Real Time Webcam Video to Screen Click Position Mapping with Eye Tracking

Julian Chu
Stanford University
450 Serra Mall, Stanford, CA
juliantc@stanford.edu

Sharon Cheng
Stanford University
450 Serra Mall, Stanford, CA
scsharon@stanford.edu

Richard Cheung
Stanford University
450 Serra Mall, Stanford, CA
rich13@stanford.edu

Abstract

Eye tracking has a myriad of applications including but not limited to cursor input, medical diagnosis, and user analytic. However, expensive hardware is required for accurate eye tracking for traditional methods. In this paper, we propose a novel approach that uses real-time multi-frame webcam data for eye tracking and map it to user mouse click positions on the screen, which differs from the traditional methods of training models with single-frame input images without laptop screenshots, aiming to use eye tracking information as an alternative cursor input. With the purpose of this study, we generated an original dataset using a self-devised data collection pipeline, containing more than 30,000 data points of webcam and screenshot images collected in the wild, which is one of the largest of its kind. Finally, our study presented promising results of mapping real-time eye movement to mouse click using multi-frame webcam data in addition to incorporating laptop screenshots inputs which generated solid saliency maps of mouse click outputs with our eye tracking model.

1. Introduction

Our project centers around the ability to track a users' eye movements using Computer Vision models. Eye tracking can be used for many purposes, like as a replacement for cursor movement, gathering analytical data, or for neuromarketing purposes. It can also be a useful tool for developers to find out where places on the screen are clicked the most, which they can use to optimize the user experience. Current methods for eye-tracking require intensive hardware, and eye-tracking as a cursor replacement methods are often criticized as inducing a lot of eye strain on the user, as users have to look at the specific location that users wish to click. Our goal is to build an inexpensive method that will allow a program to predict where the user wants to click based on their eye movement, in real time. The program should make use of a users' webcam, and not require any expensive high-resolution or infrared cameras. The implications can be huge, and can be very useful in fields like Virtual Reality, medicinal tech, and user analytics.

Current solutions for predicting where the user wants to click use a multi-step architecture and is often very slow. Solutions often first detect the eye, then the pupils, and then match that to certain coordinates on the users' screen. Though they work most of the time, users have reported that the experience is cumbersome and far from perfect. It assumes that users stare directly at the click point, and that means when these models are used as an input device, users report feeling a lot of eye strain, as they have to look and stare at directly what they want to click for it to work. This assumption is also imperfect - where a user clicks might not be where they look at directly. The equipment use is also a big hurdle, as current eye-tracking technology systems often use expensive high-resolution or infrared cameras, which means the system can't be adapted to one's smartphones or laptops, where they could be of much use. Another drawback of current systems is that the information from the camera doesn't consider the contexts of the screen. It simply tracks the iris movement - then maps it to the screen coordinates. It doesn't consider if the screen is blank, or where certain buttons are. This information and context will be especially useful to have, and can improve predictions of where users click.

2. Related Work

2.1. Eye tracking datasets

There have been numerous eye-tracking, or gaze detection datasets. Traditional methods for gaze detection require hardware that is often expensive and large, such as infrared cameras, and as such their datasets are small. Recent developments have attempted to solve this however. Xu et al crowdsourced their dataset through Amazon Mechanical Turk, which utilizes people's webcams to collect data [9]. This results in a much lower cost to collect more data. Other datasets achieved great results from just a few individual workers. For example, the MPIIGaze dataset collected over 200,000 images from just 15 individuals [10].

2.2. Eye tracking technology

There has been much work in gaze tracking over the decades. Initial attempts used infrared cameras to capture facial and pupil positioning like in FreeGaze [5]. Recently, many groups have resorted to using standard webcams or other digital cameras, as they are relatively ubiquitous and inexpensive. Many scholars in this field also first preprocess the raw images - they often go through a process of distinguishing the facial features from the images themselves [11]. In other words, they identify facial landmarks, which could be fed to various networks. For example, A. Gudi et al [1] first normalized the face, and determined the eye crops with a pre-existing model. They then pass this to a CNN model that determines a gaze angle. This allows for dimensionality reduction, and can help normalize the data against noise. There often will be differences in lighting, or facial positioning in the images, so preprocessing is paramount, as the authors explain. One possible pre-processing we can use is the model proposed in Attention Mesh: High-fidelity Face Mesh Prediction in Real-time [3], which offers a quick and efficient way to process an image into a detailed face-mesh, of which we can extract the information of the pupils from.

