Q1.

The data collection method is an extension from 8.2C API wrapper. I have added some instance variables and functions to collect pictures alongside with Accelerometer axes. One thing to note is that instead of saving the data (picture and axes) at 10 seconds interval, I use 1 second interval because I can get more meaningful information about the pattern of distinct activities. Also, I used my phone camera instead of laptop’s because my laptop’s camera is broken. I won’t go into details since the class has already been carefully explained in the 8.2C task, but here is what I have added in the API class.

# Initialize camera capture of camera\_url is provided

        if self.camera\_url:

            self.init\_camera()

def init\_camera(self) -> None:

        """Initialize the camera capture from the IP camera URL"""

        self.camera\_cap = cv2.VideoCapture(self.camera\_url)

        if not self.camera\_cap.isOpened():

            print(f"Failed to open camera at {self.camera\_url}")

def capture\_image(self, sequence\_number: int, timestamp: str) -> None:

        """Capture image from the camera, save with the sequence number and timestamp"""

        try:

            if self.camera\_cap and self.camera\_cap.isOpened():

                ret, frame = self.camera\_cap.read()

                if ret:

                    # Rotate and resize the image

                    rotated\_frame = cv2.rotate(frame, cv2.ROTATE\_90\_CLOCKWISE)

                    resized\_frame = cv2.resize(rotated\_frame, (720, 1280))

                    # Ensure the image save directory exists

                    if not os.path.exists(self.image\_save\_dir):

                        os.makedirs(self.image\_save\_dir)

                    # Construct the file path with a proper file name and extension (.jpg)

                    image\_filename = os.path.join(

                        self.image\_save\_dir, f"{sequence\_number}\_{timestamp.replace(':', '')}.jpg"

                    )

                    # Save the image and check if the operation is successful

                    if cv2.imwrite(image\_filename, resized\_frame):

                        print(f"Image saved successfully: {image\_filename}")

                        self.latest\_image\_path = image\_filename  # Update the latest image path

                    else:

                        print(f"Failed to save image: {image\_filename}")

                else:

                    print("Failed to capture image from the camera.")

            else:

                print("Camera is not initialized or not opened.")

        except Exception as e:

            print(f"Exception during image capture: {e}")

def create\_dash\_app(self) -> Dash:

        """Create the Dash app for live monitoring"""

        app = Dash(\_\_name\_\_)

        app.layout = html.Div([

            dcc.Graph(id='live-graph'),

            dcc.Interval(id='graph-update', interval=self.update\_interval),

            html.Img(id='live-image', src='', style={'width': '25%', 'display': 'block', 'margin': 'auto'}),

        ])

# Update the image displayed on the dashboard

        @app.callback(

            Output('live-image', 'src'),

            Input('graph-update', 'n\_intervals')

        )

        def update\_image(n):

            if self.latest\_image\_path:

                encoded\_image = self.encode\_image\_to\_base64(self.latest\_image\_path)

                return f"data:image/jpeg;base64,{encoded\_image}"

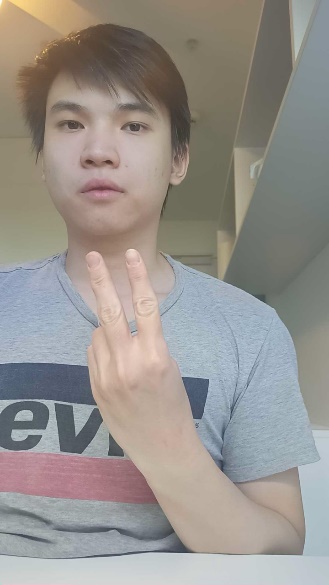
            return ''

        return app

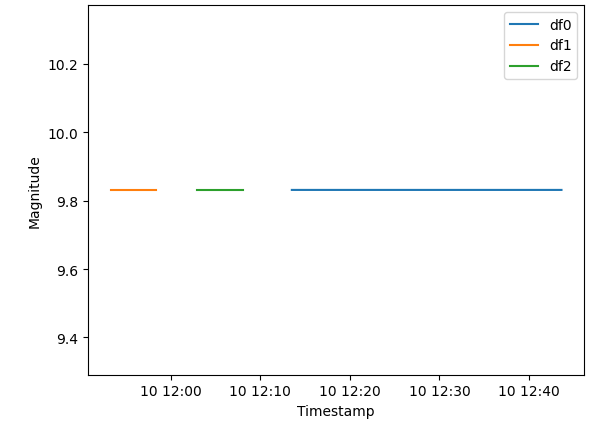
That is pretty much it, the comments have explained on the functionality of each part. I just added some code to save the picture and display the newest one on the dashboard. The visualization will be in the video.

