Smart Spectacles for Image Recognition (SSIR) for Image Recognition of Art

Northeastern University
Capstone Proposal

Team Members:
Nicla De Biasi
Angela Shen
Warren Waliggo
Denada Bakiasi
Vianna Phung

Advisor:
Dr. Kaushik Chowdhury

Abstract

This capstone design project Smart Spectacles for Image Recognition (SSIR) aims to seamlessly combine art, augmented reality, and machine learning to create a revolutionary smart glasses attachment. We aim to allow our users to explore the culture around them with ease. Our project can be broken down into two sections, hardware and software. Part III of this paper describes the overview for both hardware and software. The hardware aspect of our project aims to create the clearest display possible and the most comfortable user experience we can. This process involved creating a Bill of Materials for the attachment prototype. Part IV of this paper explains the specific hardware components and software algorithms we will use to implement the SSIR prototype. The hardware prototype will contain the main communication hub that can communicate with external components like a camera or a push-button and our backend server. The software component comprises both Online and Offline image recognition software. Offline will contain a database of images along with image descriptors which we will pull the art information from. The online portion contains both the front end and backend. The front end controls the interactions between the hardware and the user. Whereas the backend will communicate with the backend server to find the image and then send this information to the Front end to be displayed. SSIR will provide our users with an immersive art experience with exciting and innovative smart glass technology.

Table of Contents

I. Introduction

With the increasingly fast paced lifestyle, people often have the affinity toward faster and more efficient ways to recognize and identify our surroundings. The most popular among all existing recognition devices may be applications such as Shazam [1], an audio-identifying software that recognizes the origins of audio clips, and Google Images [2], a function offered by Google that identifies user-uploaded photos. All of the aforementioned utilities function by traversing the input through certain databases. While similar object recognition developments are on a rise in numbers in recent years, the accurate recognition of artworks, specifically, remains a vacancy. Some of the possible causes may be the lack of artwork organized databases. While image-recognition platforms such as Google Images guarantee results for any images of artworks in question, its resulting accuracy is irresolute due to the expansive size of its database, often returning irrelevant information to the users' inputs. Applications such as Smartify [3], a platform created for the sole purpose of museum artwork recognition, are readily available; however, its database limits to the app's collaborative museums. Moreover, Smartify's commercial offers are on a subscription basis, with a basic plan of 1800 Pounds (approximately $2193.12 in June 2022) per year. In addition to the expensiveness and scarce accessibility to accurate artwork recognition, current recognition tools often always involve the use of devices such as smartphones and tablets as its primary carrier. In the scenarios of artwork museum visits, such involvements of phones and tablets can be distracting to both the users looking for artwork recognitions and the patrons beside them. Furthermore, waking the devices (entering passwords, if necessary), opening the recognition application, taking an image of the artwork, and waiting for the results is an extensive process, and may not deliver the efficiency people usually expect in terms of quick artwork identifications.

II. Project Formulation

We propose the Smart Spectacles for Image Recognition (SSIR), which will be an additional feature to clip onto existing glasses and will allow users to get a quick image recognition of pieces of art. A key focus for SSIR was to ensure ease of use and allow the user to easily utilize it as well as take it off. The clip-on feature will allow users to have full autonomy of when SSIR is being utilized. When SSIR is clipped and turned on, it will connect to wifi and the backend database. Following the user's input to the scan button, the camera will capture an image and send it to the backend where the image will be traversed through, its features will be extracted and compared with existing images in the database. Homography will then be used in order to return the title of the closest result of the image. If a close match is found, the user will be notified and the title of the art piece will be displayed. From there, the user can choose to learn more about the piece and if they reply with the choice "yes", the title of the art piece will be sent to the backend and searched in order to find the appropriate Wikipedia page. When a plausible page is found, the link will be sent back to the external device and the first few sentences/ the summary of the art piece will be extracted. The summary will be sent to the prototype as well as the external device's default browser and the prototype will display the received strings in groups of words on SSIR. Once this process ends, the user will once again be asked for their input and if they reply with the choice "yes" again, the process will continue. If the user replies with the choice "no", the process will be halted and closed.

Figure 1. Concept Illustration of SSIR Add-on Prototype

III. Project Overview

A. Hardware Overview

The hardware on the SSIR serves the role of taking in user inputs as well as relaying and displaying the results acquired from the backend to the user. The hardware components consist of an OLED display, circuit boards, a camera, a camera lens, a USB Hub, a USB microphone and speaker, as well as a component that enables deep neural network testing. The circuit boards will be utilized to support and connect the different hardware components necessary for SSIR and will be customized as the prototyping phase continues. The hardware components that will be connected to the circuit boards will be a USB microphone and speaker, a USB Hub, the camera and its lens, and an OLED display. The USB microphone and speaker will be utilized in order to take in the user's input of "yes" or "no" and will either continue the process or halt it. The camera and its lens will be utilized as a scanner and will scan the art piece based on the user's input "yes". The OLED display will serve as a way to relay and display the results found in the scanned art piece.

The most significant hardware component will be the circuit board as it is essential in connecting our different components and without successful communication between the different components, SSIR would not be able to operate in the desired way. Successful communication between the different components will allow for the process to proceed smoothly and therefore, it is vital to ensure that the circuit board is customized to the best of our ability. The specific board that we are proposing to utilize is the Raspberry Pi Compute Module 4 I/O Board as seen in the figure below. This board features an eternal power connector, various jumpers with the ability to disable features such as wireless connectivity, as well as ports for a display and camera which are vital features that are needed for SSIR to work properly. Utilizing the Raspberry PI Computer Module 4 I/O board will also allow the group access to similar demo projects and template code that will aid in the initial prototyping phase of SSIR.

