



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

**AUTOMATIZACIÓN DE TAREAS EN SISTEMAS OPERATIVOS MEDIANTE
RECONOCIMIENTO DE GRAFICOS UTILIZANDO MACHINE LEARNING**

Walter Roberto Morales Quiñonez

Asesorado por el Ing. Marco Tulio Aldana Prillwitz

Guatemala, diciembre de 2022

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA



FACULTAD DE INGENIERÍA

**AUTOMATIZACIÓN DE TAREAS EN SISTEMAS OPERATIVOS MEDIANTE
RECONOCIMIENTO DE GRAFICOS UTILIZANDO MACHINE LEARNING**

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, DICIEMBRE DE 2022

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA



NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. Pedro Antonio Aguilar Polanco
VOCAL I	Ing. Angel Roberto Sic García
VOCAL II	Ing. Pablo Christian de León Rodríguez
VOCAL III	Ing. José Milton de León Bran
VOCAL IV	Br. Jurgén Andoni Ramírez Ramírez
VOCAL V	Br. Oscar Humberto Galicia Nuñez
SECRETARIA	Inga. Lesbia Magalí Herrera López

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Decano cuando realizó su privado
EXAMINADOR(A)	Ing. o Inga. dependiendo del género
EXAMINADOR(A)	Colocar examinadora si es Inga.
EXAMINADOR(A)	NO LLENAR SI NO HA REALIZADO PRIVADO
SECRETARIO	Secretario JD cuando realizó su privado.

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 SISTEMAS OPERATIVOS MEDIANTE RECONOCIMIENTO DE GRAFICOS UTILIZANDO MACHINE LEARNING

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería de Ciencias y Sistemas con fecha (fecha de asignación de protocolo).

Firma en original

Walter Roberto Morales Quiñonez

ACTO QUE DEDICO A:

Dios

Por darme el privilegio de culminar esta meta tan importante y por enseñarme que el camino hacia el éxito está lleno de enseñanzas y es ahí donde se aprende la 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 mi

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, Aníbal Morales, por ser compañeros de batalla y enriquecer mi conocimiento con su experiencia y habilidades.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES	III
LISTA DE SÍMBOLOS	VII
GLOSARIO	IX
RESUMEN	XI
OBJETIVOS.....	XIII
INTRODUCCIÓN	XIV
1. VISION COMPUTACIONAL	1
1.2. El sistema de vision computacional	1
1.2. Percepcion visual	2
1.3. Sistema de vision humano vs sistema de vision computacional	2
1.4. Fundamentos de la clasificacion de imagenes	7
1.5. Flujo de clasificacion	8
1.6. ¿Cómo ve una computadora una imagen?.....	9
1.7. Imagen en escala de grises.....	10
1.8. Imagen a color	10
1.9. Introduccion a OpenCV	12
1.10. Template Matching	15
1.11. Sistema coordinado de OpenCV	17
1.12. Dibujo de figuras mediante OpenCV	18

2.	REDES NEURONALES	20
2.1.	Introduccion a la Inteligencia Artificial	20
2.2.	Machine Learning.....	21
2.3.	Deep Learning.....	22
2.4.	Redes Neuronales.....	23
3.	CONTROL DE GUI	25
3.1.	Uso de la librería Pyautogui	25
3.2.	Captura de pantalla: Analisis e identificacion de coordenadas	26
3.3.	Manipulacion de ventanas y controles	27
4.	INTEGRACION	28
4.1.	Identificación de imágenes objetivo	28
4.2.	Algoritmo de búsqueda multiescalar	30
4.3.	Diagrama de Clases.....	34
4.4.	Ejecucion e Interaccion con el GUI	35
5.	EJECUCION Y REPORTES	36
5.1.	Bitácora de ejecución del script.....	36
5.2.	Uso de la clase estática Recorder y captura de metadatos	38
5.3.	Reporte de errores	41

ÍNDICE DE ILUSTRACIONES

FIGURAS

1.	Sistema de visio humano	3
2.	Sistema de vision computacional	4
3.	Similitud entre una neurona biologica y una artificial	5
4.	Red neuronal artificial.....	6
5.	Flujo de la clasificacion de imagenes	5
6.	Representacion computacional de una imagen	9
7.	Representacion de una imagen a color por medio de canales.....	11
8.	Separacion modular de OpenCV	13
9.	Busqueda de imagen objetivo	16
10.	Coordenadas en OpenCV	17
11.	Etiquetas en objetos.....	19
12.	Subcampos de la inteligencia artificial.....	20
13.	Programacion tradicional vs Machine Learning.....	21
14.	Significado de profundidad en Machine Learning	22
15.	Flujo de trabajo de una red neuronal.....	24
16.	Funcion de activacion.....	24
17.	Captura de pantalla mediante libreria.....	26
18.	Sistema coordinado mediante Pyautogui.....	26
19.	Pulsado de boton inicio de Windows mediante coordenadas	27
20.	Variantes de un mismo icono, según escala y forma	28
21.	Pirámide de iteración con 10 escalas.....	29
22.	Transformación a escala de grises y bordes.....	30
23.	Patrones identificados con escala variante	31

24.	Algoritmo de localización multiescalar	32
25.	Archivo main donde inicia la ejecución	33
26.	Diagrama de clases	34
27.	Manipulación en tiempo real con la GUI	35
28.	Analogía del flujo de ejecución con una DevOps Pipeline	37
29.	Objetos estáticos de la clase Recorder.....	38
30.	Estructura de los objetos responsables de la bitácora.....	39
31.	Recolección de los registros	40
32.	Reporte de errores.....	41

LISTA DE SÍMBOLOS

Símbolo	Significado
(símbolo)	(Breve descripción)
(símbolo)	Ordenados alfabéticamente; siglas en glosario.
m	Metro
mm	Milímetro
nm	Nanómetro

GLOSARIO

(Palabra) (Significado PUNTUAL)

English Word Palabras de otro idioma en cursiva que no sean nombres propio.

Microsoft Office Nombres propios en otros idiomas no requieren cursiva.

Tabulador Tecla que se presiona para separar concepto de significado.

Problema Se dará si el concepto es demasiado largo por lo que deberá corregir los tabuladores.

RESUMEN

Aquí empieza el resumen de la tesis que ustedes deberán escribir. No debe ser mayor de una hoja.

Entre párrafo y párrafo dejando solamente 1 espacio.

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 retroalimentación 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 Machine Learning 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 seguir así un algoritmo predefinido desde la organización que brinda soporte para que el usuario final solo tenga que ejecutarlo y la configuración deseada sea aplicada.

1. VISION 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. 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 Artificial, 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.

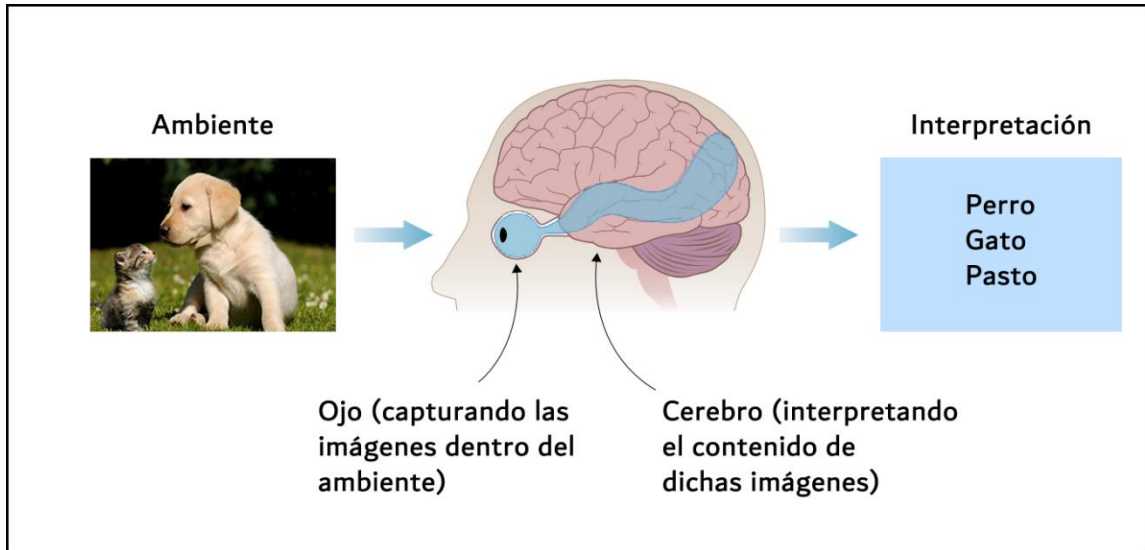
1.2. Percepción Visual

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.

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

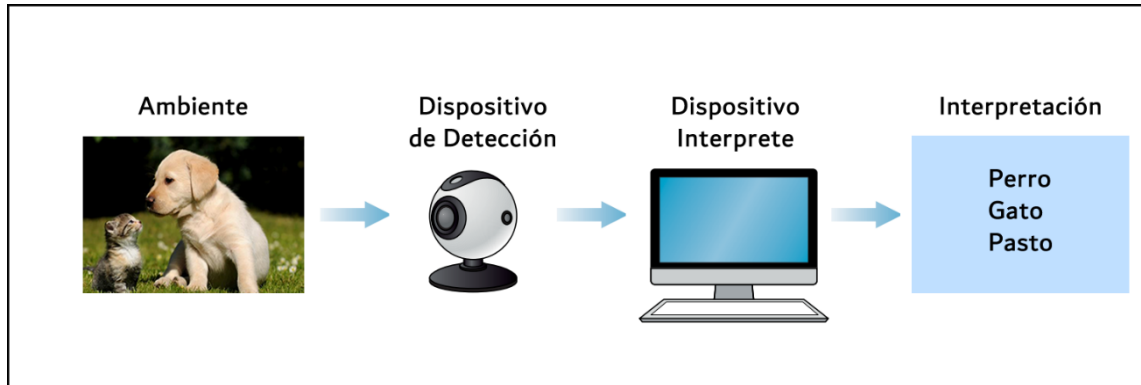
Figura 1. Sistema de visión humano



Fuente: MOHAMED, Elgendy. Deep Learning for Vision Systems
p. 5

Por otro lado, los sistemas de CV están inspirados en los sistemas naturales, estos sistemas copian el funcionamiento mediante máquinas que replican la habilidad de captar y analizar el ambiente. 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 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

