Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

AUTOMATIZACIÓN DE TAREAS EN SISTEMA OPERATIVO WINDOWS 10 MEDIANTE RECONOCIMIENTO DE GRÁFICOS UTILIZANDO OPENCV

Walter Roberto Morales Quiñonez
Asesorado por el Ing. Marco Tulio Aldana Prillwitz

Guatemala, octubre de 2023

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

AUTOMATIZACIÓN DE TAREAS EN SISTEMA OPERATIVO WINDOWS 10 MEDIANTE RECONOCIMIENTO DE GRÁFICOS UTILIZANDO OPENCV

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

WALTER ROBERTO MORALES QUIÑONEZ
ASESORADO POR EL ING. MARCO TULIO ALDANA PRILLWITZ

AL CONFERÍRSELE EL TÍTULO DE

INGENIERO EN CIENCIAS Y SISTEMAS

GUATEMALA, OCTUBRE DE 2023

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. José Francisco Gómez Rivera (a. i.)
VOCAL II	Ing. Mario Renato Escobedo Martínez
VOCAL III	Ing. José Milton de León Bran
VOCAL IV	Ing. Kevin Vladimir Cruz Lorente
VOCAL V	Br. Fernando José Paz González
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANA	Inga. Aurelia Anabela Cordova Estrada
EXAMINADOR	Ing. César Augusto Fernández Cáceres
EXAMINADOR	Ing. Herman Igor Véliz Linares
EXAMINADOR	Ing. Edgar Francisco Rodas Robledo
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

HONORABLE TRIBUNAL EXAMINADOR

En cumplimiento con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

AUTOMATIZACIÓN DE TAREAS EN SISTEMA OPERATIVO WINDOWS 10 MEDIANTE RECONOCIMIENTO DE GRÁFICOS UTILIZANDO OPENCV

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería de Ciencias y Sistemas con fecha diciembre 2022.

Walter Roberto Morales Quiñonez

Guatemala, 15 de agosto de 2023

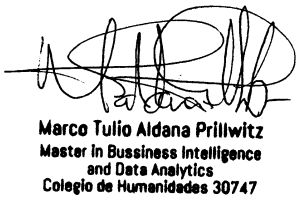
Ingeniero
Carlos Alfredo Azurdia
Coordinador de Privados y Trabajos de Tesis
Escuela de Ingeniería en Ciencias y Sistemas
Facultad de Ingeniería - USAC

Respetable Ingeniero Azurdia:

Por este medio hago de su conocimiento que en mi rol de asesor del trabajo de investigación realizado por el estudiante **WALTER ROBERTO MORALES QUIÑONEZ** con carné **200915518 y CUI 1654 22688 0301** titulado **"AUTOMATIZACIÓN DE TAREAS EN SISTEMA OPERATIVO WINDOWS 10 MEDIANTE RECONOCIMIENTO DE GRÁFICOS UTILIZANDO OPENCV",** lo he revisado y luego de corroborar que el mismo ha sido completado teniendo un progreso del 100 por ciento y que cumple con los objetivos propuestos en el respectivo protocolo, procedo a la aprobación correspondiente.

Al agradecer su atención a la presente, aprovecho la oportunidad para suscribirme,

Atentamente,



Lic. Marco Tulio Aldana Prillwitz
Colegiado Humanidades No. 30747

Universidad San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

Guatemala 7 de septiembre de 2023

Ingeniero
Carlos Gustavo Alonzo
Director de la Escuela de Ingeniería
En Ciencias y Sistemas

Respetable Ingeniero Alonzo:

Por este medio hago de su conocimiento que he revisado el trabajo de graduación del estudiante **WALTER ROBERTO MORALES QUIÑONEZ** con carné **200915518** y CUI **1654 22688 0301** titulado **"AUTOMATIZACIÓN DE TAREAS EN SISTEMA OPERATIVO WINDOWS 10 MEDIANTE RECONOCIMIENTO DE GRÁFICOS UTILIZANDO OPENCV",** y a mi criterio el mismo cumple con los objetivos propuestos para su desarrollo, según el protocolo aprobado.

Al agradecer su atención a la presente, aprovecho la oportunidad para suscribirme,

Atentamente,

Ing. Carlos Alfredo Azurdia
Coordinador de Privados
y Revisión de Trabajos de Graduación

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

SIST.LNG.DIRECTOR.8.EICCSS.2023

El Director de la Escuela de Ingeniería en Ciencias y Sistemas de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer el dictamen del Asesor, el visto bueno del Coordinador de área y la aprobación del área de lingüística del trabajo de graduación titulado: **AUTOMATIZACIÓN DE TAREAS EN SISTEMA OPERATIVO WINDOWS 10 MEDIANTE RECONOCIMIENTO DE GRÁFICOS UTILIZANDO OPENCV**, presentado por: **Walter Roberto Morales Quiñonez**, procedo con el Aval del mismo, ya que cumple con los requisitos normados por la Facultad de Ingeniería.

"ID Y ENSEÑAD A TODOS"

Ingeniero Carlos Gustavo Alonzo
DIRECTOR
Escuela de Ingeniería en Ciencias y Sistemas

Guatemala, octubre de 2023

Decanato
Facultad e Ingeniería

24189101- 24189102

LNG.DECANATO.OIE.91.2023

El Decano de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería en Ciencias y Sistemas, al Trabajo de Graduación titulado: **AUTOMATIZACIÓN DE TAREAS EN SISTEMA OPERATIVO WINDOWS 10 MEDIANTE RECONOCIMIENTO DE GRÁFICOS UTILIZANDO OPENCV**, presentado por: **Walter Roberto Morales Quiñonez** después de haber culminado las revisiones previas bajo la responsabilidad de las instancias correspondientes, autoriza la impresión del mismo.

IMPRÍMASE:

Firmado electrónicamente por: José Francisco Goméz Rivera
Motivo: Orden de impresión
Fecha: 13/10/2023 16:29:09
Lugar: Facultad de Ingeniería, USAC.

Ing. José Francisco Gómez Rivera
Decano a.i.

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DECANO a.i.
Facultad de Ingeniería

Guatemala, octubre de 2023

Para verificar validez de documento ingrese a https://www.ingenieria.usac.edu.gt/firma-electronica/consultar-documento

Tipo de documento: Correlativo para orden de impresión Año: 2023 Correlativo: 91 CUI: 1654226880301

Escuelas: Ingeniería Civil, Ingeniería Mecánica Industrial, Ingeniería Química, Ingeniería Mecánica Eléctrica, - Escuela de Ciencias, Regional de Ingeniería Sanitaria y Recursos Hidráulicos (ERIS). Postgrado Maestría en Sistemas Mención Ingeniería Vial. Carreras: Ingeniería Mecánica, Ingeniería Electrónica, Ingeniería en Ciencias y Sistemas. Licenciatura en Matemática. Licenciatura en Física. Centro de Estudios Superiores de Energía y Minas (CESEM). Guatemala, Ciudad

ACTO QUE DEDICO A:

Dios	Por darme el privilegio de culminar esta meta tan importante y por mostrarme que el camino hacia el éxito está lleno de enseñanzas y que es ahí donde se aprende humildad.
Mi madre	Sandra Patricia Quiñonez, por tener fe en mí, aun cuando la mía casi se terminaba, por sacrificios personales que hizo para que yo pudiera continuar con mis sueños.
Mis hermanos	Por acompañarme en este camino y estar a lado de mi madre mientras yo avanzaba en mis estudios.
Mi padre	Por animarme a concluir mis estudios y tener fe en mí.

AGRADECIMIENTOS A:

Universidad de San Carlos de Guatemala	Por ser la casa matriz del conocimiento y brindarme los recursos académicos necesarios para dotarme del conocimiento deseado.
Facultad de Ingeniería	Por sostener un nivel académico alto y garantizar el aprendizaje mediante cada proyecto y cada evaluación.
Mis amigos de la Facultad	Marco Chávez, Eliezer Coronado, Walter Figueroa, Erick To y Aníbal Morales, por ser compañeros de batalla y enriquecer mi conocimiento con su experiencia y habilidades.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES III
GLOSARIO V
RESUMEN XI
OBJETIVOS XIII
INTRODUCCIÓN XV

1. VISIÓN COMPUTACIONAL 1
   1.1. El sistema de visión computacional 1
   1.2. Percepción visual 2
   1.3. Sistema de visión humano vs. sistema de visión computacional 2
   1.4. Fundamentos de la clasificación de imágenes 7
   1.5. Flujo de la clasificación 8
   1.6. ¿Cómo observa una computadora una imagen? 9
   1.7. Imagen en escala de grises 10
   1.8. Imagen a color 10
   1.9. Introducción a *OpenCV* 11
   1.10. Template *Matching* 15
   1.11. Sistema coordenado de *OpenCV* 16
   1.12. Dibujo de figuras mediante *OpenCV* 17

2. REDES NEURONALES 19
   2.1. Introducción a la inteligencia artificial 19
   2.2. *Machine learning* 20
   2.3. *Deep learning* 21
   2.4. Redes neuronales 22

3. CONTROL DE GUI 27
3.1. Uso de la Librería *Pyautogui* 27
3.2. Captura de pantalla: análisis e identificación de coordenadas 28
3.3. Manipulación de ventanas y controles 29

4. INTEGRACIÓN 31
4.1. Identificación de imágenes objetivo 31
4.2. Algoritmo de búsqueda multiescalar 33
4.3. Diagrama de clases 38
4.4. Ejecución e Interacción con el GUI 39

5. EJECUCIÓN Y REPORTES 41
5.1. Bitácora de ejecución del *script* 41
5.2. Interfaz gráfica 43
5.3. Uso de la clase estática Recorder y captura de metadatos 47
5.4. Reporte de errores 50

6. DESCRIPCIÓN DE ENTREGABLES Y LICENCIA 53
6.1. Descripción del entregable final 53
6.2. Licencia BSD-2 53

CONCLUSIONES 55
RECOMENDACIONES 57
REFERENCIAS 59

ÍNDICE DE ILUSTRACIONES

FIGURAS

Figura 1. Sistema de visión humano 3
Figura 2. Sistema de visión computacional 4
Figura 3. Similitud entre una neurona biológica y una artificial 6
Figura 4. Red neuronal artificial 6
Figura 5. Flujo de clasificación de imágenes 8
Figura 6. Representación computacional de una imagen 9
Figura 7. Representación de una imagen a color por medio de canales 11
Figura 8. Separación modular de OpenCV 13
Figura 9. Búsqueda de imagen objetivo 15
Figura 10. Coordenadas en OpenCV 16
Figura 11. Etiquetas en objetos 18
Figura 12. Subcampos de la inteligencia artificial 19
Figura 13. Programación tradicional vs. machine learning 21
Figura 14. Significado de profundidad en deep learning 22
Figura 15. Flujo de trabajo de una red neuronal 24
Figura 16. Función de activación 25
Figura 17. Captura de pantalla mediante la librería 28
Figura 18. Sistema coordenado mediante Pyautogui 29
Figura 19. Pulsando el botón inicio de Windows mediante coordenadas 30
Figura 20. Variantes de un mismo icono, según escala y forma 31
Figura 21. Pirámide de iteración con 10 escalas 32
Figura 22. Transformación a escala de grises y bordes 34
Figura 23. Patrones identificados con escala variante 35

Figura 24. Algoritmo de localización multiescalar 36
Figura 25. Archivo main donde inicia la ejecución 37
Figura 26. Diagrama de clases 38
Figura 27. Manipulación en tiempo real con la GUI 40
Figura 28. Analogía del flujo de ejecución con una DevOps Pipeline 42
Figura 29. Catálogo de secuencias 43
Figura 30. Detalle de subtarea 44
Figura 31. Detalle de subtarea 46
Figura 32. Objetos estáticos de la clase Recorder 47
Figura 33. Estructura de los objetos responsables de la bitácora 48
Figura 34. Recolección de los registros 49
Figura 35. Recolección de los registros 50

GLOSARIO

ANN	Red Neuronal Artificial
API	Interfaz de Programación de Aplicaciones
Aplicación de consola	Programa que se ejecuta en segundo plano
Backend	Parte de un sistema que se encarga de procesar la información
Backpropagation	Algoritmo que permite ajustar los pesos de una red neuronal
Bot	Programa informático que efectúa automáticamente tareas preprogramadas
Bitácora	Registro de actividades
BSD	Distribución de Software Berkeley
Clase	Grupo de elementos que comparten características
Captcha	Medida de seguridad pregunta-respuesta dirigida a validar la interacción de humanos únicamente
CNN	Red Neuronal Convolucional

Convolucional Neural Network	Red neuronal artificial que imita el comportamiento de las neuronas
Coordenadas	Punto de referencia en un plano
Copyright	Derecho de autor
CV	Visión Computacional
Deep Learning	Termino asociado al aprendizaje mediante múltiples capas
Dendritas	Extensiones de la neurona que reciben señales eléctricas
DevOps	Metodología de desarrollo de software que permite la colaboración entre desarrolladores y operaciones
Diagrama de Clases	Diagrama que permite representar las clases de un sistema
DL	Aprendizaje Profundo
Frontend	Parte de un sistema que permite la interacción con el usuario
Funcion de activación	Función matemática que permite normalizar los valores de salida de una neurona

Github	Plataforma que permite almacenar código fuente
GUI	Interfaz Gráfica de Usuario
Hardware	Dispositivos electrónicos que emiten señales eléctricas.
IA	Inteligencia Artificial
IBM	Máquina de Negocios Internacionales
Inteligencia Artificial	Tecnología que permite a los sistemas informáticos aprender en base a ejemplos
Interfaz Gráfica	Conjunto de elementos visuales que permiten la interacción con un sistema
JSON	Formato de intercambio de datos.
Loss Function	función matemática que permite medir el error de una red neuronal
Machine Learning	Termino asociado a sistemas informáticos que tienen la capacidad de aprender
Matriz	Estructura de datos que almacena valores en forma de tabla
MIT	Instituto Tecnológico de Massachusetts

ML	Aprendizaje Automático
Multi-Scale	Algoritmo que permite identificar patrones en diferentes escalas
Neurona	célula del sistema nervioso que se encarga de transmitir señales eléctricas
Nodo	Elemento de una red neuronal
NumPy	librería de Python orientada al manejo de matrices
Open Source	Software cuyo Código fuente es de libre acceso
OpenCV	Conjunto de algoritmos orientados a identificar patrones en imágenes
Opex	Costos operativos asociados a un proyecto
Peso	Valor que se le asigna a una neurona
Píxel	Unidad mínima de una imagen
PNG	Gráficos de Red Portátiles
Pyautogui	Conjunto de algoritmos orientados a ejecutar comandos
Python	Lenguaje de programación de alto nivel

React	librería de Software orientada al desarrollo de páginas web
Recaptcha	Sistema que utilizan empresas para detectar el tráfico procedente de programas automatizados o bots
Recorder	Herramienta del sistema que permite grabar acciones
Red Neuronal Artificial	Algoritmo de inteligencia artificial que imita el comportamiento de las neuronas
Repositorio	Lugar donde se almacena el Código fuente de un proyecto
RGB	Rojo Verde Azul
RPA	*Robotic Process Automation*
Script	Programa que se ejecuta en tiempo de ejecución
Sinapsis	Unión entre dos neuronas
Sistema Operativo	Conjunto de programas que permiten la interacción con el hardware
Spam	Correo no deseado
Template Matching	Algoritmo de OpenCV que permite identificar patrones en imágenes

True Color	Modelo de color que utiliza 3 canales para representar un color

RESUMEN

El presente trabajo de graduación, integra librerías como *OpenCV*, *Pyautogui* y *React*. Esto para generar una un flujo de pasos que permiten la automatización de múltiples tareas. Permite al usuario crear un catálogo de secuencias que ahorran tiempo y dan una guía visual y en tiempo real de como la visión computacional puede identificar patrones con base en una plantilla.

Al ejecutar la aplicación, es posible obtener una retroalimentación visual respecto a la subtarea que detiene el motor de secuencias, esto quiere decir que existe la probabilidad de que no se encuentre presente la imagen objetivo, en tal caso, se resaltará en color rojo el paso no realizado, si por el contrario la imagen objetivo está presente, entonces el color será verde indicando el éxito de dicho objetivo.

La solución presentada tiene un carácter genérico, esto significa que puede automatizar cualquier proceso siempre y cuando se encuentre dentro de los límites de la interfaz gráfica. Por último, esta solución puede tomar control del sistema operativo gracias al lenguaje de programación Python, el cual permite una programación declarativa de fácil comprensión y mantenimiento.