2.3. Analyzing tapability

Many existing eye-tracking algorithms or models attempt to calculate where on the screen a users' gaze hits. However, they fail to take into account the context of the content on the screen, which is what our method aims to do. There has been research conducted into determining the tapability of elements on screen. Previous attempts have attempted to use a dataset on a corpus of Mobile UI screens, as well as crowdsourced workers to test and train a model that determines which elements will be tapped on the most [7]. Another paper sourced their UI screens from the RICO dataset [2] [8].

2.4. Data collection method

The trend for such existing work surrounding gaze detection, or eye tracking saliency maps, is to crowdsource data. Xu et al crowdsourced their results through an interactive game, which allowed them to naturally collect users' data. They used Amazon's Mechanical Turk as a means of distribution. Other scholars have also used a webcam-based approach [4], where they ran their Logitech webcams at 25 frames per second to capture almost half a million frames with just 38 subjects. This approach could prove to be an efficient and inexpensive method for our research, as we can use the webcams located on top of our laptops for such an experiment. Research has also been done to make such predictions as real time as possible, such as the introduction of SearchGazer [6], which operates as just a lightweight javascript library which can be plugged into any browser.

What we see from looking at the existing relevant works is that although there has been work done into how gaze angles onto a screen are calculated, and insights into the tap-ability of UI elements, no approach marries the two together. As such, our holistic approach will include calculating the users' gaze angle, taking into account the current UI elements on the screen.

3. Dataset and Features

3.1. Dataset Collection

There are two components to the datasets used in this study: 1) webcam images of subjects' faces and 2) screenshots of the subjects' laptop screens. We implemented our own data collection pipeline by writing a Python data collection script that utilizes Apple MacBook's screenshot and webcam functions to collect both sets of data. The data collection process is done in a natural setting in which we run the Python data collection script in the background while all subjects proceed on doing their daily activities on their laptops. Since we are trying to apply computer vision in a novel method, there are no existing datasets that we find relevant to our purpose of this study which is aiming to achieve real-time mapping of user eye gaze and mouse click, thus, all datasets used in this study are collected by and from the three authors of this study. We collected data from the 3 authors and their 5 close friends. We collected our data over a 3-week period to ensure that we have an ample sample size.

3.2. Dataset Analysis and Processing

Our final dataset consists of $\approx$ 31,000 samples, and we used 70% for training, 20% for validation, and 10% for testing. There are no datasets that serve similar purposes as ours in the context of this study, so it is relatively diffcult to compare the size of our dataset to those of other studies since our application apporach is novel. However, we would still like to mention that the dataset used in Google's real-time pupil tracking research study that contributed to its MediaPipe Face Mesh + Iris model has a size of $\approx$ 20,000 as a reference. As shown in **Figure 1**, each set of our samples contain 10 1280 x 720 images taken by the 13-inch MacBook Pro webcam (10 frames per second), as well as a RGB screenshot of the 1440 x 900 laptop screen captured at the instant when user mouse click is triggered and a label with the coordinates of the click position on the screen. Additional information like timestamp and whether it is a left click or a right click is also recorded, although we currently do not have use for them.

3.2.1 Webcam Image Processing

We utilized the aforementioned Google MediaPipe Face Mesh + Iris model to pre-process the webcam images to ex-

Figure 1. This is a complete set of one of our dataset samples. In one set of sample data, there are 10 webcam images captured within 1 second before user mouse click and 1 laptop screenshot at the instant of the user mouse click.

tract 3D landmark features that we hope to focus on in our training which are the landmark features that correspond to user eye movement. **Figure 2** shows a visualization of how this model works on our webcam images. To elaborate on this model, the Google MediaPipe Face Mesh model is an end-to-end neural network-based model for inferring an approximate 3D mesh representation of a human face from single camera input that utilizes attention layers, and the Iris model extends the Face Mesh model with two new components: a tiny neural network that predicts positions of the pupils in 2D, and a displacement-based estimation of the pupil blend shape coefficients. This model normalizes the facial datapoints with respect to image height and width.

