UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa

# Estudio e implementación de un algoritmo de resolución del rompecabezas Tangram utilizando visión artificial en una estación con robot colaborativo.

TREBALL DE FI D'ESTUDIS

Documento:
Memoria

Autor:
Marc Roca Ribas

Director/Directora - Codirector/Codirectora:
Marc Flor Sánchez

Titulación:
Grado en Ingeniería Electrónica Industrial y Automática

Convocatoria:
Primavera con prórroga, 2023

# 1 Resumen

Este proyecto aborda el desafío de programar un robot colaborativo (cobot) para la resolución autónoma del rompecabezas Tangram. Con un enfoque integral, el proyecto integra el aprendizaje automático y la visión por ordenador para permitir que el cobot identifique, posicione y ensamble correctamente las piezas del Tangram una vez resuelto el rompecabezas.

**Objetivos**
Los objetivos principales del trabajo han sido diseñar e implementar un conjunto de programas que controlen el proceso completo del cobot, desde la interpretación de las imágenes de las siluetas del Tangram hasta la ejecución física del ensamblaje de las piezas. Esto incluyó la creación de una aplicación para generar y procesar imágenes (problema y solución) del Tangram para el entrenamiento de un algoritmo de aprendizaje automático que pueda resolver rompecabezas Tangram nuevos, y un programa de visión por ordenador para guiar al cobot durante el ensamblaje físico.

**Metodología**
El proyecto se dividió en tareas específicas para facilitar su gestión y ejecución:

1. Desarrollo de una aplicación en Python para la creación de figuras Tangram con el objetivo de crear un dataset de entrenamiento y enviar instrucciones al cobot.
2. Implementación un algoritmo utilizando TensorFlow para entrenar un modelo que pueda resolver el rompecabezas a partir de las imágenes proporcionadas.
3. Diseño de un programa de visión por ordenador que identifica la posición y orientación inicial de las piezas en la estación robótica.
4. Creación de un programa de comunicación y controlador para el cobot, permitiendo la ejecución segura y eficiente del montaje de las piezas.

**Resultados**
Todos los componentes del proyecto se completaron con éxito excepto la resolución de los problemas Tangram por parte del aprendizaje automático. A pesar del esfuerzo dedicado, esta tarea resultó ser mucho más complicada de lo previsto inicialmente. Sin embargo, la aplicación gráfica desarrollada generó eficientemente datos para el entrenamiento del algoritmo y el envío de instrucciones al robot. La integración de la visión por ordenador permitió al cobot identificar y manipular con precisión las piezas físicas de la estación, y el robot consiguió realizar la configuración de piezas recibida por el aplicativo en la estación de forma segura y consistente.

**Conclusiones**
Si bien este proyecto no ha logrado con éxito cada uno de sus objetivos propuestos, sí que ha logrado completar la gran mayoría, así como encarrilar un estudio a posterior, y sus bases, para resolver la programación de una red neuronal capaz de resolver el rompecabezas Tangram. Durante su realización, se han adquirido nuevas y valiosas competencias en el campo del aprendizaje automático, la visión por ordenador, la automatización de robots y la programación en Python.

Este proyecto puede servir como precedente alentador para futuras investigaciones y desarrollos.

# 2 Abstract

This project tackles the challenge of programming a collaborative robot (cobot) for the autonomous resolution of the Tangram puzzle. With a comprehensive approach, the project integrates machine learning and computer vision to allow the cobot to identify, position, and correctly assemble the Tangram pieces once the puzzle is solved.

**Objectives**
The main goals of the work have been to design and implement a set of programs that control the complete process of the cobot, from the interpretation of the images of the Tangram silhouettes to the physical execution of the assembly of the pieces. This included the creation of an application to generate and process images (problem and solution) of the Tangram for the training of a machine learning algorithm that can solve new Tangram puzzles, and a computer vision program to guide the cobot during physical assembly.

**Methodology**
The project was divided into specific tasks to facilitate its management and execution:

1. Development of a Python application for the creation of Tangram figures with the aim of creating a training dataset and sending instructions to the cobot.
2. Implementation of an algorithm using TensorFlow to train a model that can solve the puzzle from the provided images.
3. Design of a computer vision program that identifies the initial position and orientation of the pieces in the robotic station.
4. Creation of a communication program and controller for the cobot, allowing the safe and efficient assembly of the pieces.

**Results**
All components of the project were successfully completed except for the resolution of the Tangram problems by machine learning. Despite the dedicated effort, this task proved to be much more complicated than initially anticipated. However, the developed graphic application efficiently generated data for algorithm training and sending instructions to the robot. The integration of computer vision allowed the cobot to identify and precisely manipulate the physical pieces from the station, and the robot successfully achieved the configuration of pieces received by the application in the station safely and consistently.

**Conclusions**
Although this project has not successfully achieved each of its proposed objectives, it has managed to complete the vast majority, as well as outline a subsequent study, and its basis, to resolve the programming of a neural network capable of solving the Tangram puzzle. During its realization, new and valuable skills have been acquired in the field of machine learning, computer vision, robot automation, and Python programming.

This project can serve as an encouraging precedent for future research and developments.

# 3 Índice

# 4 Índice de tablas



# 5 Índice de gráficas

# 6 Índice de figuras

# 7 Lista de abreviaturas/Glosario

**API:** siglas en inglés de "Application Programming Interface", que es un conjunto de herramientas y protocolos que permiten a los desarrolladores crear software y aplicaciones que se comunican con otros sistemas.

**Aprendizaje Automático:** rama de la inteligencia artificial que permite a las máquinas aprender de manera autónoma a partir de datos y experiencia, sin ser programadas explícitamente.

**CAE (Convolutional Autoencoder):** Variante de autoencoder que utiliza capas convolucionales. Especialmente eficiente para procesar imágenes y conservar estructuras espaciales.

**CNN (Convolutional Neural Network):** Tipo de red neuronal artificial inspirada en el sistema visual biológico, siendo ampliamente utilizada para el reconocimiento de imágenes.

**Cobots:** robots colaborativos diseñados para trabajar junto con los humanos en un entorno de trabajo compartido.

**Dataset:** Conjunto de datos estructurados, normalmente utilizado para análisis o para entrenar modelos de machine learning. Los datasets pueden constar de múltiples entradas, y cada entrada puede contener múltiples características o atributos.

**FlexPendant:** Es una interfaz de control táctil que se utiliza en robótica, especialmente con robots industriales, para facilitar la programación, monitorización y control del robot.

**GAN (Generative Adversarial Network):** Modelo de aprendizaje automático donde dos redes neuronales, la generadora y la discriminadora, se entrenan conjuntamente. La generadora intenta crear datos mientras que la discriminadora intenta diferenciar entre datos reales y generados.

**MAE (Mean Absolute Error):** Medida estadística que calcula el promedio de las diferencias absolutas entre las predicciones y los valores reales. Indica cuán grandes son los errores en términos absolutos, sin considerar su dirección.

**MSE (Mean Squared Error):** Medida estadística que representa la discrepancia entre los valores estimados y los valores verdaderos. Es el promedio de los cuadrados de las diferencias entre estos dos conjuntos de valores.

**Matplotlib:** biblioteca de Python utilizada para crear visualizaciones y gráficos de datos de manera fácil y rápida.

**Numpy:** biblioteca de Python utilizada para realizar cálculos científicos y matemáticos avanzados, incluyendo el procesamiento de matrices y vectores.

**OpenCV:** biblioteca de código abierto para procesamiento de imágenes y visión por ordenador, que ofrece una amplia variedad de herramientas y algoritmos para analizar y manipular imágenes.

**Pick and Place:** técnica utilizada en robótica y automatización para recoger objetos y colocarlos en otra ubicación.

**PyCharm:** Entorno de desarrollo integrado (IDE) para el lenguaje de programación Python. Ofrece herramientas para la codificación, depuración y pruebas de programas Python.

**Python:** lenguaje de programación de alto nivel y de propósito general, ampliamente utilizado en robótica, aprendizaje automático, visión por ordenador y otros campos de la informática.

**Red Neuronal Convolucional:** tipo de red neuronal especialmente diseñada para procesar datos de imágenes y reconocimiento de patrones.

**RGB/BGR:** RGB se refiere a Rojo-Verde-Azul y es el modelo estándar para la representación de colores en imágenes digitales y en la visualización de monitores. BGR es una variación del orden de los colores utilizado en ciertas bibliotecas y aplicaciones de procesamiento de imágenes.

**TensorFlow:** biblioteca de código abierto para el aprendizaje automático y la inteligencia artificial desarrollada por Google.

**Tangram:** juego de rompecabezas chino que ha sido utilizado para enseñar y demostrar conceptos de programación y visión por ordenador.

**TCP/IP (Transmission Control Protocol/Internet Protocol):** Conjunto de protocolos de comunicación utilizados para la transmisión de datos en redes de computadoras. Es el protocolo estándar para internet y otras redes.

**URScript:** Lenguaje de programación desarrollado específicamente para los robots de Universal Robots. Es utilizado para escribir scripts que controlen y programen tareas específicas que se quieren que el robot realice.

**VAE (Variational Autoencoder):** Un tipo de autoencoder que produce una codificación probabilística de las entradas, y que se utiliza especialmente en el modelado generativo.

**Visión por Ordenador:** campo de la inteligencia artificial que se centra en permitir a las máquinas "ver" y comprender el mundo visual que las rodea.

# 8 Introducción

## 8.1 Objeto

El objetivo del trabajo consiste en el diseño e implementación de un conjunto de programas que controlen el proceso por el cual un robot colaborativo (también llamado cobot) resolverá un rompecabezas dado. Para ello el alumno deberá aprender y familiarizarse con la programación de robots colaborativos, la visión artificial y los algoritmos de aprendizaje automático (machine learning en inglés).

El rompecabezas elegido es el juego chino Tangram, que consiste en determinar la posición de un conjunto de piezas geométricas a partir de la silueta de la figura que forman entre ellas.

Inicialmente se realizará la captura y el tratamiento de las imágenes de la silueta del conjunto de piezas virtualmente mediante una aplicación. Estas imágenes servirán para alimentar un algoritmo de aprendizaje automático capaz de resolver la formación de las piezas.

Una vez se disponga de la información de la posición de las piezas, será necesario convertir esta información en información utilizable por el cobot para que éste realice la secuencia "pick and place" de las piezas para formar la silueta problema físicamente en la estación.

Para ello será necesario usar la visión por ordenador para identificar correctamente la ubicación y orientación (inicial y final) de cada una de las piezas en la estación robótica.

## 8.2 Alcance

Para la consecución del objetivo del proyecto se divide el trabajo en los siguientes paquetes entregables necesarios:

### Tarea A

Diseño e implementación de una aplicación para crear figuras tangram mediante Python. El objetivo es familiarizarse con las distintas APIs y librerías de Python para interfaces gráficas y el tratamiento de imágenes.

Esta aplicación servirá para crear el conjunto de datos con el que alimentar el programa de aprendizaje automático por supervisión, utilizando las siluetas como problema de entrada y su solución de posición y orientación correspondiente como salida. De forma similar, se utilizará como interfaz para crear nuevos problemas de testeo que enviar al cobot una vez terminado el proyecto.

### Tarea B

Diseño e implementación del algoritmo de resolución del rompecabezas mediante un modelo de aprendizaje automático por supervisión creado con TensorFlow (una API de Python) y el conjunto de datos creado anteriormente mediante los siguientes pasos.

- **Recopilar y preparar datos:** será necesario recopilar un conjunto de datos de entrenamiento que contenga imágenes de los diferentes rompecabezas de Tangram y sus soluciones correspondientes. Es importante preprocesar las imágenes para transformar y extender el conjunto de datos inicial mediante las herramientas de preprocesamiento de datos que ofrece Tensorflow. También se reservará una partición del conjunto de datos con el que el modelo no se entrenará a fin de poder validar su precisión con datos nuevos.

- **Crear el modelo:** construir la red neuronal convolucional para el procesado de las imágenes. El modelo debe tener una capa de entrada que acepte imágenes, varias capas convolucionales y de pooling para extraer características de las imágenes y una capa de salida que produzca las soluciones del Tangram. Será importante ajustar los hiperparámetros, como la tasa de aprendizaje y el número de filtros en las capas convolucionales, para obtener un modelo óptimo.

- **Entrenamiento del modelo:** durante el entrenamiento, la red neuronal ajustará los pesos y sesgos de las capas para minimizar la función de pérdida, que mide la diferencia entre las soluciones producidas por el modelo y las soluciones reales. Es importante tener en cuenta que el entrenamiento puede llevar mucho tiempo y requiere de una computadora con potencia de procesamiento adecuada.

- **Validar y ajustar el modelo:** después de entrenar el modelo, se deberá validar su precisión utilizando un conjunto de datos de validación. Si la precisión es baja, se pueden realizar ajustes en el modelo, como modificar los hiperparámetros o agregar más capas, y volver a entrenarlo.

- **Probar el modelo:** probar con el conjunto de datos de validación para evaluar su capacidad para generalizar y resolver nuevos rompecabezas de Tangram. El modelo solo podrá resolver rompecabezas similares a los que ha visto en el conjunto de datos

de entrenamiento, por lo que es importante asegurarse de que el conjunto de datos de entrenamiento sea lo suficientemente diverso (en la preparación de los datos).

- **Desplegar el modelo:** integración del modelo entrenado en una aplicación que envíe los datos de salida para que el cobot pueda iniciar su tarea de componer físicamente la solución del rompecabezas.