OBJETIVOS

General

Agilizar los procesos de poca complejidad mediante bots de una sola ejecución que utilizan visión computacional para reducir así los costos operacionales relacionados al soporte técnico dedicado.

Específicos

1. Reducir considerablemente el tiempo que un agente dedica a establecer una configuración en el sistema operativo de un cliente.

2. Generar una biblioteca de iconos pertenecientes a cierto sistema operativo o aplicación con el fin de reducir los tiempos de entrenamiento de la red neuronal.

3. Producir una bitácora de ejecución del script que permita rastrear el éxito o fracaso al aplicar una configuración con el objetivo de proveer retro alimentación al usuario.

4. Publicar un conjunto de Scripts ejecutables con configuraciones comunes con el fin de obtener la mayor retroalimentación posible.

INTRODUCCIÓN

El presente informe detalla una solución basada en visión computacional que puede recrear tareas humanas (en un sistema operativo) y automatizarlas con el fin de reducir cada vez más la intervención personalizada en temas de soporte.

Tareas como instalación de aplicaciones, configuraciones o actualizaciones hoy en día aun dependen de que un agente contacte al usuario para resolver un problema, esta acción eleva considerablemente los costos operativos (OPEX) de las organizaciones.

El trabajo de investigación que aquí se detalla, pretende ahondar en los campos de visión por computadora y producir así un algoritmo que, desde el punto de vista de la organización, brinde soporte con el fin de que el usuario final solo tenga que presionar un botón.

1. VISIÓN COMPUTACIONAL

1.1. El sistema de visión computacional

> La visión computacional es un subcampo de la inteligencia artificial que provee a las computadoras y algunos otros dispositivos, la habilidad de entender tanto imágenes como vídeos incluyendo funcionalidad para procesar y analizar elementos dentro de estos medios; esta es la razón por la cual la visión computacional pertenece al campo de la inteligencia artificial, está fuertemente relacionada con los algoritmos de *Machine Learning* y *Deep Learning*. (Francois, 2021, p. 2)

La visión computacional también integra otros campos y otras tecnologías cómo:

- Matemáticas
- Física
- Tecnología de sensores
- Hardware de gráficos computacionales

Debido a que la visión computacional está íntimamente ligada con conceptos de Inteligencia Artificia, *Machine Learning* y *Deep Learning* se

utilizaran los acrónimos (CV), (AI), (ML) y (DL) respectivamente, para hacer referencia a estos temas mediante su nombre en idioma inglés.

El objetivo principal de CV está basado en la percepción visual: esta es la ciencia que capta y entiende el mundo natural atreves de imágenes y videos, utilizando estos para construir un modelo virtual del mundo físico por medio del cual los sistemas de AI y otros sistemas puedan tomar las acciones apropiadas.

1.2. Percepción visual

Este tema está relacionado analizar aquellos medios que proporcionen una entrada visual de información y observar patrones, así como objetos dentro de esta información.

> Cuando se habla de percepción significa que no solamente se captura y se almacena el ambiente en un formato digital, sino que se buscan construir sistemas que de hecho puedan entender ese ambiente a través esos medios visuales de captura de información. (Mohamed, 2020, p. 4)

1.3. Sistema de visión humano vs. sistema de visión computacional

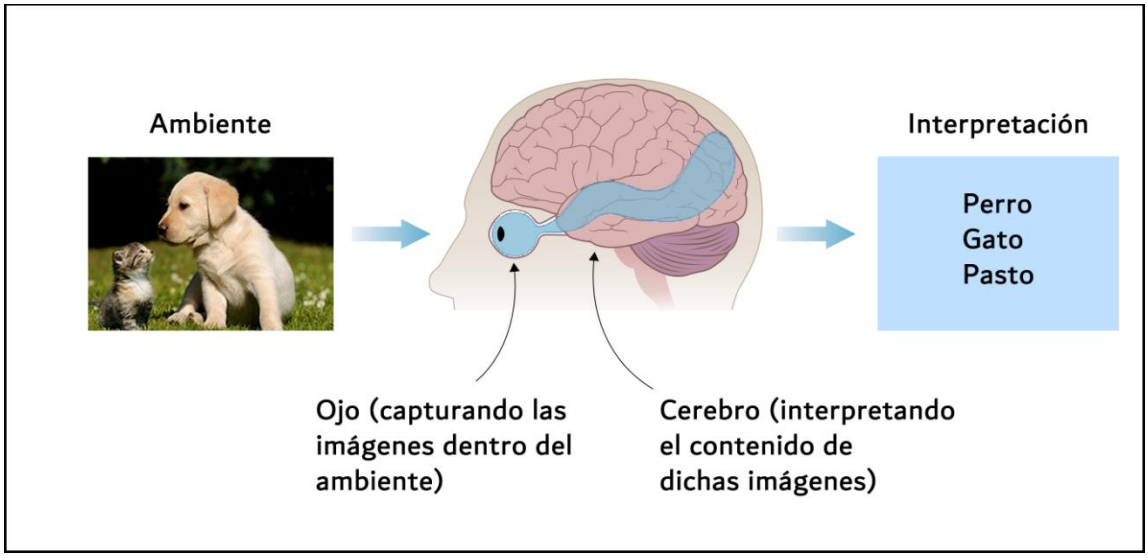
La mayoría de los inventos creados por los seres humanos han sido una representación o una copia de los sistemas naturales, en este caso los sistemas de visión computacional son muy similares al sistema de visión humano que a su vez es muy similar al sistema de visión de otras criaturas como los animales o cualquier otro organismo viviente.

> Este sistema está fuertemente apoyado en un dispositivo utilizado como un sensor óptico qué es el ojo, éste captura la luz proveniente del entorno y la convierte a señales eléctricas que viajan a nuestro cerebro y luego son interpretadas por medio de este para luego tomar una decisión. (Mohamed, 2020, p. 5)

A continuación, la Figura 1 muestra el proceso mediante un esquema muy sencillo, pero bastante práctico para ilustrar el funcionamiento del sistema de visión de los seres humanos.

Figura 1.

Sistema de visión humano

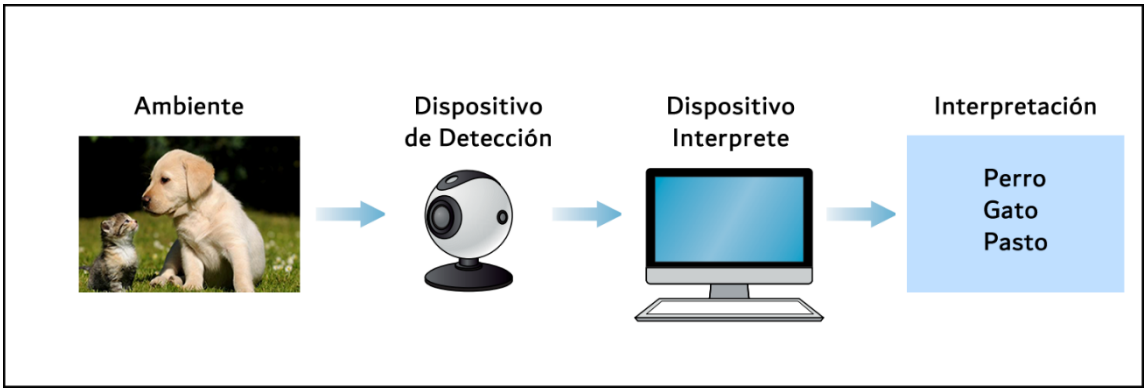


Nota. Muestra del proceso mediante un esquema muy sencillo, pero bastante práctico para ilustrar el funcionamiento del sistema de visión de los seres humanos. Obtenido de E. Mohamed (2020). *Deep Learning for Vision Systems*. (p. 5.) Editorial Manning Publicaciones Co.

“Por otro lado, los sistemas de CV están inspirados en los sistemas naturales, estos sistemas copian el funcionamiento mediante maquinas que replican la habilidad de captar y analizar el ambiente” (Mohamed, 2020, p. 6). Para imitar esta funcionalidad CV también basa su funcionamiento en dos dispositivos fundamentales: el primero para imitar el ojo humano (Dispositivo de Detección) capturando la luz captura de luz y el otro como un algoritmo que pueda interpretar esas imágenes (Dispositivo Interprete) La figura 2 muestra este esquema y es bastante parecido al esquema humano.

Figura 2.

Sistema de visión computacional



Nota. Muestra del esquema parecido al esquema humano. Obtenido de E. Mohamed (2020). *Deep Learning for Vision Systems*. (p. 7.) Editorial Manning Publicaciones Co.

El elemento fundamental de la visión computacional y el tema principal de esta tesis es quizá el dispositivo intérprete, más específicamente el algoritmo que permite tomar aquellos archivos como imágenes o vídeos y analizarlos con el fin de generar una interpretación. Un intérprete es el cerebro de CV y su papel en tomar la información generada por el dispositivo de detección e identificar patrones y objetos. Los algoritmos que permiten realizar esta tarea están

basados en cómo funciona el cerebro humano, de hecho, se podría decir que se ha utilizado la ingeniería inversa para descubrir como el cerebro clasifica las imágenes. En concreto mediante el estudio del sistema nervioso central nace el primer concepto relacionado a la Inteligencia Artificial: La Red Neuronal Artificial o ANN.

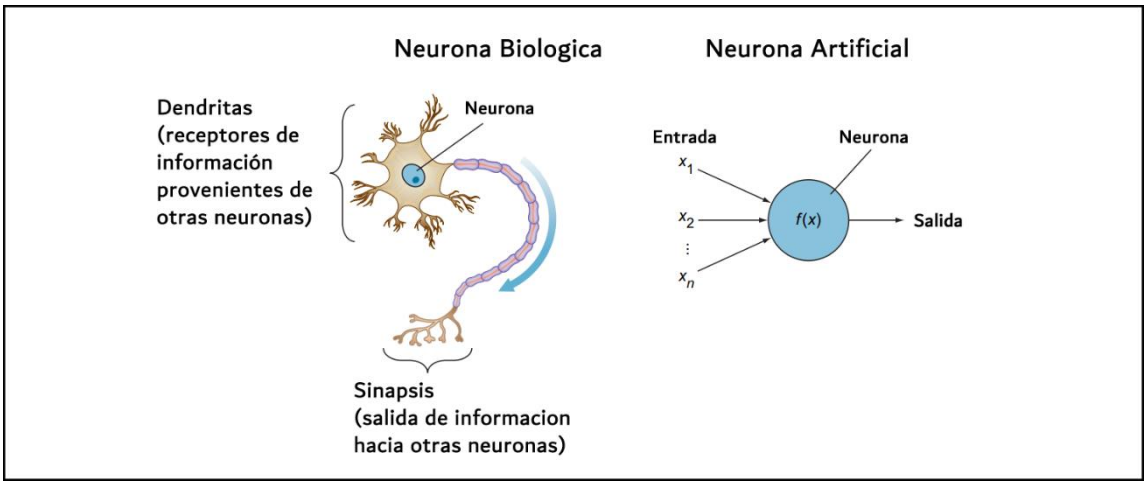
Cuando estos conceptos fueron tomando forma nació quizá el elemento más importante de este paradigma:

> El Aprendizaje. Este puede definirse como la identificación de patrones o la creación de reglas que toman un conjunto de ejemplos para poder realizar predicciones. Cuando una red neuronal consta de muchas capas de análisis, a esta arquitectura se le conoce como Aprendizaje Profundo. (Mohamed, 2020, p. 8)

A continuación, las Figura 3 y 4 muestran cuál es la analogía biológica entre una neurona y una neurona artificial, así como la estructura de una red neuronal artificial.

Figura 3.

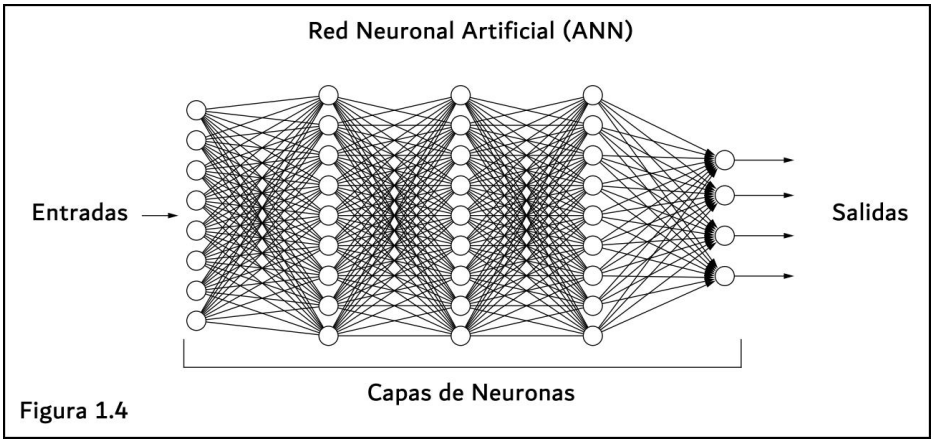
Similitud entre una neurona biológica y una artificial



Nota. Gráfica de la similitud entre una neurona biológica y una artificial. Obtenido de E. Mohamed (2020). *Deep Learning for Vision Systems*. (p. 9.) Editorial Manning Publicaciones Co.

Figura 4.

Red neuronal artificial



Nota. Gráfica de la similitud entre una neurona biológica y una artificial. Obtenido de E. Mohamed (2020). *Deep Learning for Vision Systems*. (p. 9.) Editorial Manning Publicaciones Co.

La razón por la cual en este punto se citan conceptos como las redes neuronales o el aprendizaje profundo es para poder dar un contexto de cómo la visión computacional está íntimamente basada en las herramientas de la inteligencia artificial. estos temas son abordados con mayor profundidad en el siguiente capitulo.

1.4. Fundamentos de la clasificación de imágenes

La tarea de clasificar imágenes está basada en asignar una etiqueta basada en un conjunto de categorías predefinidas. Para este trabajo las Redes Neuronales Convolucionales (CNN) son particularmente eficientes ya que permiten superar obstáculos como la escala o la rotación de un objeto en un ambiente.

“Un algoritmo clasificador de imágenes toma una imagen como entrada y produce una etiqueta que permite identificar dicha imagen. Se conoce como Clase a la categoría de salida de los datos” (Mohamed, 2020, p. 10). Las aplicaciones de este campo son muchas y pueden listarse las siguientes:

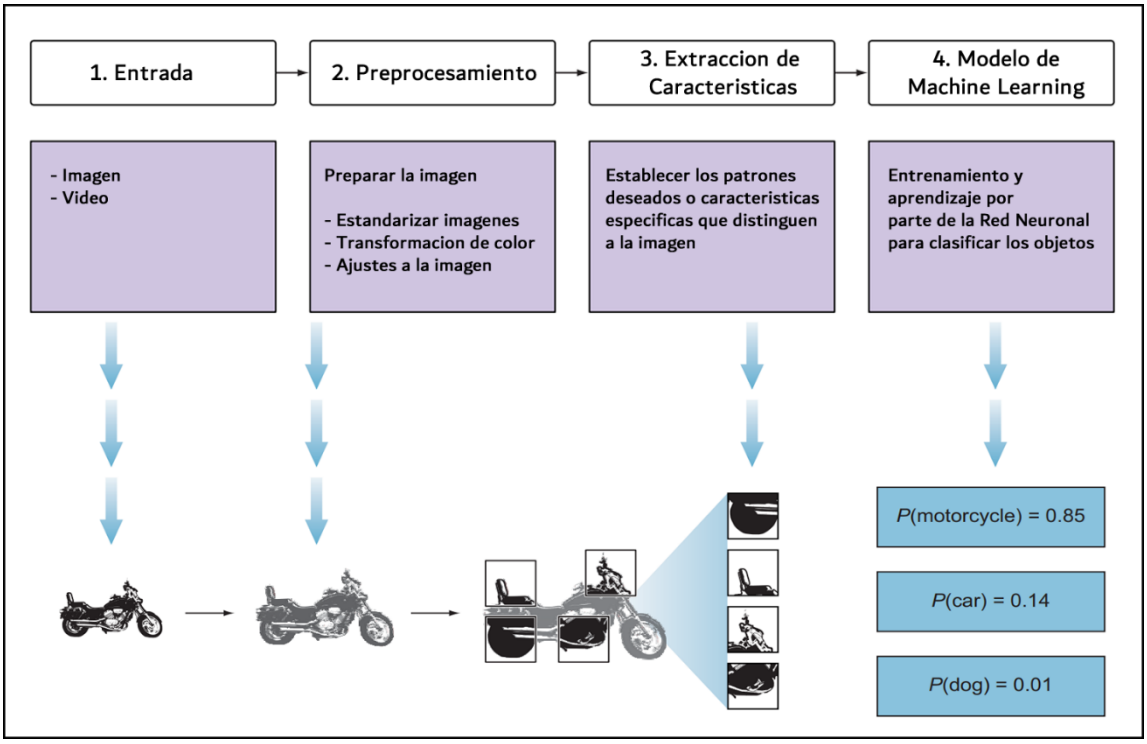
- Diagnóstico de cáncer de pulmón
- Reconocimiento de señales de tránsito
- Detección de objetos y su localización
- Reconocimiento facial

1.5. Flujo de la clasificación

A continuación, la Figura 5 muestra el proceso mediante el cual se produce la clasificación de una imagen, desde su entrada hasta su salida como objeto etiquetado perteneciente a cierta categoría o clase. Anteriormente se afirmó que existen dos elementos esenciales para iniciar este proceso, que son el dispositivo de detección y el intérprete, en el flujo mostrado a continuación se muestran más detalles sobre su interacción:

Figura 5.