Q2. The findings

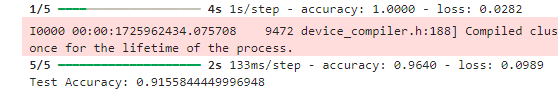
I have done 2 things to try to find pattern in my distinct activities. I need to explain the distinct activities first. The first one ‘0’ is just me working, not showing any hand gestures; the second ‘1’ is when I raise a thump up; the third '2’ is when I raise a peace sign. Each picture below corresponds to 0,1,2 label, and each picture is taken at different angles.

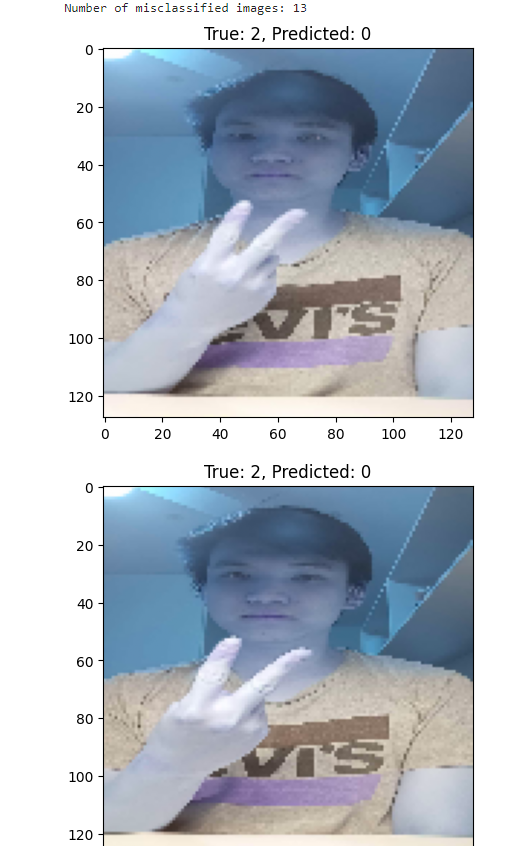


First is through the representation of linear Accelerometer Linear or the magnitude of the axes. This technique is similar to my 8.2C task when I combine the 3 axes into 1 line. Here is the time series plot.



The only pattern I can see is that different activity is recorded at different time (as the lines are disconnected). This makes me realize I need a more powerful technique to recognize the patterns of my activities, so I decided to use CNN (Convolutional Neural Network). I will explain how in section 3 but results the model gives are quite good. Using image matrices and its respective labels, I got 100% accuracy on the train set, and 91% on the test set.