### Tarea C

Diseño e implementación de un programa de visión por computador utilizando Python para identificar la posición inicial y final de las piezas a mediante una cámara a partir de los siguientes pasos:

- Familiarización e instalación de las bibliotecas necesarias, como podrían ser OpenCV para el procesamiento de imágenes, Numpy para el cálculo numérico y/o Matplotlib para la visualización de imágenes y gráficos.
- Captura de la imagen del tablero físico con las piezas en la estación robótica mediante una cámara conectada al sistema.
- Preprocesar la imagen para reducir el ruido y resaltar los bordes de las piezas utilizando técnicas como el suavizado gaussiano, la umbralización y/o la detección de bordes de Canny.
- Detectar las piezas mediante la segmentación de la imagen con el uso de técnicas como la detección de contornos y la eliminación de falsos positivos mediante el área y la forma de los contornos.
- Calcular la posición y orientación de cada pieza mediante el análisis de la geometría y la relación entre las piezas.
- Enviar los resultados de posición y orientación mediante un sistema de comunicación con el programa controlador del cobot.

### Tarea D

Diseño e implementación de un programa controlador para que el cobot pueda realizar la tarea de pick & place de las piezas. Este código deberá incluir:

- Medidas de detección de obstáculos y proximidad mediante sensores y actuadores con tal de garantizar la integración segura entre usuario y robot.
- Algoritmos para la planificación de trayectorias y la prioridad del movimiento de cada pieza, así como la necesidad de voltear la pieza romboide.
- La comunicación con el sistema de visión por computador para comprobar el siguiente paso en cada iteración.
- La verificación de la posición final de las piezas con la posición objetivo.

### Tarea E

Búsqueda de información, testear y extraer resultados, crear documentación del trabajo realizado mediante una memoria.

## 8.3 Requerimientos

La solución final deberá cumplir los siguientes requisitos y especificaciones:

1. **Seguridad**. Este es el requisito más importante del proyecto, se garantizará que en todo momento el proceso del robot es seguro para los usuarios en la estación durante toda la duración del proyecto.

2. La aplicación que sirve como interfaz de usuario a la hora de proporcionar un nuevo rompecabezas al robot deberá ser intuitiva para facilitar el uso del sistema, así como permitir la creación de Tangrams con libertad.

3. El algoritmo de aprendizaje automático deberá ser capaz de resolver la formación de las piezas de Tangram a partir de la imagen de entrada facilitada con una precisión alta significativa.

4. La aplicación de visión por ordenador que captura y efectúa el tratamiento de la imagen deberá identificar la posición y orientación de cada una de las piezas en la estación robótica independientemente de su posicional inicial, siempre y cuando las piezas no estén superpuestas. También deberá ser capaz de reconocer si faltan piezas o si es necesario voltear la pieza romboide.

5. El robot debe ser capaz de realizar la secuencia "pick and place" de las piezas con seguridad para el usuario, completando la figura con precisión, y sin apilarlas o dañarlas.

Será necesario documentar el diseño, implementación y funcionamiento del conjunto de programas, así como pruebas exhaustivas para garantizar la eficacia y fiabilidad del sistema en la resolución del Tangram.

## 8.4 Justificación

Cobots, inteligencia artificial y visión por ordenador son parte de la industria 4.0, componentes importantes de la evolución de la robótica industrial tradicional, que aprovecha el avance de las tecnologías digitales para crear robots más inteligentes, eficientes y flexibles.

A nivel global, los proyectos sobre robots colaborativos e inteligencia artificial son necesarios porque pueden ser un paso importante para mejorar la eficiencia y la seguridad en los lugares de trabajo de forma significativa, pueden aumentar la calidad de los productos, reducir el impacto ambiental de los procesos industriales y ayudar a las empresas a ser más competitivas en una economía global cada vez más competitiva.

Este proyecto integra y conecta varios de los componentes claves en esta nueva era de la robótica, que como ya se ha mencionado, son la robótica colaborativa, la inteligencia artificial y la visión por ordenador.

Se ha elegido el rompecabezas chino Tangram como problema arbitrario por ser un ejercicio a resolver especifico con el que empezar a trabajar, pero similar al que nos podríamos encontrar en algunos procesos industriales de producción o logísticos. Es decir, aplicando los criterios parecidos de resolución, podríamos resolver otros casos que, aunque específicos, sean similares a nivel de planteamiento.

Ejemplos de problemas similares podrían ser la colocación de piezas, productos o cajas en los espacios libres de un contenedor, blíster o pale. La detección de una incorrecta formación de estos y su correspondiente solución sería otro ejemplo.

Así mismo, el hecho de familiarizarse con la tecnología y los lenguajes de programación necesarios para resolver esta tarea especifica, y la adquisición de las competencias correspondientes, prepara al estudiante para adaptarse a nuevos retos que pueda plantear la industria ya no solo en el futuro, sino también en el presente.


*Figura 1. Industria 4.0*

# 9 Antecedentes y/o revisión del estado de la cuestión

Esta sección se propone explorar la historia y evolución del tangram, así como su relación y aplicabilidad en campos modernos como el aprendizaje automático, la visión por ordenador y la robótica colaborativa. A través de este pequeño análisis, se busca entender cómo un juego tradicional puede ofrecer *insights* y desafíos para la tecnología moderna y cómo, a través de la convergencia de estas disciplinas, se puede mejorar y enriquecer la comprensión humana sobre problemas complejos y su solución.

## 9.1 Tangram

El tangram es un juego de origen chino que consiste en formar figuras geométricas utilizando un set de siete piezas (también llamadas "tans") de diferentes formas. El set de figuras está compuesto por cinco triángulos (dos grandes, uno mediano y dos pequeños), un cuadrado y un paralelogramo (romboide).

Todas las piezas se originan a partir del cuadrado, que se considera la unidad de medida:

- *1 Cuadrado*
  - *Tamaño de los lados: 1*
  - *Área: 1*
- *1 Romboide*
  - *Tamaño de los lados: 1 y √2*
  - *Altura: √2/2*
  - *Área: 1*
- *2 Triángulos rectángulos isósceles grandes*
  - *Tamaño hipotenusa: 2√2*
  - *Tamaño de los catetos: 2*
  - *Área: 2*
- *1 Triángulo rectángulo isósceles mediano*
  - *Tamaño hipotenusa: 2*
  - *Tamaño de los catetos: √2*
  - *Área: 1*
- *2 Triángulos rectángulos isósceles pequeños*
  - *Tamaño hipotenusa: √2*
  - *Tamaño de los catetos: 1*
  - *Área: ½*

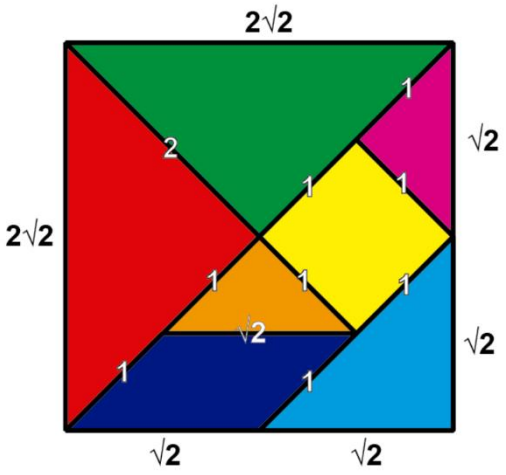



*Figura 2. Dimensiones Tangram*

El área total de todas las piezas siempre sumará 8.

De estas siete piezas, el romboide es único, en el sentido de que no tiene simetría de reflexión, sino solo simetría de rotación, por lo que su imagen especular solo se puede obtener dándole la vuelta. Por lo tanto, es la única pieza que puede necesitar voltearse al formar ciertas formas.

El objetivo del tangram es utilizar todas las piezas sin solaparlas para formar figuras planas, como animales, objetos, letras, números, etc. Se considera una actividad educativa que fomenta la creatividad, el pensamiento lógico, la resolución de problemas y la percepción espacial.

El tangram se ha convertido en un juego popular en todo el mundo y se utiliza en el ámbito educativo como una herramienta para enseñar geometría, matemáticas y habilidades cognitivas. Además, el tangram ha inspirado numerosos juegos y rompecabezas que utilizan la unión de las piezas geométricas para crear figuras y patrones de siluetas, con el objetivo de resolver que configuración de piezas da lugar a esa silueta.

Estos rompecabezas son los que se tratarán de resolver en este proyecto mediante el uso de aprendizaje automático, siendo este un problema con una complejidad muy elevada, ya que para cada silueta puede haber varias soluciones, y en algunos casos la similaridad entre los problemas puede llevar a confusión, como en el caso de las **paradojas**.

### 9.1.1 Paradojas tangram

Una paradoja del tangram es una falacia aparente en la composición de figuras: dos figuras compuestas con el mismo conjunto de piezas, una de las cuales parece ser un subconjunto propio de la otra.

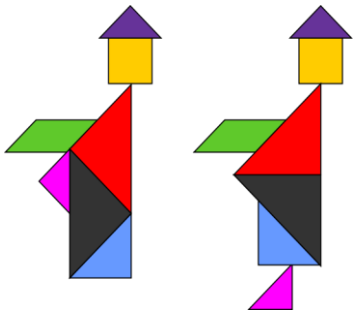


*Figura 3. Paradoja de los dos monjes*

Una paradoja famosa es la de los dos monjes, atribuida a Dudeney, que consta de dos formas similares, una con un pie y la otra sin pie. En realidad, el área del pie está compensada en la segunda figura por un cuerpo sutilmente más grande.

Explicación de la paradoja de los dos monjes:

En la figura 1, las longitudes de los lados están etiquetadas asumiendo que el cuadrado tiene lados unitarios.

En la figura 2, la superposición de los cuerpos muestra que el cuerpo sin pies es más grande por el área del pie. El cambio en el área a menudo pasa desapercibido ya que√2 está cerca de 1.5.

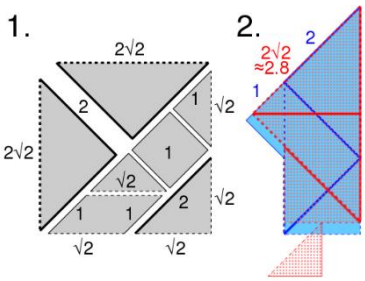



*Figura 4. Explicación paradoja*

En otra paradoja del tangram, Magic Dice Cup, cada una de las copas se compone utilizando las mismas siete formas geométricas. Pero la primera copa está entera, y las otras contienen huecos de diferentes tamaños.

La figura de la izquierda es un poco más corta que las otras dos. La del medio es ligeramente más ancha que la de la derecha, y la de la izquierda es aún más estrecha.

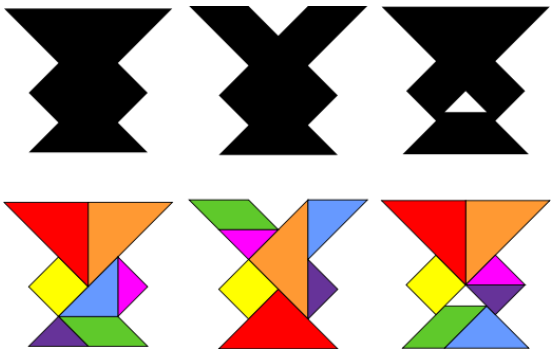


*Figura 5. Paradoja Magic Dice Cup*

## 9.2 Aprendizaje Automático y Redes Convolucionales

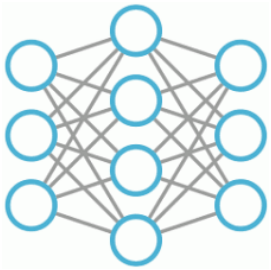


*Figura 6. Red neuronal*

El Aprendizaje Automático, un subcampo de la Inteligencia Artificial (IA), ha sido objeto de estudio por varias décadas. Durante este tiempo, ha evolucionado de modelos simples a sistemas más sofisticados y robustos que pueden aprender de la experiencia y mejorar su rendimiento de manera autónoma. Un desarrollo crucial en este campo ha sido el de las Redes Neuronales Convolucionales (CNN, por sus siglas en inglés), una técnica de Aprendizaje Profundo (Deep Learning) especializada en el procesamiento de imágenes.

Las CNN se inspiran en el modelo biológico del sistema visual humano y actualmente es una herramienta fundamental en la Visión por Ordenador, capaz de identificar y clasificar objetos en imágenes y videos con un alto grado de precisión.

Desde entonces, han sido desarrolladas distintas arquitecturas de CNN como VGG, ResNet, Inception y EfficientNet, expandiendo las posibilidades de las aplicaciones de la Visión por Ordenador. Las mejoras en la eficiencia computacional y el crecimiento exponencial de los conjuntos de datos de imágenes disponibles han permitido la proliferación de estas técnicas.

## 9.3 Visión por Ordenador

La Visión por Ordenador es otro subcampo de la IA que se centra en dar a las máquinas la capacidad de "ver" e interpretar visualmente el mundo. Aunque el campo se remonta a los años 60, ha sido la combinación de hardware más potente, grandes cantidades de datos visuales y algoritmos avanzados de aprendizaje automático lo que ha llevado a los recientes avances significativos.

La Visión por Ordenador abarca una amplia gama de aplicaciones, desde el reconocimiento de imágenes y la detección de objetos hasta la segmentación semántica y la reconstrucción 3D. En términos de evolución, los primeros sistemas se basaban en técnicas de procesamiento de imágenes y patrones de reconocimiento más básicos. Sin embargo, con la introducción de técnicas de aprendizaje profundo, la Visión por Ordenador ha hecho un gran salto en términos de rendimiento y aplicabilidad.