Flujo de clasificación de imágenes



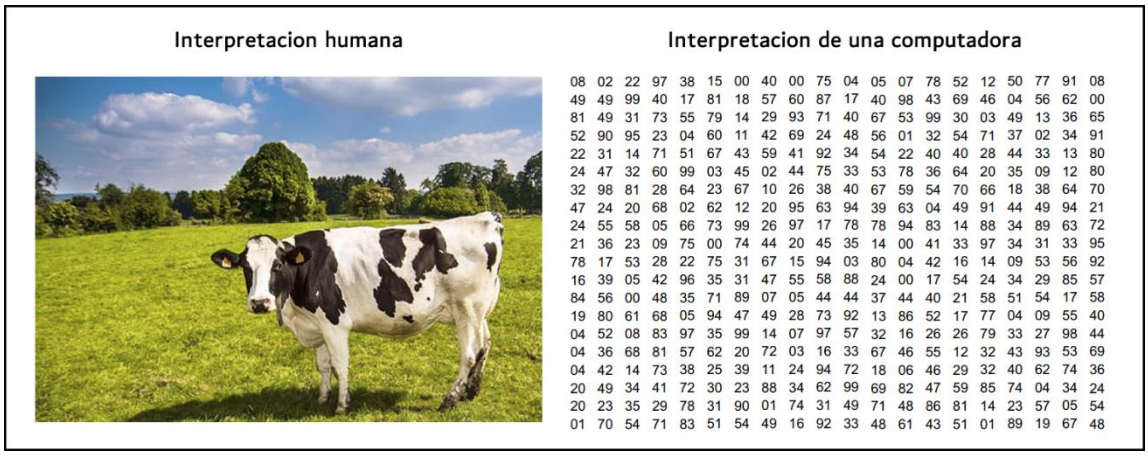
Nota. Gráfica de proceso mediante el cual se produce la clasificación de una imagen, desde su entrada hasta su salida. Obtenido de E. Mohamed (2020). *Deep Learning for Vision Systems*. (p. 18.) Editorial Manning Publicaciones Co.

1.6. ¿Cómo observa una computadora una imagen?

Una computadora percibe una imagen de una forma matemática y más específicamente como una matriz de dos dimensiones con valores numéricos asociados a cada píxel, donde cada valor representa la intensidad en el espectro de color. La figura 6 muestra esta comparativa entre el visón humano y lo que una computadora interpreta como imagen. (Mohamed, 2020, p. 21)

Figura 6.

Representación computacional de una imagen



Nota. Gráfica comparativa entre el visón humano y lo que una computadora interpreta como imagen. Obtenido de E. Mohamed (2020). *Deep Learning for Vision Systems*. (p. 21.) Editorial Manning Publicaciones Co.

1.7. Imagen en escala de grises

La matriz mostrada en la figura 6 representa una imagen de 24 x 24 pixeles, esto quiere decir un total de 576 pixeles. Cada valor asociado a un píxel representa la intensidad de brillo, donde 0 corresponde al color negro y 255 corresponde al color blanco. Para un dispositivo interprete es mucho más sencillo trabajar con imágenes en escala de grises. Convertir una imagen a color a una en escala de grises es una estrategia utilizada con el fin de reducir la complejidad computacional.

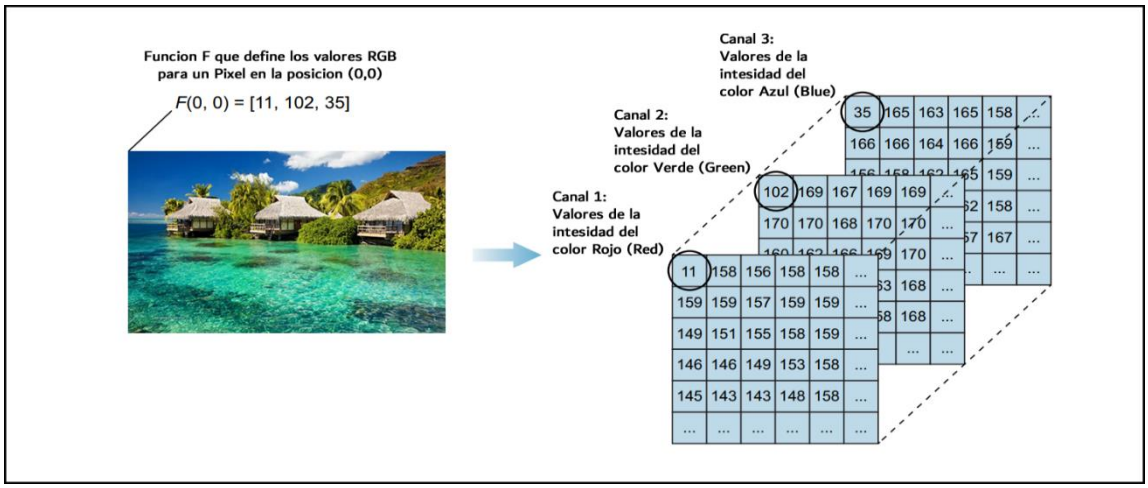
1.8. Imagen a color

A diferencia de la escala de grises, una imagen a color está compuesta por tres matrices superpuestas una sobre la otra. Cada matriz recibe el nombre de canal y utiliza el estándar RGB que corresponde al acrónimo *Red Green Blue*. El solapamiento de estos valores genera en el ojo humano la ilusión de tener una imagen a color. Mediante este concepto se define también la función F(x,y) que define los valores RGB para determinado pixel.

> La combinación de los valores RGB permiten obtener cualquier color deseado dentro del espectro de los 24 bits por píxel, esto significa 16,777,216 colores posibles para cada píxel, a esta configuración se le conoce como *True Color* ya que es aproximadamente el rango máximo de colores que el ser humano puede distinguir. La figura 7 muestra más a detalle este concepto. (Mohamed, 2020, p. 21)

Figura 7.

Representación de una imagen a color por medio de canales



Nota. Gráfica representativa de una imagen a color por medio de canales. Obtenido de E. Mohamed (2020). *Deep Learning for Vision Systems*. (p. 21.) Editorial Manning Publicaciones Co.

1.9. Introducción a OpenCV

OpenCV es una librería basada en C/C++ y programación altamente optimizada que brinda a una computadora capacidades de análisis de imágenes y video en tiempo real. Esta librería está registrada mediante una licencia BSD la cual permite que su uso sea totalmente gratuito tanto de forma académica como comercial. Es quizá la librería más reconocida y ampliamente utilizada en el campo de la industria y la investigación. A continuación, se detallan algunos aspectos del porque su gran popularidad:

- Librería Open Source de visión computacional bajo licencia BSD.

- Posee más de 2,500 algoritmos optimizados.

- Está preparada para ser integrada con otros *frameworks* de *machine learning* y *deep learning*.

- Multiplataforma.

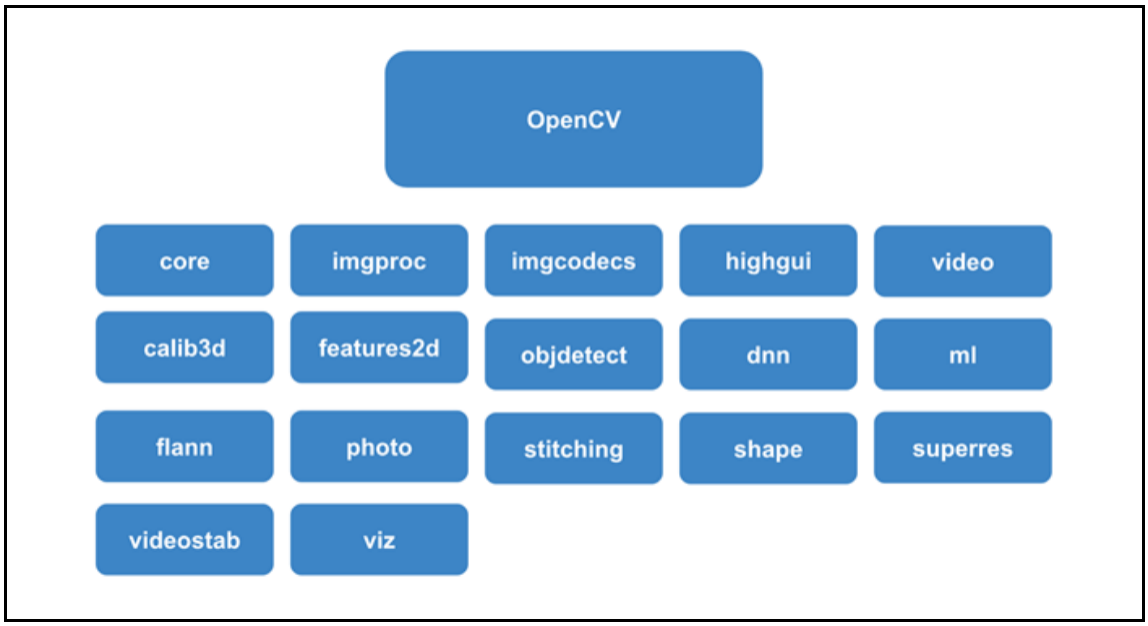
- Poséela una gran comunidad donde destacan entidades como el MIT y Standford que dan mantenimiento a su código.

- Grandes compañías como Google, Microsoft e IBM ya la implementan de forma empresarial.

La versión 2 de OpenCV está dividida en diferentes módulos. Este nuevo enfoque permite utilizar cada módulo por separado para resolver problemas específicos asociados a la visión computacional. La Figura 8 muestra este esquema que luego será descrito brevemente.

Figura 8.

Separación modular de OpenCV



Nota. Ejemplo de esquema de separación modular de *OpenCV*. Obtenido de M. Valliappa, y R. Gillard (2021). *Practical Machine Learning for Computer Vision*. (p.18.) Editorial O'Reilly Media, Inc.

- Core: estructuras de datos y funciones básicas que serán utilizadas por otros módulos.

- Imgproc: módulo de procesamiento de imágenes que incluye filtros, transformaciones geométricas, histogramas entre otros.

- Imgcodecs: lectura de y escritura de imágenes.

- Videoio: interfaz de captura de video.

- Highgui: interfaz que brinda controles para mostrar imágenes, así como capturar eventos del ratón y del teclado.

- Video: módulo de análisis de video que incluye algoritmos de seguimiento de objetos en movimiento

- Objdetect: módulo de detección de objetos mediante la definición de clases como: caras, ojos, personas, automóviles entre otros.

- Dnn: módulo que proporciona una interfaz para generar redes neuronales profundas

- Ml: módulo *machine Learning* que continue un conjunto de librerías que pueden ser usadas para propósitos de clasificación y regresión

- Flann: módulo de búsqueda por agrupamiento y búsqueda en espacios multidimensionales por medio de algoritmos *fast, nearest, neighbor* y *search.*

- Photo: módulo dedicado a la fotografía computacional

- Shape: módulo dedicado a analizar escalas y similitud entre una imagen y un entorno, también puede es aplicable a realizar comparaciones entre dos objetos.

- Superres: módulo de manipulación de la resolución que contiene un conjunto de algoritmos que puede mejorar la resolución de una imagen.

- Videostab: módulo dedicado a la estabilización de video.

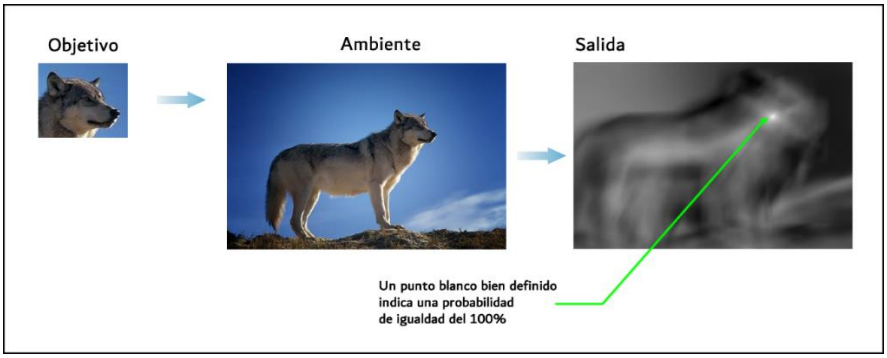
1.10. Template *Matching*

OpenCV utiliza este método para encontrar la localización de una imagen objetivo dentro de un ambiente o imagen más grande. Para esta tarea en específico se utiliza la función cv2.matchTemplate. Este método utiliza el desplazamiento de la imagen objetivo atreves de regiones superpuesta sobre la imagen para encontrar cierta similitud. Es importante destacar que este análisis de barrido realiza múltiples comparaciones basadas en algoritmos de DL que le permiten ajustar la comparación mediante un valor probabilístico de éxito.

Como se ha mostrado en apartados anteriores de este capítulo, el primer paso consiste en transformar la imagen a escala de grises para reducir la complejidad computacional. La salida de este método será una imagen en escala de grises que identificará mediante un pequeño circulo blanco el área donde puede estar la imagen objetivo. La precisión o difusión de este círculo o punto blanco indica que tan alta es la probabilidad de similitud entre la región y el objetivo. La imagen continuación muestra el funcionamiento de este método.

Figura 9.

Búsqueda de imagen objetivo



Nota. Funcionamiento del método *OpenCV*. Elaboración propia, realizado con Paint.

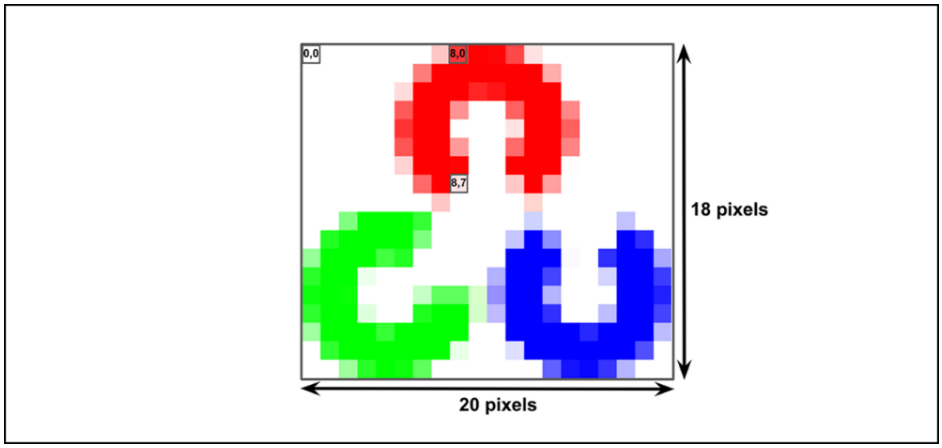
Para los objetivos de esta tesis y para el campo en general de RPA, *Template Matching* es ideal, ya que una aplicación o un sistema operativo no varía sus iconos y sigue reglas de estandarización. Quizá se generen discrepancias respecto al color o al tamaño, para lo cual *OpenCV* tiene diferentes configuraciones y estrategias que permiten hacer posible esa correlación.

1.11. Sistema coordenado de *OpenCV*

OpenCV brinda una forma bastante practica de indexar cada píxel perteneciente a una imagen. Este sistema de coordenadas (x,y) tiene su origen (0,0) en la esquina superior izquierda. El concepto ubicar un píxel y relacionarlo con un plano coordenado es muy importante ya que casi todas las acciones que conciernen a la detección de objetos producen su localización mediante estas coordenadas. La imagen a continuación muestra el logo de *OpenCV* acotado por estas coordenadas.

Figura 10.