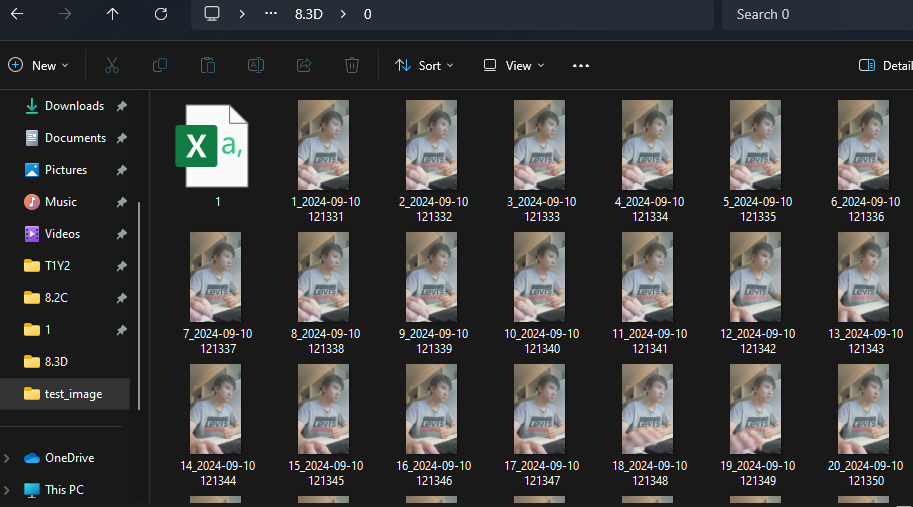


I have used similar camera angle on both the training and testing set because we only have limited amount of data.

Q3. Annotation file and how did I recognize patterns in images

I have recorded and saved the csv file for the Accelerometer axes and pictures for each label in folder like this.

A black background with blue lights

Description automatically generated

I transformed those images into matrices, scaled them and added them to a dataframe using this code:

dir = r"/mnt/c/Users/tomde/OneDrive/Documents/Deakin/Deakin-Data-Science/T1Y2/SIT225 - Data Capture Technologies/Week 8 - Using smartphone to capture sensor data/8.3D/2"

# Define the image size for the model

image\_size = (128,128)

# Function to extract year '2024' from picture names

def contain\_2024(filename):

    return re.search(r'2024', filename) is not None

# List all files in the directory

files = os.listdir(dir)

# Filter for image files containing '2024' in the filename

image\_filenames\_2024 = [file for file in files if file.endswith(('.jpg', '.png')) and contain\_2024(file)]

image\_filenames\_2024

# List to store image matrix

image\_data = []

# Loop through the filtered filenames and convert each image to a matrix

for filename in image\_filenames\_2024:

    img\_path = os.path.join(dir, filename)

    # Read the image using OpenCV

    image = cv2.imread(img\_path)

    # Resize the image to a uniform size (128x128)

    image\_resized = cv2.resize(image, image\_size)

    # Normalize the image (values between 0 and 1)

    image\_resized = image\_resized / 255.0

    # Convert the image to a CuPy array (move to GPU)

    image\_matrix = np.array(image\_resized)

    # Append the matrix along with the filename

    image\_data.append({

        'Filename': filename,

        'ImageMatrix': image\_matrix

    })

df\_images = pd.DataFrame(image\_data)

# Combine with df0

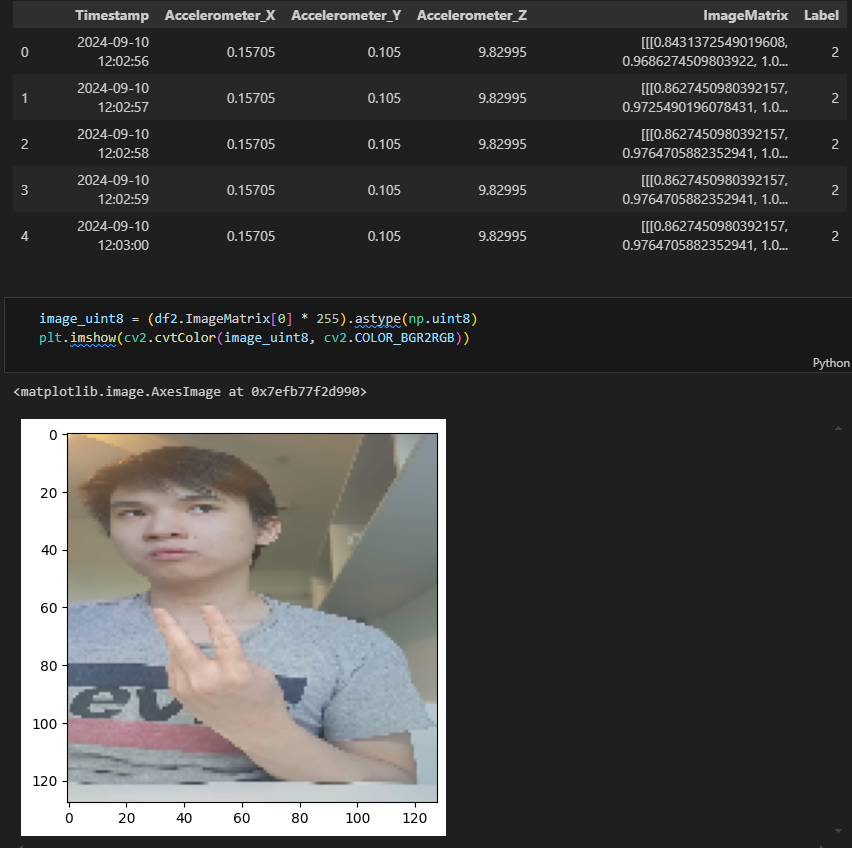
df2 = pd.read\_csv('2/1.csv')

