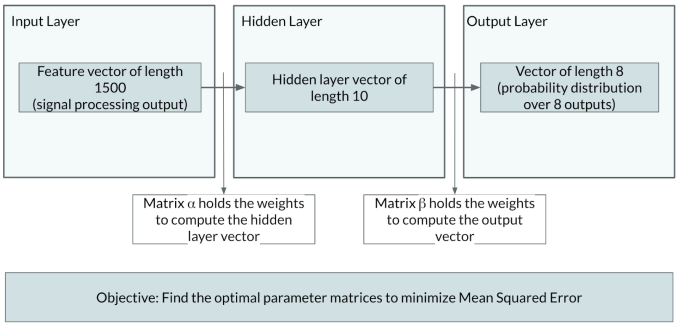

Fig. 10. Block diagram for neural network. Note: a similar diagram was used in Mohini and Shilika's Final Report for 18-794: Pattern Recognition Theory. Both diagrams were created by Mohini and Shilika.

3. *Testing and Results*

To determine the accuracy of the speech recognition algorithm, we performed two different checks. After training the model with 105 training data samples, we tested the algorithm with 50 testing data samples. We performed this test 50 times, and, as seen in the figure below, the results had six outliers. Excluding the outliers, the average error rate for the training and testing error are 0.05 percent and 64.5 percent respectively. The outliers are caused because our algorithm is non-deterministic. Since we are using Stochastic Gradient Descent to update the weight matrices for every training data sample within every epoch, our algorithm has an element of randomness. With every run, the algorithm will create a slightly different model to predict unseen data, resulting in varying training and testing error rates[3]. However, as seen in the figure below, the overwhelming majority of the runs stayed consistent.

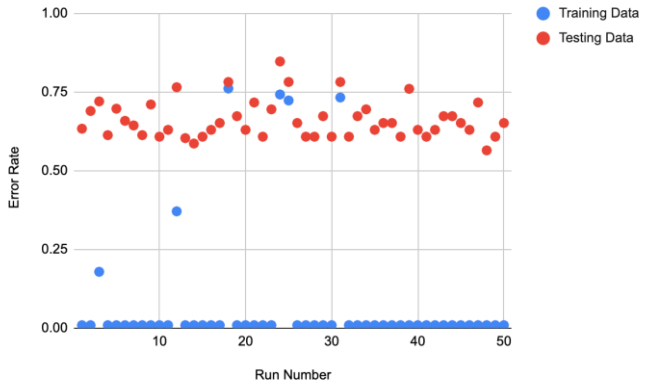

Fig. 11. Training and testing error rates for 50 separate runs.

Our second check involved using the same model, but varying the testing data sample sizes from 50 to 80. We ran each combination multiple times and plotted the most consistent result in the figure below. Through this analysis, we were able to confirm that our testing accuracy was not a fluke based on the data samples we included and remained steady as we added more data.

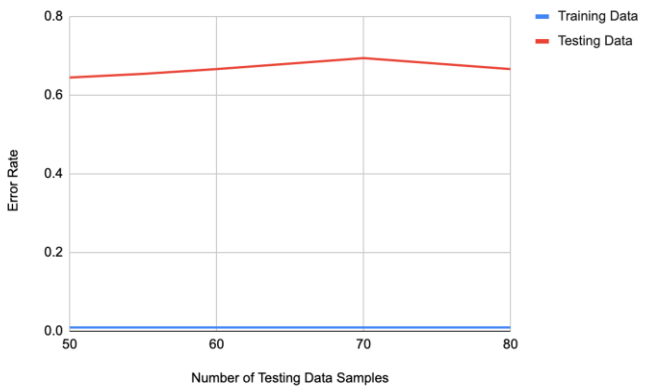

Fig. 12. Training and testing error rate for number of testing samples ranging from 50 to 80.

As discussed in the "Design Trade Studies" section, despite attempting multiple methodologies to increase the accuracy of the speech recognition model, we were unable to. We achieved an accuracy of 35 percent, which is 30 percent below our desired accuracy. In order to cross-check our neural network accuracy with an accuracy given by another classification method, we ran our algorithm using python's in-built Gaussian Naive Bayes function provided in the sklearn.naive_bayes library[23]. This model incorrectly predicted 29 out of 50 points, rendering an error rate of 58 percent. This is seven percent above the neural network error rate of 65 percent. The difference in the accuracy between the neural network approach and Gaussian Naive Bayes approach is not significant. This analysis leads us to believe that the representation of the audio signal could have been further fine-tuned, as neither model was able to classify more than half of the testing data correctly. Modifications include performing different processing techniques on the input signal. Additionally, these results show that a probabilistic approach such as Gaussian Naive Bayes or Gaussian Mixture Modelling may have been better to use, as the accuracy is slightly higher when using Gaussian Naive Bayes rather than a convolutional neural network approach

VI. Project Management

A. Schedule

Our full Gantt chart is in Appendix X. This schedule varies slightly from the original schedule we included in the Design Report. From the beginning, we made our schedule very detailed, so that each part was broken down into several parts. The schedule was adjusted to account for changes in design decisions as discussed in the "Design Trade Studies" section. It includes the tasks that all team members are responsible for (e.g. abstract, proposal presentation), a research phase, and an implementation phase. The three main components of the requirements - facial detection, signal processing, and machine learning, were divided among each team member. All team members worked on designing the web application.