Coordenadas en OpenCV



Nota. Logo de *OpenCV* acotado por coordenadas. Elaboración propia, realizado con Paint.

1.12. Dibujo de figuras mediante *OpenCV*

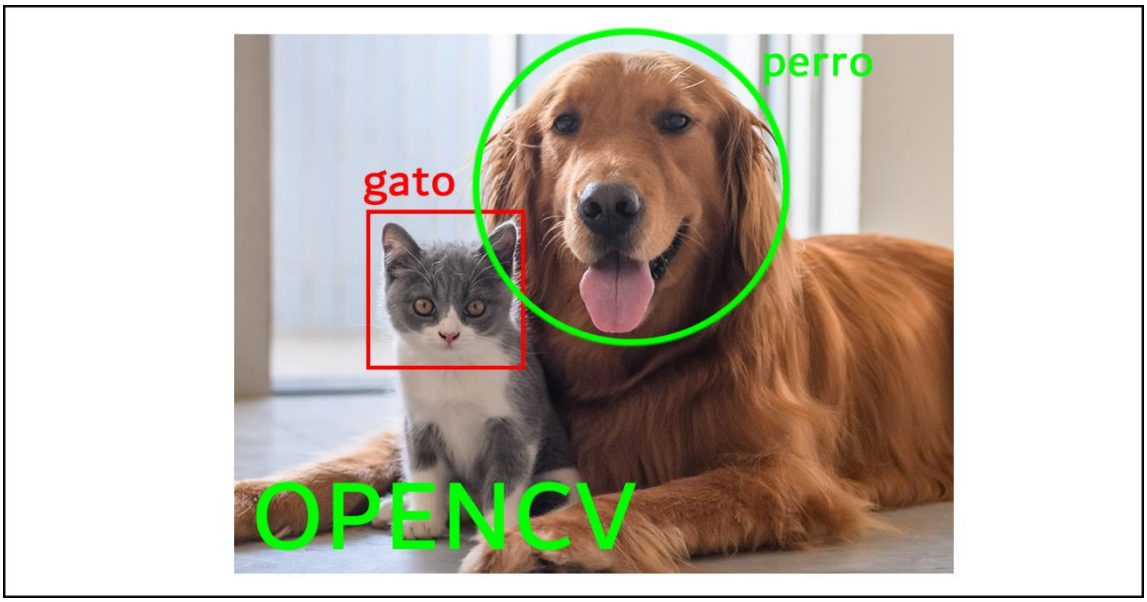
Ahora que ya se han definido conceptos como la composición de imágenes y el sistema coordenado de *OpenCV* es posible hablar sobre la manipulación directa de los pixeles, esto con el fin de dibujar o graficar recuadros de colores llamativos una vez los objetos deseados sean localizados en la imagen ambiente. La librería nos ofrece las siguientes funciones para dibujar la figura que se desee una vez se haya identificado su posición:

- cv2.line()
- cv2.circle()
- cv2.rectangle()
- cv2.ellipse()
- cv2.putText()

La imagen mostrada a continuación muestra la interacción de estas funciones sobre una imagen, nuevamente se hace énfasis en la importancia de encontrar la posición (x, y) o punto central donde se encontró la posible correlación para poder dibujar o etiquetar los objetos que se deseen.

Figura 11.

Etiquetas en objetos



Nota. Interacción de las funciones sobre una imagen. Elaboración propia, realizado con Paint.

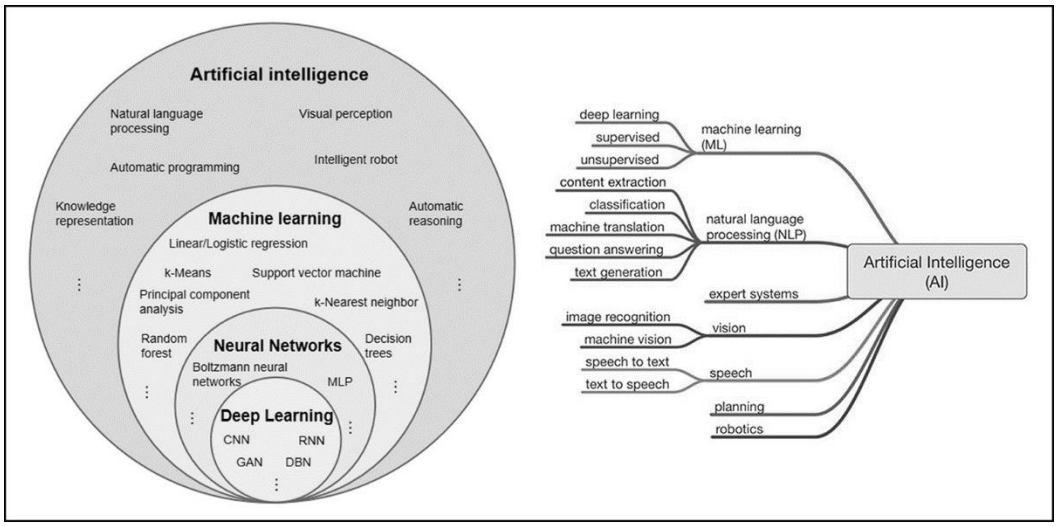
2. REDES NEURONALES

2.1. Introducción a la inteligencia artificial

De forma práctica se puede definir a la inteligencia artificial como la automatización de aquellas tareas de carácter repetitivo que regularmente están asociadas a los humanos. A la inteligencia artificial se suelen asociar los conceptos de *machine learning* y *deep learning*, sin embargo, estos son solo subcampos de una diciplina mucho más grande. La Figura 13 muestra esta descripción e ilustra otros campos involucrados:

Figura 12.

Subcampos de la inteligencia artificial



Nota. Gráfica de subcampos de inteligencia artificial involucrados. Elaboración propia, realizado con Paint.

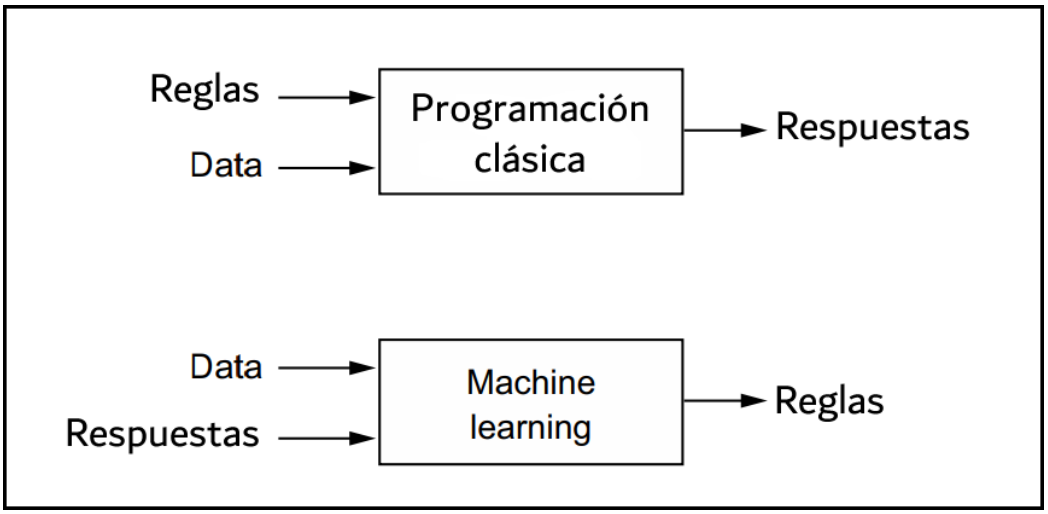
A su vez, se puede integrar a la inteligencia artificial como una rama de la Ciencia de los Datos, encargada de simular e imitar el comportamiento humano atreves de una computadora.

2.2. *Machine learning*

> Este es el concepto que permite asociar a la inteligencia artificial con una de las características más potentes y poderosas del ser humano como lo es el aprendizaje y la deducción. La programación clásica o imperativa definía claramente un flujo de reglas, las cuales para un conjunto de datos X producían una salida F(X), la cual era totalmente predecible, una característica de los algoritmos determinísticos, sin embargo, *Machine Learning* cambia esta perspectiva y permite tomar un conjunto de datos X y un conjunto de resultados Y para inferir o deducir las reglas que modelan cierto comportamiento. (Francois, 2021, p. 3)

Figura 13.

Programación tradicional vs. machine learning



Nota. Gráfica de programación tradicional vs. *machine learning*. Elaboración propia realizado con Paint.

Machine learning comenzó a florecer a partir de 1990, por esa época las inferencias se hicieron más precisas gracias a los avances en el hardware y en bases de datos cada vez más grandes.

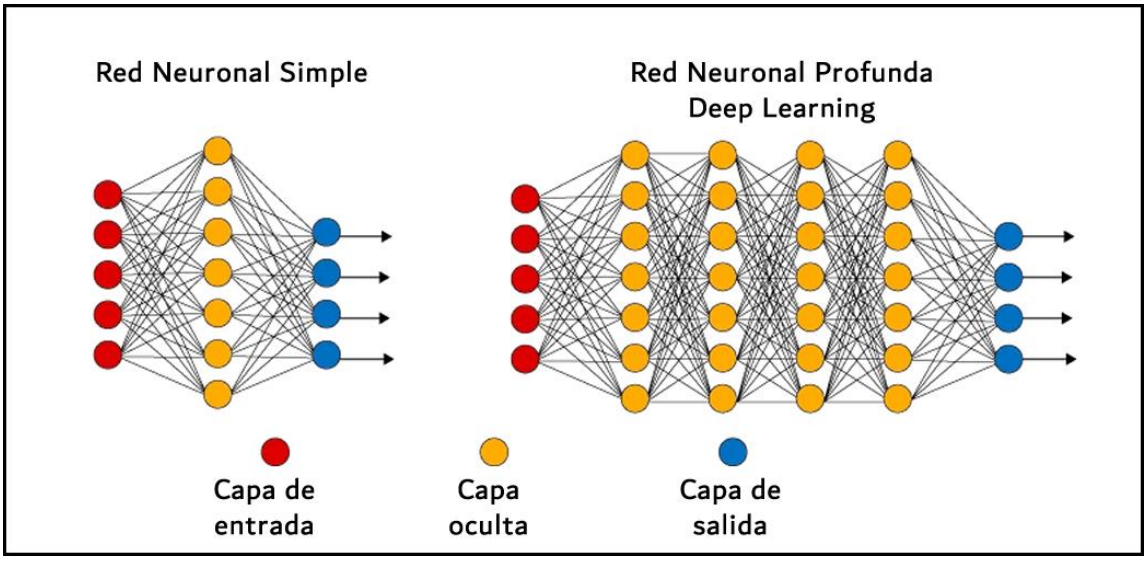
2.3. *Deep learning*

"El punto más fuerte detrás de este enfoque es el hecho que *Machine Learning* utiliza enormes conjuntos de datos, tan grandes que el análisis estadístico clásico puede ser impráctico para este propósito" (Francois, 2021, p. 7). Como resultado se obtiene un método ideal para analizar grandes volúmenes de información que mediante funciones matemáticas permiten dar paso a la deducción o inferencia.

Habiendo definido lo anterior, es posible introducir el concepto de *deep learning* ya que este es un subcampo especifico de *machine learning* que permite asociar una representación para cada dato X del conjunto masivo de información. *deep learning* no solo aporta un significado a la información que procesa, sino que su nombre Deep deriva de la profundidad o número grande de N de capas por las cuales la información pasará en su camino para obtener un mejor acercamiento a los datos deseados. La Figura 14 muestra las N capas de un esquema al que puede llamársele *deep learning*

Figura 14.

Significado de profundidad en deep learning



Nota. Esquema al que puede llamársele *deep learning.* Elaboración propia, realizado con Paint.

2.4. Redes neuronales

En el primer capítulo se expuso que existen ciertas similitudes entre una neurona humana y una neurona artificial, en este capítulo se profundizara más a

detalle acerca de los elementos fundamentales que definen a un conjunto de neuronas conectadas, el tipo de relación y las bases matemáticas que existen detrás del concepto de *deep learning*. Los siguientes elementos esenciales para el trabajo de clasificación en una red neuronal:

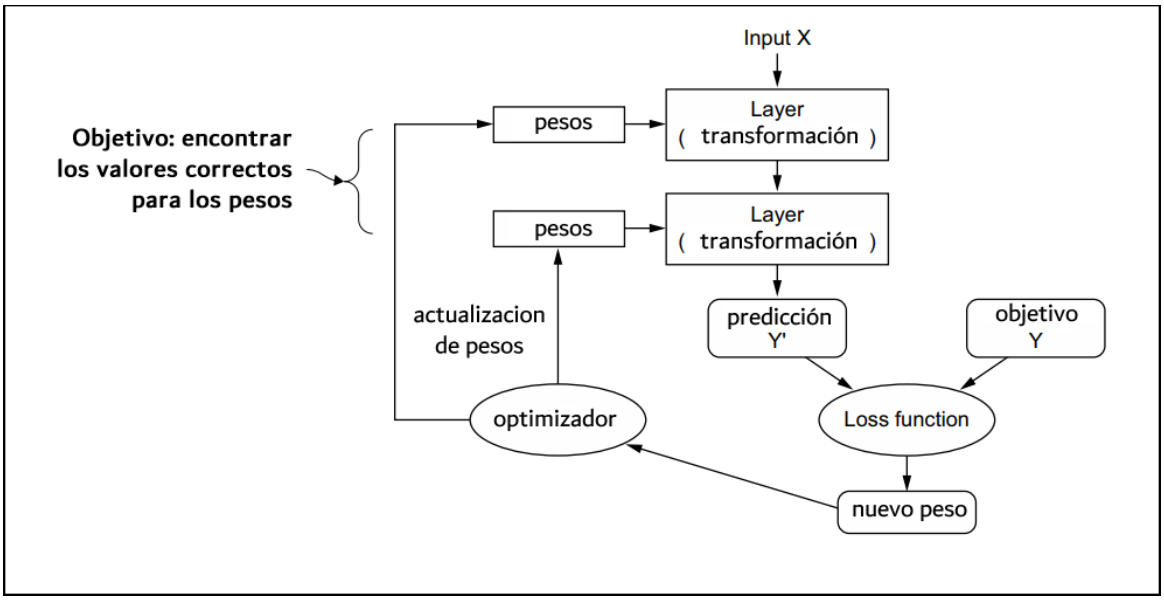
- Capa de entrada
- Pesos o parámetros de las capas
- Capas ocultas
- Capa de salida
- Función de costo
- Función de activación

> El funcionamiento de una red neuronal se basa en un conjunto de datos conocido como capa de entrada, luego son redirigidos a las capas ocultas las cuales realizan transformaciones a dichos datos cada vez que pasan por ellas. Como lo ilustra la figura 15 la salida de una neurona es la entrada de otra, en este contexto, el peso representa que tan lejos se encuentra el resultado obtenido del deseado, este peso debe de recalibrarse cada o ajustarse cada vez que una capa oculta realiza una trasformación al dato. (Francois, 2021, p. 9)

La Figura 15 muestra este flujo de información a medida que es categorizado por medio de la red y la Figura 16 muestra la función de activación que permite generar la transformación requerida envase a la actualización del nuevo peso.

Figura 15.

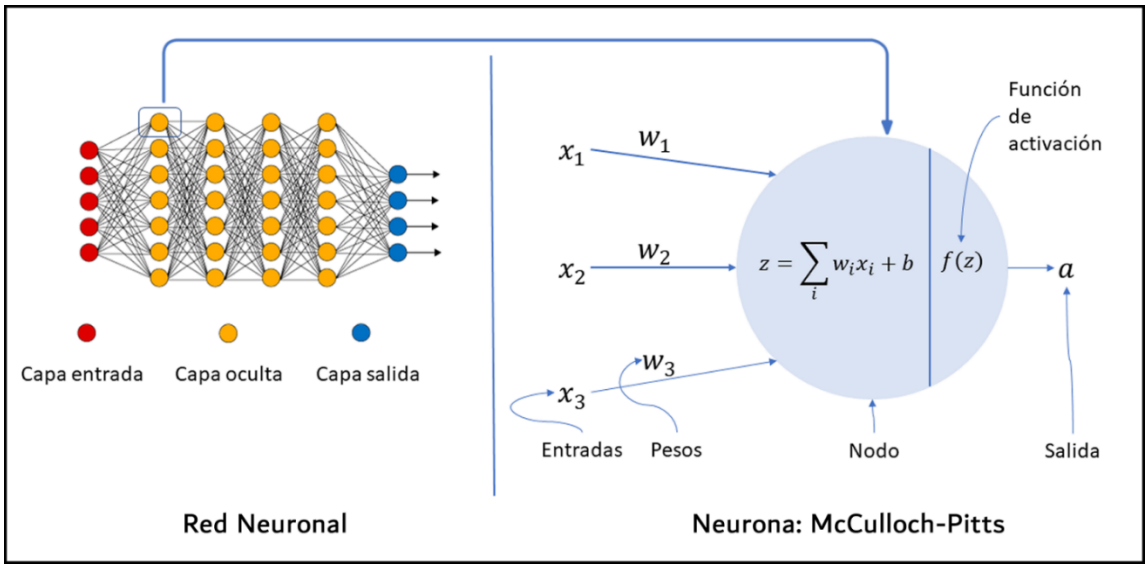
Flujo de trabajo de una red neuronal



Nota. Gráfica de flujo de información. Obtenido de C. Francois (2021). *Deep Learning with Python*. (p. 10.) Editorial Manning Publicaciones Co.