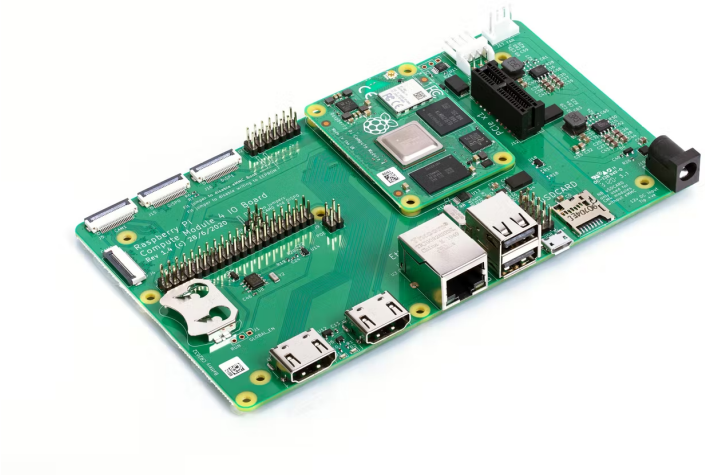


Figure 2. Ideal Raspberry Pi Model for SSIR

In addition to the board, the specific camera and camera lens that we are proposing to use are the Raspberry Pi HQ Camera and Lens as seen in the figure below. The camera and lens are

manufactured by Raspberry Pi and are compatible with our proposed board, which will ease the process of connecting the external components as there are existing guides on how to do so. The Raspberry Pi HQ Camera and Lens allow for the capturing of clear and sharp images which are vital in the process of identifying and relaying information on the scanned object. Additionally, the camera and lens have the ability to manually adjust light levels and focus of the object which will increase the chance of SSIR's success in displaying the wanted information.

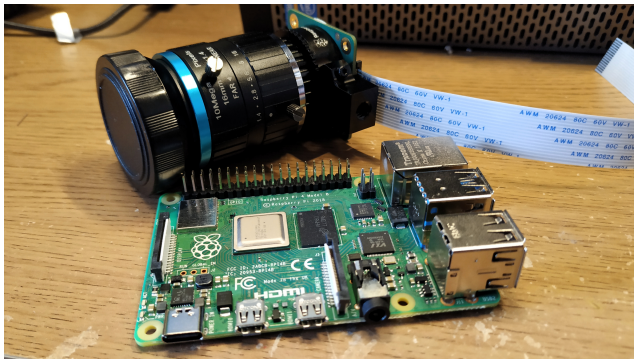


Figure 3. Raspberry Pi Model with Camera

B. Software Overview

The software end of SSIR will handle the communications between the smart-glasses add-on prototype model, the backend server, and potential, optional external devices such as smartphones and earphones.

The backend server will be the foundation to the software functionalities of SSIR. It will be responsible for receiving the image taken by the prototype and traversing it through the open or available databases such as the National Gallery of Art [4]. If prompted, we may also need to create a database of our own, yet many more difficulties such as data regulation will then arise, thus the possibility of this implementation is still doubtful.

The backend server will be connected to the prototype model via internet, which is made possible by the built-in Wi-Fi connection feature and thus can be configured using Raspberry Pi's imager applications. Once connected, the prototype model will be able to begin communications with the backend applications, creating an IoT network.

Analyzing the strengths of our groups, we will most likely be implementing the backend with the C++ language. However, considering that the Raspberry Pi models support both C++ and Python, we may also write certain source code in Python for its advantaged choice of external libraries. If we decide to do so, we will either need to ensure that all members of our team have some basic knowledge of the Python language or devise a way to connect the code implemented with different languages. The detailed machine learning and recognition algorithms to be utilized by the backend is explained further in this paper.

IV. Design Strategies and Analysis

A. Hardware Design Strategies and Analysis - Nicla

The rise in popularity of Augmented Reality (AR) in recent years led to great strides in this sector of technology. Beginning with mostly head-mounted displays (HMD) these strides lead to a variety of AR display types. When beginning the mechanical frame design, our first concern was determining the simplest optical path in order to maximize the light intensity of whatever display we choose to use. Referring to the system diagram below, we're using a transparent graphical OLED breakout to display the user input ques and information found using our image recognition software. The user will be asked the question "yes or no" by our software in order to prompt the re-scan feature or give more information. At this stage, we brainstormed how much information would actually display on the screen. This feature will be finalized once the prototype is built because of possible changes in the optical see-through display depending on the accuracy of the design below.

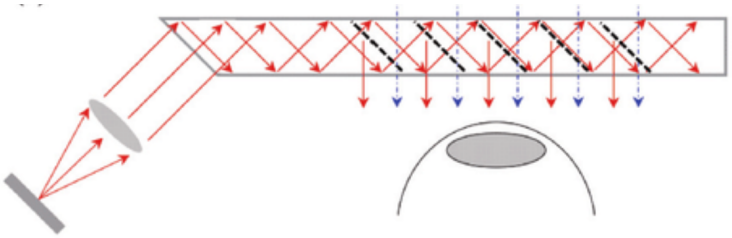


Figure 4. Proposed Optical Path

Our proposed optical path will utilize a biconvex lens to refract the image of the OLED on a rectangular pane of semi-transparent acrylic. The transparency of the acrylic will be 40% aiming

to allow the user to have a clear view of the display without disrupting their vision. Further optics research will be required in the fall and spring to ensure a clear display.

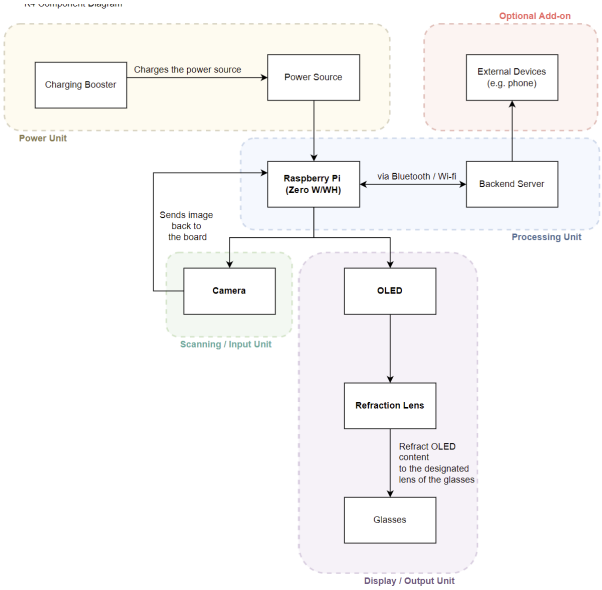


Figure 5. Simplified System Diagram

Next, we break down this system into its separate units and discuss how they will individually function. The output unit design is not yet finalized.

B. Component Design

When designing the hardware and software components of the project, two main hardware challenges immediately came into play. The worldwide shortage of semiconductors could potentially limit our access to certain circuit boards, or at least jack the price up. The second was finding a small circuit board that has all the necessary components and computational power.