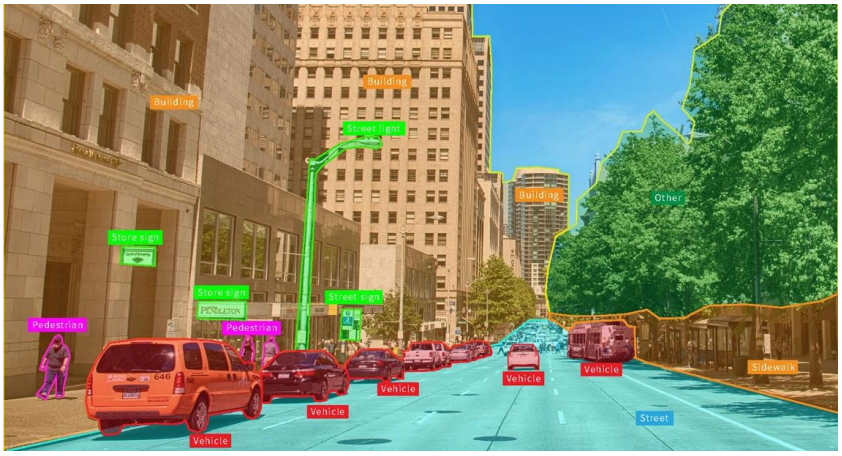



*Figura 7. Visión por Ordenador: identificación de objetos y segmentación semántica de imágenes*

Hoy en día, estos sistemas están presentes en una multitud de aplicaciones en la vida cotidiana, como cámaras de vigilancia inteligentes, sistemas de identificación biométrica, asistentes personales, vehículos autónomos, y más.

## 9.4 Robots Colaborativos

Los robots colaborativos, o cobots, son un desarrollo relativamente reciente en el campo de la robótica. Estos robots están diseñados para interactuar y trabajar en proximidad directa con los humanos en un espacio compartido, o directamente en colaboración con ellos en tareas.

La robótica colaborativa difiere de la robótica tradicional en que se enfoca en la seguridad y la interacción con los humanos, en lugar de solo en la eficiencia y la automatización de tareas. Los primeros cobots fueron introducidos a finales de los años 90 y principios del 2000, pero es en la última década donde su desarrollo ha despegado.

Los cobots modernos incorporan una variedad de tecnologías, incluyendo en algunos casos la Visión por Ordenador, para permitirles interactuar de manera segura y efectiva con los humanos y su entorno. Se han utilizado en una variedad de contextos, desde la fabricación y la logística hasta la asistencia sanitaria y el cuidado personal.

Las Redes Neuronales Convolucionales y los algoritmos de Visión por Ordenador han desempeñado un papel esencial en el desarrollo de los cobots, permitiéndoles "ver" y entender su entorno para interactuar de forma más eficiente y segura.

La robótica colaborativa, impulsada por el Aprendizaje Automático y la Visión por Ordenador, representa una evolución de la robótica hacia sistemas que pueden trabajar en armonía con los humanos, en lugar de reemplazarlos. Este campo continúa evolucionando a medida que avanzamos en el desarrollo de algoritmos más sofisticados y en la integración de estos sistemas en diversos entornos de trabajo.

*Figura 8. Robots colaborativos*

# 10 Descripción

En la siguiente sección, se proporciona una descripción detallada de las utilidades (tecnológicas o no) implementadas en el desarrollo de este proyecto, categorizadas en dos grandes conjuntos: hardware y software. Los componentes de hardware se refieren a los equipos físicos empleados en el proyecto, abarcando desde dispositivos de computación como ordenadores personales, hasta entidades robóticas, cámaras y las piezas que componen el juego Tangram. Por otro lado, el software hace referencia a cualquier conjunto intangible de instrucciones o programas de computación que se han ejecutado en un sistema informático con propósitos específicos.

## 10.1 Descripción del hardware

A continuación, se detallan las herramientas físicas utilizadas durante el proyecto con una breve descripción y como han sido utilizados durante el trabajo.

### 10.1.1 Piezas físicas Tangram

Se aprovechan las piezas de un juego infantil como conjunto de piezas que utilizará el robot en su tarea de pick & place y que tendrá que capturar la cámara por visión por ordenador.

Las piezas son de madera, todas tienen una altura de 5 mm y las siguientes medidas:

- *Cuadrado, 1 unidad*
  - *Tamaño de los lados: 40 mm.*
  - *Área: 1600 mm2.*
  - *Peso:*
- *Romboide, 1 unidad*
  - *Tamaño de los lados: 40 mm y 56,56 mm.*
  - *Altura: 113,14 mm.*
  - *Área: 1600 mm2.*
  - *Peso:*
- *Triángulos rectángulos isósceles grandes, 2 unidades*
  - *Tamaño hipotenusa: 113,14 mm.*
  - *Tamaño de los catetos: 80 mm.*
  - *Área: 3200 mm2.*
  - *Peso:*
- *Triángulo rectángulo isósceles mediano, 1 unidad*
  - *Tamaño hipotenusa: 80 mm.*
  - *Tamaño de los catetos: 56,56 mm.*
  - *Área: 1600 mm2.*
  - *Peso:*
- *Triángulos rectángulos isósceles pequeños, 2 unidades*
  - *Tamaño hipotenusa: 56,56 mm.*
  - *Tamaño de los catetos: 40 mm.*
  - *Área: 800 mm2.*
  - *Peso:*

*Figura 9. Piezas físicas Tangram*

El área total de todas las piezas siempre sumará 12800 $mm^2$.

Debido a las limitaciones de la pinza del robot (descrita mas adelante), se decide crear unos pequeños agarres para que la herramienta pueda manipular las piezas de forma sencilla y consistente. Se diseñan en **Fusion360** (programa sencillo de diseño 3D gratuïto) dos piezas muy simples (un rectangulo y un cuadrado) con el tamaño adecuado para nuestro propósito. El cuadrado se utilizará en la pieza tangram cuadrado, y el resto de piezas llevarán el rectangulo. Se la añade un pequeño orificio a las piezas que servirá de guía para fijarlas con un tornillo tirafondos.

El objetivo es que la pinza del robot agarre las piezas por los lados que hacen 18,5 mm, que es la distancia media que recorren las pinzas.

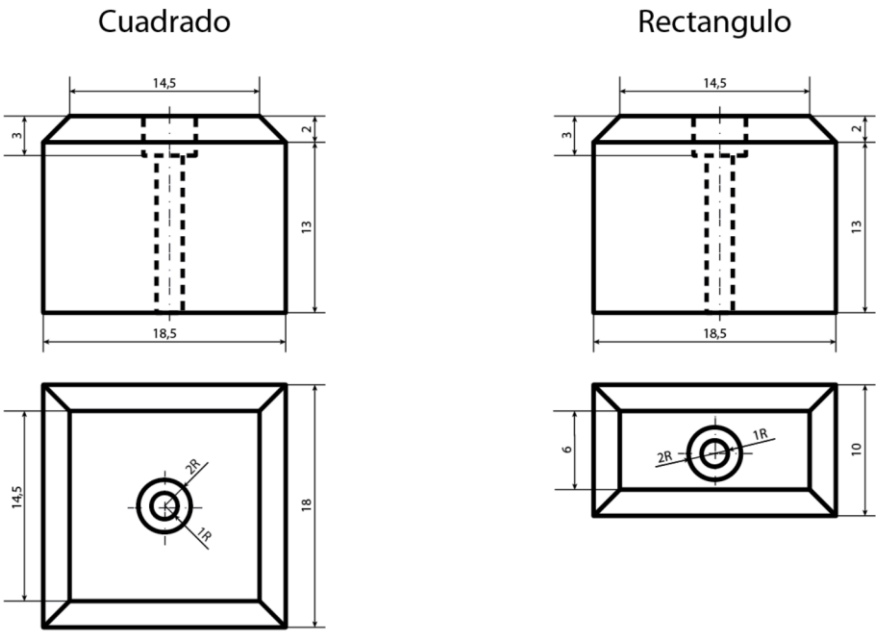



*Figura 10. Medida de las piezas de agarre impresas en 3D*

El diseño se exporta a STL y las piezas se imprimen en PLA de color blanco en una impresora **Creality CR-10**, con una altura de capa de 0,4 mm y un relleno del 15%.

Las piezas se unirán al los tans mediante un pequeño tirafondos, midiendo antes los centroides de los tans y creando un agujero que sirva de guía y conecte con el agujero guía de las piezas impresas.

*Figura 11. Imágenes de la unión de las piezas.*

De esta manera las piezas quedan bien fijadas y listas para ser usadas.

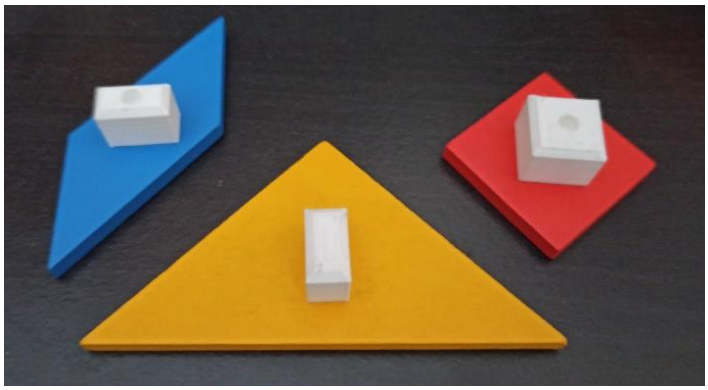


*Figura 12. Piezas tangram con agarre acabadas.*

Durante la realización de la visión por ordenador se detecta que las piezas de agarre de color blanco generan problemas a la hora de detectar correctamente las piezas, así que se decide pintar de color parecido a su pieza tangram para solucionarlo.


*Figura 13. Pintado de las piezas de agarre.*

### 10.1.2 Ordenador de escritorio

Este dispositivo es el ordenador principal donde se realizará el proyecto, un PC de escritorio configurado en el pasado para gaming. Es una máquina potente y eficiente, suficientemente equipada para manejar la alta carga de trabajo asociada con la programación en Python, el procesamiento de datos y el entrenamiento de redes neuronales CNN.

Dispone de un disco duro SSD NVMe de 1TB, que proporciona velocidades de lectura y escritura extremadamente rápidas, ideal para acceder a grandes conjuntos de datos de manera eficiente.

En este ordenador se programan tanto la aplicación en Python como el código del robot y su simulación.

#### 10.1.2.1 Características y especificaciones

- Sistema operativo: Windows 10 Pro
- Procesador: AMD Ryzen 5 1600X Six-Core Processor 3.90 GHz
- Disco duro: 1TB SSD NVme + 3TB HDD
- Memoria RAM: 16,0 GB DDR4
- Gráfica: Nvidia Geforce 1080, 8 GB GDDR5X

### 10.1.3 Ordenador portátil MSI GL75 Leopard

*Figura 14. Portatil MSI.*

El MSI GL75 Leopard 10SEK-040XES es un potente portátil diseñado para brindar una experiencia un alto rendimiento en tareas demandantes. El procesador Intel Core i7-10750H, un chip de seis núcleos, ofrece velocidades impresionantes y capacidades multitarea, ideal para programación, edición y otras tareas intensivas. Equipado con 16GB de RAM, este portátil asegura una multitarea fluida y un rendimiento rápido al ejecutar aplicaciones y programas pesados. Cuenta con un SSD de 1TB, lo que garantiza tiempos de carga rápidos, un arranque veloz del sistema y espacio suficiente para la carga de archivos pesados. Incorpora una tarjeta NVIDIA RTX 2060, conocida por su potencia y capacidad para manejar juegos modernos en configuraciones gráficas altas y ofrecer características avanzadas como trazado de rayos en tiempo real.

Este ordenador portátil, una vez traspasado el proyecto desde el ordenador de escritorio, será el encargado de conectarse a la controladora del robot para ejecutar el programa y comunicarse con el robot para realizar los test necesarios y realizar la solución final del proyecto.

#### 10.1.3.1 Características y especificaciones

- Sistema operativo: Windows 11 Home
- Procesador Comet lake i7-10750H+HM470)
- Memoria DDR IV 8GB*2 (2666MHz)
- Almacenamiento 1TB NVMe PCIe Gen3x4 SSD
- Pantalla 17.3" FHD (1920*1080), IPS-Level 120Hz Thin Bezel
- Controlador gráfico GeForce® RTX 2060, GDDR6 6GB
- Gb LAN
- Intel Wi-Fi 6 AX201(2*2 ax) + BT5
- Cámara de portátil HD type (30fps@720p)
- Batería 6 celdas de Ion de litio 51 Whr
- Dimensiones 397 x 268.5 x 29 mm
- Peso 2.6 Kg

### 10.1.4 Brazo robótico UR3

El UR3 es un brazo robótico industrial colaborativo (cobot) fabricado por Universal Robots. Es el modelo más pequeño y ligero de la serie UR de Universal Robots, que también incluye el UR5 y el UR10, cada uno con diferentes capacidades de carga y alcance.

El brazo es compacto pero potente, con una carga útil máxima de 3 kg (de ahí su nombre, "UR3") y un alcance de hasta 500mm. Está diseñado para la automatización de tareas ligeras y de precisión en espacios limitados.

El UR3 es un brazo robótico colaborativo versátil y seguro, ideal para empresas de cualquier tamaño que busquen aumentar su eficiencia y productividad a través de la automatización.

*Figura 15. Brazo robótico UR3*

#### 10.1.4.1 Características útiles

- **Flexibilidad:** El UR3 es extremadamente versátil, capaz de realizar una amplia gama de tareas, desde ensamblaje y soldadura hasta pintura y pegado. Su tamaño compacto y su diseño ligero también permiten que se integre fácilmente en cualquier espacio de trabajo.
- **Seguridad:** El UR3 ha sido diseñado para trabajar de manera segura junto a los humanos. Incorpora una serie de características de seguridad, incluyendo la capacidad de detenerse si detecta un contacto inesperado, lo que hace que sea seguro para el trabajo colaborativo en entornos industriales.
- **Fácil programación:** Los robots de Universal Robots, incluyendo el UR3, son conocidos por su facilidad de programación. Los usuarios pueden programar el UR3 utilizando una interfaz gráfica de usuario intuitiva o mediante un lenguaje de scripting más avanzado.
- **Rápida implementación:** El UR3 es rápido y fácil de configurar, con tiempos de implementación que pueden ser tan cortos como una sola tarde.
- **Alta precisión:** El UR3 tiene una repetibilidad de ±0.1 mm, lo que lo hace adecuado para tareas de alta precisión.
- **Bajo costo de operación:** El UR3 es un robot eficiente energéticamente y su mantenimiento es relativamente bajo, lo que contribuye a reducir los costos operativos.