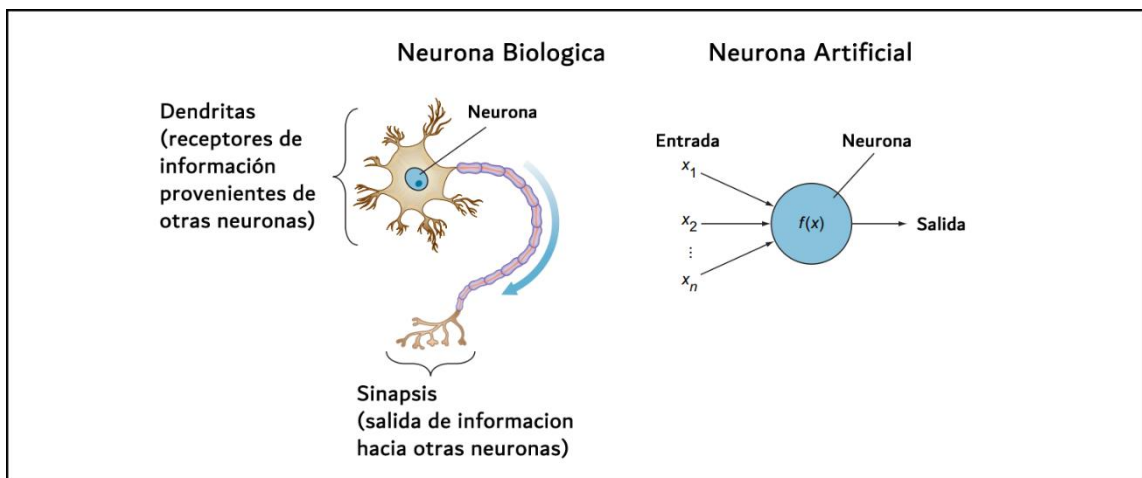


Fuente: MOHAMED, Elgendy. Deep Learning for Vision Systems
p. 7

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. A continuación, la 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.

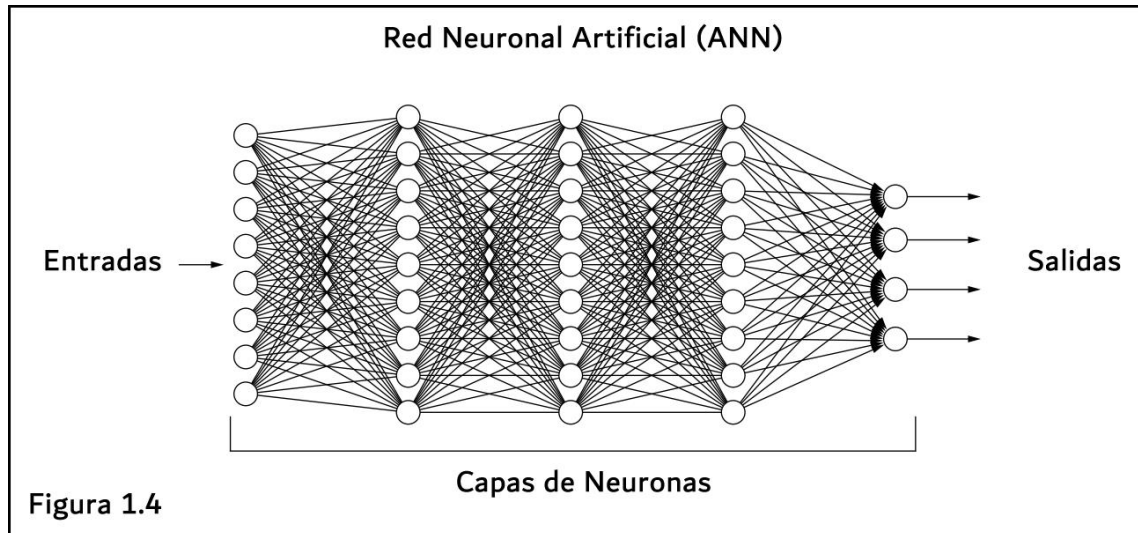
Figura 3. Similitud entre una neurona biológica y una artificial



Fuente: MOHAMED, Elgendy. Deep Learning for Vision Systems

p. 9

Figura 4. Red Neuronal Artificial



Fuente: MOHAMED, Elgendy. Deep Learning for Vision Systems
p. 9

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 capítulo.

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 son particularmente eficientes ya que permiten superar obstáculos como la escala o la rotación de un objeto en un ambiente.

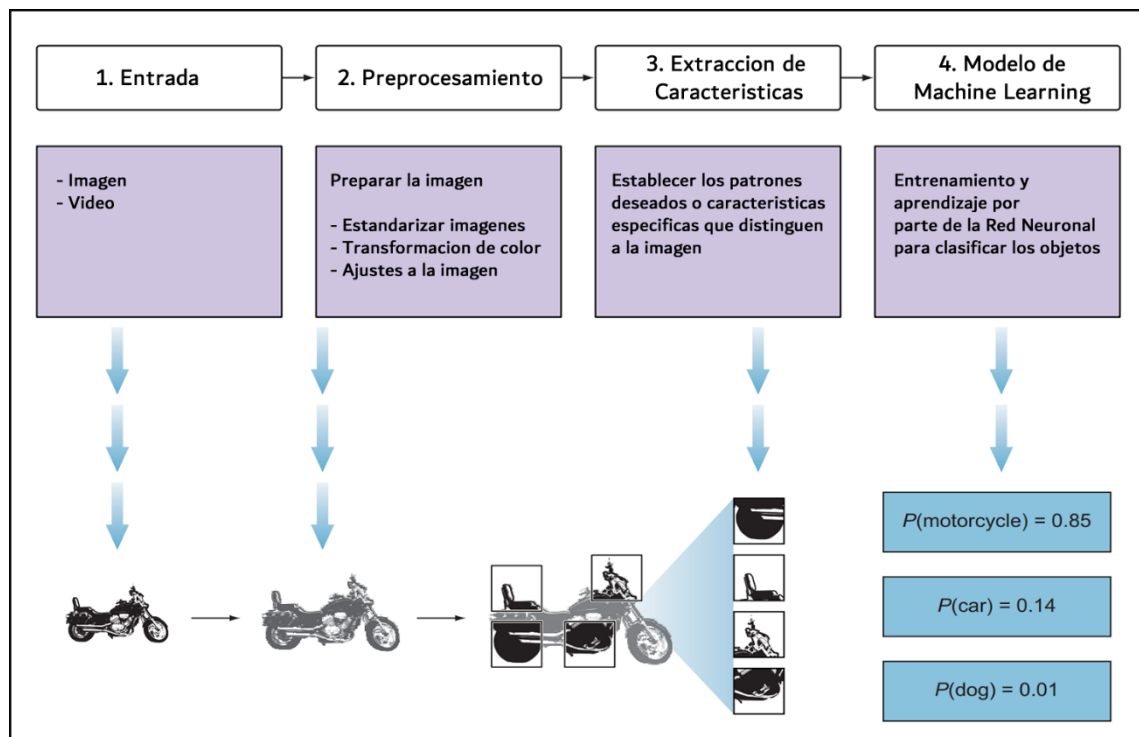
Un algoritmo clasificador de imágenes toma una imagen como entrada y produce una etiqueta que permite identificar dicha imagen. Se conoce como Clase a la categoría de salida de los datos. Las aplicaciones de este campo son muchas y pueden listarse las siguientes:

- 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



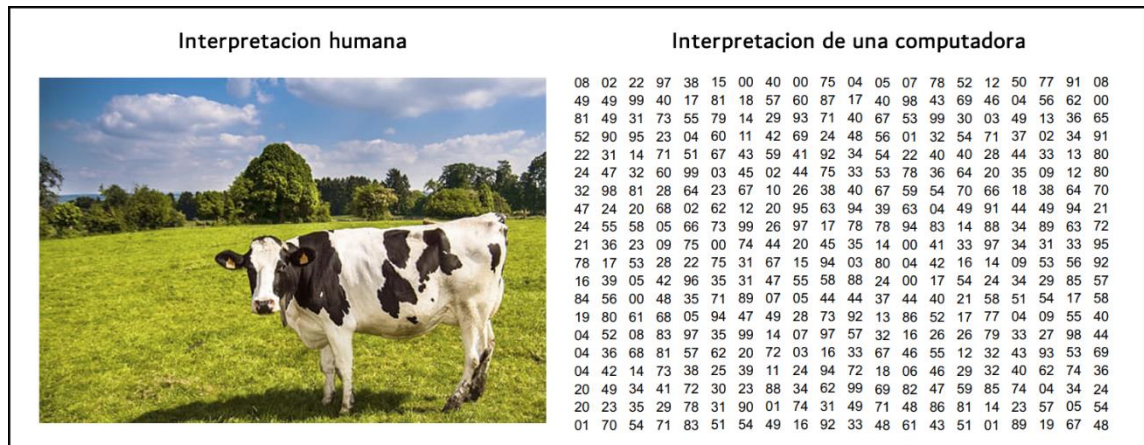
Fuente: MOHAMED, Elgendy. Deep Learning for Vision Systems

p. 18

1.6. ¿Cómo ven 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.

Figura 6. Representación computacional de una imagen



Fuente: MOHAMED, Elgendy. Deep Learning for Vision Systems

p. 21

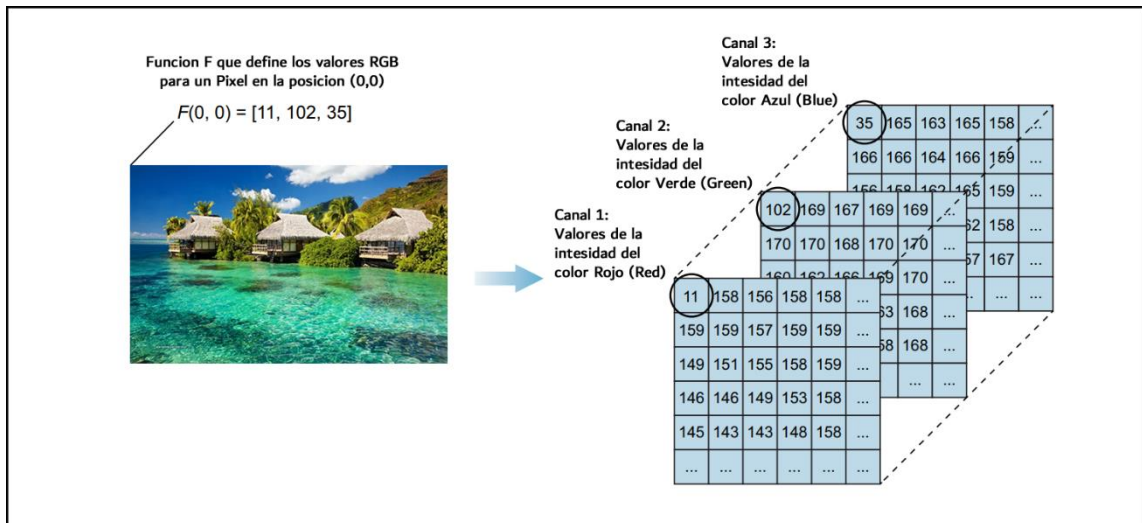
1.7. Imagen en escala de grises

La matriz mostrada en la figura 6 representa una imagen de 24 x 24 píxeles, esto quiere decir un total de 576 píxeles. 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 píxel. 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.