df2['ImageMatrix'] = df\_images['ImageMatrix']

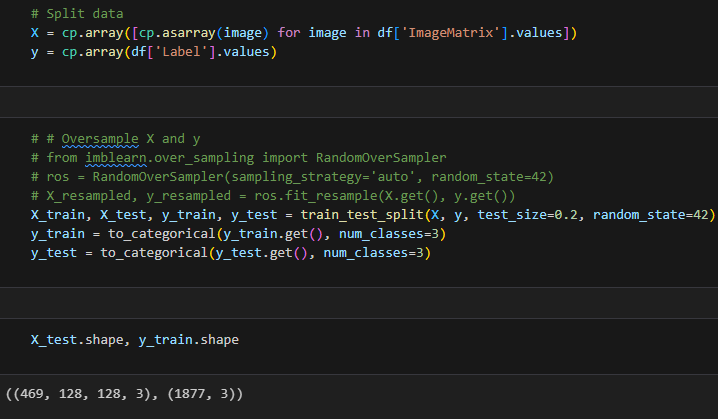
df2['Label'] = 2

df2.head()

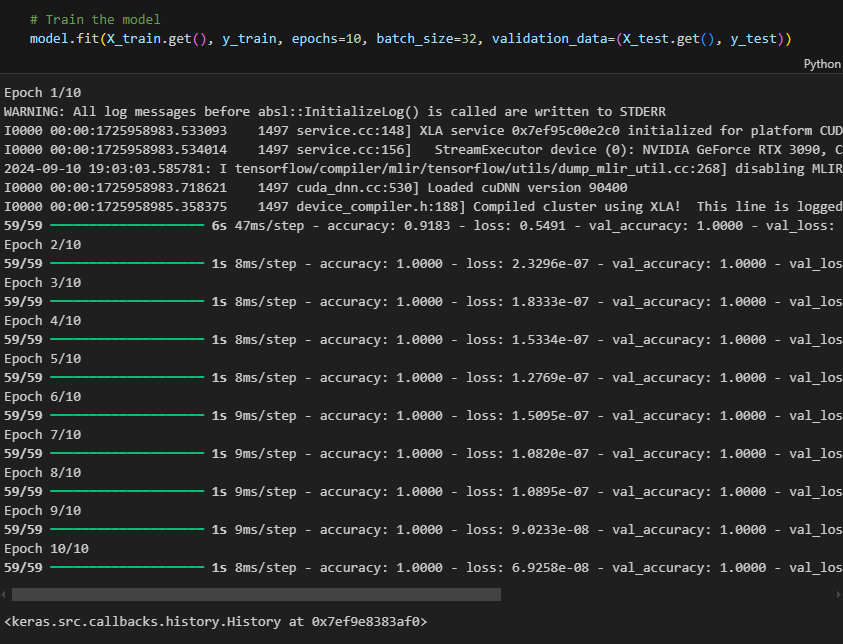
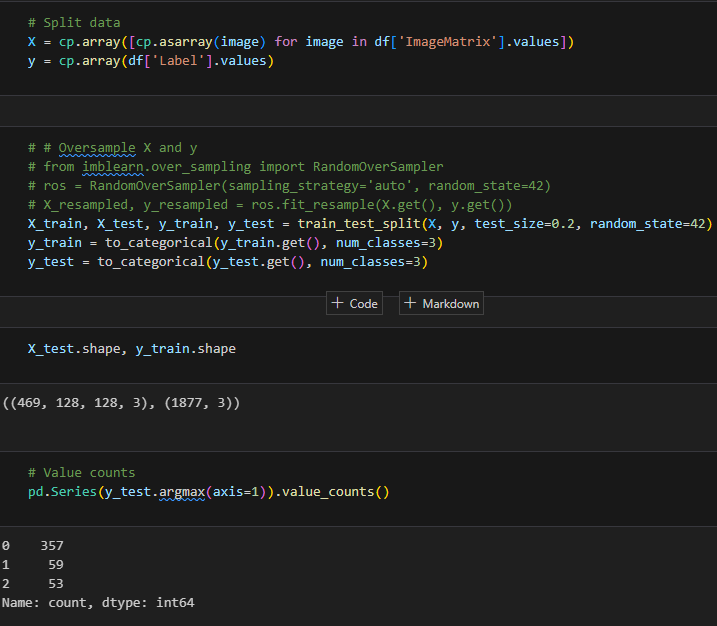
The output is