### 10.1.4.2 Especificaciones

*Tabla 1. Especificaciones UR3*

| | | |
|---|---|---|
| **Rendimiento** | | |
| Repetibilidad | ±0,1 mm / ±0,0039 in (4 mil.) | |
| Rango de temperatura ambiente | 0–50° | |
| Consumo de energía | Mín. 90 W, estándar 125 W, máx. 250 W | |
| Operación de colaboración | 15 funciones avanzadas de seguridad regulables. Función de seguridad con certificación TÜV NORD. Probado de acuerdo con las normas: EN ISO 13849:2008 PL d | |
| **Especificación** | | |
| Carga útil | 3 kg / 6,6 lb | |
| Alcance | 500 mm / 19,7 in | |
| Grados de libertad | 6 articulaciones giratorias | |
| Programación | Interfaz gráfica del usuario PolyScope con pantalla táctil de 12" con soporte | |
| **Movimiento** | | |
| Movimiento del eje del brazo robot. | Radio de acción | Velocidad máxima |
| Base | ± 360° | ± 180°/s |
| Hombro | ± 360° | ± 180°/s |
| Codo | ± 360° | ± 180°/s |
| Muñeca 1 | ± 360° | ± 360°/s |
| Muñeca 2 | ± 360° | ± 360°/s |
| Muñeca 3 | Infinita | ± 360°/s |
| Herramienta típica | | 1 m/s / 39,4 in/s |
| **Funciones** | | |
| Clasificación IP | IP64 | |
| Clase ISO Sala limpia | 5 | |
| Ruido | 70dB | |
| Montaje del robot | Todos | |
| Puertos de E/S en | Entrada digital | 2 |
| herramienta | Salida digital | 2 |
| | Entrada analógica | 2 |
| | Salida analógica | 0 |
| E/S de fuente de aliment. en herramienta | 12 V / 24 V 600 mA en herramienta | |
| **Características físicas** | | |
| Huella | Ø 128 mm | |
| Materiales | Aluminio, plásticos de PP | |
| Tipo de conector para herramientas | M8 | |
| Long. cable del brazo robótico | 6 m / 236 in | |
| Peso con cable | 11 kg / 24,3 lb | |

### 10.1.5 Herramienta pinzas paralelas DHPS-10-A


*Figura 16. Pinza paralela.*

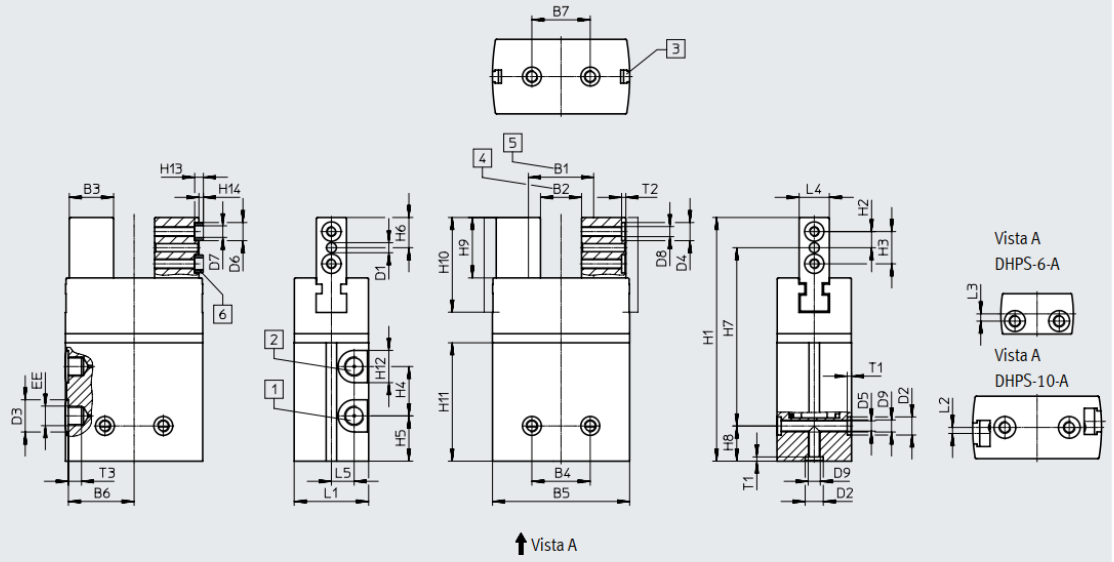
Las pinzas neumáticas paralelas Festo de tamaño 10 ofrecen una carrera por mordaza de 3 mm con una precisión máxima de sustitución de ⩽0.2 mm. Están diseñadas con dos mordazas que funcionan con un modo de doble efecto y ofrecen un movimiento de sujeción en paralelo. Utilizan una guía deslizante para un movimiento guiado y operan a una presión de 0.4 MPa a 0.8 MPa.

La frecuencia de trabajo máxima es de 4 Hz. El cuerpo principal está hecho de una aleación de forja de aluminio anodizado duro, y las mordazas están construidas con acero inoxidable de alta aleación. Son adecuadas para un rango de temperatura ambiente de 5 °C a 60 °C y tienen un peso de 68 g.

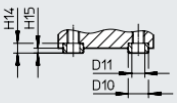
Pinza paralela DHPS

Hoja de datos

**Dimensiones** Descargar datos CAD → www.festo.com




[1] Conexión de aire comprimido para apertura
[2] Conexión de aire comprimido para cierre
[3] Ranura para sensor de proximidad
[4] Posición básica en DHPS-...-A y DHPS-...-A-NC
[5] Posición básica en DHPS-...-A-NO
[6] Casquillos para centrar ZBH (a partir del tamaño 10: 4 unidades incluidas en el suministro)
[7] Interfaz de fijación: Casquillos para centrar ZBH para la fijación de la pinza (2 unidades incluidas en el suministro)




| Tamaño [mm] | B1 ±0,5 | B2 ±0,5 | B3 −0,03 | B4[1] | B5 ±0,1 | B6 | B7[1] | D1 Ø H8 | D2 Ø H8 | D3 Ø | D4 Ø H8 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 6 | 10 | 6 | 5,5 | 11 | 18 | 8,65 | 11 | 1,5 | 5 | 7 | – |
| 10 | 21,8 | 15,8 | 7 | 16 | 32 | 15,4 | 16 | 2 | 5 | 7 | 5 |
| 16 | 27,8 | 17,8 | 13 | 25 | 47 | 22,65 | 25 | 3 | 7 | 7 | 7 |
| 20 | 30 | 17 | 17,5 | 25 | 55,6 | 26,65 | 25 | 4 | 7 | 10 | 7 |
| 25 | 35,4 | 20,4 | 22 | 29 | 68,2 | 32,65 | 29 | 4 | 9 | 16 | 9 |
| 35 | 56 | 31 | 27 | 33 | 88 | 42,25 | 33 | 5 | 12 | 16 | 9 |

1) Tolerancia para taladro centrador ±0,02 mm; tolerancia para rosca ±0,1 mm

*Figura 17. Hoja de datos 1, pinzas paralelas DHPS.*

Hoja de datos

| Tamaño [mm] | D5 ø +0,1 | D6 ø h7 | D7 ø | D8 | D9 | D10 ø h7 | D11 ø | EE | H1 | H2 | H3[1] |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 6 | 2,5 | – | – | M2 | M3 | – | – | M3 | 45,5 | 2,9 | 5,8 |
| 10 | 2,5 | 5 | 3,2 | M3 | M3 | 5 | 3,2 | M3 | 66 | 4 | 8 |
| 16 | 3,3 | 7 | 5,3 | M4 | M4 | 5 | 3,2 | M3 | 80 | 5,5 | 11 |
| 20 | 3,3 | 7 | 5,3 | M4 | M4 | 7 | 5,3 | M5 | 101 | 7 | 14 |
| 25 | 5,1 | 9 | 6,4 | M5 | M6 | 9 | 6,4 | G1/8 | 121 | 8 | 16 |
| 35 | 6,4 | 9 | 6,4 | M6 | M8 | 12 | 10,3 | G1/8 | 142 | 8,5 | 17 |

| Tamaño [mm] | H4 | H5 | H6 | H7 ±0,2 | H8[2] | H9 | H10 | H11 | H12 | H13 –0,2 | H14 –0,3 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 6 | 15 | 4 | 5 | 33 | 7,5 | 9,55 | 15,8 | 25,3 | 7 | – | – |
| 10 | 15,5 | 10,5 | 7,5 | 51 | 7,5 | 15,2 | 23 | 35 | 7 | 2,4 | 1,2 |
| 16 | 18 | 11 | 10 | 62,5 | 7,5 | 20 | 32,5 | 38,1 | 7 | 3 | 1,4 |
| 20 | 23 | 16 | 12,5 | 81 | 7,5 | 25 | 39,5 | 50 | 10 | 3 | 1,4 |
| 25 | 24,5 | 22,5 | 15 | 88,5 | 17,5 | 30 | 47 | 58,8 | 16 | 4 | 1,9 |
| 35 | 29 | 24 | 16 | 108,5 | 17,5 | 32 | 53 | 65,3 | 16 | 4 | 1,9 |

| Tamaño [mm] | H15 –0,2 | H16 –0,3 | L1 | L2 | L3[1] | L4 –0,05 | L5 | T1 +0,1 | T2 +0,1 | T3 +0,5 |
|---|---|---|---|---|---|---|---|---|---|---|
| 6 | – | – | $10^{+0.1}$ | – | 1,8 | 5 | 1,5 | 1,2 | – | 3,5 |
| 10 | 2,4 | 1,2 | $15,5^{+0.1}$ | 1,5 | – | 7 | 5 | 1,2 | 1,2 | 5 |
| 16 | 3 | 1,4 | $22^{+0.1}$ | – | – | 10 | 7 | 1,6 | 1,6 | 6 |
| 20 | 3 | 1,4 | $30_{\pm 0.1}$ | – | – | 12 | 9 | 1,6 | 1,6 | 6 |
| 25 | 4 | 1,9 | $37_{\pm 0.1}$ | – | – | 15 | 11,3 | 2,1 | 2,1 | 6,5 |
| 35 | 4 | 1,9 | $45^{+0.1}$ | – | – | 20 | 13,5 | 2,6 | 2,1 | 6,5 |

1) Tolerancia para taladro centrador ±0.02 mm; tolerancia para rosca ±0.1 mm
2) Tolerancia para taladro centrador –0.05 mm; tolerancia para rosca ±0.1 mm

**Referencias de pedido**

| Tamaño | De doble efecto Sin muelle de compresión | | De simple efecto o con aseguramiento de la fuerza de sujeción En apertura | | En cierre | |
|---|---|---|---|---|---|---|
| [mm] | N.º art. | Código del producto | N.º art. | Código del producto | N.º art. | Código del producto |
| 6 | **1254039** | **DHPS-6-A** | – | | – | |
| 10 | **1254040** | **DHPS-10-A** | **1254041** | **DHPS-10-A-NO** | **1254042** | **DHPS-10-A-NC** |
| 16 | **1254043** | **DHPS-16-A** | **1254044** | **DHPS-16-A-NO** | **1254045** | **DHPS-16-A-NC** |
| 20 | **1254046** | **DHPS-20-A** | **1254047** | **DHPS-20-A-NO** | **1254048** | **DHPS-20-A-NC** |
| 25 | **1254049** | **DHPS-25-A** | **1254050** | **DHPS-25-A-NO** | **1254051** | **DHPS-25-A-NC** |
| 35 | **1254052** | **DHPS-35-A** | **1254053** | **DHPS-35-A-NO** | **1254054** | **DHPS-35-A-NC** |

*Figura 18. Hoja de datos 2, pinzas paralelas DHPS.*

### 10.1.6 Estación robótica

La estación robótica se encuentra en el laboratorio de robótica y control avanzado del edificio TR11 del campus de Terrassa. Se trata de una mesa con propósito educativo de la marca Festo con las siguientes características:

- Carrito/mesa con railes para montajes modulares, con unas dimensiones de 700 x 700 mm. La altura de la mesa se encuentra a 1000 mm del suelo. Dispone de ruedas para poder mover la estación.
- Brazo robot UR3.
- Pantalla FlexPendant para controlar y programar el cobot mediante PolyScope 5.3, conectada a la controladora.
- Botonera de control con parada de emergencia.
- Semáforo para señales.
- Válvula neumática para conectar pinza neumática.
- Marco vertical modular para montar accesorios como cámaras.

A esta configuración se le añade una balda blanca marcada con cintas negras donde se manipularán las piezas del tangram. Las cintas negras marcan el recuadro donde el robot deberá recoger y montar las formaciones de piezas.

En el marco vertical se fijará la cámara, que irá conectada al portátil situado en el escritorio al lado de la estación robótica.

La estación está bien iluminada ya que dispone de dos puntos de luz directamente encima de la estación, esto evita la creación de sombras que podrían provocar problemas a la hora de capturar las imágenes con la cámara.