B. Team Member Responsibilities

Mohini - Mohini was in charge of the machine learning algorithm for the technical interview section and ensuring that the accuracy of the neural network matched our expected accuracy of 65 percent. She also worked with Shilika to form the input to the neural network, which was a result of the signal processing portion of our project.

Shilika - Shilika was in charge of the signal processing algorithm for the technical interview section. She worked on ensuring that the output was a reasonable representation of the audio that was spoken by the user, and continuing to make enhancements and improvements to hone the output.

Jessica - Jessica was in charge of the facial detection portion for the behavioral interview section. She worked on researching and implementing the facial detection algorithm, and providing real-time feedback to the user. This included detecting facial features and alerting the user of subpar eye contact and screen alignment.

C. Budget

While iRecruit was originally going to require AWS credits for EC2, we decided to no longer deploy the web application. This was due to the complex nature of the code layout, where both Python and Java were used, as well as the Python OS library to run files from the command line (given the different setups of Mac versus Windows)[20]. This made it difficult to deploy iRecruit with the various file paths and languages, and the requirement of the command line. Since AWS EC2 was not used, iRecruit did not cost us anything to create.

D. Risk Management

There were a couple of risk factors to consider for each portion of iRecruit that we needed to consider. For the facial detection part, the biggest risk factor was the inability to meet our accuracy expectation of 80 percent. To mitigate this, we used the OpenCV library Haar Cascades, which historically have an accuracy rate of about 95 percent[27]. Because the bulk of the initial facial detection was done using Haar Cascades and we only used the distinct nose and mouth coordinates from facial landmark detection, the accuracy continued to stay high, and the losses in accuracy were likely due to the tracking and alert portions. We were able to exceed our accuracy goal of 80 percent for all 3 options. Other risk factors included factors that may have affected performance, such as the contrast between different facial features, lighting, and background. The contrast between the iris and the rest of the eye may vary depending on the user. To account for this, we wanted to allow users to set their threshold manually instead of having a hard set threshold. This should have also taken care of the lighting. However, we were unable to implement the setting threshold feature in time given our schedules and capacities, and this may have been another potential reason for losses in accuracy. For the background, we recommended on the "Behavioral" web page that the user be positioned in front of an empty background, so that the face is apparent on the screen. Having an empty background allows for the system to detect the facial features. From testing, this detection worked much better on an empty background as opposed to a non-empty/noisy background.

There are two main risk factors to consider in the signal processing portion. The first risk factor is not being able to reach the accuracy expectations we have set for ourselves. We

are aiming for a 65 percent accuracy on the overall speech processing algorithm. The output of the signal processing algorithm will have a big impact on the overall accuracy of our speech recognition implementation. We believe that we weren't able to mitigate this risk factor as well as we would have liked, as we hypothesize this is one reason that the accuracy of our speech recognition algorithm fell short. The second risk factor is ensuring that pitch and loudness will not have an effect on our signal processing algorithm. For example, different people speak at different volumes and have different pitches. These factors should not have a major input on the overall output. We believe we were able to mitigate this risk factor to the best of our ability as different people speaking the same word resulted in similar spectrogram representations.

The biggest risk factor for machine learning is that our model may not work at all. As mentioned previously in this report, speech recognition is an extremely demanding task which requires many talented engineers spending many years working on it. Our speech recognition attempt was not a complete failure as our accuracy of 35 percent is more than double of the chance accuracy of 12.5 percent. However, the accuracy was a lot lower than we hoped to achieve. One possible reason is that the testing data samples are significantly different from our training data, and our algorithm did not generalize to unseen data samples well. There are many factors beyond our control, such as the pitch, frequency, and volume of the audio signal, that may contribute to a lower than expected accuracy for the algorithm. Another risk factor relates to not being able to determine the optimal parameters for our model. This includes determining the optimal number of training data samples to build a model that does not overfit to any particular data. We found it extremely surprising that adding more than 105 samples of training data reduced the accuracy of the algorithm. We hypothesize it is because the added training data did not have enough variety from the already existing training data. Other parameters we experimented with are the number of hidden layers, the number of epochs, and size of training and test dataset.