3.2.2 Laptop Screenshot Processing

We normalized the original 1440 x 900 laptop screenshot size to let the new x-axis of the screenshots have a range of [-1.4, 1.4] and the new y-axis have a range of [-0.9, 0.9]. Then, we downsized the original 1440 x 900 pixel values to 80 x 50. We also normalized the screenshot pixel value range from the original RGB value range of [0, 255] to [0, 1].

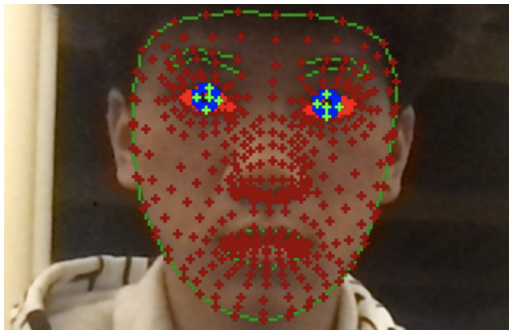

Figure 2. This is a visualization of how Google's MediaPipe Face Mesh + Iris 3D landmark model works on webcam images.

4. Methods

4.1. Baseline Method

4.1.1 Single-Frame Webcam Image with FC Net

Referencing previous works, we decided to construct our baseline method using a multi-step method. Shown in **Figure 3**, we will first input single-frame webcam images of the subjects' faces, then use the pretrained Google MediaPipe Face Mesh + Iris model to process input data and get the 3D landmarks for the subjects' eye region as well as well as different facial landmarks as specified in our data processing section. This could help the model learn things like head pose and orientation that are useful for predicting click position. In total, 478 feature points are extracted from each image.

We then flatten the 478 3D landmarks coordinates to get a one dimensional vector of 1434 feature points, which are than fed into two fully connected layers to generate a final output of a one dimensional vector of 2 elements, which corresponds to the normalized x and y coordinates respectively.

4.2. Proposed Methods

4.2.1 Multi-Frame Webcam Image with FC Net

Shown in **Figure 4**, the architecture of The multi frame webcam image model is similar to that of the baseline model, but uses the previous 10 frames of webcam video as input to the model, as opposed to only using the final frame. This model is created to test the hypothesis that having multiple frames is useful in predicting mouse click coordinates.

We have tested different hyper parameters like layer size and number of layers on this model. We also trained a model that has the exact same layer size and model architecture as the baseline, changing only the input layer (from 1434 inputs to 1434 x 10 inputs, flattened to 14340) This is so that we can have a fair comparison between having multiple frame and having only the last frame, keeping model

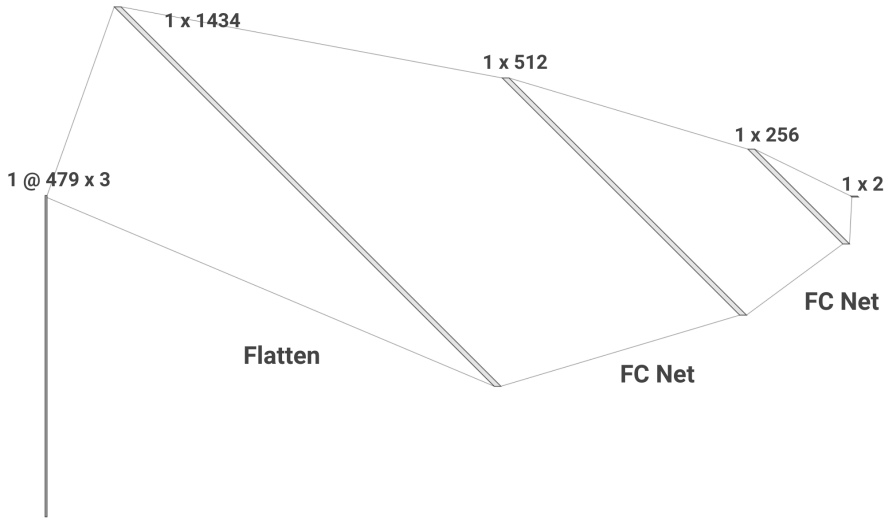

Figure 3. This is the model architecture for our baseline model for single-frame webcam image with FC net.