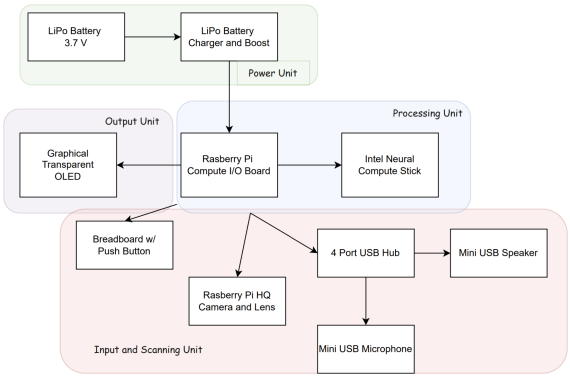


Figure 6. Electrical Component Diagram

Processing Unit

An advantage of using Raspberry Pi technology is it largely eliminates the challenge of finding external compatible components because of the wide variety of add-ons the company provides. The Raspberry Pi Zero W/WH both fit our requirements for Bluetooth and WiFi connectivity and fits our size constraints. The Raspberry Pi Zero W is 60 x 30 mm but lacks a sufficient amount of power pins to connect the necessary components to successfully complete our project [14]. In order to conduct machine learning algorithms necessary for image recognition a CPU with much more computational power. For this reason, we selected the Raspberry Pi Compute Module 4 I/O Board which does add an additional size constraint given that it's 90x160 mm [15]. This model contains two USB-2 ports which we need to separate the communication channels of the backend server vs the hardware communication. This second USB port is where the Intel Neural 2 Compute Stick will plug into the communication board. The CPU on the Intel Neural Compute Stick will run our image recognition software. Making the Raspberry Pi our main communication hub.

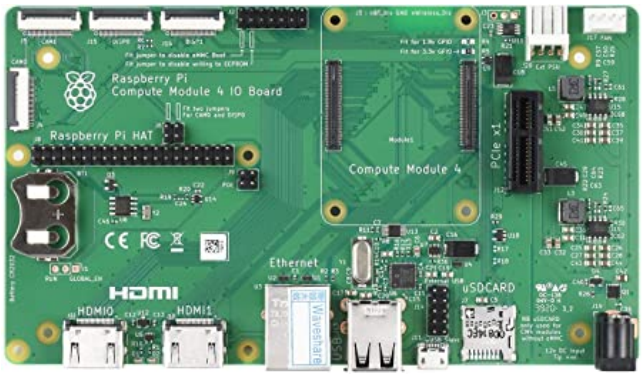


Figure 7. Raspberry Pi Compute Module 4 IO Board

Power Unit

When designing this section of the prototype, our main concerns were making the system rechargeable and the obvious that it'll power on. A Raspberry Pi needs 5V to power on and the typical LiPo battery outputs between 3.7 - 4.2V. We opted to use a battery charger that can both recharge our LiPo battery and simultaneously act as a boost circuit. The battery will plug Vin and GND pins in the Micro-LiPo Charger then that 5V output will be connected to the external power connector pins in the Raspberry Pi and power the circuit. This charger can be plugged into the wall via USB to recharge the battery at the user's convenience.

Scanning and Input Unit

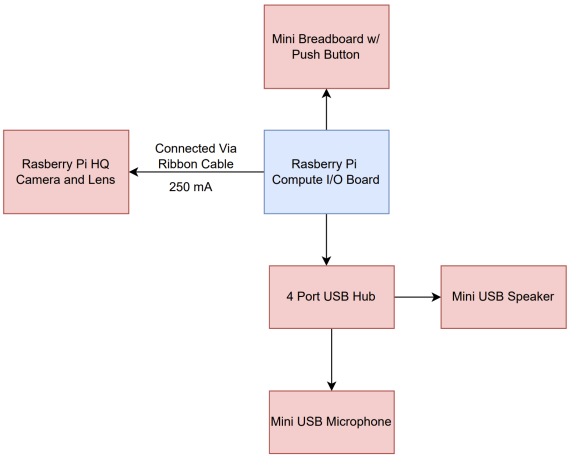


Figure 8. Scanning and Input Unit Schematic

This unit of the prototype consists of a Raspberry Pi HQ camera, Raspberry Pi HQ camera lens, Raspberry Pi 4 port USB Hub, and a push-button. The push-button will be connected to a small breadboard connected via jumper cables to a GPIO pinout in the Raspberry Pi.. A simple python script will be implemented to allow this button to trigger the camera to scan, connect and send the image back to the Raspberry Pi. The Pi will then send the image to our backend server for processing. The camera and camera lens will connect to the main circuit board using a ribbon cable. The camera requires 250 mA transmitted through a ribbon cable. We will be using Raspberry Pi's 4-port USB Hub to connect the speaker and microphone to our communication hub.

B. Software Design Strategies and Analysis

AI and Machine Learning are now a part of our society. It is fascinating to see how many computations that would take years for an individual to make, only take seconds when using AI and ML. One example of that would be identifying a painting, their artist and getting context around it. In this section the software design strategy to make such an application possible via wearable tech will be discussed and analyzed. As shown in the image below, there are two main parts: offline and online.