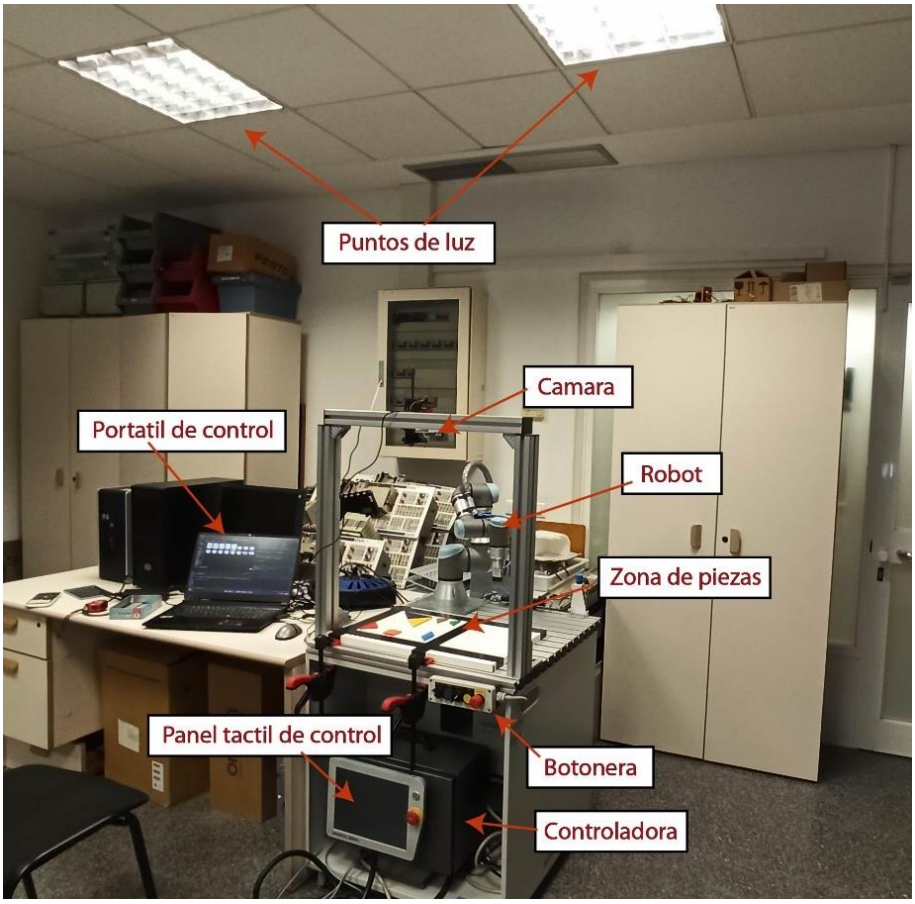



*Figura 19. Foto de la estación robótica del laboratorio.*


*Figura 21. Botonera estación robótica.*

*Figura 20. Válvula de vacío para pinza Festo.*

### 10.1.7 WebCam Creative Live Cam Sync 1080p V2

Esta cámara para videoconferencias ofrece una resolución nítida de 2 MP con calidad Full HD 1080p a una velocidad de 30 fotogramas por segundo (fps). Está diseñada con un amplio campo de visión de 77°, y aunque es ideal para capturar entornos de reuniones y conferencias, también es útil para abarcar todo el espacio de la estación robótica. Sus dimensiones compactas son de 83 x 58 x 60 mm, facilitando su colocación en diversos espacios de trabajo, en nuestro caso se instalará en un racking en lo alto de la estación ayudándonos de sus articulaciones. La cámara incluye un cable de 1.8 metros de largo, lo que proporciona suficiente alcance para su conexión al portátil desde la estación.


*Figura 22. Cámara Creative.*

*Tabla 2. Especificaciones cámara*

| Especificaciones | |
|---|---|
| Resolución | 2 MP, Full HD 1080p, 30 fps |
| Campo de visión | 77° |
| Dimensiones | 83 x 58 x 60 mm |
| Largo del cable (de un extremo a otro) | 1.8m / 5.9 ft |
| SmartComms Kit | ✓ |
| Tapa de privacidad | ✓ |
| Micrófonos integrados | Micrófonos duales mejorados |
| Compatibilidad con plataformas | Windows PC, macOS |

## 10.2 Descripción de software

En esta sección se detallan los diferentes programas utilizados en la realización del proyecto, cada uno con una breve descripción y una lista de sus características más útiles por las que han sido elegidas.

### 10.2.1 Python 3.5


*Figura 23. Python, lenguaje de alto nivel*

Python es un lenguaje de programación interpretado, de alto nivel y de propósito general creado por Guido van Rossum en 1991. Destaca por su legibilidad y sintaxis clara, que favorece un código limpio y fácil de leer.

Python admite varios paradigmas de programación, incluyendo programación procedural, orientada a objetos y funcional. Es extensible en C y C++, y tiene interfaces para muchas llamadas al sistema y bibliotecas, así como a diversos sistemas de ventanas.

Además, Python es dinámico y tiene tipos de datos incorporados de alto nivel, como números flexibles de precisión variable y listas, lo que hace que la programación con Python sea una experiencia fluida y eficiente.

**Características útiles**

- **Fácil de aprender y usar:** La sintaxis de Python es simple y similar al inglés, lo que la hace fácil de leer y entender. Esto hace que Python sea un excelente lenguaje para iniciarse en la programación.
- **Versátil y de propósito general:** Python puede ser utilizado para una amplia gama de tareas, desde el desarrollo web y la creación de guiones hasta el análisis de datos, la inteligencia artificial y el aprendizaje automático.
- **Gran ecosistema de bibliotecas y marcos:** Python cuenta con una extensa biblioteca estándar, y una gran cantidad de paquetes de terceros (como NumPy, Pandas, Matplotlib y TensorFlow, entre otros) que extienden aún más sus capacidades.
- **Comunidad activa y creciente:** Python tiene una comunidad de desarrolladores muy activa y amigable que contribuye a mejorar el lenguaje y a desarrollar un gran número de recursos de aprendizaje y librerías de código abierto.
- **Uso en la industria y en la academia:** Python es ampliamente utilizado en la industria y en la investigación académica, lo que significa que hay muchas oportunidades de trabajo para los programadores de Python, y una gran cantidad de investigación y desarrollo ocurre en Python.
- **Interoperabilidad:** Python puede interactuar con otros lenguajes de programación como C, C++, Java, y más, lo que permite crear soluciones más eficientes y personalizadas.

- **Plataforma independiente:** Python es un lenguaje de programación interpretado, que significa que se puede ejecutar su código en cualquier sistema operativo sin necesidad de cambiarlo.

### 10.2.2 PyCharm 2023.1

*Figura 24. PyCharm, entorno de desarrollo integrado*

PyCharm es un entorno de desarrollo integrado (IDE) que está diseñado específicamente para Python, un popular lenguaje de programación de alto nivel. Fue desarrollado por JetBrains, una compañía que ha producido una serie de IDEs para varios lenguajes de programación.

PyCharm proporciona una serie de características que son útiles para la programación en Python, incluyendo la finalización de código inteligente, inspecciones de calidad de código, navegación de proyectos fácil, soporte para desarrollo web, soporte para bases de datos SQL y otras.

**Características útiles**

- **Características robustas:** PyCharm tiene una amplia variedad de características que están diseñadas para hacer la vida del programador mucho más fácil, incluyendo resaltado de sintaxis, autocompletado de código, capacidad de refactorización, soporte de control de versiones y una multitud de plugins que pueden usarse para personalizar la experiencia de programación.

- **Depuración y pruebas:** PyCharm viene con un potente depurador y un corredor de pruebas. Estas características permiten depurar el código paso a paso y realizar pruebas unitarias fácilmente.

- **Interfaz gráfica de usuario:** PyCharm tiene una interfaz gráfica de usuario bien diseñada que es intuitiva y fácil de usar, lo que hace que la programación sea más eficiente y menos propensa a errores.

- **Soporte para varios marcos y bibliotecas:** PyCharm ofrece soporte para una gran cantidad de marcos de trabajo y bibliotecas de Python, lo que permite trabajar con facilidad en variedad de proyectos.

### 10.2.3 Tensorflow 12.0


*Figura 25. TensorFlow, biblioteca de código abierto para aprendizaje automático*

**TensorFlow** es una biblioteca de software de código abierto creada por Google Brain Team. Se utiliza para la computación numérica y se basa en el concepto de gráficos de flujo de datos, en los que los nodos del gráfico representan operaciones matemáticas, mientras que los bordes del gráfico representan los tensores (arrays multidimensionales) que fluyen entre los nodos.

TensorFlow es particularmente conocido por su utilidad en el desarrollo de aplicaciones de aprendizaje automático (machine learning) y redes neuronales profundas (deep learning), pero también es útil en cualquier contexto que requiera cálculos numéricos intensivos.

**Características útiles**

- **Flexibilidad:** TensorFlow puede ser utilizado tanto para investigación como para producción. Permite crear desde simples operaciones de álgebra lineal hasta algoritmos de aprendizaje automático complejos.
- **Escala de operación:** TensorFlow es capaz de ejecutarse tanto en CPUs como en GPUs, lo que lo hace útil para una variedad de hardware. También puede ejecutarse en varios sistemas operativos y puede escalarse de un solo dispositivo hasta clústers de servidores.
- **Visualización con TensorBoard:** TensorBoard es una herramienta de visualización incluida en TensorFlow que permite visualizar tus modelos de aprendizaje automático y entender, depurar y optimizar tus programas.
- **APIs de alto y bajo nivel:** TensorFlow proporciona tanto APIs de alto nivel (como Keras) para la simplicidad y la consistencia, como APIs de bajo nivel para la flexibilidad y el control.
- **Adopción amplia y comunidad activa:** TensorFlow es utilizado por una amplia gama de organizaciones, desde startups hasta empresas de tecnología muy grandes. Tiene una comunidad muy activa que contribuye con nuevas funcionalidades y mejoras, así como con tutoriales y soluciones a problemas comunes.
- **Soporte para redes neuronales profundas y algoritmos de aprendizaje automático:** TensorFlow viene con una amplia gama de herramientas y bibliotecas integradas que facilitan la creación y el entrenamiento de modelos de aprendizaje automático y redes neuronales profundas.

### 10.2.4 OpenCV


*Figura 26. OpenCV, biblioteca libre de visión artificial*

**OpenCV** (Open Source Computer Vision Library) es una biblioteca de software de código abierto que se utiliza para el procesamiento de imágenes y la visión por ordenador. Fue lanzada por Intel en 2000 y está escrita en C/C++, aunque también ofrece bindings completos para Python, Java y MATLAB/OCTAVE.

OpenCV proporciona una amplia gama de algoritmos para varias tareas relacionadas con la visión por ordenador, incluyendo la captura y el procesamiento de imágenes, el reconocimiento de objetos, el análisis de movimiento, la extracción de características, la segmentación de imágenes y el aprendizaje de máquinas, entre otros.

**Características útiles**

- **Open Source y multiplataforma:** OpenCV es de código abierto, lo que significa que puedes usarla y modificarla libremente. Además, es multiplataforma, compatible con Windows, Linux, macOS, iOS y Android.
- **Funcionalidades amplias:** OpenCV proporciona más de 2500 algoritmos optimizados de visión por ordenador que son útiles para varias tareas, como el análisis de imágenes, la detección y el reconocimiento de rostros, la estimación de pose, la realidad aumentada, el seguimiento de objetos, la extracción de características 3D, la calibración de cámaras, el aprendizaje de máquinas y más.
- **Integración con otros lenguajes y bibliotecas:** OpenCV se integra fácilmente con otras bibliotecas populares, como NumPy, TensorFlow y PyTorch. También proporciona bindings para Python, Java y MATLAB/OCTAVE, lo que significa que se puede utilizar OpenCV con el lenguaje de programación preferido.
- **Ampliamente adoptada:** OpenCV es utilizada por miles de personas en todo el mundo, en industrias que van desde la robótica y los vehículos autónomos hasta la realidad virtual y la inteligencia artificial. También es utilizada en la academia para la investigación y la enseñanza.
- **Comunidad activa y soporte:** Al ser una biblioteca de código abierto, OpenCV cuenta con una comunidad de desarrolladores y usuarios activa que contribuye regularmente con nuevas mejoras y características. Además, existen muchos recursos de aprendizaje y tutoriales disponibles.

### 10.2.5 URSim

*Figura 27. URSim, simulación por software de Universal Robots*

**URSim** es una herramienta de simulación de software proporcionada por Universal Robots. Esta herramienta es esencialmente una versión virtual de los robots colaborativos de Universal Robots y se utiliza para simular el movimiento y las operaciones de estos robots en un entorno virtual antes de implementarlas en el mundo real.

URSim está diseñado para funcionar en un ordenador personal y proporciona una interfaz de usuario muy similar a la del controlador de robot real.

**Características útiles**

- **Desarrollo y prueba segura de programas:** URSim permite a los usuarios desarrollar y probar programas para robots Universal Robots en un entorno seguro y controlado. Esto significa que se puede probar nuevas funciones y movimientos sin riesgo de dañar el robot o el entorno.
- **Formación y aprendizaje:** URSim es una excelente herramienta para aprender a programar y operar robots de Universal Robots. Permite a los nuevos usuarios familiarizarse con la interfaz y las funcionalidades del robot sin necesidad de acceder a un robot físico.
- **Facilidad de uso:** URSim tiene una interfaz de usuario intuitiva que facilita el desarrollo de programas. También incluye una serie de plantillas y ejemplos de programas para facilitar el inicio de la programación.
- **Planificación y optimización:** Con URSim se puede simular diferentes escenarios y configuraciones para optimizar la eficiencia de los robots. Esto puede ser especialmente útil en la fase de planificación de una nueva línea de producción o de un nuevo proceso de automatización.

### 10.2.6 Otros programas

Durante el transcurso del proyecto se utilizan en menor medida otros programas como **Adobe Illustrator**, **Fusion360**, **Word**, **Excel**, la app **Cámara** de Windows y **Cura**.

# 11 Metodología y desarrollo

En esta sección se describe el desarrollo del proyecto y la metodología aplicada, que en su mayoría se ha realizado de forma secuencial, siguiendo la planificación descrita en el apartado **Alcance**.

El desarrollo se divide en cuatro grandes apartados:

1. La aplicación en python para crear problemas y soluciones tangram, para crear datasets de entrenamiento y el control del robot.
2. La preparación, diseño y entrenamiento de la red neuronal que resuelva el rompecabezas tangram.
3. La visión por ordenador para detectar las piezas físicas en el tablero de la estación robótica.
4. La comunicación con el robot y su secuencia de instrucciones de movimiento.

Aunque la gran mayoría del trabajo realizado durante este proyecto ha sido desarrollar código, en este documento se ha realizado un esfuerzo por sintetizar y describir cada programa a un nivel funcional, y no tanto al detalle técnico. Durante esta memoria no se encontrará prácticamente código insertado, solo las explicaciones de los mismos. Por ello, si se desea entender en profundidad como se ha logrado algún proceso de código, se recomienda tener a mano los programas adjuntos en el anexo de este proyecto.