Figura 7. Representación de una imagen a color por medio de canales



Fuente: MOHAMED, Elgendy. Deep Learning for Vision Systems

p. 22

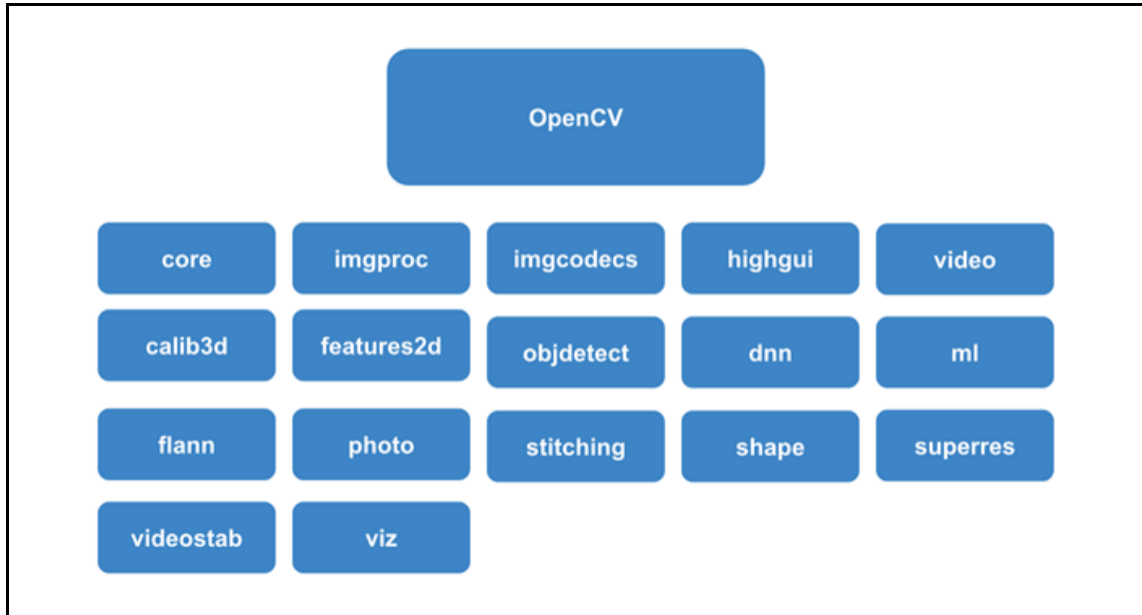
1.9. Introducción a OpenCV

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 Stanford 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



Fuente: VILLAN FERNANDEZ, Alberto. Mastering OpenCV 4 with Python
p. 18

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

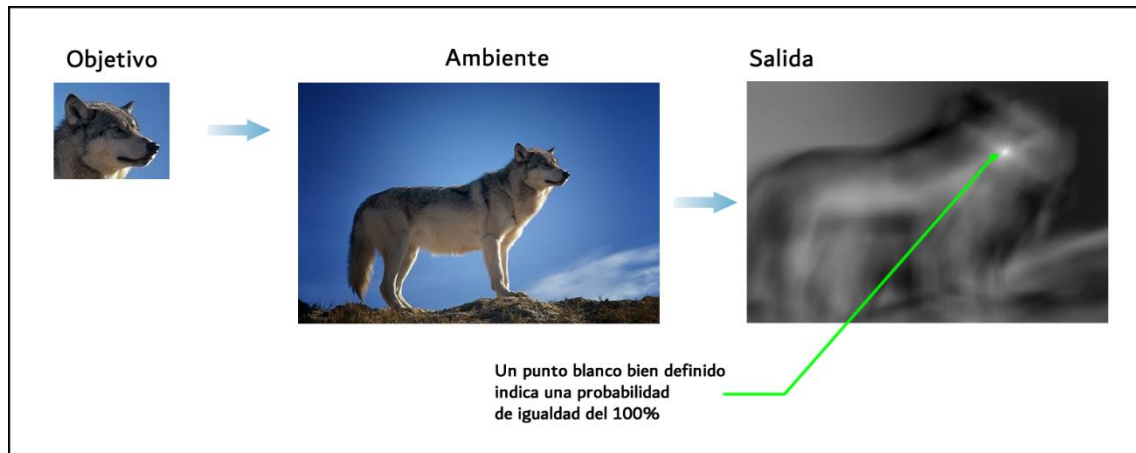
- Video: Modulo de análisis de video que incluye algoritmos de seguimiento de objetos en movimiento
- Objdetect: Modulo de deteccion de objetos mediante la definición de clases como: caras, ojos, personas, automóviles entre otros.
- Dnn: Modolo que proporciona una interfaz para generar redes neuronales profundas
- Ml: Modulo machine learning que continue un conjunto de librerías que pueden ser usadas para propósitos de clasificación y regresión
- Flann: Modulo de búsqueda por agrupamiento y búsqueda en espacios multidimensionales por medio de algoritmos fast, nearest, neighbor y search
- Photo: Modulo dedicado a la fotografía computacional
- Shape: Modulo dedicado a analizar escalas y similitud entre una imagen y un entorno, también puede es aplicable a realizar comparaciones entre dos objetos.
- Superres: Modulo de manipulación de la resolución que contiene un conjunto de algoritmos que puede mejorar la resolución de una imagen.
- Videostab: Modulo 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 a través de regiones superpuestas sobre la imagen para encontrar cierta similitud. Es importante destacar que este análisis de barrido realiza múltiples comparaciones basadas en algoritmos de Deep Learning 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 círculo 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



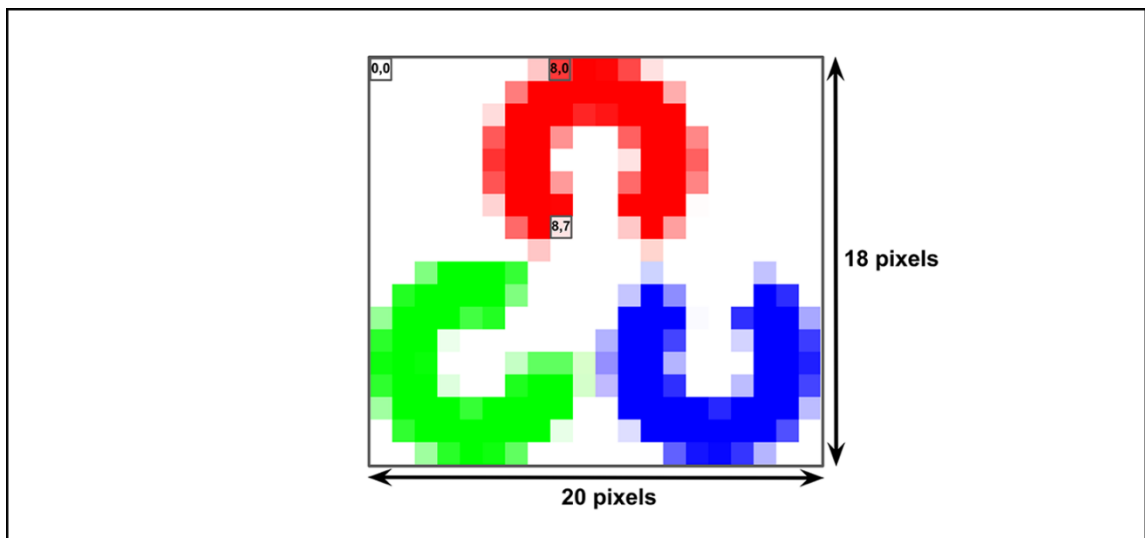
Fuente: elaboración propia.

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



Fuente: elaboración propia.

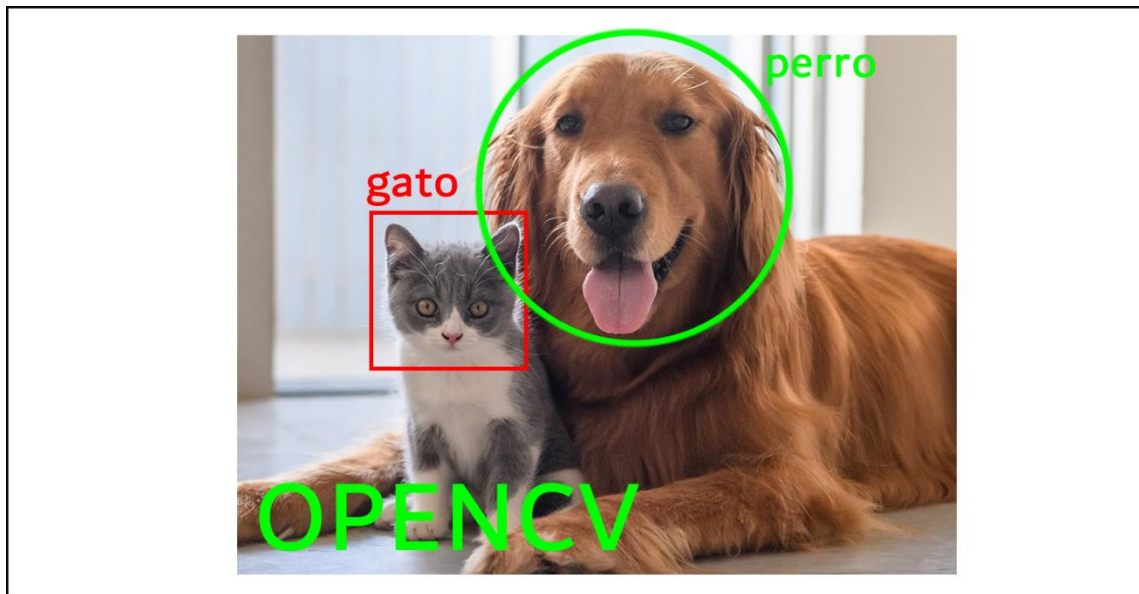
1.12. Dibujo de figuras mediante OpenCV

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



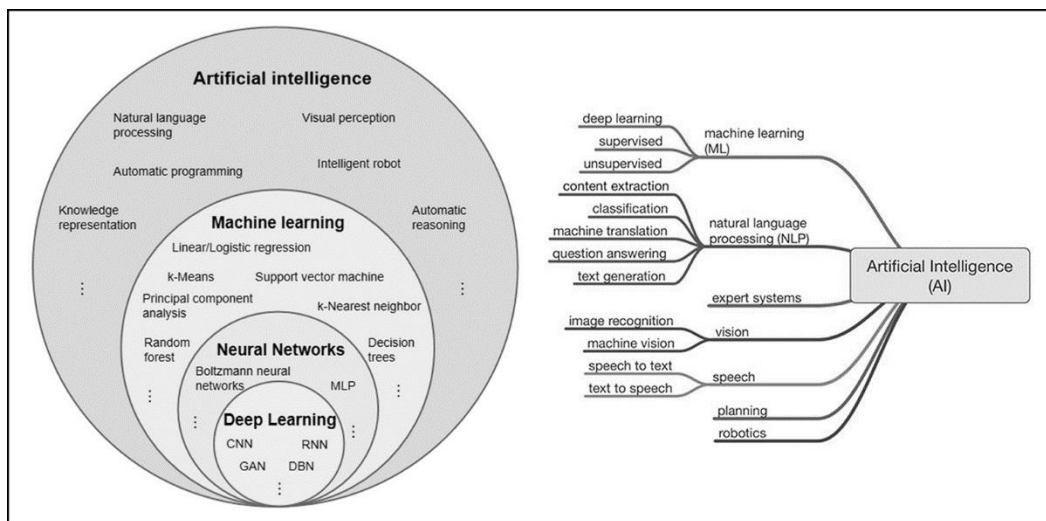
Fuente: elaboración propia.