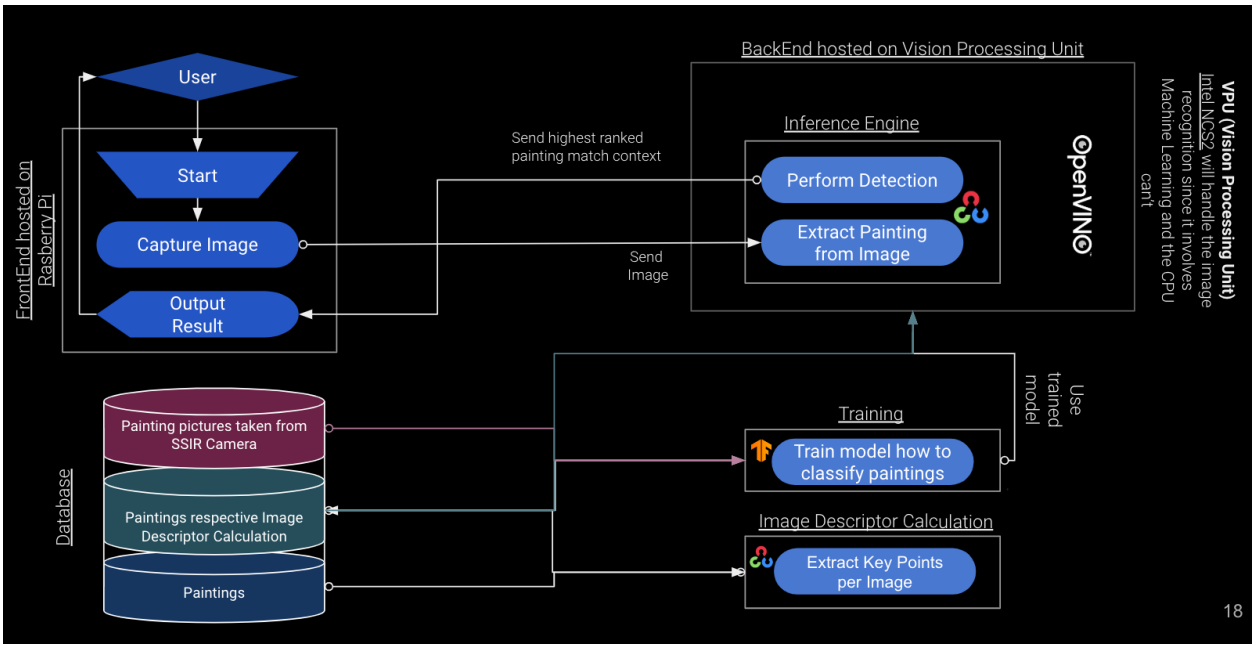


Figure 9. Software Design Architecture

A. Offline

Offline is what will be done before deploying the software program to Raspberry Pi. Offline we will create a database with images of popular paintings and their respective context. The context will include the name of the artist, the piece's title, and a description on what the artist was trying

to present via their art. For each image in that database the visual descriptor will be computed. A visual descriptor in computer vision is kind of like memory for example after an individual looks at a painting and tries to recall it minutes later. Visual descriptors describe elementary characteristics such as the shape, the color, the texture or the motion, etc.

To perform the feature extraction OpenCV will be used. OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was created to provide a common platform for computer vision applications and to speed up the integration of machine perception in daily life.

There are three major algorithms in OpenCV that extract key features from images. The first is SIFT (Scale-Invariant Feature Transform), created in 2004 by D. Lowe in the University of British Columbia to solve the problem of scale variance for feature extraction. There are two main parts to this algorithm, first detect the key points and then extract the key-point descriptors.

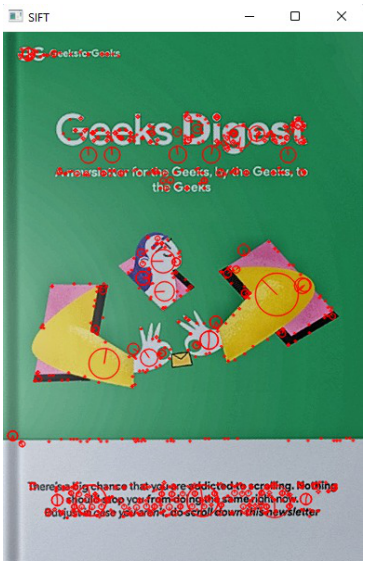


Figure 10. Using SIFT to extract key features

The second option is the FAST algorithm. As the name suggests, this algorithm has a very fast computing time. FAST just provides us with the key points, thus we need to use other methods like SIFT and to compute descriptors.

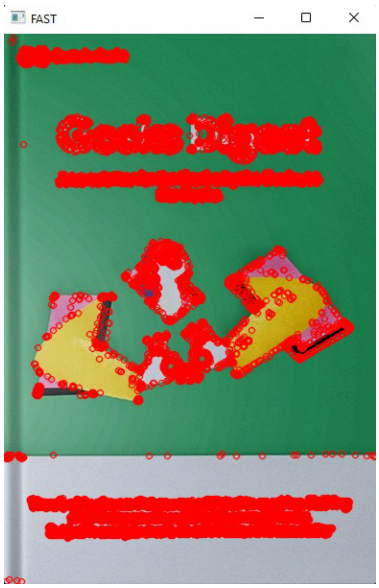


Figure 11. Using FAST to Extract Key Features

The last option is ORB (Oriented FAST and Rotated Brief).Comparing ORB to SIFT and FAST, it is much more efficient at identifying the image's features. However, ORB is intended to locate less features in the image since it picks up the most crucial details faster. Despite this, this algorithm is still regarded as being quite effective when compared to other feature-detection methods.

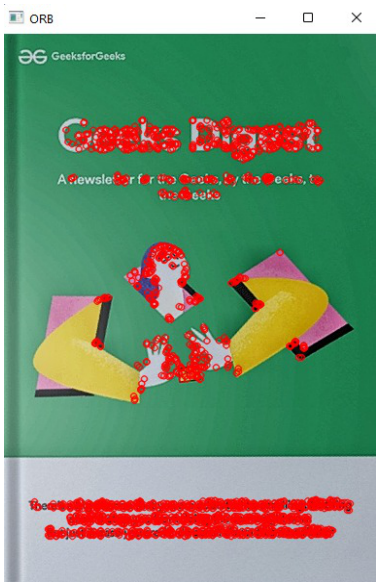


Figure 12. Using ORB to Extract Key Features

After researching further into the comparison between these three algorithms, ORB seemed to be the triumph.With quick computation, ORB consistently outperforms the competition and is demonstrated to be robust to illumination and rotational shifts. Therefore ORB will be used to extract the key features from each of the paintings. Once extracted the visual descriptor calculations for each painting will be added to the database.

Pictures using the Raspberry Pi HD camera and lens will be taken in various angles and lighting and included in the database; some of which will be of the already included paintings in the database . A machine learning model will be developed and trained through this database. TensorFlow is a robust open source machine learning platform. It has an extensive, versatile range of features, libraries, and support systems that enable researchers to push the boundaries of ML and developers to easily build and deploy ML-powered applications. Tensorflow provides already pretrained ML models which can be further trained and customized for the task it needs

to accomplish. An already pre-trained model on image data is ResNet50, one of CNNs(Convolutional Neural Network) architectures.

A neural network is an artificial intelligence technique that instructs computers to process data in a manner modeled after the human brain. A type of machine learning process that utilizes interconnected nodes or neurons in a layered structure that resembles the human brain. It generates an adaptive system that computers can use to learn from their mistakes and continuously improve. Thus, artificial neural networks attempt to solve complex problems with greater accuracy. Convolutional neural networks (ConvNets or CNNs) is one of the main ways to recognize and classify pictures.