## 11.1 Aplicativo para la creación problemas tangram mediante Python

A continuación, se detalla el proceso de diseño y creación del programa que servirá como aplicación grafica del proyecto.

En el primer apartado se describe cómo funciona la aplicación ya terminada para, más adelante en los siguientes apartados, definir como se ha programado cada parte y las razones de cada decisión de diseño.

Como ya se ha comentado, la aplicación tiene dos objetivos claros:

- Servir como plataforma para crear un dataset útil para el aprendizaje automático mediante el entrenamiento de redes neuronales. Para ello será necesario poder formar configuraciones de forma sencilla y rápida para poder generar un dataset lo más grande posible en un tiempo razonable.
- Servir como interfaz gráfica para crear problemas y enviar la solución al robot para que este la replique en el tablero físico de la estación.

### 11.1.1 Funcionamiento de la aplicación a nivel usuario

Al iniciar el programa nos aparece un mensaje de bienvenida con las instrucciones básicas del funcionamiento de la aplicación. Si bien no es algo realmente necesario para el programa es una *feature* muy sencilla de implementar y que ayuda a que el programa se sienta terminado y funcional.

Haciendo clic en Aceptar podremos visualizar la pantalla del programa, que tiene un tamaño de 1280x720 pixeles (resolución HD). A la izquierda de la pantalla encontraremos las botoneras de control separadas por grupos. En el centro se encuentra el lienzo donde

crearemos las piezas tangram y donde podremos moverlas y girarlas individualmente. A la derecha y arriba, con fondo blanco, se encuentra el lienzo donde se creará la silueta de la configuración de piezas presente en el lienzo central. Esta silueta también es movible y se puede girar para facilitar la creación rápida de diferentes problemas. Por último, a la derecha abajo y con fondo negro, se visualizará la solución de las piezas según la configuración de la silueta del lienzo que hay justo encima. Este lienzo, sin embargo, no es interactuarle para evitar diferencias entre el problema y la solución a la hora de guardar.

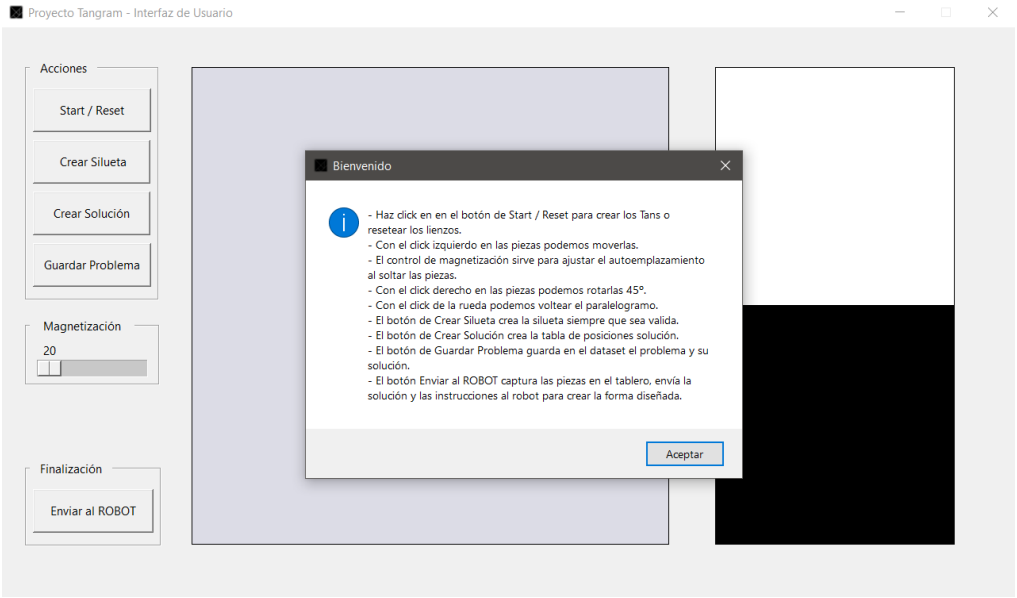



*Figura 28. Mensaje de bienvenida al abrir la aplicación.*

#### 11.1.1.1 Botones de control y lienzos interactivos

Se describen aquí las acciones que podemos llevar a cabo como usuarios utilizando los botones situados en la parte izquierda de la pantalla:

- **Start / Reset:** con este botón podremos crear las piezas tans en el centro de lienzo central. Inicialmente las piezas aparecen formando un cuadrado, que suele ser la configuración básica del Tangram. Si ya existieran elementos en cualquiera de los lienzos, todos son borrados para empezar de nuevo de cero.

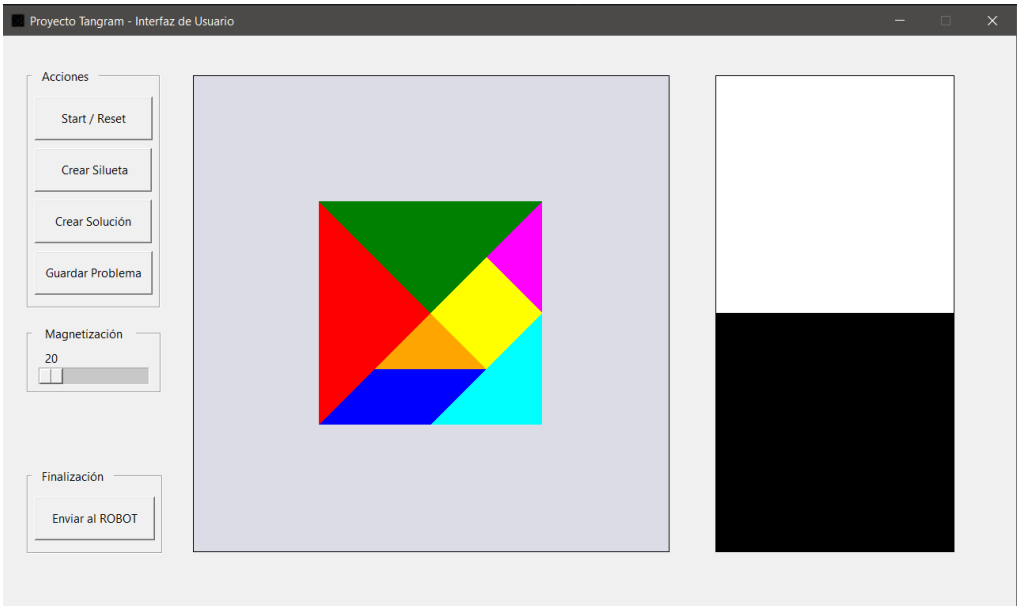



*Figura 29. Tans creados inicialmente con el botón Start / Reset.*

- **Crear Silueta:** al clicar en este botón se creará en el centro lienzo blanco la silueta formada por las piezas del lienzo central.

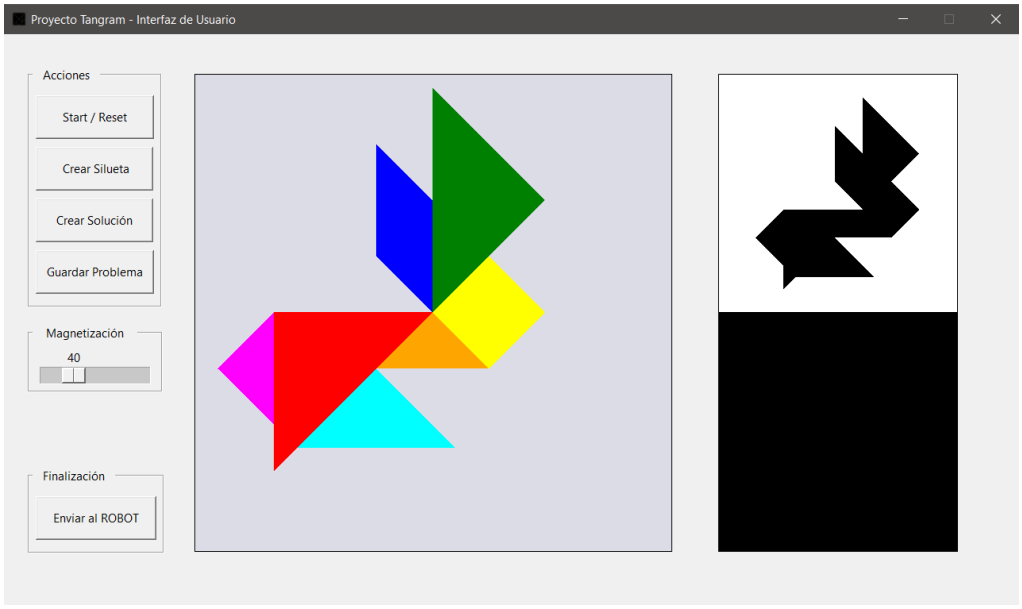



*Figura 30. Creación de la silueta mediante el botón Crear Silueta.*

Sin embargo, si alguna pieza se encuentra superpuesta, separada o conectada solo por un vértice, o si forman un agujero que más de uno de sus vértices coincida con el perímetro, nos aparecerá un mensaje de error y no se generará la silueta. Esto se ha diseñado así para asegurarnos que tenemos una silueta continua y siempre sin solapar piezas (la limitación de los agujeros es a nivel de código por el uso de librerías externas).

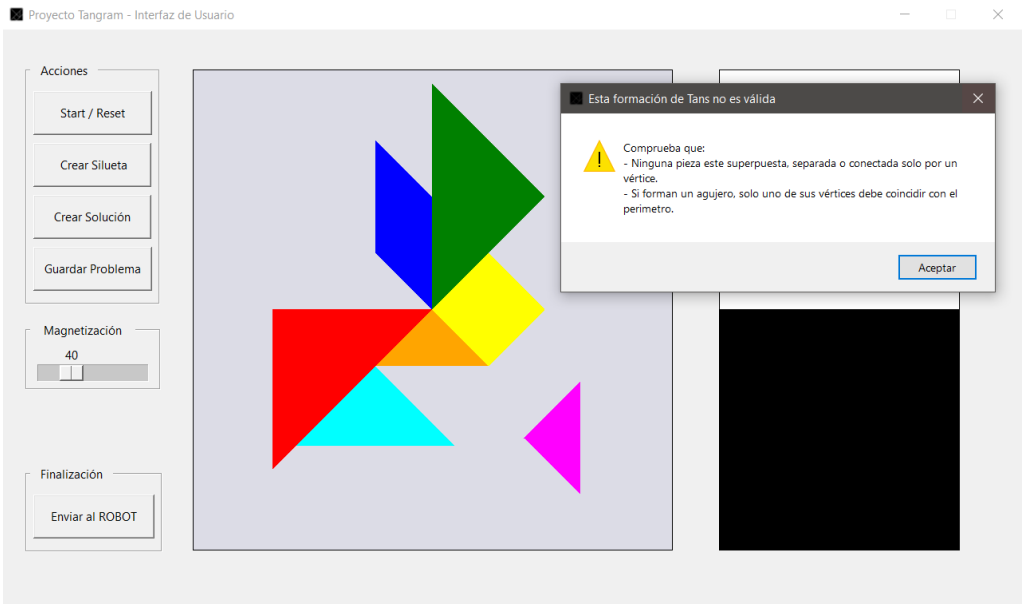
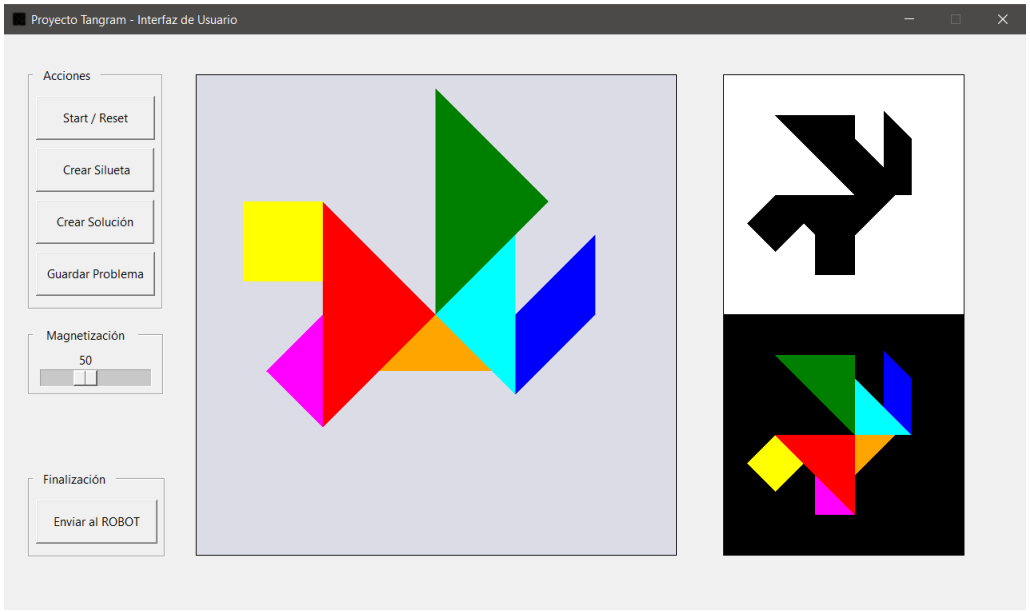



*Figura 31. Ejemplo del mensaje de error al crear siluetas no admisibles.*

- **Crear Solución:** con este botón crearemos en el lienzo negro la visualización de la configuración de piezas individuales del problema silueta situado encima con la misma orientación y posición.




- **Guardar Problema:** al clicar este botón se guardarán en el dataset de imágenes las imágenes del lienzo silueta (fondo blanco) y el lienzo solución (fondo negro) en formato jpeg. Las imágenes se guardan con el nombre **"problemimage_{i}.jpeg"** y **"solutionmimage_{i}.jpeg"** respectivamente, donde "i" es el siguiente valor numérico que no exista anteriormente en la carpeta (con el fin de no sobre escribir archivos).