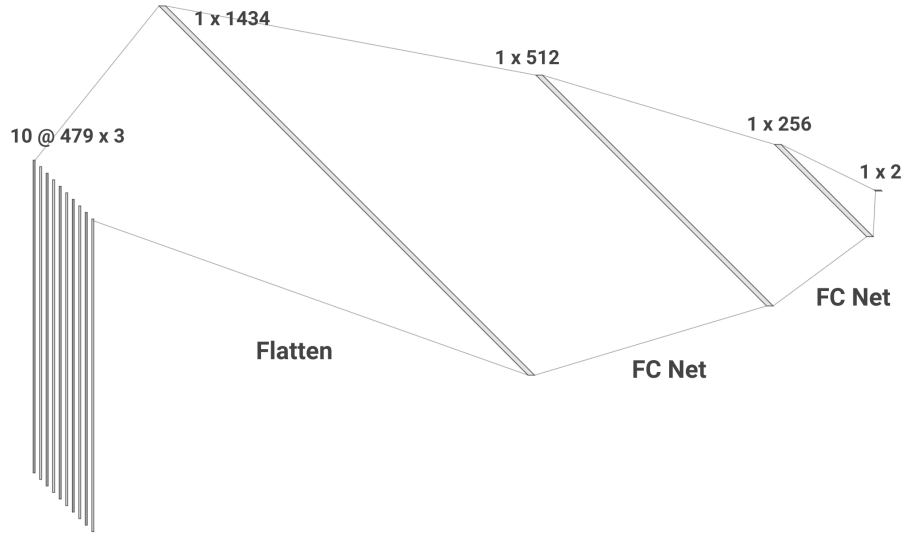

Figure 4. This is the model architecture for our proposed model 1 for multi-frame webcam image with FC net.

size and architecture constant.

We hypothesize that the model will not only look at the frame right before the mouse click, but also the other frames, as users could look at the object meant to be clicked at before they actually click on it. We predict that the model will perform better than the baseline.

4.2.2 Multi-Frame Webcam Image with FC Net + Laptop Screenshot with CNN

Shown in **Figure 5**, this model uses both the multi frame webcam facial landmarks as well as the screenshot of the computer screen at the moment the user trigger the click. The multi frame webcam video is processed similar to the Multi-Frame Webcam Image with FC Net mentioned above. The screenshot is downsampled to 80x50 pixels with RGB values, then fed into a simple convolutional neural network. The output of the the convolutional network is that flattened