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 disciplina mucho más grande. La figura 13 muestra esta descripción e ilustra otros campos involucrados:

Figura 12. Subcampos de la Inteligencia Artificial



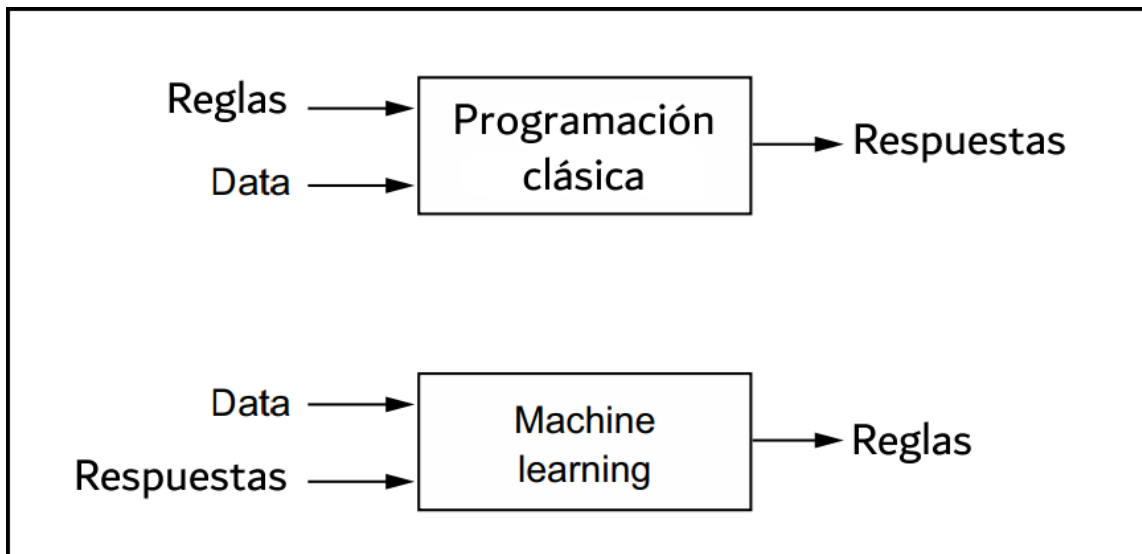
Fuente: elaboración propia.

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 a través 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 poder inferir o deducir las reglas que modelan cierto comportamiento.

Figura 13. Programación tradicional vs Machine Learning



Fuente: elaboración propia.

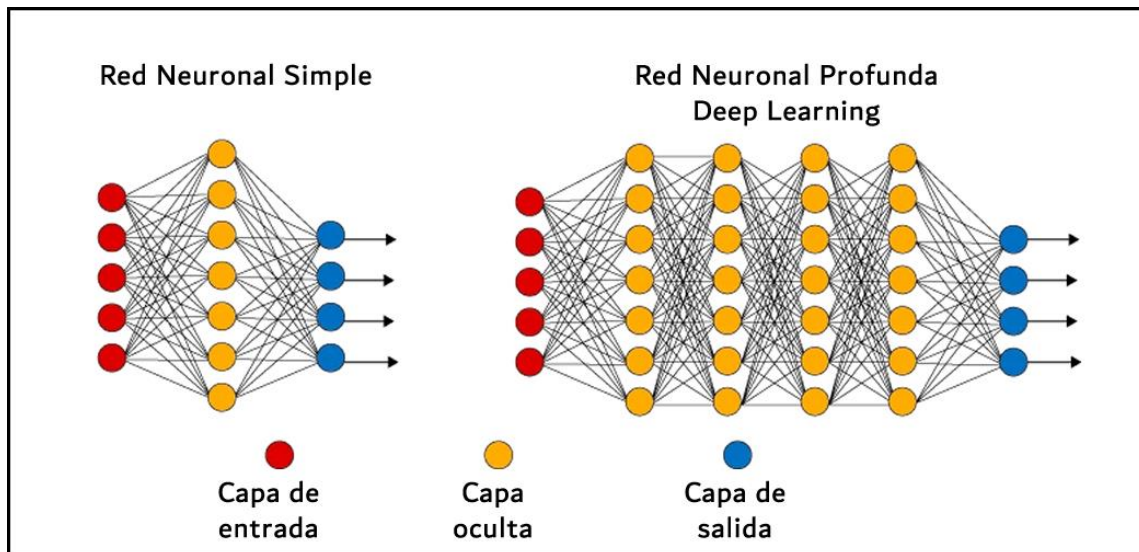
Machine Learning comenzó a florecer a partir de 1,990, por esa época las inferencias se hicieron mas precisas gracias a los avances en el hardware y en bases de datos cada vez más grandes.

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. 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 específico 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 15 muestra las N capas de un esquema al que puede llamársele Deep Learning

Figura 14. Significado de profundidad en Deep Learning



Fuente: elaboración propia.

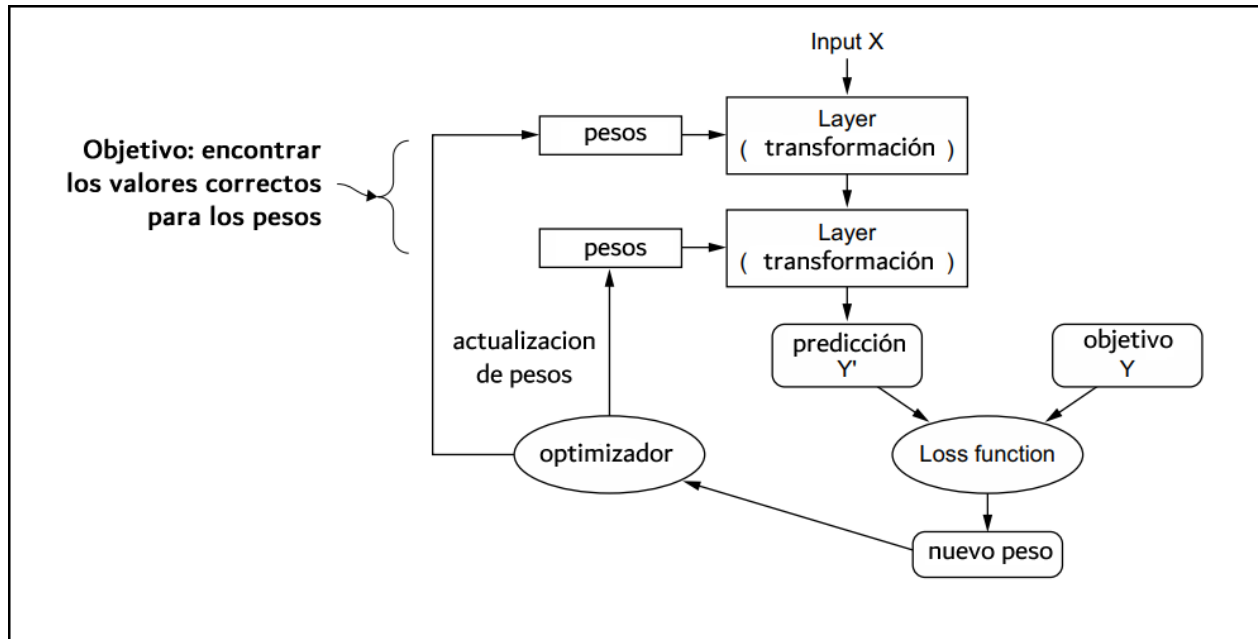
2.4. Redes Neuronales

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

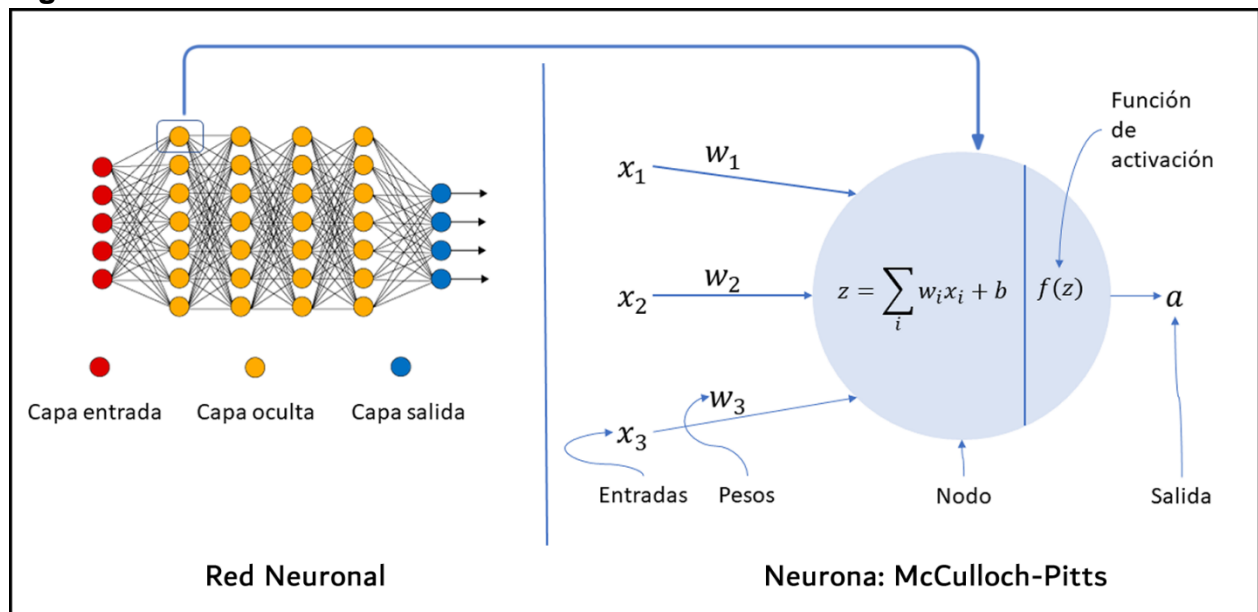
El funcionamiento de una red neuronal se basa en un conjunto de datos conocido como capa de entrada, luego son redirigidos a las capas ocultas las cuales realizan transformaciones a dichos datos cada vez que pasan por ellas. Como lo ilustra la figura 15 la salida de una neurona es la entrada de otra, 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 transformación al dato. La figura 16 muestra este flujo de información a medida que es categorizado por medio de la red y la figura 17 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