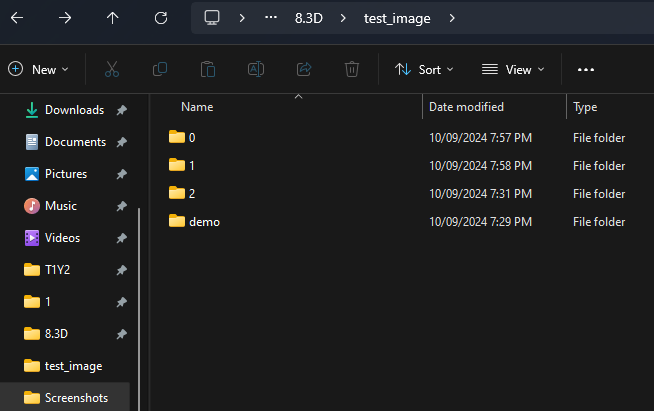
The above example is just for Label 2, but I have done the same for the other two labels. Next, I combine the three dataframes into one. Then, I splitted the data into X and y, and in the correct format for CNN.



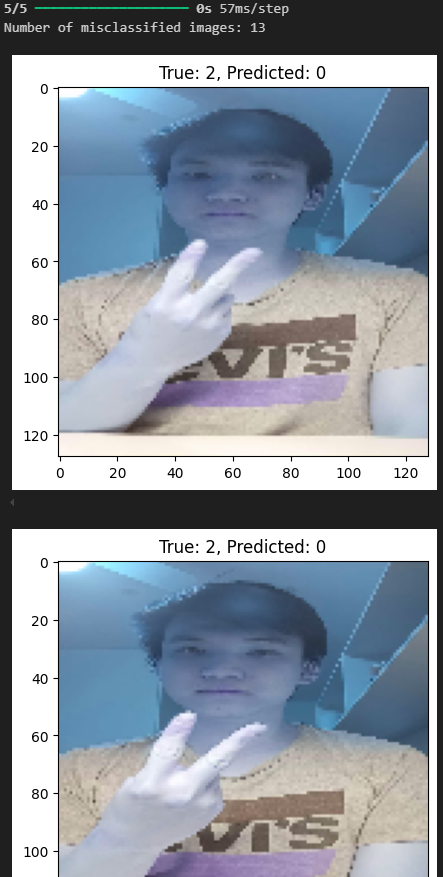
Then I train the CNN model.



Then I created a new folder containing the test set. The test set photos are taken in roughly the same angle as the train ones.



There are a total of 154 test samples, and I scaled those down and fit into the model.



The model got 13 wrong out of 154. From this, I conclude that for complex pattern recognition like this (those the deals with pictures), we need cutting-edge machine learning models. Although, I do not know which neuron of the model captures which pixel, or how can it recognize patterns in complex image matrices since deep learning is a black box, I do know that this proves complex pattern analysis requires machine learning.

Q4.