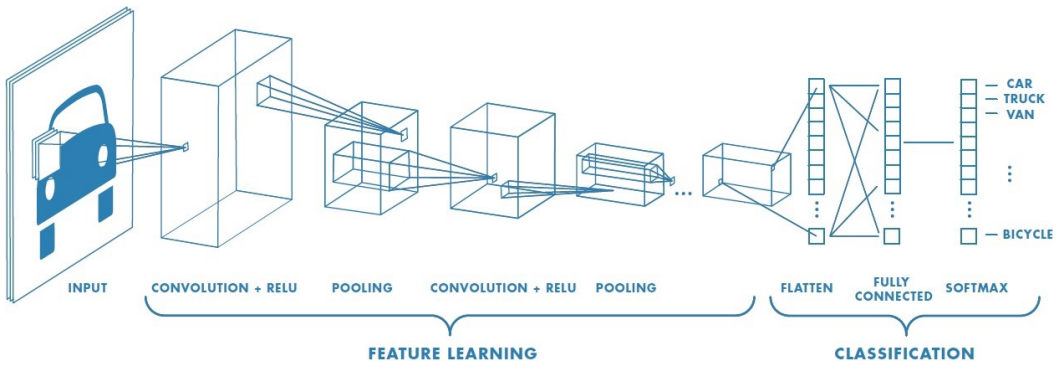


Figure 13. Convolutional neural networks (ConvNets or CNNs) architecture

The image above details the complete process of CNN to classify an inputted image. Convolution is a technique for extracting features from an input image. By learning visual attributes from small input data squares, convolution preserves the link between pixels. Two

inputs, such as an image matrix and a filter or kernel, are required for this mathematical procedure. Applying different filters to an image through convolution allows for the conduction of procedures like edge detection, blur, and sharpening.

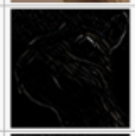
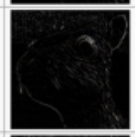
Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Figure 14. Output of applying different filters through convolution

When the photos are too huge, the section on pooling layers will lower the number of parameters. Spatial pooling, also known as subsampling or downsampling, lowers the dimensionality of each map while preserving crucial data.

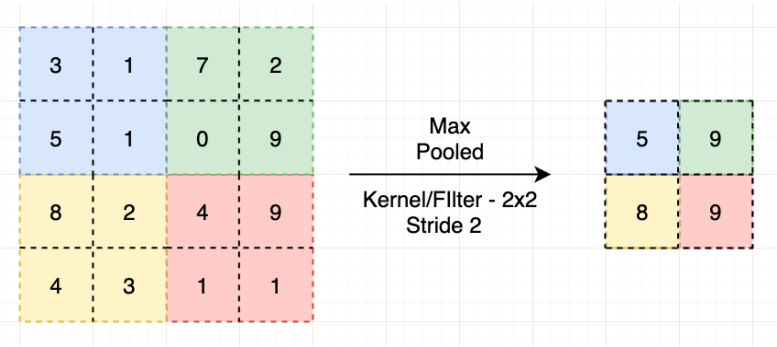


Figure 15. Applying max pooling to a matrix.

We flatten our matrix into a vector and feed it into a fully linked layer, much like a neural network. Finally, we have an activation function to classify the outputs, as shown in the picture below.

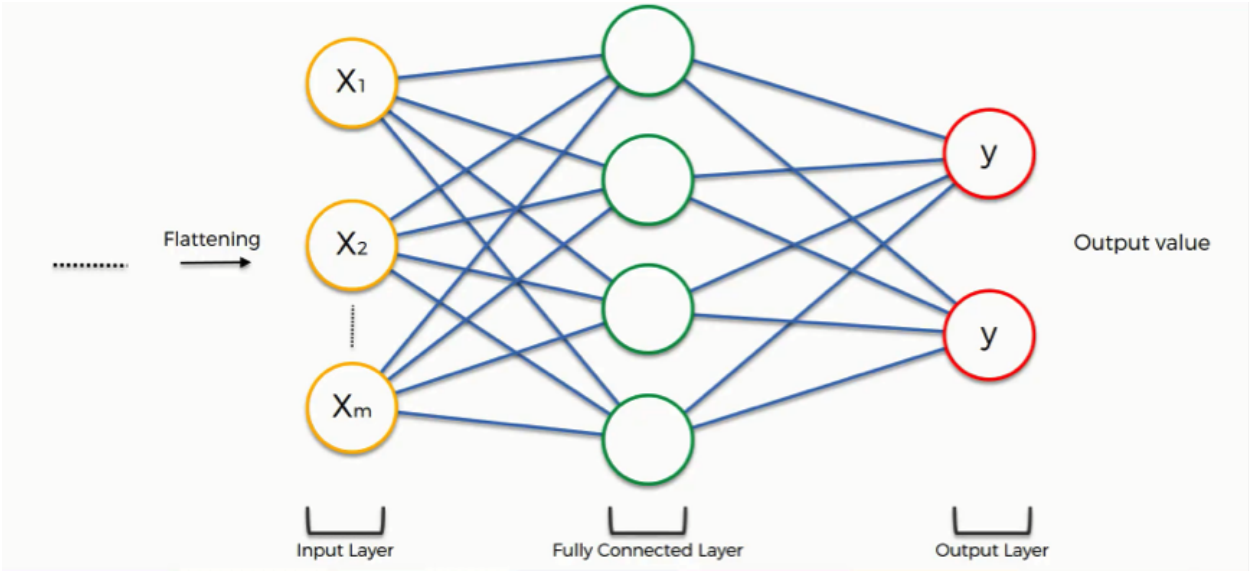


Figure 16. Convolution Neural Network(CNN) Classification Diagram

ResNet, short for Residual Network, uses residual learning as opposed to making an effort to learn specific attributes. Residual can be described as removing the learned features from the input. ResNet's central concept is that ResNet uses skip connection to add the output from an earlier layer to a later layer. The architecture of ResNet50 is shown below.