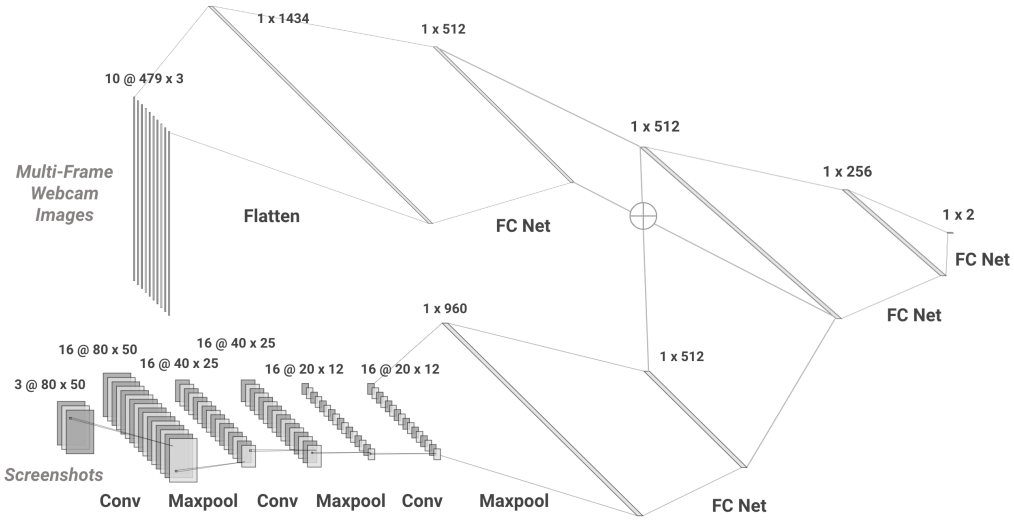

Figure 5. This is the model architecture for our proposed model 2 for multi-frame webcam image with FC net + laptop screenshot with CNN.

to a one dimension vector of 960 elements, which then go through a feed forward layer to create a vector embedding, which is added to the first hidden layer of the FC net of the multi frame webcam video. We hypothesize that this will improve the network performance as this gives the network knowledge about the context of which the user is looking at. By knowing what the user is looking at, as well as finding areas that has a higher chance of being cliecked, for example, button like objects vs background images, the model can achieve a higher accuracy on users' click position.

4.2.3 Determining the importance of each frame with average total saliency

To explore whether having multiple frames actually help the model prediction user's click location, other than comparing the performance of models using and not using multiple frames as input, we will calculated the total gradient with respect to each frame, averaged across the entire test dataset. The average total saliency of facial landmark frame i is given by

$$\frac{1}{N}\sum_{k}^{N}\sum_{j}^{m}(\frac{\partial x_{l_{ij}}^{k}}{\partial y_1}+\frac{\partial x_{l_{ij}}^{k}}{\partial y_2})$$

where $x_{l_{ij}}^{k}$ is the j^{th} feature of the flattened facial landmarks of the i^{th} frame of the webcam video, m is the number of features in the flattened facial landmarks vector, y_1 and y_2 are the output of the model predicting the click coordinates, and N is the test dataset size.

4.2.4 Exploring what the model looks for in screenshot with saliency maps

To explore how the model process the screenshot and whether the model actually look for areas that has higher likelihood of click, we will compute the saliency map of the screenshot, with respect to the output of the model. The saliency of screenshot pixel $x_{s_{ij}}$ is given by

$$\frac{\partial x_{s_{ij}}}{\partial y_1}+\frac{\partial x_{s_{ij}}}{\partial y_2}$$

5. Experimental Results and Discussion

5.1. Results

5.1.1 Training and Validation

We have trained a total of 6 models as shown in **Figure 6**. The baseline model has two hidden layers, the first one with 512 neurons and the second one with 256 neurons, both with ReLU as activation, before feeding it to the output layer to output the normalized x and y coordinates.

We have trained 3 different feed forward models that has all 10 frames of webcam image as the input. The first one has the same architecture as the baseline model (2 hidden layer with 512 and 256 neurons respectively). We also experimented with models that are larger and smaller. The larger model has two hidden layers with 4096 and 256 neurons respectively, and smaller model has only one hidden layer with 128 neurons. All hidden layer has relu as activation

We trained 2 different models for the screenshot + webcam architecture. Both of them uses a simple convolutional