Fuente: CHOLLET, Francois. Deep Learning with Python
p. 10

Figura 16. Función de activación



Fuente: PONCE, Jahaziel. Blog personal sobre inteligencia artificial

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 librería Pyautogui que integra un set completo de herramientas para controlar eventos del sistema operativo tales como la interfaz gráfica, teclado y ratón. 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
- Envío 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 específica. 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 18 muestra como la librería toma una captura y la guarda como archivo png.

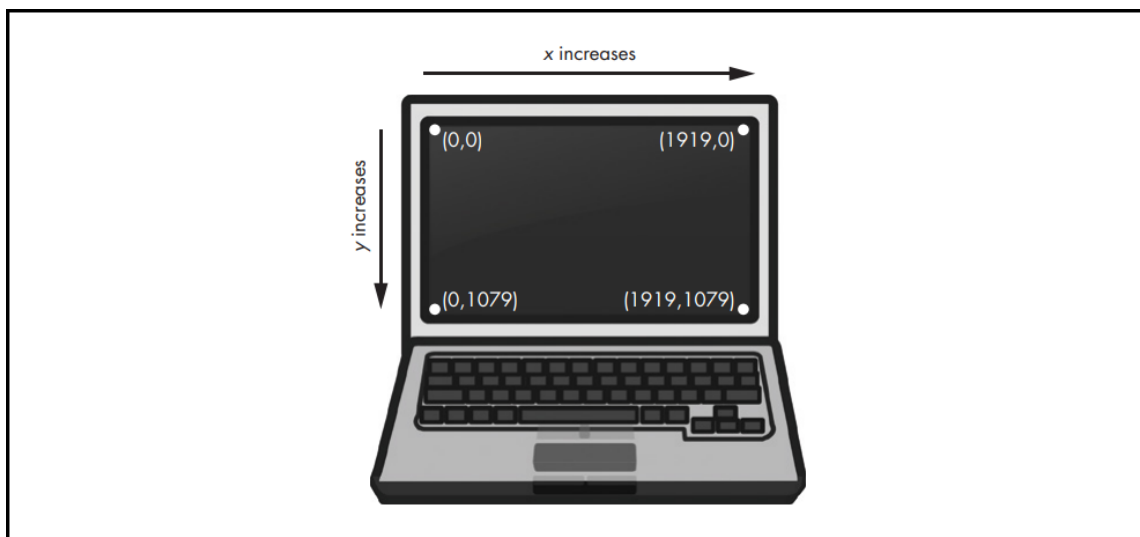
Figura 17. Captura de pantalla mediante la librería

```
main.py U x
exes > main.py > ...
1  import pyautogui
2
3  def main():
4
5      myScreenshot = pyautogui.screenshot()
6      myScreenshot.save(r'./screenshot.png')
7
8  if __name__ == "__main__":
9      main()
10
```

Fuente: 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 resultarían en un error irreparable de la aplicación. La coordenada origen inicia en el punto (0,0) que está ubicado en la esquina superior izquierda. La figura 19 muestra esta descripción.

Figura 18. Sistema coordenado mediante Pyautogui

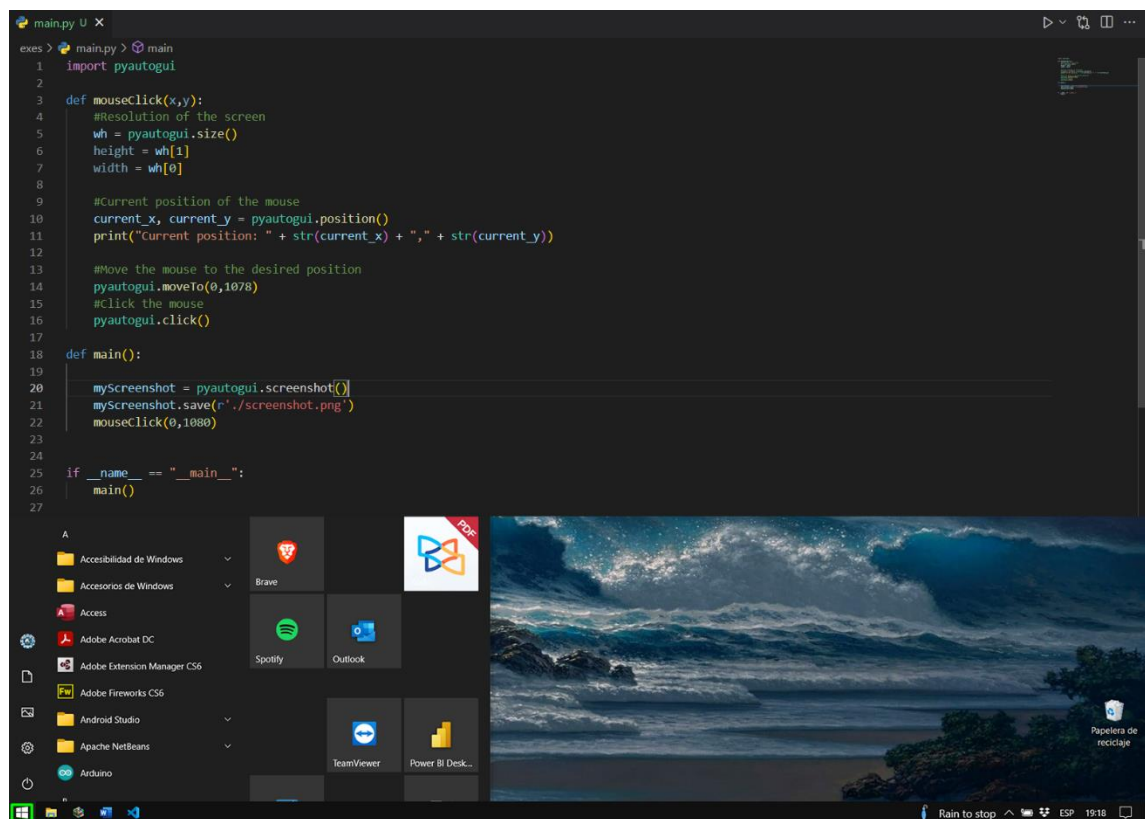


Fuente: SWEIGART, Al. Automate The Boring Stuff with Python

3.3. Manipulación de ventanas y controles

Cuando las coordenadas están establecidas y OpenCV a identificado la posición específica de los controles, entonces es posible programar las interacciones con botones y cajas de texto. **Pyautogui** permite programar pulsaciones mediante una coordenada específica, así mismo tiene retrasos que permiten darle a la aplicación o sistema operativo el tiempo suficiente para procesar la petición, la figura 20 muestra el código necesario para mover el cursor y realizar una pulsación.

Figura 19. Pulsando el botón inicio de Windows mediante coordenadas



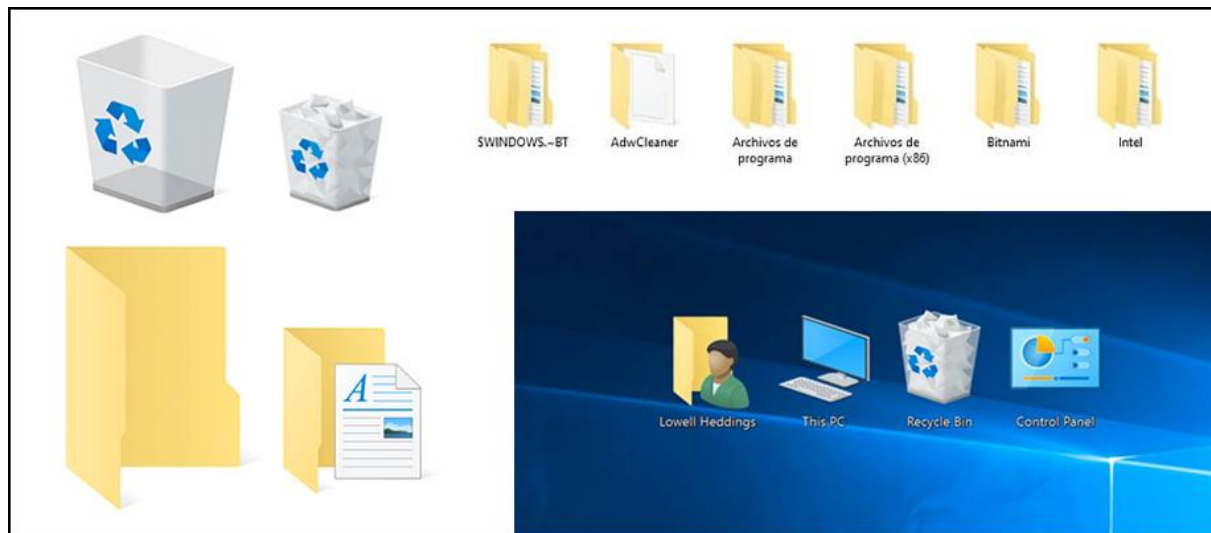
Fuente: elaboración propia.

4. INTEGRACION

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 poder 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.