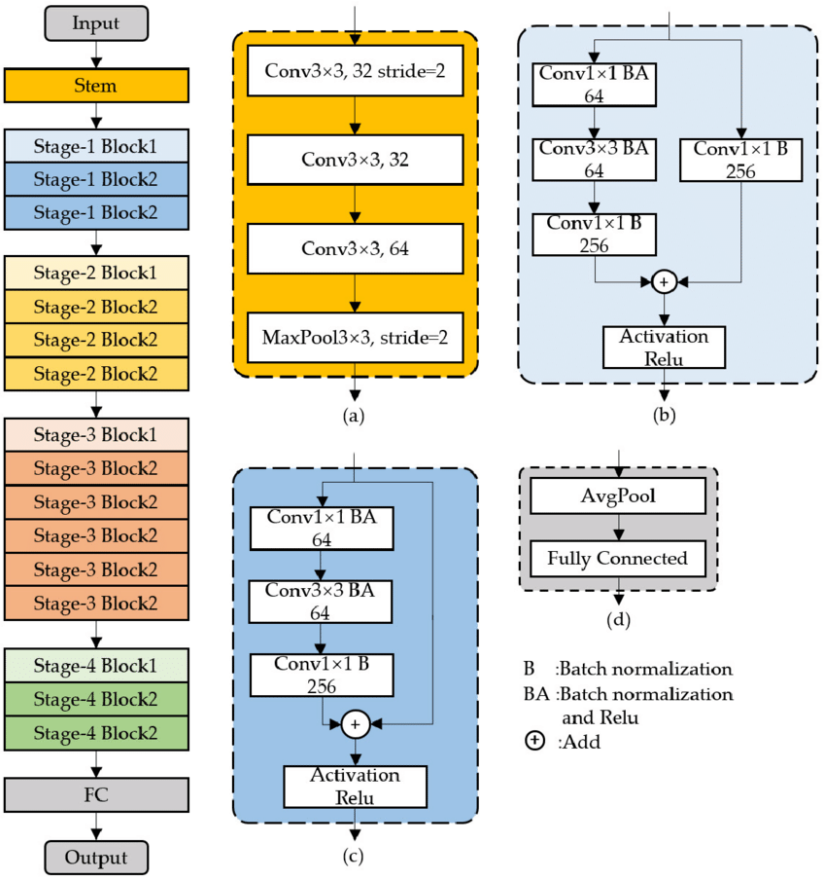


Figure 17. ResNet50 Architecture

To train the model there are various optimization algorithms. The properties of a neural network, such as weights and learning rate, can be modified using an optimizer, which is an algorithm or a technique, to lessen losses. The most popular optimizers are AdaGrad, Adam andAdaDelta.

Adaptive Gradient (AdaGrad) scales each parameter's alpha in accordance with its history of gradients (prior steps), which is essentially accomplished by dividing the current

gradient in the update algorithm by the total number of previous gradients. So , when the gradient is large, alpha is reduced, and vice versa.

$$W_t = W_{t-1} - (lr_t * (\nabla_{W_{t-1}} L))$$

$$g_t = \nabla_{W_{t-1}} L$$

$$W_t = W_{t-1} - (lr_t * g_t)$$

$$lr_t = \frac{lr}{\sqrt{\alpha_{t-1} + \varepsilon}}$$

$$\alpha_{t-1} = \sum_{i=1}^{t-1} g_i^2$$

Function 1. AdaGrad Optimizer Algorithm

With the previous approach, AdaGrad, the learning rate decreased over a large number of iterations, which caused the process to converge slowly. The AdaDelta algorithm gives a suggestion to prevent this: use an exponentially decaying average.

$$W_t = W_{t-1} - (lr_t * g_t)$$

$$lr_t = \frac{lr}{\sqrt{eda_{t-1} + \varepsilon}}$$

$$eda_{t-1} = (\gamma eda_{t-2}) + ((1 - \gamma) * g_{t-1}^2)$$

Function 2. AdaGrad Optimizer Algorithm

To change the learning rate with the AdaDelta method, the exponentially decaying averages of the square of the gradients was stored. With Adam's optimizer, both the first order of moment and second order moment of the gradient are stored.

$$W_t = W_{t-1} - \alpha * \frac{m_t^*}{\sqrt{v_t^* + \varepsilon}}$$

$$0 < \beta_1, \beta_2, \alpha < 1$$

. . .

EDA for 1st order moment:

$$m_t = \beta_1 * m_{t-1} + (1 - \beta_1) * g_t$$

EDA for 2nd order moment:

$$v_t = \beta_2 * v_{t-1} + (1 - \beta_2) * g_t$$

Bias Correction:

$$m_t^* = \frac{m_t}{1 - \beta_1^t}$$

$$v_t^* = \frac{v_t}{1 - \beta_2^t}$$

Function 3. Adam Optimizer Algorithm

Using one of the optimizers the model will be trained. A technique we can use to cut down on the computational time for training while not losing precision is by freezing layers of ResNet50. This means keep a few layers hidden to not train hence the weights will remain the same. Certain layers are frozen from training due to various reasons, for example wanting to train the newly added layers to an already pre-trained model.

Resnet50 constructs 50 neural networks, and in order to train them, a lot of space is required. To address this, Resnet50 should be run on Google Colab, which offers 13 GB of space. Google Colab, which allows anyone to develop and run arbitrary Python code in the browser, is especially suited for machine learning.

Once the model has been trained it should be evaluated using graphs and matrices. Then it should be tested using pictures that we have taken with the Raspberry Pi camera and have not used in the training to identify. Now the model can be exported and used as needed.

B. Online

The online part contains two main systems: the front end and the backend. The front end focuses on the visual elements the user will interact with. The back end with the server side that users can't see.

In the front end the user interacts with a button at the side of their glasses, which will start the process of capturing a painting and identifying it. Once the user hits the button thus taking the picture the backend process starts. The image received in the back end can be represented as a grid where each square is a pixel meaning it contains Red Green and Blue intensity values as shown in the picture below.

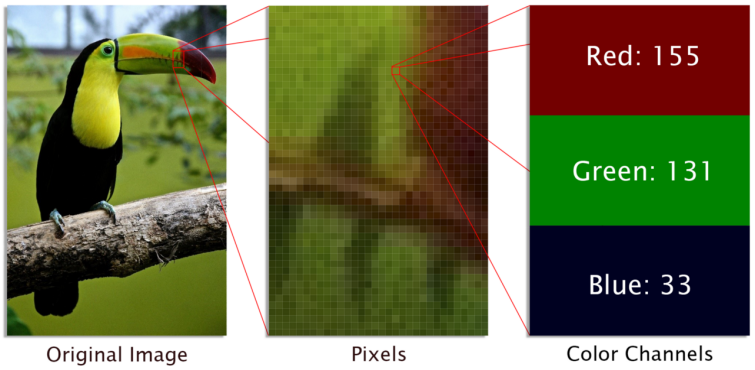


Figure 18. Image Displayed as Grid of Pixels

We can modify these values to extract the painting from the image. First gaussian blur needs to be applied to the image so that we can reduce the noise introduced due to poor temperature and illumination. The Gaussian Blur is applied to an image by calculating the transformation to apply to each pixel by using the Gaussian Function as displayed below..

$$G(x, y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Function 4. Gaussian Function

In the Gaussian function displayed above x and y are distances from the origin in the horizontal and vertical axes, respectively, and σ is the standard deviation of the Gaussian distribution. This formula creates a surface with concentric circles that have a Gaussian distribution from the center point.