- **Magnetización (slicer):** con este botón deslizante podemos ajustar la distancia de magnetización de las piezas, una utilidad que hará que al soltar una pieza que estemos moviendo se ajuste al vértice más próximo de otra pieza. De esta manera,

es mucho más sencillo no solapar piezas, ya que sería muy sencillo no ajustarlas correctamente por unos pocos pixeles si no estuviera activa.

- **Enviar a ROBOT:** al clicar en este botón enviaremos la información de la solución para que el robot pueda empezar a formarla físicamente al tablero. Este botón inicia todo el proceso de detección de las piezas físicas del tablero mediante visión por ordenador y la comunicación con el robot y sus instrucciones. Se detalla mas extensamente en su apartado específico.

Además de los botones, podemos interactuar con cada pieza creada en el lienzo central, o con la silueta, con los botones del ratón. Con el clic izquierdo podemos mover las piezas arrastrándolas. Con el clic derecho las rotaremos 45º en sentido antihorario, y por último, presionando el botón de la rueda del ratón podremos voltear la pieza paralelogramo (está pieza es la única que tiene habilitada esta opción porque es la única que la requiere).

### 11.1.2 Decisiones de diseño

Durante el planteamiento y desarrollo de la aplicación se han tomado multitud de decisiones de proceso y diseño, se describen a continuación las mas relevantes para la consecución de los objetivos del trabajo.

**Interfaz gráfica y lienzos**

Con el objetivo de realizar un dataset grande y variado se decide programar dos lienzos por separado donde en uno podamos modificar la configuración de las piezas individualmente y en otro tratarlas como un conjunto (la silueta). De esta manera, realizada una configuración, podemos trasladar en varias posiciones distintas todo el conjunto, tomando una instancia problema y solución para cada una, ahorrándonos tiempo. Seguidamente podemos modificar un par de piezas individualmente y repetir el proceso.

Si no se hubiera realizado así, para rotar o mover todo el conjunto deberíamos hacerlo individualmente para cada pieza, haciendo del proceso una tarea mucha más tediosa.

Añadir un atajo de teclado en la tecla espacio para crear la solución y guardar el problema automáticamente también es una decisión en la misma línea, tomada en mitad del proceso de realización del dataset.

**Reflejo del paralelogramo**

Aunque por código se ha implementado la posibilidad de realizar un reflejo del paralelogramo (siendo esta la única pieza que lo necesitaría) se decide no utilizar esta funcionalidad durante la generación del dataset. La razón para ello es simplificar el proceso de aprendizaje automático de la red neuronal. Si se incluyeran las dos opciones del paralelogramo, la red tendría primero que realizar una tarea de clasificación para saber cuál de los dos lados está presente en la silueta, y después realizar la regresión para encontrar la posición y orientación de cada pieza por separado. El problema ya es bastante profundo como para abordar además esta tarea de clasificación y se considera que esta simplificación no devalúa el objetivo del proyecto.

### Ángulos de los polígonos

Con el fin de simplificar la tarea de la red neuronal y del funcionamiento del aplicativo se decide limitar los ángulos de orientación de las piezas a múltiplos de 45º. Aún con esta limitación la cantidad y diversidad de configuraciones es infinita.

Además, al extraer la información de la posición de las piezas simplificaremos los ángulos (calculados desde el centroide hasta el vértice más cercano) los únicos ángulos distintos que puede adoptar el polígono:

- 2 para el cuadrado, que tiene una simetría rotacional de 90º
- 4 para el paralelogramo, con simetría rotacional de 180º
- 8 para los triángulos, sin simetría rotacional.

Por lo tanto, si la aplicación detecta que el cuadrado se encuentra a 135º, mediante código los simplificará a 45º.

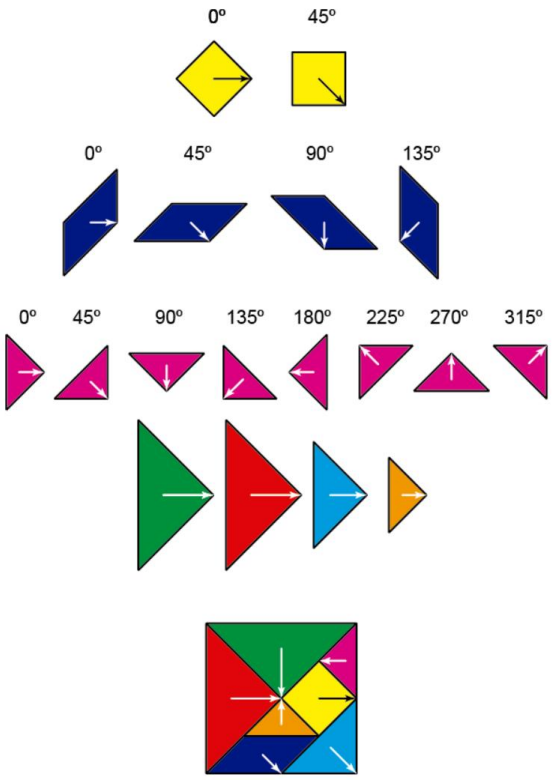



*Figura 32. Ángulos de las piezas en aplicación gráfica.*

### Variedad del dataset

Durante la investigación y documentación sobre el entrenamiento de redes neuronales en todas y cada una de las fuentes consultadas se remarca la importancia de este para lograr un éxito durante el entrenamiento. En algunos casos, se da a entender que este es el proceso más crítico a la hora de conseguir los objetivos. Con esto en mente, se decide no generar un único dataset, sino crear dos complementarios.

El primer dataset constará de las imágenes de la silueta en blanco y negro que servirán como problema, junto a un archivo numpy con el vector de las 21 posiciones de las piezas ([x, y, θ] x7). Este dataset se usará para la red neuronal convolucional (CNN).

A la vez, también se generará en otra carpeta, un segundo dataset con las mismas imágenes problema, pero en vez de una solución numérica, se guardará la imagen solución asociada al problema. Este dataset se utilizará si la CNN no es capaz de resolver el problema y necesitamos implementar una red Convolucional Autoencoder (CAE).

De esta manera si en los primeros entrenamientos no obtenemos los resultados deseados no necesitaremos realizar otro dataset porque ya estará creado.

### 11.1.3 Estructura de los scripts del programa.

Se ha hecho un esfuerzo por estructurar y comentar el código de la forma más clara posible, aunque debido al tamaño del mismo esto no siempre ha sido posible. El siguiente diagrama ilustra conceptualmente los archivos Python con sus clases y métodos, y que archivos importan otros (mediante flechas) antes de su apartado donde se describen más adelante. Los programas de entrenamiento y testeo funcionan aparte del programa principal.

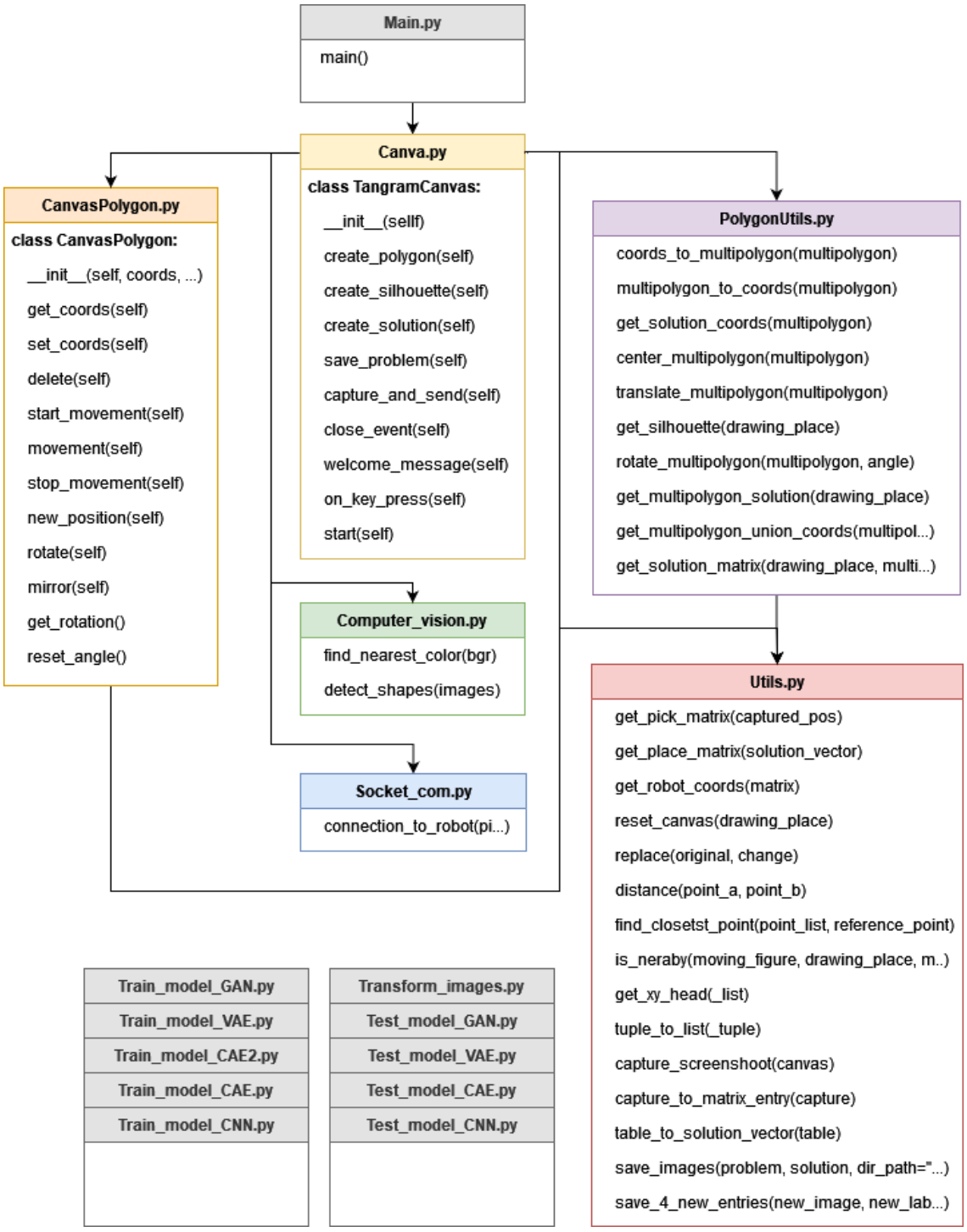



*Figura 33. Estructura del código creado durante el proyecto.*

### 11.1.4 Librerías utilizadas

- **tkinter** es la biblioteca estándar de Python para la creación de interfaces gráficas de usuario (GUI). Proporciona una gran variedad de controles como botones, etiquetas y cuadros de texto, que se utilizan en una aplicación de escritorio.

- **shapely:** es una biblioteca de Python para la manipulación y el análisis de objetos geométricos planos. Está basada en las bibliotecas de geometría plana de JTS y GEOS. Permite realizar operaciones como buffer, merging, creation, y análisis espacial en objetos geométricos.

- **numpy:** es una biblioteca de Python para la computación científica. Proporciona un objeto de matriz multidimensional de alto rendimiento, y herramientas para trabajar con estas matrices. También ofrece capacidades de álgebra lineal, transformada de Fourier, y generación de números aleatorios.

- **matplotlib:** es una biblioteca de Python para la creación de gráficos estáticos, animados e interactivos en Python. Proporciona una interfaz similar a MATLAB y es muy flexible y potente para la visualización de datos.

- **math:** es un módulo de Python que proporciona funciones matemáticas definidas por el estándar C. Estas funciones no pueden usarse con números complejos, para ello se debe utilizar el módulo cmath.

- **os:** es un módulo de Python que proporciona una forma de usar funcionalidades dependientes del sistema operativo, como leer o escribir en el sistema de archivos, iniciar o matar procesos, leer variables de entorno, entre otras.

- **glob:** es un módulo de Python que encuentra todos los nombres de ruta que coinciden con un patrón especificado de acuerdo a las reglas de la línea de comandos del shell de Unix.

- **tensorflow:** es una biblioteca de código abierto para aprendizaje automático y aprendizaje profundo para llevar a cabo tareas complejas de cálculo numérico. Fue desarrollada por Google y se utiliza en una variedad de tareas, principalmente en el entrenamiento de redes neuronales profundas.

- **sklearn (scikit-learn):** es una biblioteca de Python para aprendizaje automático. Proporciona una selección de algoritmos de aprendizaje supervisados y no supervisados. Incluye herramientas para modelado predictivo como clasificación, regresión, clustering, entre otros.

- **tabulate:** es una biblioteca de Python para crear tablas en una variedad de formatos como HTML, Markdown, y LaTeX. Puede tomar datos de varias fuentes, incluyendo listas y diccionarios.

- **socket:** es una biblioteca en Python que proporciona acceso a la interfaz de sockets de Berkeley. Permite a los programas Python comunicarse a través de la red utilizando protocolos como TCP/IP y UDP. Se utiliza comúnmente para crear

clientes y servidores de red, permitiendo la comunicación entre diferentes sistemas y dispositivos en una red.

- **time:** es un módulo en Python que proporciona funciones para trabajar con tiempos y fechas. Ofrece funciones para determinar el tiempo actual, convertir entre diferentes representaciones de tiempo, y pausar la ejecución de un programa por un tiempo determinado.

- **datetime:** es un módulo en Python que suministra clases para manipular fechas y tiempos. Ofrece capacidades para crear objetos de fecha y hora, formatear y analizar cadenas de fecha/hora, y realizar aritmética de fecha/hora.

- **pyautogui:** es una biblioteca de Python utilizada para automatizar la interacción con la interfaz gráfica de usuario (GUI) de un ordenador. Permite controlar el ratón, el teclado y la pantalla, lo que es útil para la automatización de tareas y la realización de pruebas de software.