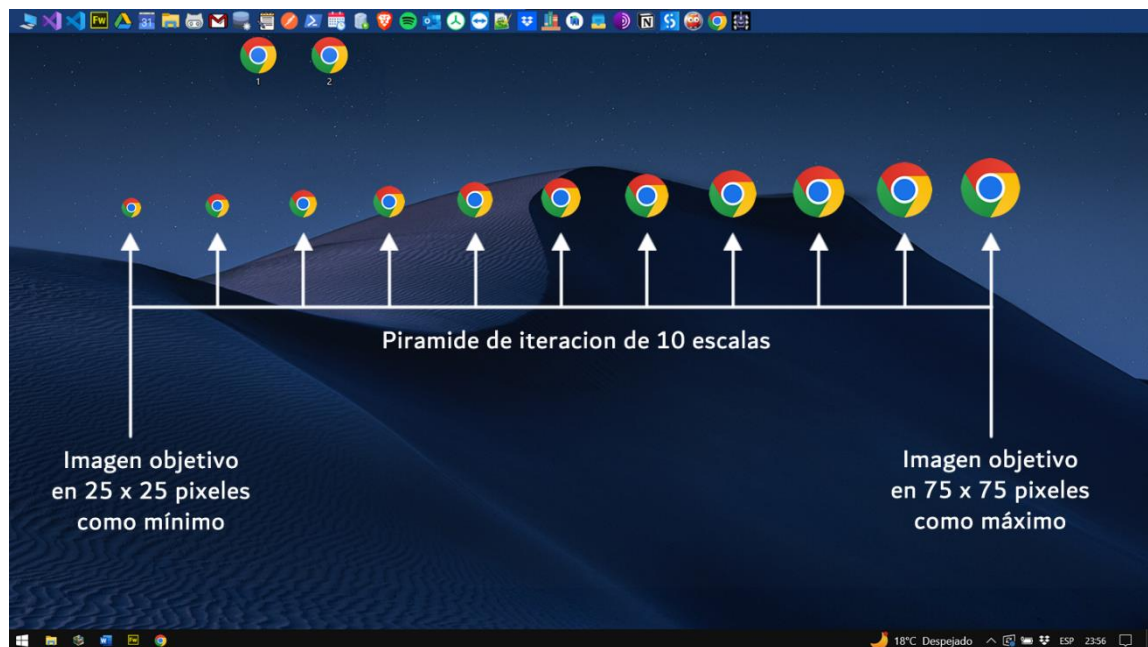
Figura 20. Variantes de un mismo icono, según escala y forma



Fuente: elaboración propia.

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



Fuente: elaboración propia.

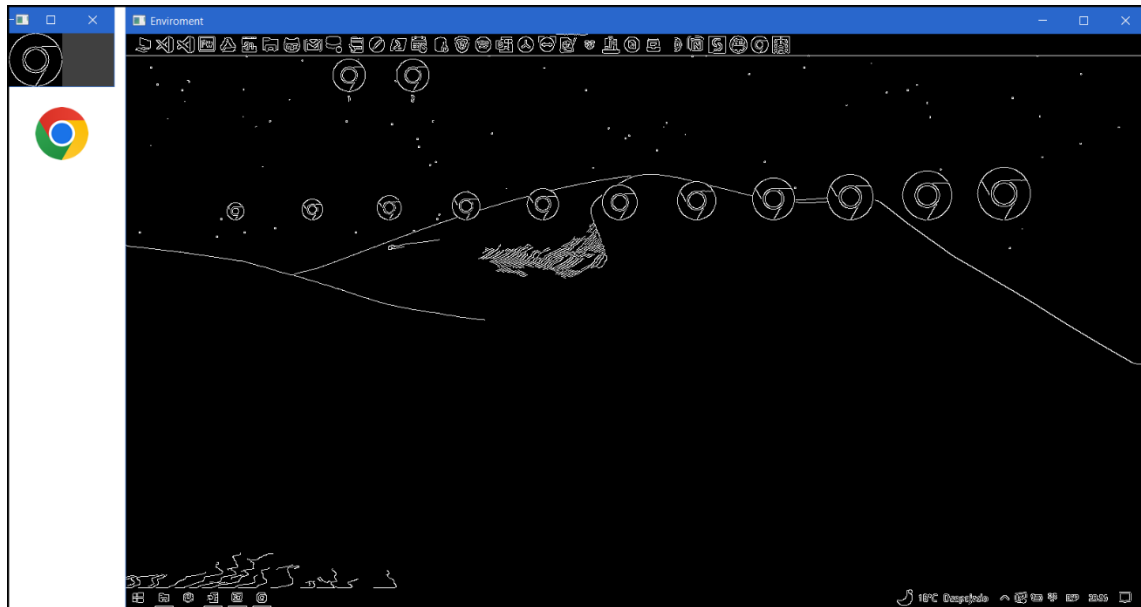
Uno de los problemas de **Template Matching** como detector de objetos es que tiene la desventaja de funcionar solo en casos muy específicos, principalmente cuando la plantilla coincide a la perfección con el objeto de que se busca en la imagen ambiente. De tal manera que si el objeto aparece en menor 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 20 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 canny. La imagen 22 muestra este resultado

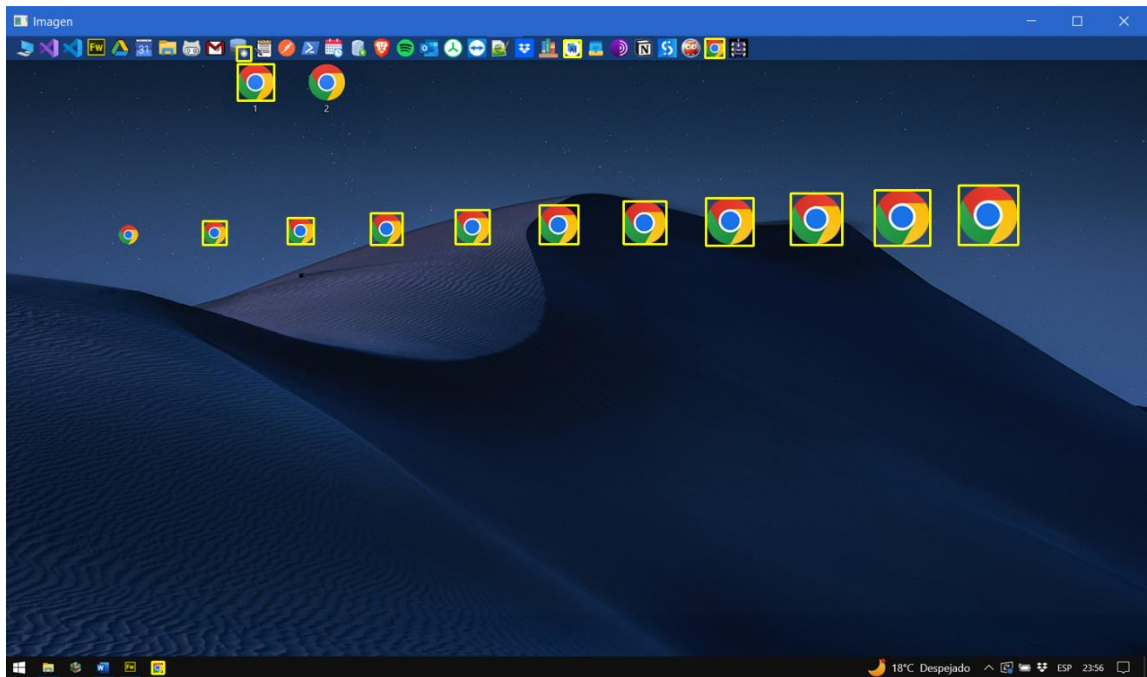
Figura 22. Transformación a escala de grises y bordes



Fuente: 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 poder ver los aciertos de esta ejecución se dibujarán recuadros 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

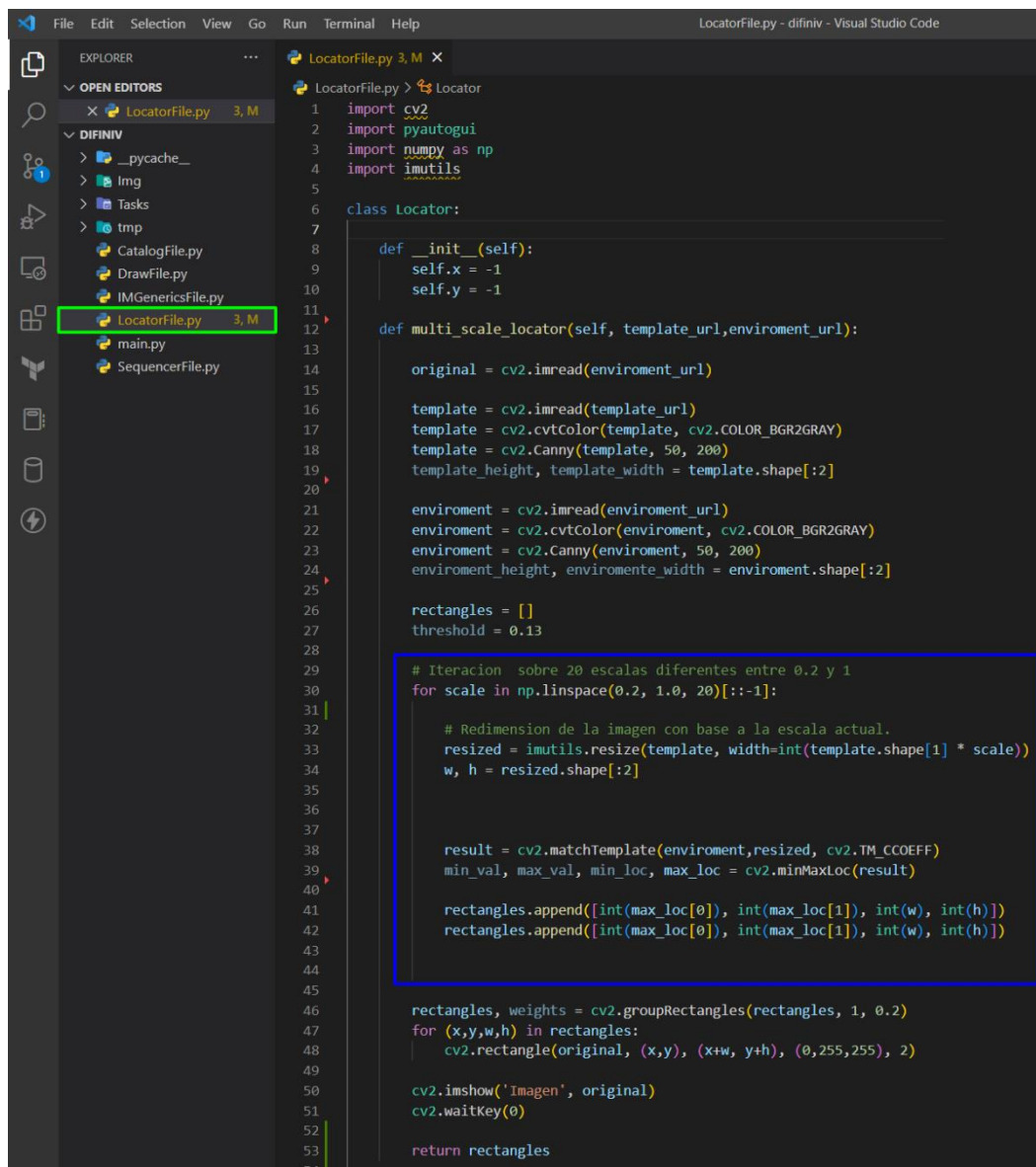


Fuente: elaboración propia.