A convolution matrix is constructed using values from this distribution and then applied to the original image. The new value for each pixel is a weighted average of that pixel's neighborhood. The value of the initial pixel, which has the highest Gaussian value, is given the most weight, and as the distance between pixels grows, so do their weights. So, as a result even though the image is blurred the boundaries and edges are preserved.

To determine where the painting is in the image the edges which can represent the painting's outline that we want to extract need to be detected. Edges are where intensity of pixels changes drastically, and taking the derivative is how we can determine where these changes occur. Once the edges are identified we can extract the painting from the image. To do this process automatically we will need to train our own classifier using machine learning.

Once the painting has been extracted from the image, the painting needs to be identified. To do so we will use OpenCV and the model we trained through TensorFlow. We can do that through OpenVino. OpenVINO covers Computer vision workloads on Intel hardware. It speeds up application development and increases performance. Using a library of preset functions and pre-optimized kernels. We can use OpenVino since we will hand the back end inference computation to the Intel Neural Compute Stick. These computations are usually conducted by the CPU but since we are using machine learning the Raspberry Pi CPU can not handle it. With Intel NCS2 one can develop and deploy convolutional neural networks (CNNs) on low-power applications that require real-time inferencing.

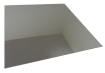
Using OpenCV we will then run the ML model to rank which painting the input matches the most with. Based on the top match we will output the painting context to the user through a speaker.

V. Cost Analysis

The table below contains all the components needed to develop the Smart Spectacles for Image Recognition of Art (SSIR) together with the total cost. Currently there is a shortage of Raspberry Pi board and Intel Neuron Compute Stick. These two components have increased in price which also fluctuates due to being very low in stock. Since SSIR will start to be developed at the start of January of 2023, there is a hope that this shortage will be subdued together with their prices.

Table 1. Cost Table of SSIR Prototype Implementation

Product	Unit Price	Quantity	Total Price
 Transparent Graphical OLED Breakout	$42.95	2	$85.90
 Rechargeable 5V Circuit	$14.95	2	$29.90
 **Raspberry Pi Compute I/O board**	**$39.99 - $99**	**2**	**$79.99 - $198**
 Li-po Battery	$10.49	2	$20.98
 Raspberry Pi HQ Camera	$50.00	1	$50.00
 Raspberry Pi HQ Camera Lens	$25.00	1	$25.00
 Connector Wires	$4.95	1	$4.95
 Flex Connector	$6.41	1	$6.41
 Push Button	$5.95	4	$23.80

 Mini USB Microphone	$16.59	2	$33.18
 Mini USB Speaker	$14.95	2	$29.90
 Raspberry 4 Port USB Hub	$12.99	1	$12.99
 **Intel Neural Compute Stick 2**	**$83.41 - $201.73**	**1**	**$83.41 - $201.73**
 BiConvex Lens	$1.75	4	$6.99
 Acrylic See-Through Mirror	$16.99	1	16.99
			$497.31 - $725.64

VI. Division of Tasks

Our team has decided that the best method to organize our design and testing process is to divide our tasks into two camps; hardware and software.

The hardware side of our project will be mainly focused on the construction of our smart glasses. These individuals will design and construct the electronics to be incorporated into our prototype. The synthesis of our Raspberry Pi, OLED, and cameras - among other components -

will all be handled by the hardware group. Alongside this, the group will also handle external device integration, such as connectivity with smartphones.

Our software side will deal with our image recognition software. This for the most part will be focused on designing our image database that will be used by our machine learning. Tasks will also include training the machine learning software to recognize our desired images.

Both sides will of course also rigorously test their portion of the project. The hardware side will test basic power activation, and validate that all components of the smart glasses - from camera to Raspberry Pi - are all running correctly. Intermittent testing will also take place on our image recognition to ensure proper functionality.

All members of the project will test the final design of the prototype after previous work is complete. The desired goal of this testing is to ensure a working prototype that will be ready for presentation. Below can be observed our division of tasks in full:

Task Name	Assignment	Start Date	End Date	Day #	Duration
Milestone #1 - Preparation					
Project Brainstorming	All	5/10	5/25	0	15 days
Market Research	All	5/25	6/13	15	19 days
Process and Component Breakdown	Angela	6/13	6/24	34	11 days
Workflow Breakdown	Warren	6/13	6/24	34	11 days
Initial Proposal	All	6/23	6/24	44	1 days
Milestone #2 - Hardware					
Smart Glasses Hardware Design	Warren	1/9	2/6	244	28 days
External Device Integration	Nicla	1/9	2/7	244	29 days
Smart Glass Basic Testing and Validation	Vianna	1/9	2/8	244	30 days
Milestone #3 Software - Image Recognition					
Design Image Database	Angela, Denada	1/9	2/28	244	50 days
Machine Learning Training	Angela, Denada	1/9	3/1	244	51 days
Image Recognition Testing and Validation	Angela, Denada	2/9	3/23	275	42 days
Final Testing and Validation	All	3/23	4/28	317	36 days
Milestone #4 - Presentation					
Capstone Final Report	All	3/3	4/28	297	56 days
Capstone Final Presentation	All	3/3	4/29	297	57 days

Figure 19. Division of Tasks

VII. Proposed Timeline

The proposed timeline for this project is an estimated 297 days in total. Much of the initial work has already been completed, including market research, component breakdown, and the initial proposal.