Figura 16.

Función de activación



Nota. Flujo de información a medida que es categorizado por medio de la red. Obtenido de J. Ponce (2022). *Blog personal sobre inteligencia artificial*. (https://jahazielponce.com/blog/), consultado el 25 de febrero de 2023. De dominio público.

3. CONTROL DE GUI

3.1. Uso de la librería *Pyautogui*

> Por excelencia, Python es el lenguaje utilizado en proyectos de inteligencia artificial y scripting de automatización. Para este último propósito se utilizará la Librearía *Pyautogui* que integra un set completo de herramientas para controlar eventos del sistema operativo tales como la interfaz gráfica, teclado y ratón. (Sweigart, 2020, p. 473)

Posee una API bastante sencilla y fácil de utilizar, es multi plataforma y posee las siguientes características además de las ya mencionadas:

- Identificación de controles
- Integración con *OpenCV*
- Creación y lectura de archivos
- Envió de correos electrónicos
- Control promedio de la interfaz grafica
- Tomar captura de pantallas

- Despliegue de alertas

3.2. Captura de pantalla: análisis e identificación de coordenadas

Realizar la captura de pantalla es posible mediante el comando *screenshot* que almacena el objeto en una variable que luego puede utilizarse para guardar dicha captura en una ubicación especifica. Esta captura de pantalla será crucial para los objetivos de la aplicación que se desarrolla, ya que en este se localizaran las coordenadas de los controles a manipular. La Figura 17 muestra como la librería toma una captura y la guarda como archivo png.

Figura 17.

Captura de pantalla mediante la librería

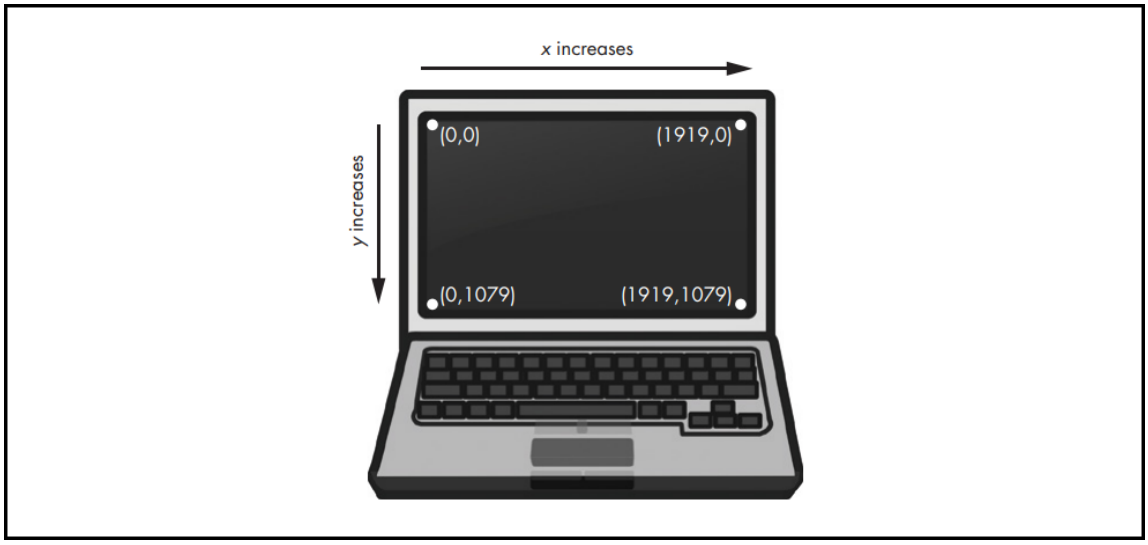
Nota. Captura de pantalla. Elaboración propia.

Otra funcionalidad muy importante es la captura de la resolución de pantalla, la resolución es diferente para cada equipo de cómputo y brinda las bases para establecer los límites de acción ya que acciones fuera de las coordenadas máximas de resolución resultaran en un error irrecuperable de la

aplicación. La coordenada origen inicia en el punto (0,0) que está ubicado en la esquina superior izquierda. La Figura 18 muestra esta descripción.

Figura 18.

Sistema coordenado mediante Pyautogui



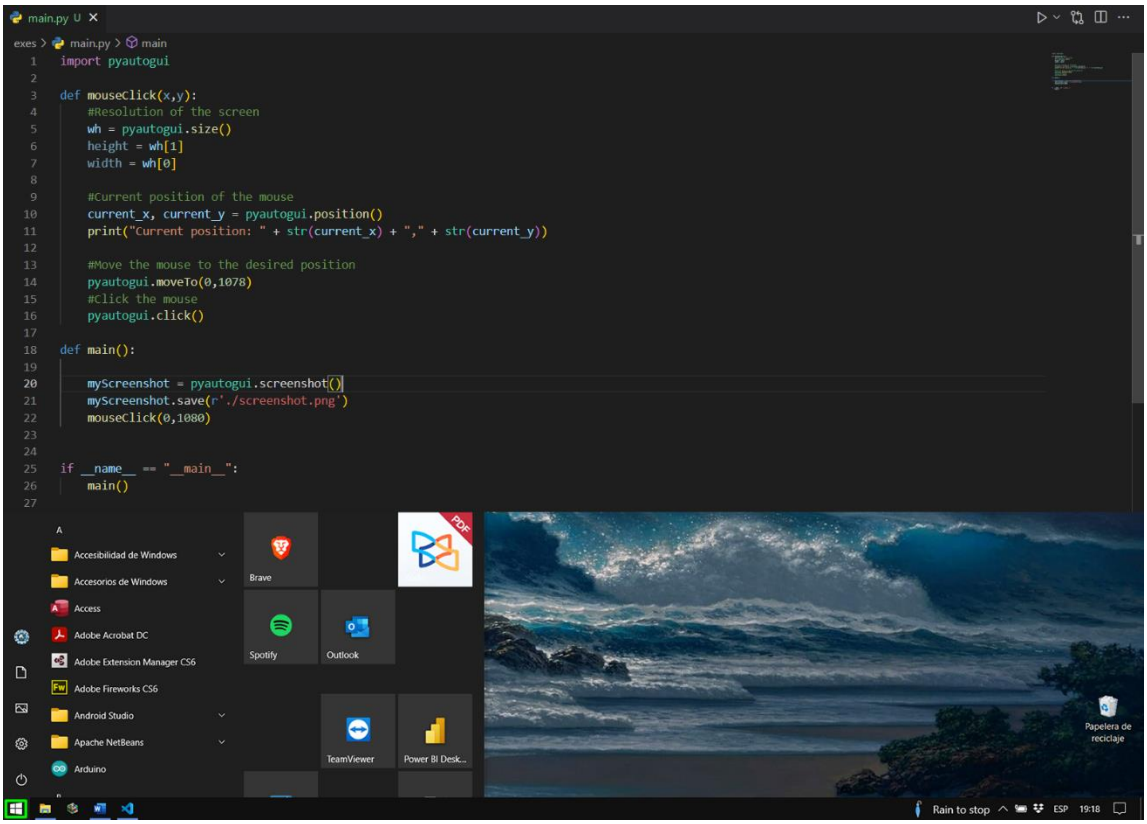
Nota. Ejemplo de captura de la resolución de pantalla. Obtenido de A. Sweigart (2020). *Automate the Boring Stuff with Python*. (p. 22.) Editorial No Starch Press.

3.3. Manipulación de ventanas y controles

"Cuando las coordenadas están establecidas y *OpenCV* a identificado la posición especifica de los controles, entonces es posible programar las interacciones con botones y cajas de texto" (Sweigart, 2020, p. 478). *Pyautogui* permite programar pulsaciones mediante una coordenada especifica, así mismo tiene retrasos que permiten darle a la aplicación o sistema operativo el tiempo suficiente para procesar la petición, la Figura 19 muestra el código necesario para mover el cursor y realizar una pulsación.

Figura 19.

Pulsando el botón inicio de Windows mediante coordenadas



Nota. Captura de pantalla. Elaboración propia.

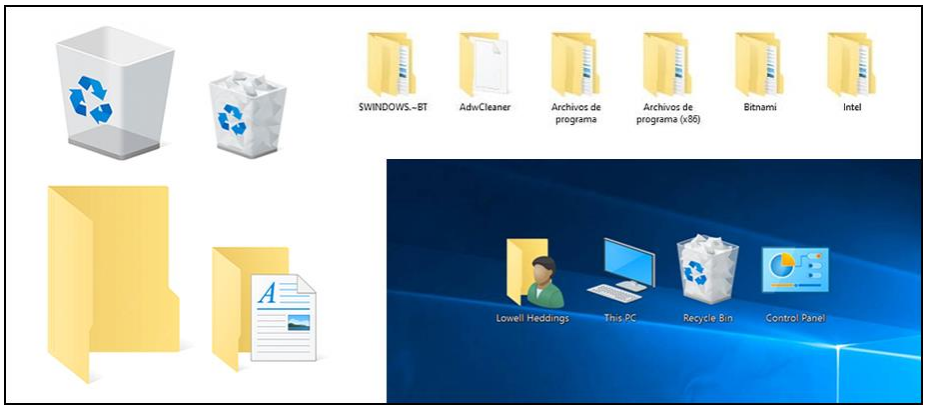
4. INTEGRACIÓN

4.1. Identificación de imágenes objetivo

El primer paso en la construcción de un sistema de reconocimiento de imágenes objetivo mediante *OpenCV*, es definir la estrategia de búsqueda. Como se definió en capítulos anteriores, se utilizará la técnica de *Template Matching* será optimizada para identificar variaciones en los temas o colores de cada sistema operativo y en la escala o tamaño. El objetivo es determinar la posición X, Y de cada objetivo sin importar sus variantes y así obtener la mayor precisión posible. La Figura 20 muestra como los iconos o las imágenes objetivo pueden variar según cada sistema operativo e ilustra el reto que representa identificar un patrón en estas variantes.

Figura 20.

Variantes de un mismo icono, según escala y forma



Nota. iconos o imágenes objetivo pueden variar según cada sistema operativo. Elaboración propia, realizado con Paint.

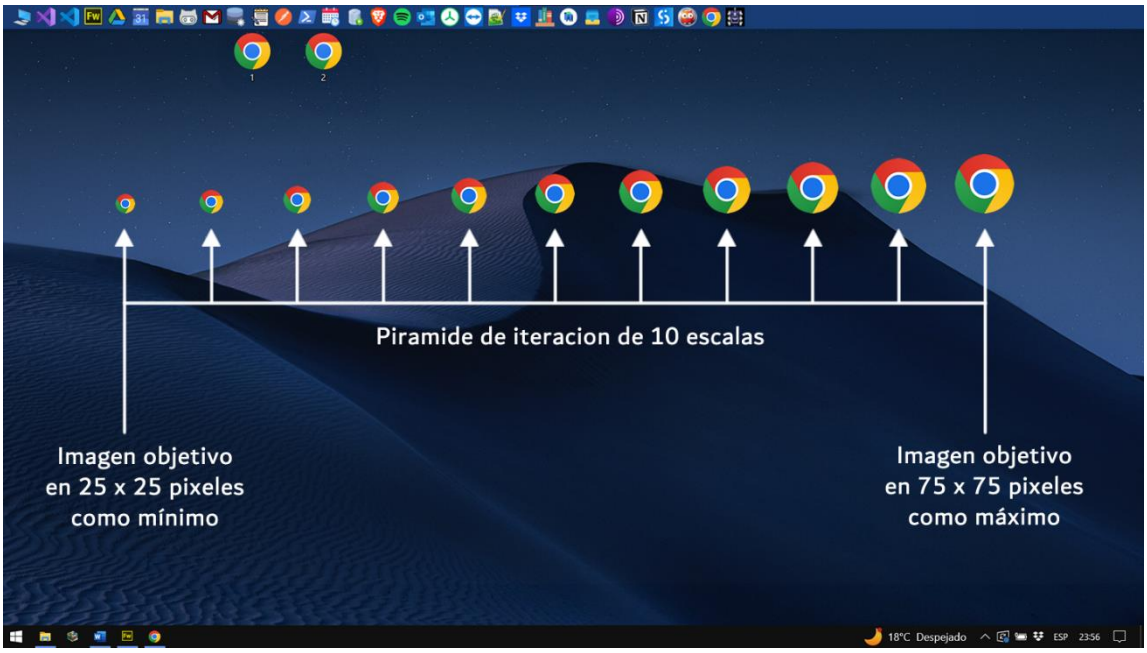
Como se ha mostrado en la figura, estas variaciones añaden complejidad al momento de identificar un patrón; con el fin de reducir esta complejidad, es importante dividir y crear módulos que se encarguen de cada aspecto que pueda cambiar, el primero de estos aspectos a trabajar es la variación en el tamaño.

En este sentido, la estrategia se basa en realizar múltiples barridos de la imagen objetivo sobre el entorno. A esta técnica se le conoce como *Multi-scale Template Matching.*

La Figura 21 muestra como el análisis se realiza reduciendo en cierta proporción la imagen objetivo para encajar con la imagen ambiente.

Figura 21.

Pirámide de iteración con 10 escalas



Nota. Captura de pantalla sobre el análisis se realiza reduciendo en cierta proporción la imagen. Elaboración propia.

Uno de los problemas de *Template Matching* como detector de objetos es que tiene la desventaja de funcionar solo en casos muy específicos, principalmente cuando la plantilla coincide a la perfección con el objeto de que se busca en la imagen ambiente; de tal manera que, si el objeto aparece en menor o mayor tamaño en la imagen, a pesar de ser exactamente igual a la imagen objetivo, con este algoritmo tradicional no será posible detectar una similitud, sin embargo con un pequeño ajuste es posible eliminar esta limitante.

Mediante la creación de una pirámide de imágenes o dicho de una manera más sencilla: buscar la imagen objetivo sobre la imagen ambiente a diferentes escalas.

De ser posible primero se hace una búsqueda de la imagen objetivo en la imagen ambiente en tamaño normal, luego buscando por la mitad de esta y así sucesivamente dependiendo de cuántas veces se quiera llevar a cabo la búsqueda.

Este proceso involucra más pasos y por lo tanto involucra más tiempo y recursos. Como muestra la Figura 21 el estándar que se manejará será de 10 escalas.

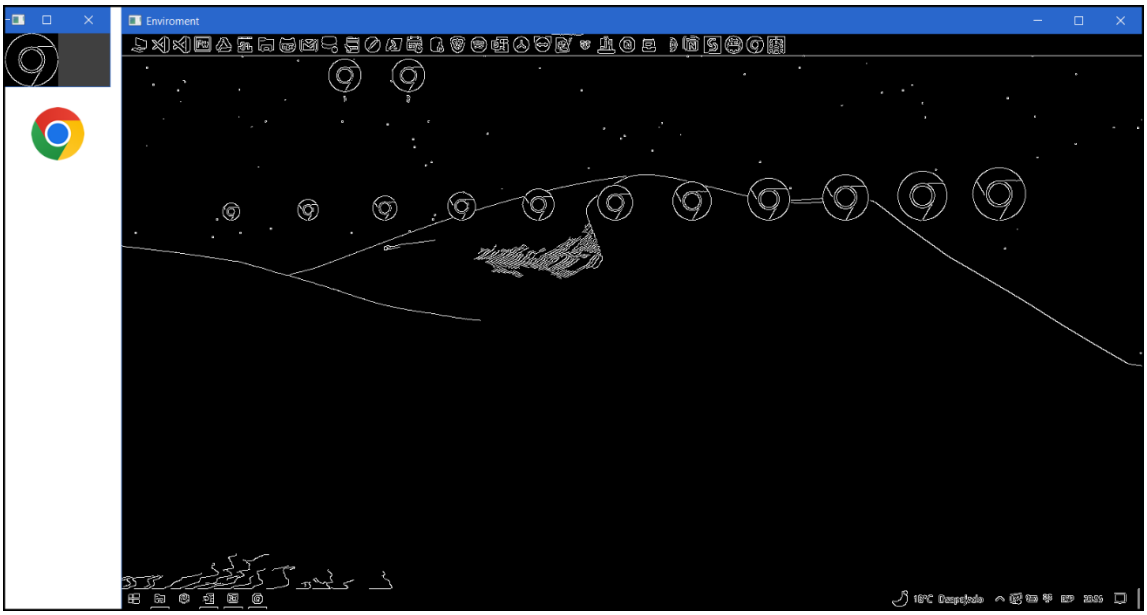
4.2. Algoritmo de búsqueda multiescalar

En busca de reducir la complejidad, existen algunas transformaciones que se aplicaran a ambas imágenes, a la imagen objetivo y a la imagen ambiente. Se aplicará primeramente una escala de grises y luego se obtendrán los bordes mediante el algoritmo de cani.