VII. Related Work

There are multiple current resources available for practicing behavioral and technical interviewing. For behavioral interviews, there are several articles that exist that give tips and commonly asked behavioral questions. These articles exist across many platforms, including many job-related sites, such as Indeed and Glassdoor. For technical interviews, there are several platforms that allow users to practice their technical and coding skills. This includes *HackerRank* and *LeetCode*, where users are able to select which topic they want to practice with and are given a question and an IDE to code on.

VIII. Summary

A. Future Work

If we do choose to continue working beyond the semester, we have improvements that we would like to make. For the facial detection portion, as mentioned in the "Risk Management" section, we would like to allow users to set their threshold manually instead of having a hard set threshold to account for contrast differences between facial features and lighting. This may help improve the accuracy and the accessibility of the system, as it would provide for better facial detection and take more into account users of various backgrounds and circumstances. We would also like to improve the accuracy of the first option for the facial detection portion, as this option had the lowest accuracy with the integration of both eye contact and screen alignment. Several of the negative tests were due to incorrect alerts of subpar eye contact when the user actually had subpar screen alignment. To improve this, we would calculate a new "center" for the eyes as the user moves off screen, so that the system does not falsely alert them of bad eye contact. Instead of simply having one initial frame of reference, we could update the frame of reference regularly, so that there are new centered eye coordinates to account for movement. Finally, another improvement would be to test the facial detection system on more users. Due to scheduling and time limits, it was difficult to test the system on multiple users, so Jessica conducted all of the current tests. It would be ideal to have a larger and more representative set of users test the system, including users who are non-team members, to get a better idea of the accuracy and performance.

Regarding the speech recognition model, there are multiple factors that can be improved to higher the accuracy. First, we would reconsider the best techniques used to process the raw data. This would include exploring options to remove background noise or a tool to normalize frequencies of the different energy components. We would also explore various filter bank options and weigh the pros and cons of each use case. Second, we would use a probabilistic model such as Gaussian Mixture Modelling or Gaussian Naive Bayes to classify the data, instead of a convolutional neural network. One advantage of using a probabilistic model is that significantly less training data is needed for classification purposes. Generally, probabilistic models are faster to compute with and can learn with less data. Furthermore, we would create our training data with more variety of pitch, frequency, and volume. This would involve recording the voices of people of all genders, ethnicities, and age groups so that our training data can be representative of the population, as a whole.

B. Lessons Learned

This was an important application to consider as many of us, as graduating seniors, are experiencing the hardships of the job recruiting process. Interviewing is rough and based on our personal experiences, we decided that having a centralized platform would make practicing for an interview less stressful. In terms of the design and implementation process of our project, we suggest having clear goals, deadlines, and visions for the end product before starting the implementation. Documenting the small success points along the way helped the end goal seem more feasible and attainable.

References

[1] Banerjee, Debdeep, and Kevin Yu. "Robotic Arm-Based Face Recognition Software Test Automation." *IEEE Access*, vol. 6, 10 July 2018, pp. 37858–37868., doi:10.1109/ACCESS.2018.2854754.