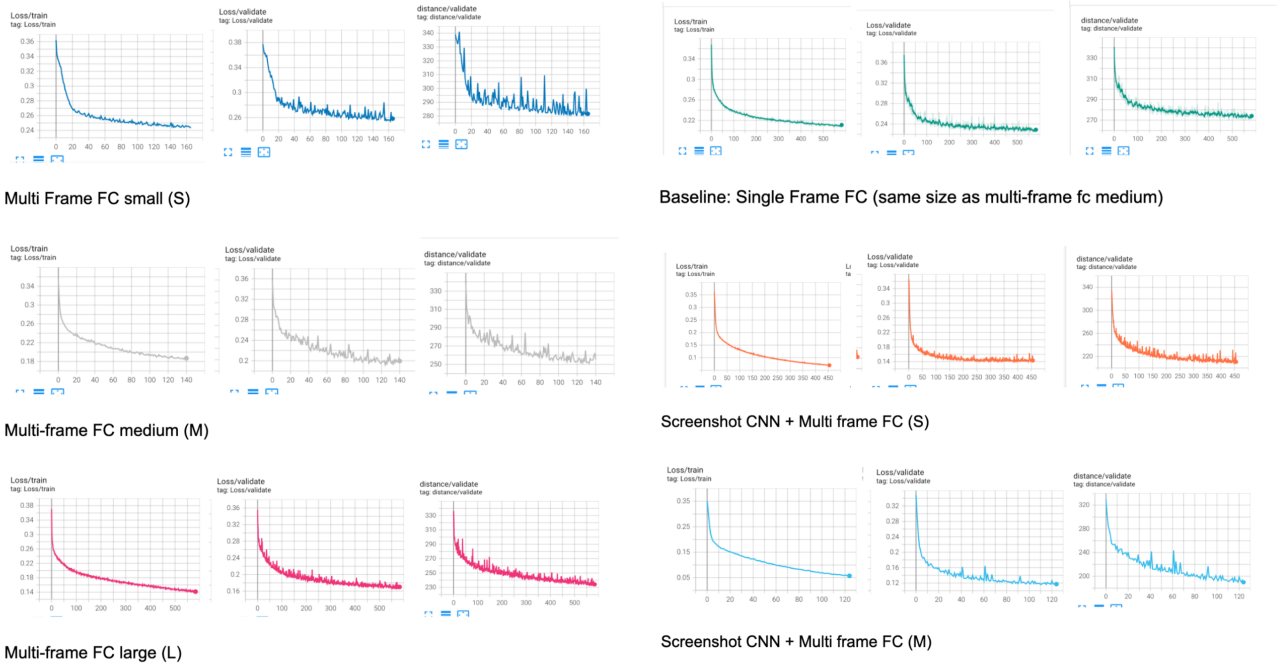

Figure 6. Training logs of all models

neural network which consist of 3 repeated block consisting of the following architecture: 2D convolutional layer with 16 3x3 filters, relu activation, and a 2D maxpooling with 2x2 filters and stride 2. The output of the the convolutional network is that flattened to a one dimension vector of 960 elements, which then go through a feed forward layer to create a vector embedding, which is added to the first hidden layer of the FC net of the multi frame webcam video model, which has the same architecture as the medium size and smaller size model of the 10 frames of webcam image FC net.

Model	MSE Loss	avg. Dist (cm)
Baseline	0.24	2.29
Multi FC (S)	0.26	2.3
Multi FC (M)	0.20	1.8
Multi FC (L)	0.16	1.95
With screenshot (S)	0.14	1.74
With screenshot (M)	0.11	1.5

The models are trained with square mean error. Other than the loss, we also has the average euclidean distance of predicted click from true click as a metric.

5.2. Discussion

5.2.1 The Addition of a Multi-Frame Approach

As aforementioned in the Training and Validation section, using a multi-frame approach proves to yield more accurate

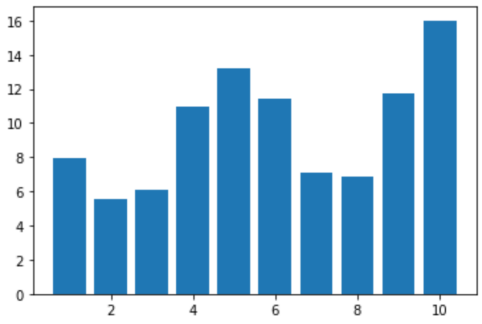

Figure 7. This is a graph of the average total saliency per frame for the total 10 frames per dataset.

results as the average distances are smaller. After calculating the average total saliency for each of the 10 frames for all datasets, we can see that in **Figure 7**, the highest saliency score belongs to the frames towards the end and the middle of during the 1-second duration before user mouse click. This means where the user would be looking at on the screen is the most important towards the end and the middle of the 1 second period before the user clicks the mouse. This can potentially be linked to user behavior as we can expand our finding to not only use our eye tracking model as an alternative cursor but also to help with user analytic.