La Figura 22 muestra este resultado

Figura 22.

Transformación a escala de grises y bordes

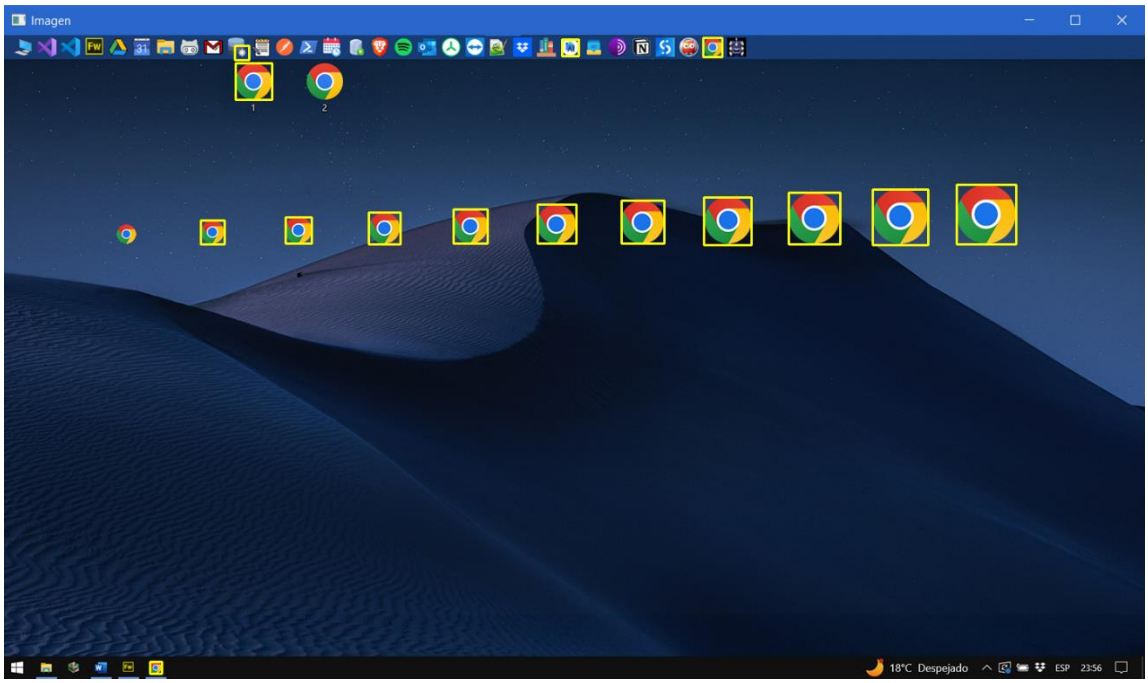


Nota. Captura de pantalla de transformación a escala de grises y bordes. Elaboración propia.

Las transformaciones que permite realizar *OpenCV* son particularmente útiles para eliminar todos aquellos elementos que no son relevantes para el análisis. El siguiente paso contempla realizar el barrido por medio de escalas. Para ver los aciertos de esta ejecución se dibujarán recuerdos de color verde que serán colocados en las coordenadas obtenidas. Se espera obtener una lista de todas las coincidencias posibles, que en el mejor de los casos sería igual a 15 aciertos. La Figura 23 muestra el resultado del algoritmo

Figura 23.

Patrones identificados con escala variante

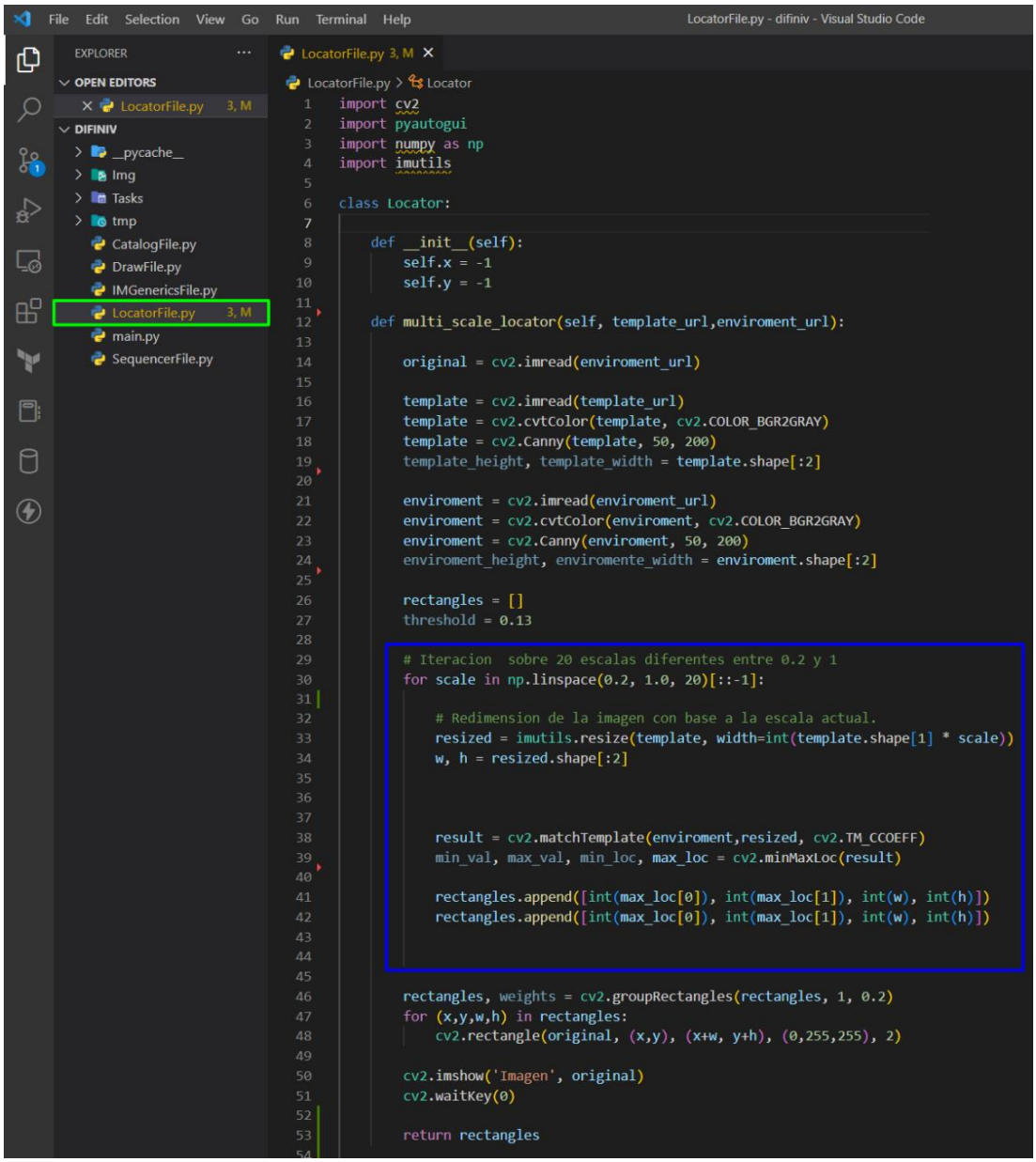


Nota. Captura de pantalla de los resultados del algoritmo. Elaboración propia.

El resultado es bastante satisfactorio, se lograron identificar casi todos los elementos que coinciden con el patrón. Es importante destacar que existen coincidencias erróneas como las dos que se encuentran en la parte superior de la imagen, así mismo existen dos elementos que no fueron seleccionados, esto no quiere decir que el algoritmo no es adecuado para los propósitos de esta tesis, sino que siempre se debe tomar en cuenta la probabilidad de que la búsqueda falle. El ejemplo presentado es una prueba extrema de las variaciones de escala en un sistema y esa es la razón por la cual puede afirmarse que este resultado aceptable, a continuación, se describirán las fases de este algoritmo:

Figura 24.

Algoritmo de localización multiescalar

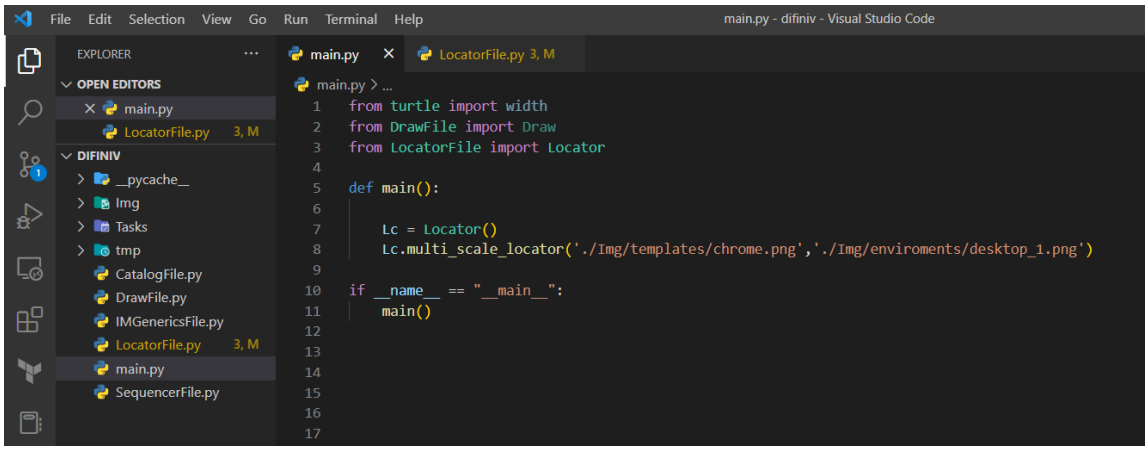


Nota. Captura de pantalla de algoritmo de localización multiescalar. Elaboración propia.

La figura mostrada anteriormente muestra la estructura del proyecto llamado automatov-opencv. Este inicia su ejecución por medio de un archivo maestro llamado main.py el cual inicializa un objeto del tipo Locator. Este objeto utilizara el método multi_scale_locator el cual recibe como parámetros la imagen objetivo y el ambiente. La Figura 25 muestra esta ejecución:

Figura 25.

Archivo main donde inicia la ejecución



Nota. Captura de pantalla de archivo main donde inicia la ejecución. Elaboración propia.

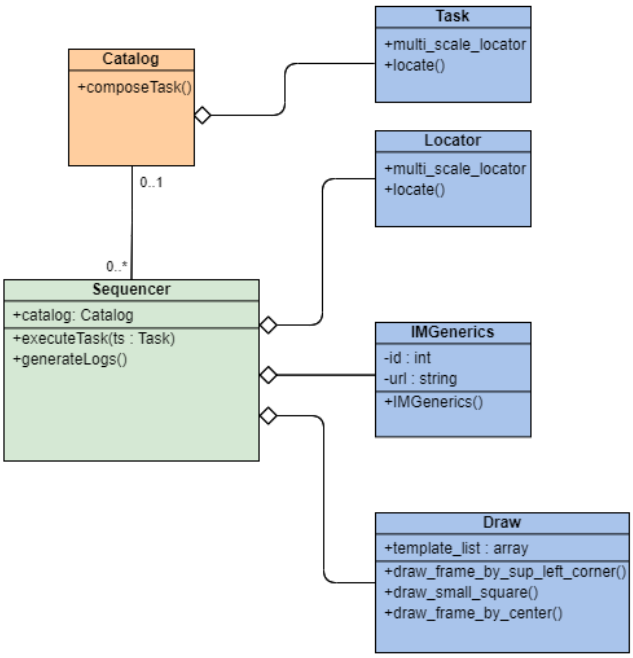
Por otro lado, y regresando al contexto de la Figura 23, el recuadro azul hace énfasis en el ciclo repetitivo que permite realizar el barrido de la imagen objetivo a diferentes escalas. Es importante hacer notar el uso de la librería NumPy como un conjunto de herramientas que permiten trabajar de una manera más sencilla sobre colecciones de datos y arreglos. Por último, este objeto retorna un arreglo que contiene las coordenadas o “Rectángulos” que superpuestos en la imagen ambiente localizan todas las coincidencias encontradas y mostradas mediante la Figura 22.

4.3. Diagrama de clases

A continuación, se muestra la interacción de clases que tienen lugar en este proyecto con el fin dar un enfoque basado en el paradigma de objetos. Es importante hacer notar que la clase *Sequencer* es la que permite ejecutar un conjunto de tareas predefinidas mediante un catálogo, no obstante, el secuenciador puede generar tareas que no estén incluidas en este catálogo, es decir tareas personalizadas. La clase *Locator* como se explicó anteriormente identifica las coordinas de la imagen objetivo, mientras que la clase *IMGenerics* brinda un conjunto iconos comunes en el sistema operativo Windows para evitar la necesidad de capturar imagen por imagen. Por último, la clase Draw permite dibujar los rectángulos en tiempo real mediante la interacción con el GUI.

Figura 26.

Diagrama de clases



Nota. Descripción del diagrama de Clases. Elaboración propia, realizado con Paint.

4.4. Ejecución e Interacción con el GUI

La interacción directa con la GUI del sistema operativo es realizada por dos clases únicamente: *Task* y *Draw. Task* permite invocar las herramientas que brinda *Pyautogui* que brindan control sobre posicionamiento del cursor, presión sobre el cursor derecho e izquierdo, manejo del portapapeles y combinación de teclas. Por otro lado, la solución que se propone en esta tesis incluye una retroalimentación visual al momento de ejecutar cada paso del secuenciador. Mediante la librería win32gui es posible escribir directamente a la memoria de video y localizar o escribir en tiempo real. La Figura 27 muestra el algoritmo que manipula la interfaz gráfica.

Figura 27.

Manipulación en tiempo real con la GUI

```
class Draw:
    def draw_small_square(self,x, y,color):
        dc = win32gui.GetDC(0)
        win32gui.SetPixel(dc, x, y, color)
        win32gui.SetPixel(dc, x + 1, y + 1, color)
        win32gui.SetPixel(dc, x + 2, y + 2, color)
        win32gui.SetPixel(dc, x + 3, y + 3, color)

    def draw_frame_by_sup_left_corner(self,x, y, width, height,color):
        dc = win32gui.GetDC(0)

        win32gui.SetPixel(dc, x, y, color)
        thickness = 3
        for i in range(x, x + width):

            for j in range(thickness):
                win32gui.SetPixel(dc, i, y + j, color)
                win32gui.SetPixel(dc, i, y + height - j, color)

        for i in range(y, y + height):

            for j in range(thickness):
                win32gui.SetPixel(dc, x + j, i, color)
                win32gui.SetPixel(dc, x + width - j, i, color)

    def draw_frame_by_center(self,x, y, width, height,color):
        self.draw_frame_by_sup_left_corner((x - (width // 2)),(y - (height // 2)),width,height,color)


    def draw_circle(self,x, y, radius):

        dc = win32gui.GetDC(0)
        green = win32api.RGB(124,252,0)
        for i in range(x-radius,x+radius):
            for j in range(y-radius,y+radius):
                if (i-x)**2 + (j-y)**2 <= radius**2:
                    win32gui.SetPixel(dc, i, j, green)
                    time.sleep(0.06)
```

Nota. Captura de pantalla de manipulación en tiempo real con la GUI. Elaboración propia.