[2] Brownlee, Jason. “How to Avoid Overfitting in Deep Learning Neural Networks.” *Machine Learning Mastery*, 6 Aug. 2019, machinelearningmastery.com/introduction-to-regularization-to-reduce-overfitting-and-improve-generalization-error/.
[3] Brownlee, Jason. “Why Do I Get Different Results Each Time in Machine Learning?” *Machine Learning Mastery*, 26 Aug. 2020, machinelearningmastery.com/different-results-each-time-in-machine-learning/.
[4] Dey, Sandipan. Hands-On Image Processing with Python: Expert Techniques for Advanced Image Analysis and Effective Interpretation of Image Data. Packt Publishing Ltd., 2018.
[5] “Dlib C++ Library.” *Dlib*, dlib.net/python/index.html.
[6] “Face Detection Using Haar Cascades.” OpenCV, opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_objdetect/py_face_detection/py_face_detection.html.
[7] Fayek, Haytham. *Speech Processing for Machine Learning: Filter Banks, Mel-Frequency Cepstral Coefficients (MFCCs) and What's In-Between*. 21 Apr. 2016, haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html?fbclid=IwAR1jJ1skvbHHXXYoa3aSpxBPCjwgzHACi26xlIAQc3T4nsh2QUoQ2OA5M0I.
[8] Filonov, Stepan. “Tracking Your Eyes with Python.” *Medium*, 22 Mar. 2019, medium.com/@stepanfilonov/tracking-your-eyes-with-python-3952e66194a6.
[9] “Gaussian Mixture Model.” *Brilliant Math & Science Wiki*, brilliant.org/wiki/gaussian-mixture-model/.
[10] Guennouni, Souhail, et al. “A Comparative Study of Multiple Object Detection Using Haar-Like Feature Selection and Local Binary Patterns in Several Platforms.” *Modelling and Simulation in Engineering*, vol. 2015, 31 Dec. 2015, pp. 1–8., doi:10.1155/2015/948960.
[11] “How Neural Networks Work - A Simple Introduction.” *Explain That Stuff*, 17 June 2020, www.explainthatstuff.com/introduction-to-neural-networks.html.
[12] Jack, Srujan. “Face Detection Using Dlib HOG.” *Medium*, 17 July 2020, medium.com/mlcrunch/face-detection-using-dlib-hog-198414837945.
[13] Lab, Sagara Idea. “What Is Django and Why Is It Used?” *Medium*, Medium, 4 Feb. 2020, medium.com/@sagarajkt/what-is-django-and-why-is-it-used-2dafdc75ce67.
[14] Mallick, Satya. “Blob Detection Using OpenCV (Python, C++).” *Learn OpenCV*, 17 Feb. 2015, www.learnopencv.com/blob-detection-using-opencv-python-c/.
[15] Mallick, Satya. “Facemark : Facial Landmark Detection Using OpenCV.” *Learn OpenCV*, 19 Mar. 2018, www.learnopencv.com/facemark-facial-landmark-detection-using-opencv/.
[16] Mallick, Satya. “How to Find Frame Rate or Frames per Second (Fps) in OpenCV (Python / C++) ?” *Learn OpenCV*, 12 Nov. 2015, www.learnopencv.com/how-to-find-frame-rate-or-frames-per-second-fps-in-opencv-python-cpp/.
[17] Oluwatosin, Otulagun Daniel. “Facial Landmarks and Face Detection in Python with OpenCV.” Medium, 24 Jan. 2020, medium.com/analytics-vidhya/facial-landmarks-and-face-detection-in-python-with-opencv-73979391f30e.
[18] “100 Behavioral Interview Questions to Help You Find the Best Candidates.” *Top Echelon*, 7 Dec. 2020, www.topechelon.com/blog/placement-process/top-behavioral-interview-questions-list-examples/.
[19] “Precision and Recall.” *Wikipedia*, Wikimedia Foundation, 10 Nov. 2020, en.wikipedia.org/wiki/Precision_and_recall.
[20] “Python | Os.system() Method.” *GeeksforGeeks*, 20 June 2019, www.geeksforgeeks.org/python-os-system-method/.
[21] Ruder, Sebastian. *An Overview of Gradient Descent Optimization Algorithms*. 20 Mar. 2020, ruder.io/optimizing-gradient-descent/.
[22] Scheidler, Pete. “Understanding the Basics of Fourier Transforms.” *EnDAQ Blog for Data Sensing and Analyzing*, blog.endaq.com/fourier-transform-basics.
[23] Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011. https://scikit-learn.org/stable/about.html#citing-scikit-learn.
[24] S. K. Kopparapu and M. Laxminarayana, "Choice of Mel filter bank in computing MFCC of a resampled speech," *10th International Conference on Information Science, Signal Processing and their Applications (ISSPA 2010)*, Kuala Lumpur, 2010, pp. 121-124, doi: 10.1109/ISSPA.2010.5605491.
[25] “Structural Analysis and Shape Descriptors.” *OpenCV 2.4.13.7 Documentation*, docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html.
[26] *Understanding FFTs and Windowing*. National Instruments, download.ni.com/evaluation/pxi/Understanding%20FFTs%20and%20Windowing.pdf.
[27] Viola, Paul, and Michael Jones. “Rapid Object Detection Using a Boosted Cascade of Simple Features.” *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2001, pp. I-I., doi:10.1109/CVPR.2001.990517.
[28] *The World's Leading Online Programming Learning Platform*. leetcode.com/.

IX. Appendix I. Schedule

Abstract							
	Proposal presentation		Design Review stage				Final stage
		Research facial detection and begin implementation					
		Research signal processing and begin implementation					
		Research machine learning and begin implementation					
		Begin WebApp wireframes					
		Begin WebApp wireframes					
			Eye contact (initial setup, off-center alerts)				
			Signal Processing implementation				
			Design pages for WebApp				
			Initial signal processing implementation				
				Facial landmark detection for screen alignment (nose)			
				Review neural network and assist with implementartion			
				Initial neural network implementation			
				Mel filter banks and fine tuning of signal processing			
				Update WebApp page for behavioral			
					Screen alignment for mouth and behavioral questions		
					Work on unfinished web applcaition components		
					Create training data and integrate signal processing with machine learning		
	Legend				Update WebApp pages (dashboard, technical, behavioral tips)		
		All				Integrate eye contact/screen alignment, profile section	
		Jessica				Work on neural network	
		Shilika				Improve accuracy of neural network	
		Mohini				Integrate speech recognition algorithm into webapp	
							Testing and cleanup
							Testing and cleanup
							Testing and cleanup

X. Appendix II. Enlarged Block Diagrams for Facial Detection and Machine Learning