Moreover, the average total saliency per frame graph also supports our findings of how a multi-frame approach is bet-

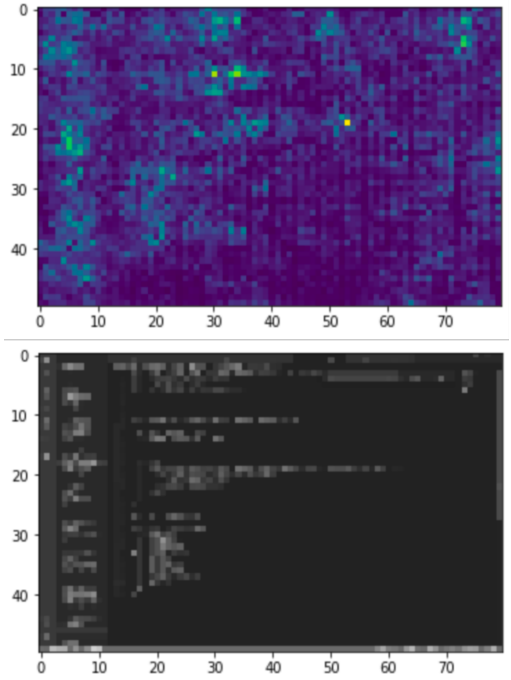

Figure 8. This is a comparison of one of the laptop screenshots for a coding interface and its corresponding saliency map of mouse click.

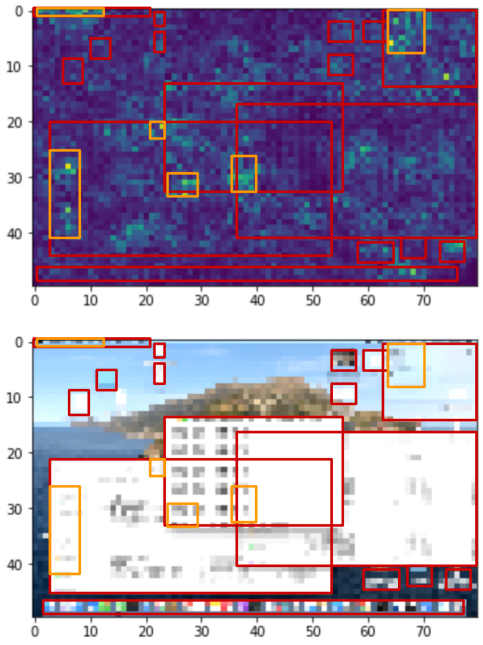

Figure 9. This is a comparison of one of the laptop screenshots for a desktop interface with several opened windows and its corresponding saliency map of mouse click. The red boxes are specific features on the desktop including desktop files, menu and application bar, opened windows, notifications. The orange boxes highlights the areas with high saliency and maps them to areas that tend to receive more mouse clicks.

ter than a traditional single-frame approach when it comes to mapping eye tracking with mouse click activity. **Figure 7** shows that the saliency scores for each frame don't vary significantly. This means that by using a single-frame approach which only considers the 10-th frame (the webcam image captured at the instance of mouse click), we are losing much information from the 1-th to 9-th frame. Therefore, only with a multi-frame approach can we consider the cumulative eye tracking information to provide a more accurate model to generate better mouse click coordinates.

5.2.2 The Addition of Laptop Screenshots

The addition of laptop screenshot to train our eye tracking model that maps to mouse click, like the addition of a multi-frame approach, was proved to improve the accuracy of the model. Even though in our model, we reduced the screenshot resolution by a significant amount and added max-pooling layers due to our nature of the model, the addition of screenshots provided more information for the model to learn the mouse click locations. Our model is not trying to generate mouse clicks at the exact location of the user eye gaze, but rather for the model to learn where the user would click to make the generated click valid. For instance, we do not require the model to generate a click at an exact location on a button that the user clicked, but rather to generate clicks at any location on the button to make that button click valid.