5. EJECUCION Y REPORTES

5.1. Bitácora de ejecución del *script*

Como su nombre lo indica, la bitácora de ejecución registra cada uno de los pasos en la ejecución del script, este elemento es la piedra angular de proyecto. ¿Cómo saber en qué lugar fallo? ¿Cómo darle retroalimentación al usuario? He ahí la importancia de este módulo. Dicho componente se ha dividido en elementos atómicos que juntos forman un árbol de ejecución y registran cada paso. Esta división se explica a continuación:

- Secuencia: es una serie de pasos que culminan con el éxito o el fracaso de un algoritmo. Este módulo secuenciador permite escribir los pasos de la automatización en alto nivel y con un lenguaje bastante intuitivo.

- Paso: un paso conlleva un contexto e involucra el seguimiento de múltiples tareas. Para fines de este trabajo investigativo, los pasos solo pueden contener una lista de tareas.

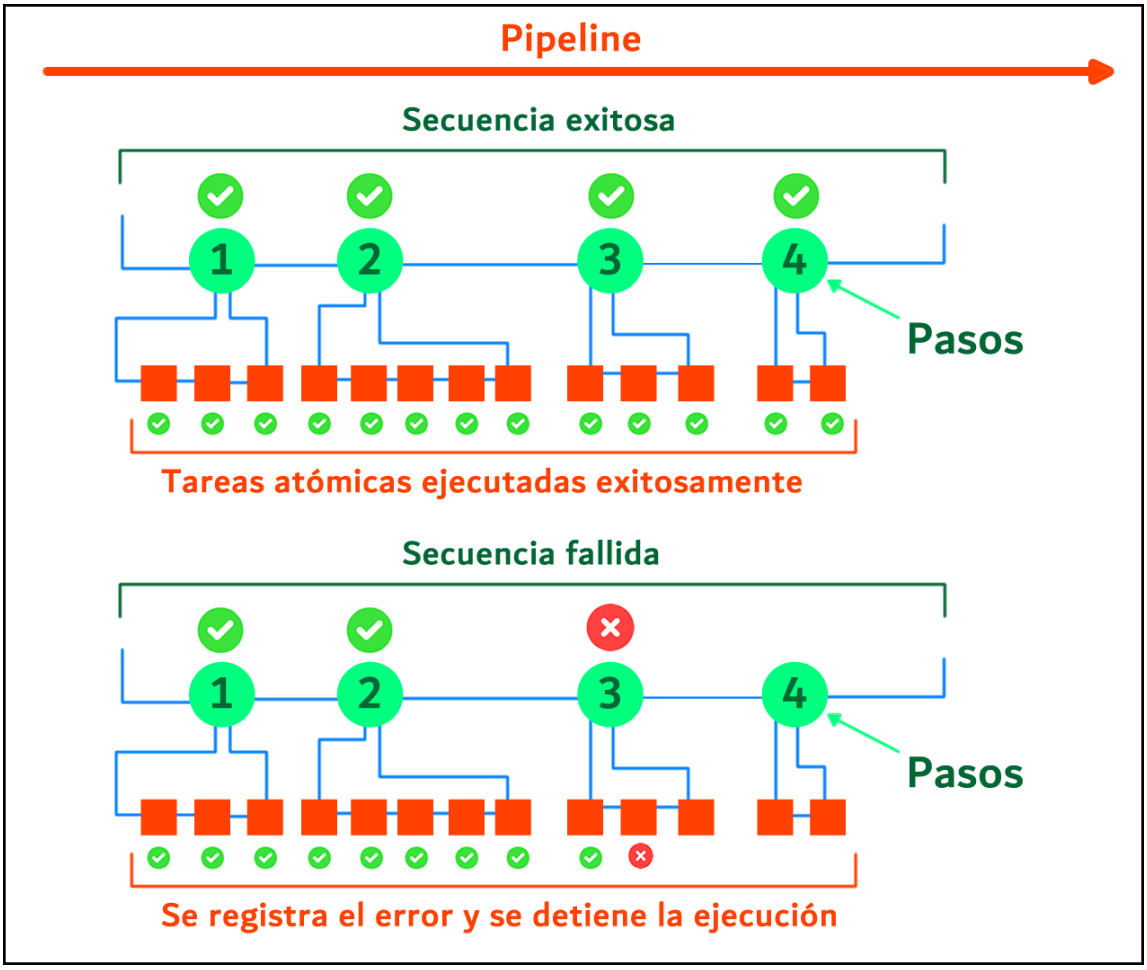
- Tarea: indica la interacción con la Interfaz gráfica y la API del sistema operativo, estas tareas son atómicas pues no poseen subtareas

Por excelencia la mejor estructura para representar un árbol es el formato JSON el cual permite el anidamiento y la visualización de los datos de una manera sencilla. En este aspecto la meta data juega un papel muy importante ya que etiqueta y señala en qué momento se puede producir un error. Para comprender mejor esto se pude pensar un una tubería o una secuencia de pasos

como los que podemos ver en la cultura DevOps, la figura 28 ilustra mejor esta idea:

Figura 28.

Analogía del flujo de ejecución con una DevOps Pipeline



Nota. Gráfica de analogía del flujo de ejecución con una *DevOps Pipeline*. Elaboración propia, realizado con Paint.

Como puede observarse, al producirse un error el secuenciador detiene automáticamente la ejecución y devuelve el control al usuario. Un error se

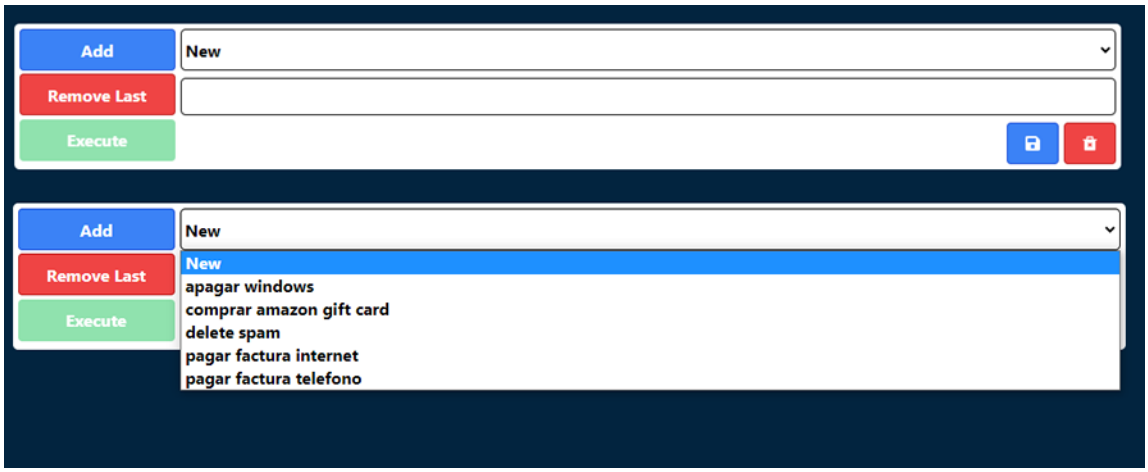
produce en una tarea y esta puede ser debido a que la red neuronal de *OpenCV* no logra identificar la imagen objetivo, al suceder esto devolverá un valor booleano falso y este detendrá de manera inmediatamente las acciones del script. Cada paso es registrado, sea exitoso o no mediante el árbol de ejecución.

5.2. Interfaz Gráfica

La interfaz gráfica consta de un catálogo donde se almacenan las secuencias previamente creadas. La herramienta provee de las funcionalidades necesarias para actualizar y eliminar pasos según las necesidades del usuario. A continuación, la Figura 29 muestra esta interfaz:

Figura 29.

Catálogo de secuencias



Nota. Captura de pantalla del catálogo de secuencias. Elaboración propia.

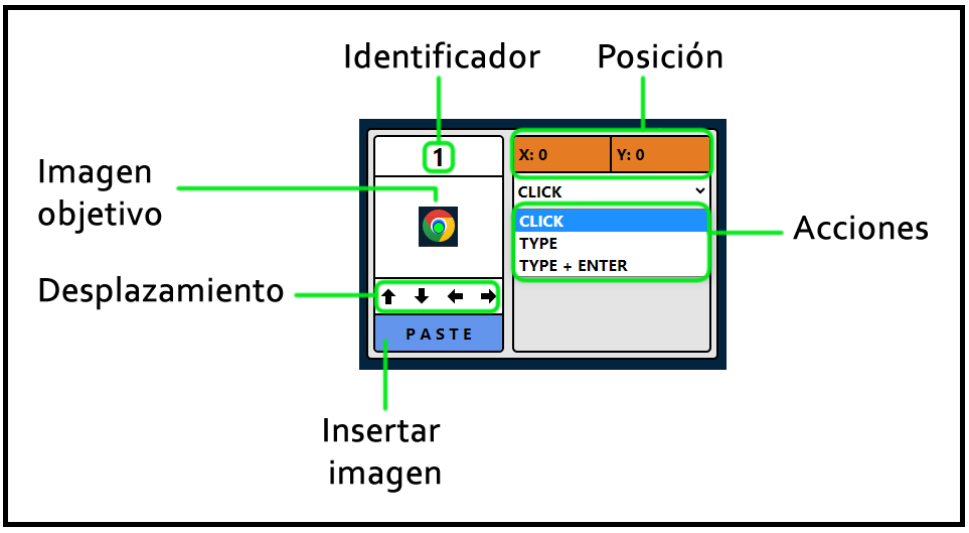
Como puede observarse, existe un menú que despliega las secuencias guardadas con un nombre personalizado, en este caso, utilizadas para la automatización de pagos, compras y para apagar el equipo. Los botones

ubicados al lado izquierdo son utilizados para agregar, remover pasos a una secuencia, tanto para secuencias nuevas, como para editar alguna existe. En la esquina inferior derecha encontramos los botones necesarios para crear una secuencia nueva o guardar una actualización, también para eliminar alguna.

Como se me mencionó en los capítulos anteriores, cada secuencia consta de una serie de pasos detallados y para este propósito la interfaz gráfica utilizar los siguientes controles para identificar la imagen objetivo y definir que acción se realizará una vez que se tenga localizada. Para colocar una nueva imagen en su espacio correspondiente, se debe de utilizar la siguiente combinación de tecla**s**: Windows + Shift + S para recortar el área deseada y el botón pegar para insertar la misma dentro de la tarea. La Figura 30 muestra esta descripción a detalle esta descripción e identifica la función de cada uno de los controles:

Figura 30.

Detalle de subtarea

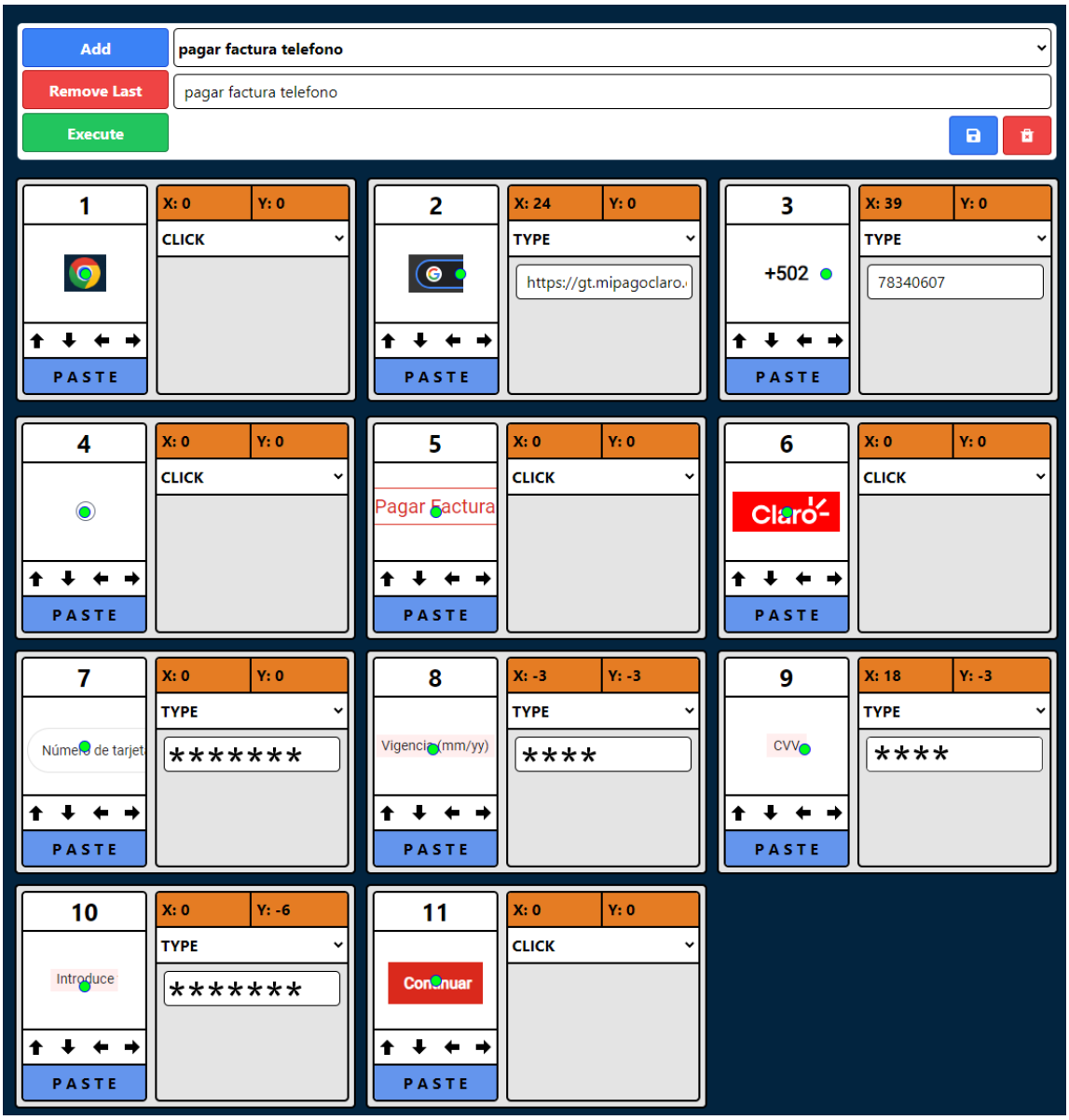


Nota. Detalle de subtarea. Elaboración propia, realizado con Paint.

El segmento que incluye los controles de cada tarea es muy parecido a una pequeña tarjeta y para cada secuencia la interfaz gráfica proporciona un conjunto de tareas o tarjetas que incluyen la configuración anteriormente mostrada. Se puede apreciar una imagen completa de la interfaz gráfica seleccionando una de las secuencias previamente guardada, en este caso se selecciona la opción: Eliminar Spam con lo que automáticamente las tarjetas de configuración son cargadas. La Figura 31 muestra como luce aplicación en este escenario:

Figura 31.

Detalle de subtarea



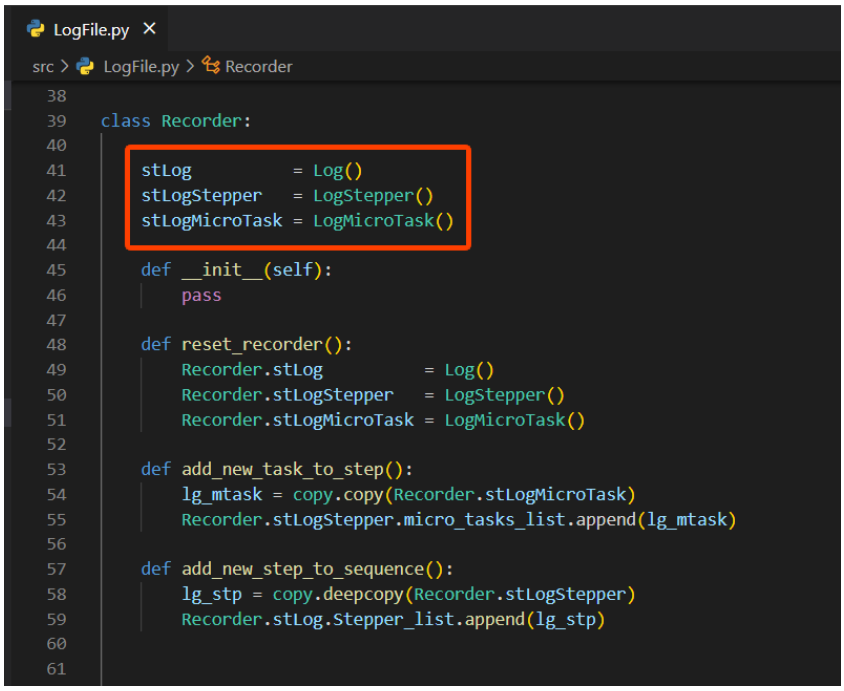
Nota. Captura de pantalla de detalle de subtarea. Elaboración propia.