Our timeline has much of our actual project work projected to be completed in the Spring Semester of 2023. The division of work - hardware and software - is expected to be completed concurrently. With the return of Spring Semester, production is expected to take place between January and March. The goal is to have a completed prototype by the end of March. From there, the remaining weeks of April will be spent testing and validating our prototype and preparing for our final presentation. This timeline can be observed below:

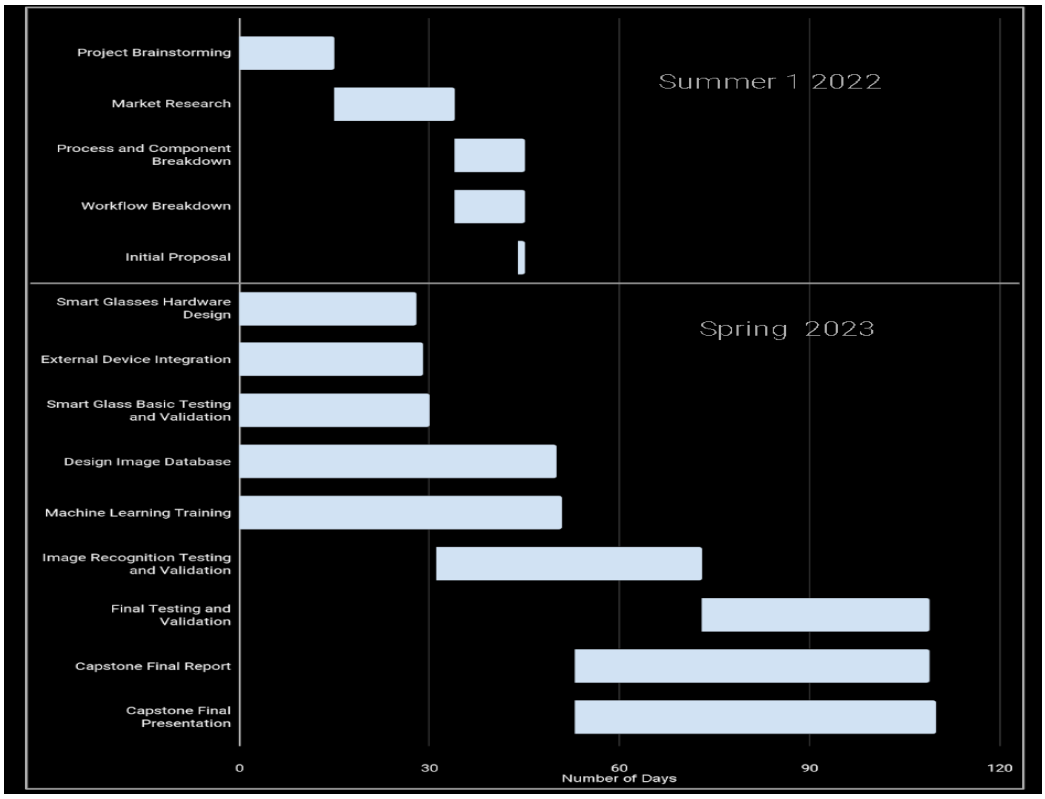


Figure 20. Proposed Timeline

VIII. Conclusion

Our overall objective is to create a product that will allow the user to maintain their immersive experience when looking at artistic pieces. We hope to develop a device that will accurately recognize art, and through machine learning give the user a detailed listing of the various characteristics of the piece they are viewing. We will provide this experience using wearable technology, in this case smart glasses that will work in tandem with our software to deliver the information directly in front of the user.

Our clip-on hardware - which will consist of a Raspberry Pi, camera lens, an OLED, and push button among other components - will allow the device to quickly scan the image the user desires more information of. Using our computational device we will be able to link ourselves to an external server which will serve as the central database of information pertinent to art pieces. As discussed prior, using various software algorithms and mathematical computations, we will be able to correctly identify the art piece and provide the user with the information that they seek of what they are currently viewing.

Using the connection of machine learning and smart glass technology, we hope to provide a seamless experience for the user in a reliable, efficient, and accessible manner. This product should enhance the cultural awareness of the public, and leave the user more informed of the rich artistic history that surrounds them.

IX. Appendices

List of Figures

List of Figures

List of Functions

X. References

[1] "Shazam", Shazam, 2022. [Online]. Available: https://www.shazam.com/home.

[2] "Google Images", Images.google.com, 2022. [Online]. Available: https://images.google.com/.

[3] "The World's Most Downloaded Museum App", Smartify, 2022. [Online]. Available: https://about.smartify.org/.

[4] "National Gallery of Art", Nga.gov, 2022. [Online]. Available: https://nga.gov/.

[5] "Working with the Pi: Getting Online, I/o and Command Lines." *Blog - Raspberry Pi - element14 Community*, community.element14.com/products/raspberry-pi/b/blog/posts/working-with-the-pi-getting-online-i-o-and-command-lines.

[6] "Overview of Intel® Distribution of Openvino™ Toolkit." *Intel*, https://www.intel.com/content/www/us/en/developer/tools/devcloud/edge/learn/openvino.html.

[7] Gus. "Building Your Own Raspberry Pi Google Assistant." *Pi My Life Up*, 31 Jan. 2022, https://pimylifeup.com/raspberry-pi-google-assistant/.

[8] Perminov, Alexey. "Running Tensorflow Model Inference in Openvino." *OpenCV*, 25 Dec. 2020, https://opencv.org/running-tensorflow-model-inference-in-openvino-2/.

[9] "Image Classification : Tensorflow Core." *TensorFlow*, https://www.tensorflow.org/tutorials/images/classification.

[10] Stephanie_Maluso. "Introducing Openvino™ Integration with Tensorflow." *CodeProject*, CodeProject, 18 May 2022, https://www.codeproject.com/Articles/5331792/Introducing-OpenVINO-integration-with-TensorFlow.

[11] *Adam: A Method for Stochastic Optimization - Arxiv*. https://arxiv.org/pdf/1412.6980.

[12] Kumar, Satyam. "Overview of Various Optimizers in Neural Networks." *Medium*, Towards Data Science, 9 June 2020, https://towardsdatascience.com/overview-of-various-optimizers-in-neural-networks-17c1be2df6d5.

[13] Sagar, Ram. "What Does Freezing a Layer Mean and How Does It Help in Fine Tuning Neural Networks." *Analytics India Magazine*, 10 Oct. 2020, https://analyticsindiamag.com/what-does-freezing-a-layer-mean-and-how-does-it-help-in-fine-tuning-neural-networks/.

[14] "Allegro - Raspberry Pi Zero datasheets," *Raspberry Pi*. [Online]. Available: https://datasheets.raspberrypi.com/rpizero/raspberry-pi-zero-mechanical-drawing.pdf. [Accessed: 28-Jun-2022].

[15] "Raspberry pi datasheets," *Raspberry Pi*. [Online]. Available: https://datasheets.raspberrypi.com/cm4io/cm4io-datasheet.pdf. [Accessed: 28-Jun-2022].