We generated saliency maps for mouse click and compared them to the laptop screenshots at that corresponding instance. As shown in **Figure 8**, the high saliency areas mostly lie on the left of the screen, which if we map it to the actual laptop screenshot, it corresponds to the menu bar of the Visual Studio Code interface. The other high saliency areas lie around the code (text) area, which is also valid since that will be the areas that the users would tend to click on more when editing the code.

Figure 9 shows a more complex interface. As shown in this figure, there are several opened windows on the desktop interface which also contains other features like files, menu and application bars, notifications, and more. The red boxes are the outlines of the specific features, and when mapped to the saliency map of mouse click, we can see that the high saliency areas mostly lie within the red boxes, or in real-life, the certain features that users would tend to click on. More specifically, we also highlighted the highest saliency areas, which is shown within the orange boxes. When mapped to the actual laptop screenshot, we found out that these areas are valid as well since they either correspond to the menu area that users would most likely click on (ex: apple icon,

file, edit, as supposed to window or help), the menu bar on the left side of the finder window, or at boundary areas of windows.

6. Conclusion / Future Work

We presented a solution where not only is the users' gaze determined by the webcam data, but also the content on the screen. The content is taken via a series of screenshots, while the raw webcam images are feed through an existing model to generate a face mesh. These two different data points are then fed through their own CNN networks first before being concatenated and fed through another network. Our approach uses a python script, which runs in the background of individual's Macbooks to automatically collect these screenshots and face mesh data whenever the user clicks on the screen. Though the preliminary results are promising, there are various improvements we can make in the future.

Our current approach uses the pixel values on the screen to determine where the user will click. However, if we are able to get more detailed information about what's on the display, for example, the HTML content of what the user has clicked, and feed that data into the model, perhaps predictions might be more accurate.

One future work needed is to improve the screenshot processing modules. Currently, the screenshot is down sampled to an image of 80x50, with 3 additional layers of maxpooling applied to it, which loses a significant amount of information. Looking at the sample screenshots shows that important features like words and buttons are not really clearly visible at all. Future work should be done exploring ways of increasing the screenshot resolution without significantly increasing model size, such as first cropping out only the region that the user is likely looking at based on only webcam eye tracking input, then using the cropped screenshot to fine tune the output. Ways of processing the screenshot without losing information about the location of things in the screenshot can also be explored. For instance, we should explore not using maxpooling.

Another future work needed is exploring ways to combine the screenshot and webcam information. For instance, attention mechanism can be explored, treating the web cam eye tracking data as the query and the screenshot as the context.

7. Contributions & Acknowledgement

We thank Dr. Feifei Li, Jiajun Wu, Ruohan Guo, and Mihir Patel for your generous help and guidance. We would also like to thank Lavender Chen, Solomon Kim, Annie Ma, Yoonju Kim, and Yoonjoo Hwang for helping with webcam and screenshot data collection. Our implementation utilizes Google MediaPipe's Face Mesh + Iris model.

References

[1] A. G. et al. Efficiency in real-time webcam gaze tracking, 2020. 2

[2] D. et al. Rico: A mobile app dataset for building data-driven design applications, 2017. 2

[3] G. et al. Attention mesh: High-fidelity face mesh prediction in real-time, 2020. 2

[4] M. et al. Webcam-based eye movement analysis using cnn, 2017. 2

[5] O. et al. Freegaze: a gaze tracking system for everyday gaze interaction, 2002. 2

[6] P. et al. Searchgazer: Webcam eye tracking for remote studies of web search, 2017. 2

[7] S. et al. Modeling mobile interface tappability using crowdsourcing and deep learning, 2019. 2

[8] S. et al. Predicting and explaining mobile ui tappability with vision modeling and saliency analysis, 2022. 2

[9] X. et al. Turkergaze: Crowdsourcing saliency with webcam based eye tracking, 2015. 1

[10] Z. et al. Appearance-based gaze estimation in the wild, 2015. 1

[11] Z. et al. A deep learning-based approach to video-based eye tracking for human psychophysics, 2021. 2