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

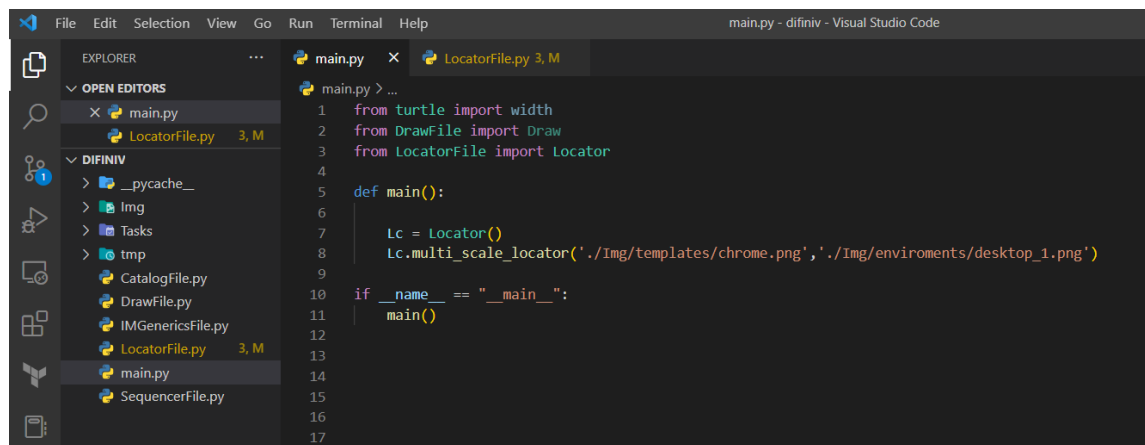


```
LocatorFile.py 3, M X
LocatorFile.py > Locator
1 import cv2
2 import pyautogui
3 import numpy as np
4 import imutils
5
6 class Locator:
7
8     def __init__(self):
9         self.x = -1
10        self.y = -1
11
12    def multi_scale_locator(self, template_url, enviroment_url):
13
14        original = cv2.imread(enviroment_url)
15
16        template = cv2.imread(template_url)
17        template = cv2.cvtColor(template, cv2.COLOR_BGR2GRAY)
18        template = cv2.Canny(template, 50, 200)
19        template_height, template_width = template.shape[:2]
20
21        enviroment = cv2.imread(enviroment_url)
22        enviroment = cv2.cvtColor(enviroment, cv2.COLOR_BGR2GRAY)
23        enviroment = cv2.Canny(enviroment, 50, 200)
24        enviroment_height, enviromente_width = enviroment.shape[:2]
25
26        rectangles = []
27        threshold = 0.13
28
29        # Iteracion sobre 20 escalas diferentes entre 0.2 y 1
30        for scale in np.linspace(0.2, 1.0, 20)[::-1]:
31
32            # Redimension de la imagen con base a la escala actual.
33            resized = imutils.resize(template, width=int(template.shape[1] * scale))
34            w, h = resized.shape[:2]
35
36
37
38            result = cv2.matchTemplate(enviroment, resized, cv2.TM_CCOEFF)
39            min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(result)
40
41            rectangles.append([int(max_loc[0]), int(max_loc[1]), int(w), int(h)])
42            rectangles.append([int(max_loc[0]), int(max_loc[1]), int(w), int(h)])
43
44
45
46        rectangles, weights = cv2.groupRectangles(rectangles, 1, 0.2)
47        for (x,y,w,h) in rectangles:
48            cv2.rectangle(original, (x,y), (x+w, y+h), (0,255,255), 2)
49
50        cv2.imshow('Imagen', original)
51        cv2.waitKey(0)
52
53        return rectangles
54
```

Fuente: elaboración propia.

La figura mostrada anteriormente muestra la estructura del proyecto llamado **difiniv**. 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 24 muestra esta ejecución:

Figura 25. Archivo main donde inicia la ejecución



```
1 from turtle import width
2 from DrawFile import Draw
3 from LocatorFile import Locator
4
5 def main():
6
7     Lc = Locator()
8     Lc.multi_scale_locator('./Img/templates/chrome.png', './Img/enviroments/desktop_1.png')
9
10 if __name__ == '__main__':
11     main()
12
13
14
15
16
17
```

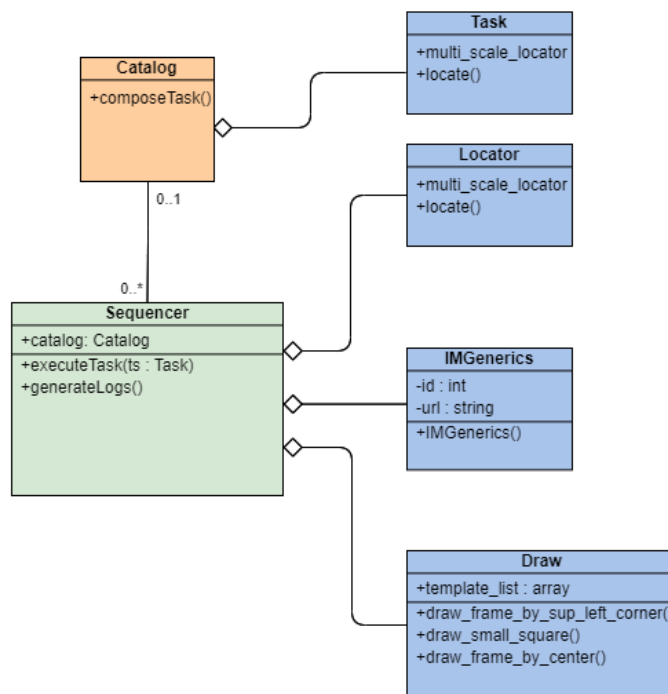
Fuente: elaboración propia.

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 mas 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 coordenadas 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



Fuente: elaboración propia.

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 26 muestra el algoritmo que manipula la interfaz grafica