5.3. Uso de la clase estática Recorder y captura de metadatos

Un buen sistema de bitácora y de registro de ejecución debe ser totalmente invisible, es decir no debe de comprometerse con la programación de la aplicación y únicamente debe ser un observador de los movimientos. Con este fin, se ha decidido utilizar el patrón de diseño de la clase estática. Específicamente se ha creado la clase Recorder básicamente tiene 3 objetos de tipo estático. Estos son fácilmente referenciales, pero son de carácter volátil así que deben ser registrados en un árbol de memoria cada vez que se salga de una tarea o de algún paso en específico. La Figura 32 muestra cómo funciona esta clase:

Figura 32.

Objetos estáticos de la clase Recorder

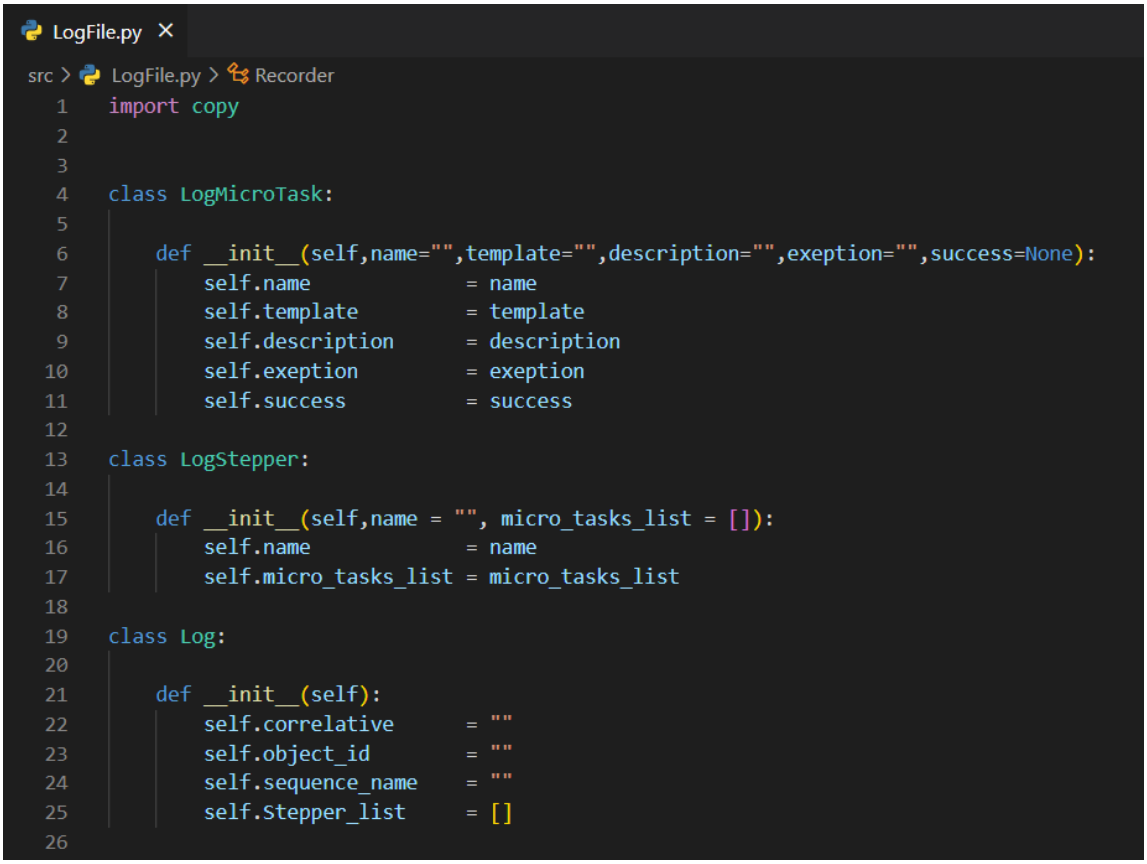


Nota. Captura de pantalla de objetos estáticos de la clase Recorder. Elaboración propia.

Los objetos señalados en la Figura 33 corresponden a los contenedores de metadatos. Básicamente poseen un identificador y una lista de los pasos o de tareas. El objeto que más proporciona información es corresponde a las tareas, aquí llamadas Micro Tareas por considerarse puramente atómicas, a continuación, la figura 30 muestra la estructura de estos objetos, que como se dijo anteriormente son volátiles y reutilizables.

Figura 33.

Estructura de los objetos responsables de la bitácora

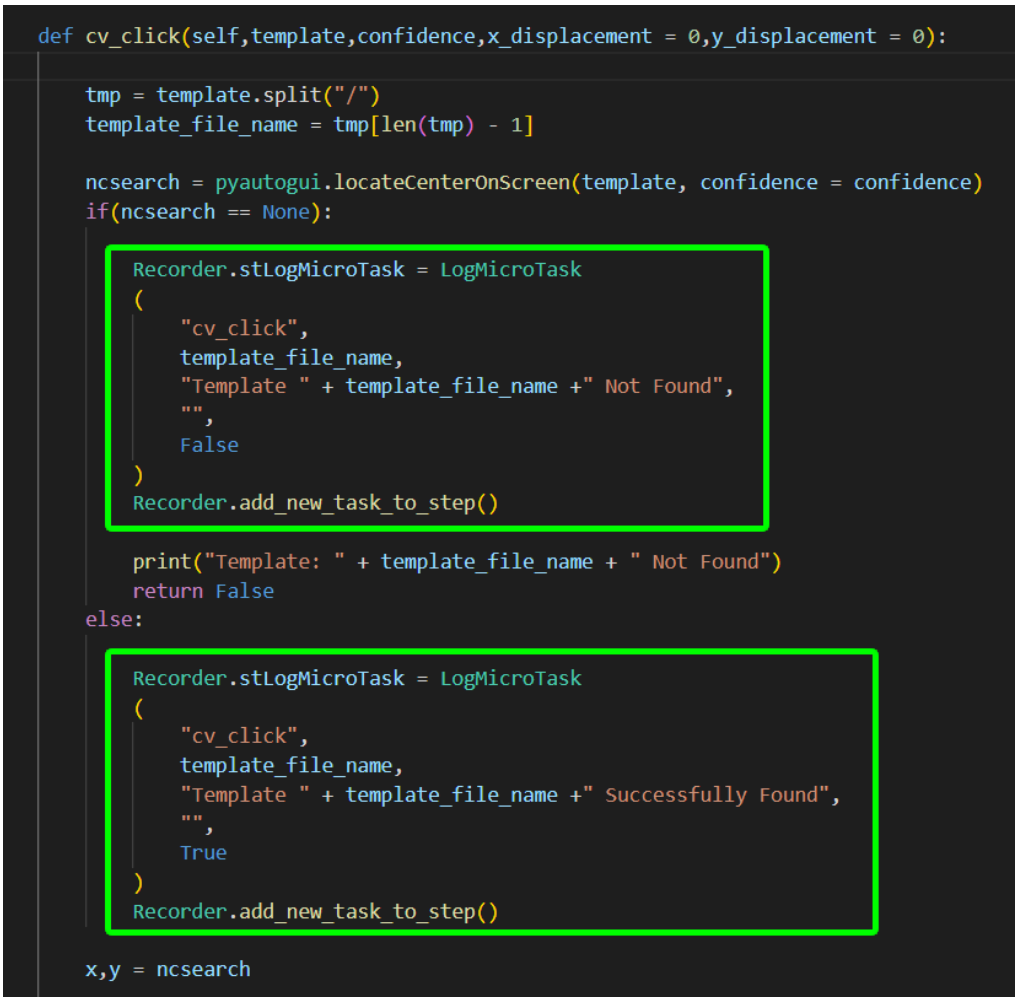


Nota. captura de pantalla de estructura de los objetos responsables de la bitácora. Elaboración propia.

A continuación, se muestra como el sistema de bitácora interactúa y registra cada uno de los pasos de la ejecución, como se puede observar en la Figura 34 los objetos de la clase estática en ningún momento se compromete la lógica de programación y su presencia es meramente como un observador. Mediante una tarea atómica cv_click podemos ver como se realiza este registro.

Figura 34.

Recolección de los registros



```
def cv_click(self,template,confidence,x_displacement = 0,y_displacement = 0):

    tmp = template.split("/")
    template_file_name = tmp[len(tmp) - 1]

    ncsearch = pyautogui.locateCenterOnScreen(template, confidence = confidence)
    if(ncsearch == None):
        Recorder.stLogMicroTask = LogMicroTask
        (
            "cv_click",
            template_file_name,
            "Template " + template_file_name +" Not Found",
            "",
            False
        )
        Recorder.add_new_task_to_step()

        print("Template: " + template_file_name + " Not Found")
        return False
    else:
        Recorder.stLogMicroTask = LogMicroTask
        (
            "cv_click",
            template_file_name,
            "Template " + template_file_name +" Successfully Found",
            "",
            True
        )
        Recorder.add_new_task_to_step()

    x,y = ncsearch
```

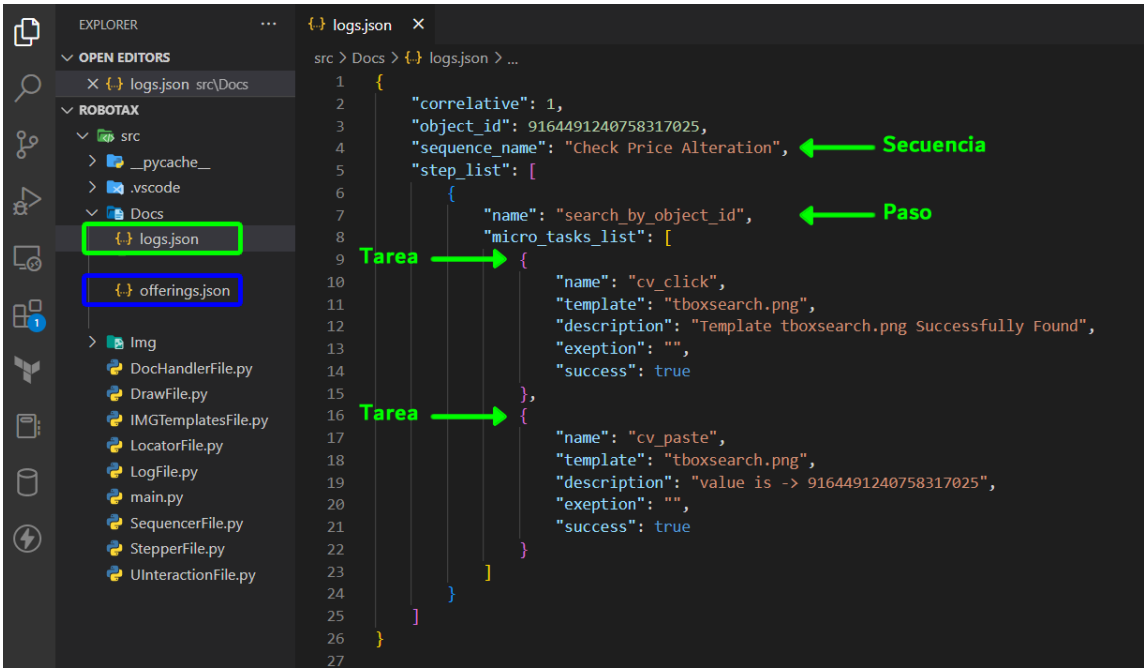
Nota. Captura de pantalla de la recolección de los registros. Elaboración propia.

5.4. Reporte de errores

Los pasos mostrados anteriormente respecto al manejo de la ejecución y colección de información son almacenados en un objeto raíz declarado al momento de iniciar el secuenciador, este es el encargado de tomar la información volátil almacenada en los objetos estáticos y llenar el árbol de ejecución. Posteriormente se tiene un método que traduce estos objetos anidados a un archivo son. La Figura 35 muestra cómo se observa el reporte de ejecución.

Figura 35.

Recolección de los registros



Nota. Captura de pantalla de reporte de ejecución. Elaboración propia.

Como puede apreciarse la estructura de este archivo llamado logs.json refleja justamente un árbol que registra todos los movimientos realizados por el secuenciador. Es importante anotar aquí que la fuente de datos se encuentra en el archivo offerings.json marcado con azul y el contexto en el que fue probado este prototipo es en la automatización de un conjunto de ofertas.

6. DESCRIPCIÓN DE ENTREGABLES Y LICENCIA

6.1. Descripción del entregable final

Este trabajo de investigación está dividido en 3 aplicaciones que en conjunto generan la ejecución del motor de visión computacional OpenCV. Las mismas se describen a continuación:

- Ampliación Front-end: GUI
- Aplicación Back-end: Servidor
- Aplicación de Consola: Ejecutor de *OpenCV*

El desarrollo explicado en la sección de ejecución y reportes fue actualizado y puede ser encontrado en el siguiente repositorio: https://github.com/Gualtix/automatov-opencv

6.2. Licencia BSD-2

Se ha elegido una licencia del tipo BSD-2. La Licencia BSD es una licencia permisiva, lo que significa que permite a los usuarios hacer casi cualquier cosa con el software, incluyendo su uso en software propietario. Bajo las cláusulas de esta licencia se permite la redistribución en fuente y en binario con o sin modificación, siempre que se cumplan las condiciones siguientes:

- Las redistribuciones en fuente deben retener la nota de copyright y listar estas condiciones y la limitación de garantía.

- El software desarrollado bajo este trabajo de investigación NO provee ninguna garantía ni se hace responsable por cualquier daño ocasionado por su mala implementación o mal funcionamiento.

- Este programa se proporciona tal cual, sin garantías expresas ni implícitas, tales como su aplicabilidad comercial o su adecuación para un propósito determinado. En ningún caso el propietario será responsable de ningún daño causado por su uso (incluyendo pérdida de datos, de beneficios o interrupción de negocio).

CONCLUSIONES

1. El reconocimiento de imágenes en un entorno mediante la utilización de una plantilla o imagen objetivo tiene una fiabilidad que está directamente relacionada con la complejidad de la gráfica, rotación y escala.

2. El retraso o tiempo de espera entre una tarea y otra es muy importante debido a que este garantiza que la renderización este completa antes de realizar alguna acción; la configuración actual de la solución establece un retraso estático de 3 segundos, sin embargo, el trabajo de investigación establece que también es posible utilizar el método de visión computacional para detectar variantes entre una pantalla y otra.

3. La prueba de validación humana es quizá el obstáculo más grande para la ejecución de la solución, sin embargo, la identificación de letras o imágenes por parte de Captcha y Recaptcha están basadas en identificar patrones por medio de plantillas que es justamente el método que se utiliza para identificar los iconos en la pantalla. Esto quiere decir que la librería *OpenCV* puede ser utilizada para resolver esta limitación.

4. La tolerancia o umbral de aceptación para identificar una imagen como igual o similar está dada en un rango de 0.6 a 1 donde el número más bajo significa que la tolerancia es más alta y puede admitir ligeras variaciones en tamaño o forma y el número más alto, indica que el criterio es estricto y la imagen objetivo debe ser idéntica.

5. Mediante la ejecución de secuencias prealmacenadas se ha comprobado que el tiempo y la complejidad al realizar tareas repetitivas se reduce considerablemente.

RECOMENDACIONES

1. Implementar esta solución para todas aquellas tareas que pueden llegar a ser monótonas y que consumen una cantidad considerable de tiempo.

2. Evitar utilizar esta solución para automatizar tareas que requieran la utilización de datos sensibles como lo pueden ser cambio de contraseñas o información confidencial.

3. Tomar en cuenta que la ejecución de este algoritmo se basa en la probabilidad de éxito y dependiendo del ambiente en que se utilice puede producir resultados erróneos o totalmente inesperados.

4. Evitar interrumpir la ejecución de la secuencia mientras esta sigue su curso ya que puede perderse la continuidad de pasos y generar un resultado no deseado

5. Evitar usar esta solución para resolver pruebas de verificación humana o Captcha por lo que no es viable su uso en secuencias que incluyan este paso ya que la ejecución terminará al ser incapaz de continuar.

REFERENCIAS

Francois, C. (2021). *Deep Learning with Python*. Editorial Manning Publicaciones Co.

Mohamed, E. (2020). *Deep Learning for Vision Systems*. Editorial Manning Publicaciones Co.

Ponce, J. (29 de diciembre de 2022). *Blog personal sobre inteligencia artificial*. Jahaziel. https://jahazielponce.com/blog/

Sweigart, A. (2020). *Automate the Boring Stuff with Python*. Editorial No Starch Press.