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)
```

Fuente: elaboración propia.

5. EJECUCION Y REPORTES

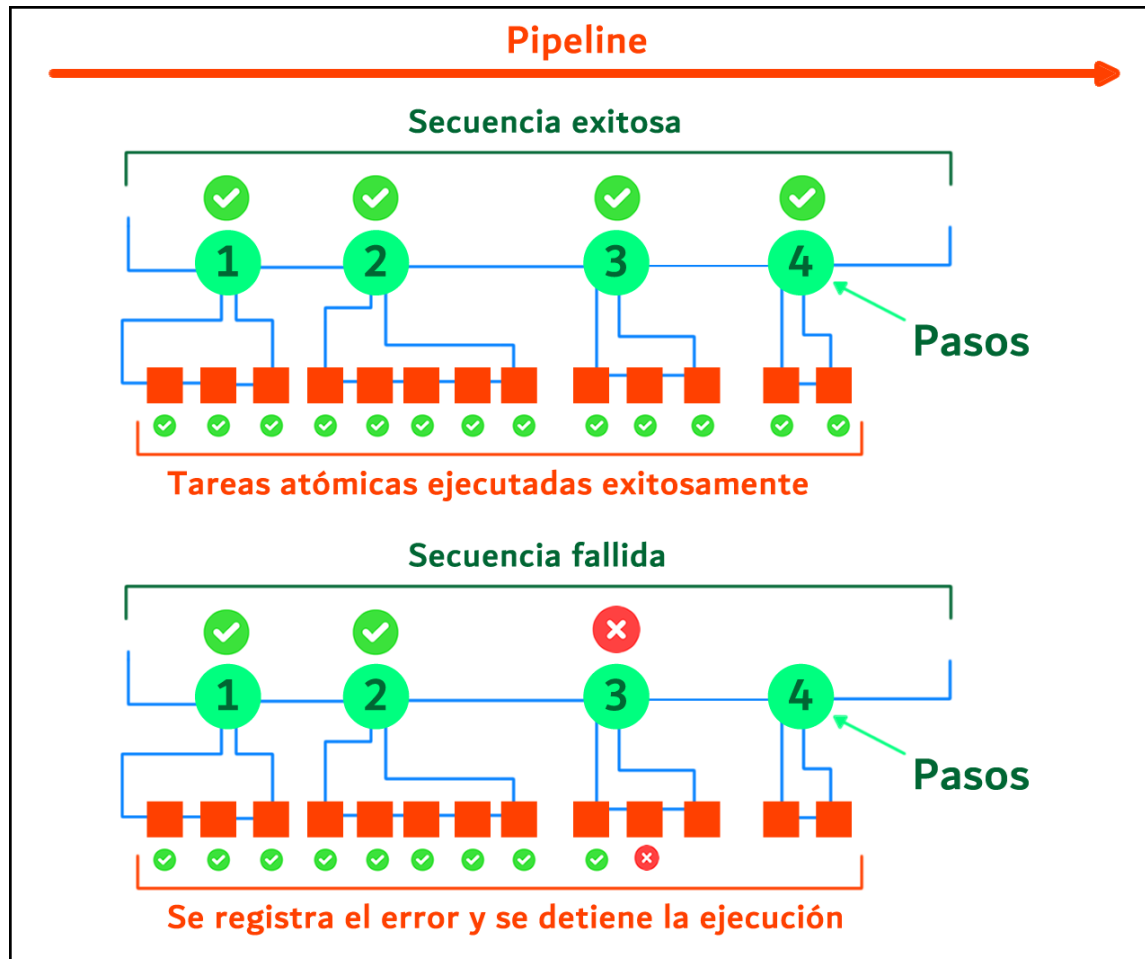
5.1 Bitácora de ejecución del script

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 grafica 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 que momento se puede producir un error. Para comprender mejor esto se puede pensar en 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



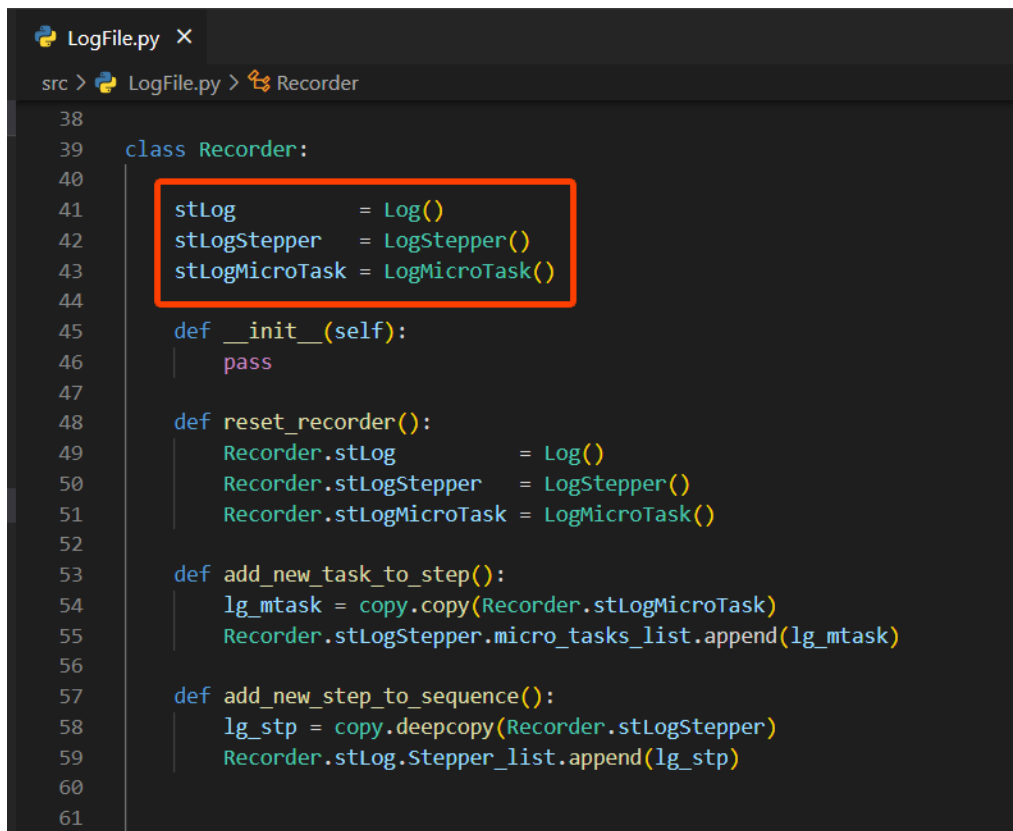
Fuente: elaboración propia.

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 inmediata las acciones del script. Cada paso es registrado, sea exitoso o no mediante el árbol de ejecución.

5.2 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 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 29 muestra cómo funciona esta clase:

Figura 29. Objetos estáticos de la clase Recorder

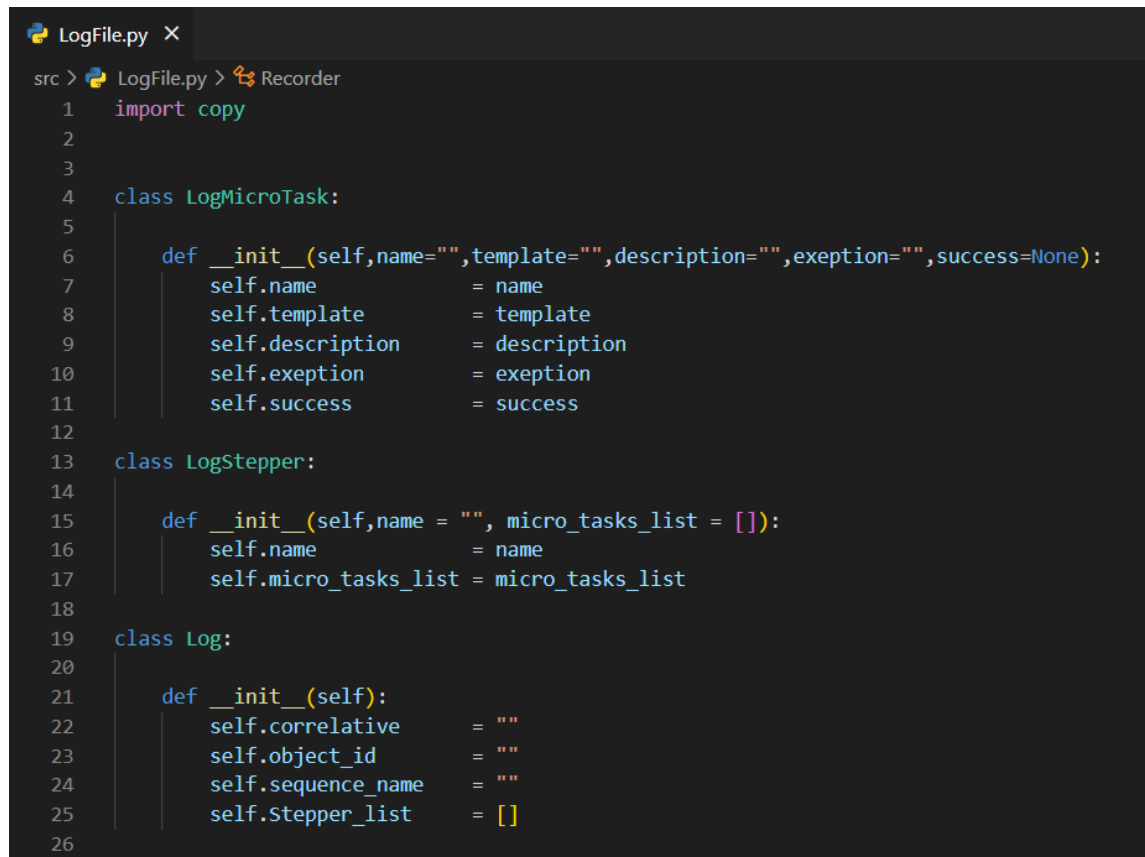


```
LogFile.py X
src > LogFile.py > Recorder
38
39 class Recorder:
40
41     stLog = Log()
42     stLogStepper = LogStepper()
43     stLogMicroTask = LogMicroTask()
44
45     def __init__(self):
46         pass
47
48     def reset_recorder():
49         Recorder.stLog = Log()
50         Recorder.stLogStepper = LogStepper()
51         Recorder.stLogMicroTask = LogMicroTask()
52
53     def add_new_task_to_step():
54         lg_mtask = copy.copy(Recorder.stLogMicroTask)
55         Recorder.stLogStepper.micro_tasks_list.append(lg_mtask)
56
57     def add_new_step_to_sequence():
58         lg_stp = copy.deepcopy(Recorder.stLogStepper)
59         Recorder.stLog.Stepper_list.append(lg_stp)
60
61
```

Fuente: elaboración propia.

Los objetos señalados en la Figura 29 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 30. Estructura de los objetos responsables de la bitácora



```
LogFile.py X
src > LogFile.py > Recorder
1  import copy
2
3
4  class LogMicroTask:
5
6      def __init__(self,name="",template="",description="",exeption="",success=None):
7          self.name          = name
8          self.template      = template
9          self.description    = description
10         self.exeption       = exeption
11         self.success        = success
12
13     class LogStepper:
14
15         def __init__(self,name = "", micro_tasks_list = []):
16             self.name          = name
17             self.micro_tasks_list = micro_tasks_list
18
19     class Log:
20
21         def __init__(self):
22             self.correlative    = ""
23             self.object_id      = ""
24             self.sequence_name  = ""
25             self.Stepper_list   = []
26
```

Fuente: 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 31 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 31. 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()

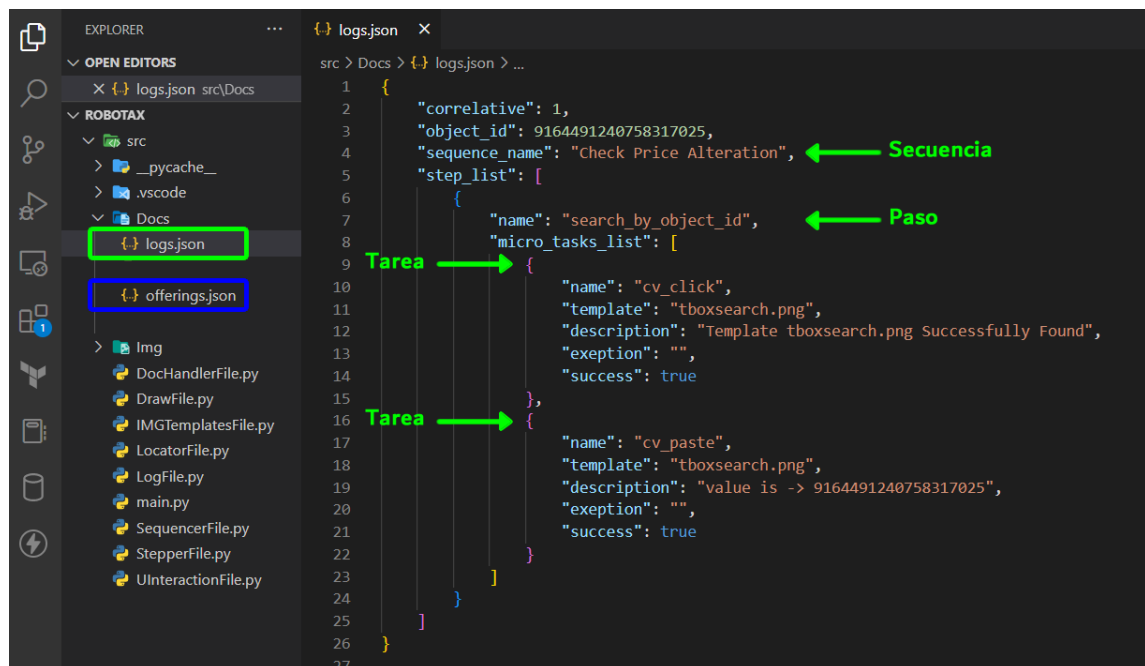
    x,y = ncsearch
```

Fuente: elaboración propia

5.3 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 32 muestra como se ve el reporte de ejecución

Figura 32. Recolección de los registros



Fuente: 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.

Guatemala, 07 de noviembre de 2022

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 SISTEMAS OPERATIVOS MEDIANTE RECONOCIMIENTO DE GRAFICOS UTILIZANDO MACHINE LEARNING”**, lo he revisado y luego de corroborar que el mismo se encuentra con un avance superior al 75 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,



Marco Tulio Aldana Prillwitz
Master in Business Intelligence and Data Analytics
Colegio de Humanidades 30747

El desarrollo explicado en la sección de ejecución y reportes fue actualizado utilizando el mismo repositorio, pero en la carpeta v2. En el siguiente hipervínculo puede encontrarse esta actualización:

Repositorio:

<https://github.com/Gualtix/tisis/tree/master/v2>