### 11.1.5 Descripción de los scripts desarrollados

Se describe a continuación cada uno de los archivos script que se utilizan en durante la ejecución de la aplicación gráfica.

#### 11.1.5.1 Main.py

Este script Python inicializa y ejecuta el script **TangramCanvas.py.**

Primero, importa la clase **TangramCanvas** del archivo **canva.py** en la carpeta scripts. Luego define una función ***main()***, que crea una instancia de **TangramCanvas** y llama a su método ***start()*** para comenzar el módulo en cuestión.

Finalmente, si este archivo se ejecuta directamente (no se importa como módulo en otro script), se llama a la función ***main().*** Esto se verifica con la línea *"if __name__ == '__main__':",* que es un patrón común en Python utilizado para determinar cómo se está ejecutando un script.

#### 11.1.5.2 Canva.py

Este es el código principal del programa, define la clase **TangramCanvas**, que es la aplicación gráfica interactiva para construir los problemas de Tangram, la base de la aplicación.

A continuación, se describen los métodos en detalle:

- ***__init__(self):*** Este es el constructor de la clase, donde se inicializan todos los componentes visuales de la aplicación utilizando tkinter. Se definen los lienzos (canvas), donde el usuario podrá mover las piezas del Tangram, crear siluetas y visualizar soluciones. También se crean los botones para interactuar con la aplicación, "**Start / Reset**", "**Crear Silueta**", "**Crear Solución**", y "**Guardar**

**Problema**" (explicados más adelante). Adicionalmente, se crea un control deslizante para ajustar la magnetización de las piezas.

- ***create_polygon(self):*** Este método se utiliza para crear las siete piezas iniciales del Tangram en el lienzo "**DrawingPlace**" borrando antes todas las piezas que hubiera en él. Se ejecuta cuando el usuario hace clic en el botón "**Start / Reset**". Para cada pieza, se especifica un conjunto de coordenadas de inicio, un color y dos etiquetas, cuyos detalles se encuentran en las constantes definidas al inicio del script.

- ***create_silhouette(self):*** Este método genera un polígono "silueta" (la imagen problema con la que se alimentará la red neuronal) en el lienzo "**SilhouettePlace**" a partir de las posiciones actuales de las piezas en el lienzo "**DrawingPlace**". La silueta se crea solo si las piezas cumplen ciertas condiciones: no deben estar superpuestas, separadas, o conectadas solo por un vértice. Si las piezas forman un agujero, solo uno de sus vértices debe coincidir con el perímetro. Si estas condiciones no se cumplen, se muestra un mensaje de advertencia y no se crea la silueta. Se ejecuta al hacer clic en el botón "**Crear Silueta**".

- ***create_solution(self):*** Este método crea una tabla de solución, con los valores de posición [x, y] del centroide de cada pieza y su orientación en el lienzo **"SilhouettePlace".** Esta tabla se muestra en la consola y las piezas se redibujan en el lienzo **"SolutionPlace"**. Los datos de esta tabla, así como las imágenes de los dos lienzos, serán usadas como datos de salida para el entrenamiento de la red neuronal.

- ***save_problem(self):*** Este método guarda la silueta y su solución en archivos, captura imágenes de la silueta y de la solución y las guarda en archivos. Además, convierte la captura y la tabla de solución en una matriz y un vector, respectivamente, y guarda estas estructuras en archivos.

- ***capture_and_send():*** Este método inicializa la captura de video desde una cámara externa y configura su resolución. Después, captura un fotograma (frame) de la cámara para su posterior procesamiento, si la captura es exitosa, utiliza la función ***detect_shapes*** para identificar y obtener las posiciones de las piezas en la estaciñón en la imagen capturada. Seguidamente libera los recursos de la cámara y destruye cualquier ventana de visualización creada. Crea las matrices para las posiciones de "recogida" y "colocación" de las formas detectadas, redondeando las coordenadas para cada posición y convierte las matrices de posiciones en cadenas de texto que serán enviadas al robot. Hecho esto, establece una conexión TCP/IP con el robot y envía las coordenadas de "recogida" y "colocación" para que el robot realice las operaciones correspondientes.

- ***close_event(self):*** Este método cierra la ventana de la aplicación, terminando el programa.

- ***welcome_message(self):*** Este método muestra un mensaje de bienvenida e instrucciones al usuario cuando se carga el programa.

- **on_key_press(self, event):** Este método llama a las funciones ***create_solution(self)*** y ***save_problem(self)*** cuando el usuario pulsa la tecla "espacio". Se utiliza para realizar el dataset mucho más rápido.

- ***start(self):*** Este método inicia la aplicación, iniciando el bucle principal de la interfaz de usuario de tkinter.

Las constantes al principio del script, como ***START_COLOR***, ***START_TAG***, ***START_LABEL***, y ***START_COORDS***, definen las características iniciales de las piezas del Tangram.

Las variables globales ***multipolygon_solution*** y ***solution_table*** almacenan la solución del problema del Tangram, es decir, la primera como una lista de polígonos para ser usada con la librería **shapely** en otros scripts y la segunda con la información de salida con la que queremos entrenar la red neuronal. Estas se actualizan cuando se llama a los métodos ***create_silhouette*** y ***create_solution*** (relacionados con sus respectivos botones).

#### 11.1.5.3 CanvasPolygon.py

Este código define la clase ***CanvasPolygon***, que se utiliza para manipular polígonos en los lienzos de la aplicación gráfica. Cada polígono representa una de las piezas del Tangram.

El constructor ***__init__*** inicializa el polígono en el lienzo.

Los parámetros se usan para:

- ***coords:*** representan las coordenadas de los vértices del polígono.

- ***color:*** *define* el color del polígono.

- ***tag:*** etiqueta para identificar el tipo de polígono.

- ***label:*** otra etiqueta para identificar específicamente el polígono (dentro del mismo tipo).

- ***canvas:*** el lienzo donde se dibuja el polígono.

- ***magnet_slider***: el valor de magnetización, ajustado por el control deslizante, para la magnetización entre polígonos.

Las funciones **tag_bind** se utilizan para vincular ciertos eventos de mouse a funciones específicas:

- **Button-1:** Cuando se hace clic en un polígono con el botón izquierdo del ratón ('<Button-1>'), se llama a la función ***start_movement***. Cuando se mueve el ratón mientras se mantiene pulsado el botón izquierdo (***'<Motion>'***), se llama a la función ***movement***. Y cuando se suelta el botón izquierdo del ratón (***'<ButtonRelease-1>'***), se llama a ***stop_movement***. De esta manera, se puede manipular el polígono al interactuar con el ratón.

- **Button-3:** Este evento se dispara al hacer clic con el botón derecho del ratón, esto llama a la función ***rotate***, que rota el polígono 45 grados en el sentido de las agujas del reloj alrededor de su primer vértice declarado.

- **Button-2:** Este evento se dispara al hacer clic con el botón del medio del ratón (conocido como el botón de desplazamiento). Esto llama a la función ***mirror***, que refleja el polígono a través de su punto medio. Sin embargo, esta función solo se activa si la etiqueta del objeto comienza con "p", es decir, sólo se aplica cuando se hace clic en el paralelogramo (romboide).

A continuación, se describen los métodos:

- ***get_coords:*** Este método devuelve las coordenadas del polígono. No toma ningún argumento y simplemente devuelve el valor almacenado en el atributo ***self.coords*** de la instancia del objeto.

- ***set_coords:*** Este método se utiliza para establecer las coordenadas del polígono. Toma como argumento una lista de coordenadas y le asigna al atributo ***self.coords*** de la instancia del objeto.

- ***delete:*** Este método se utiliza para eliminar el polígono del lienzo. Llama al método ***delete*** de ***self.canvas***, pasándole el ID del polígono.

- ***start_movement:*** Este método se activa cuando el usuario hace clic en el polígono (el evento ***<Button-1>***). Comienza a mover el polígono si es movible, capturando la posición inicial del cursor.

- ***movement:*** Este método se activa cuando el usuario mueve el cursor (el evento ***<Motion>***). Si el polígono está en movimiento, se calculan las nuevas coordenadas y se mueve el polígono.

- ***stop_movement:*** Este método se activa cuando el usuario suelta el botón del mouse (el evento ***<ButtonRelease-1>***). Detiene el movimiento del polígono y si el polígono está cerca de otro, lo une con el polígono más cercano (magnetización).

- ***new_position:*** Este método se utiliza para actualizar las coordenadas del polígono. Toma como argumentos la distancia que el polígono debe moverse en el eje x y en el eje y, calcula las nuevas coordenadas y luego actualiza las coordenadas del polígono.

- ***rotate:*** Este método se activa cuando el usuario hace clic con el botón derecho del mouse en el polígono (el evento ***<Button-3>***). Rota el polígono 45 grados alrededor de su primer punto.

- ***mirror:*** Este método se activa cuando el usuario hace clic con el botón del medio del mouse en el polígono (el evento ***<Button-2>***). Realiza una reflexión vertical del polígono en el lienzo, pero solo si se trata del paralelogramo (romboide).

- ***get_rotation:*** Este método estático devuelve el ángulo de rotación global del polígono en el lienzo silueta. No toma ningún argumento y devuelve el valor

almacenado en la variable global ***rotated_angle***. Esta variable se utiliza para saber cuántas veces ha sido rotado el polígono "silueta" en su lienzo.

- ***reset_angle:*** Este método estático restablece el ángulo de rotación global del polígono en el lienzo silueta a 0. No toma ningún argumento y actualiza el valor de la variable global ***rotated_angle***.

#### 11.1.5.4 PolygonUtils.py

Este script es una colección de funciones que manipulan ***Polygons*** y ***MultiPolygons*** utilizando la biblioteca **shapely**. A continuación, se describen las funciones resumidamente:

- ***coords_to_multipolygon:*** Esta función toma una lista de listas de coordenadas. Cada lista de coordenadas representa un polígono. La función convierte cada lista de coordenadas en un objeto ***Polygon***, y luego agrupa todos estos objetos ***Polygon*** en un objeto ***MultiPolygon***.

- ***multipolygon_to_coords:*** Esta función toma un objeto ***MultiPolygon*** y devuelve una lista de listas de coordenadas, donde cada lista de coordenadas representa un polígono.

- ***get_solution_coords:*** Esta función toma un objeto ***MultiPolygon*** y devuelve una lista de listas de coordenadas, donde las coordenadas de cada polígono se han reducido a la mitad para ser visualizadas en el lienzo solución.

- ***center_multipolygon:*** Esta función centra un objeto ***MultiPolygon*** en el lienzo, trasladándolo al centro (300, 300) en función de sus límites actuales.

- ***rotate_multipolygon:*** Esta función rota un objeto ***MultiPolygon*** en un ángulo dado en grados.

- ***get_multipolygon_solution:*** Esta función toma un objeto ***DrawingPlace*** y devuelve el ***MultiPolygon*** centrado que representa la solución del problema de tangram. Las figuras de la ***DrawingPlace*** se convierten en un ***MultiPolygon***, que luego se centra en el lienzo.

- ***get_multipolygon_union_coords:*** Esta función toma un objeto ***MultiPolygon***, une todos los polígonos en un solo objeto ***Polygon*** y devuelve una lista de coordenadas que representan el contorno exterior e interior del ***Polygon***. Si el objeto no es de tipo ***Polygon*** o su área no es aproximadamente 80000, se imprime un mensaje de error y se devuelve 0.

- ***get_solution_matrix:*** Esta función toma un objeto **DrawingPlace** y un objeto **MultiPolygon** y devuelve una matriz que representa la solución del problema de tangram. La matriz incluye la etiqueta de cada polígono, las coordenadas del centroide, y el ángulo de rotación en grados.

#### 11.1.5.5 Utils.py

Esta es otra colección de diferentes funciones diversas que no valía la pena separar en más archivos, cada una de ellas tiene un propósito muy específico, ya sea en la manipulación de datos gráficos, en la conversión de unidades para el uso del robot, o en la gestión de archivos:

Al inicio del programa se declara una matriz constante llamada **WAIT_POS** con los puntos de coordenadas donde el robot deberá depositar las piezas a modo de espera para apartarlas de la zona de montaje y que no puedan ser solapadas durante los movimientos.

Descripción de las funciones:

- ***reset_canvas(drawing_place):*** Esta función recibe un objeto ***drawing_place*** como argumento, que es un lienzo de Tkinter. La función elimina todos los polígonos del lienzo al recorrer cada figura y llamar al método ***delete()***.

- ***replace(original, change):*** La función toma dos listas: ***original***, que es la lista de coordenadas del polígono original, y ***change***, que contiene un par de puntos que se han encontrado más cerca uno del otro. Luego modifica las coordenadas del polígono original en función de la diferencia entre los puntos en ***change***.

- ***get_pick_matrix(captured_pos):*** Esta función coge la matriz de coordenadas de píxeles de las piezas detectadas por la visión artificial y la transforma a milímetros y radianes para que pueda ser usada por el robot. Se utiliza relaciones y desplazamientos específicos para que las coordenadas coincidan con la zona del tablero (explicados más en detalle en su apartado específico). Añade también a la matriz las posiciones donde deberá ir a buscar el robot una vez las tenga apartadas, extendiendo la matriz con la constante ***WAIT_POS*** definida al inicio del script. Devuelve la lista que se usará para mover el robot a esas posiciones.

- ***get_place_matrix(solution_vector):*** Similar a ***get_pick_matrix***, pero en este caso, utiliza el vector de solución creado en la aplicación y lo transforma de forma similar a la función anterior. En este caso primero se coge la constante ***WAIT_POS*** y luego se le añade la matriz modificada, ya que primero se dejarán (place) las piezas en la posición de espera y luego en su posición final.

- ***get_robot_coords(matrix):*** Convierte una matriz de coordenadas en una lista de cadenas de texto que representa cada posición de destino del robot. Esta es la lista final necesaria para enviar al robot mediante la conexión TCP/IP cliente servidor.

- ***distance(point_a, point_b):*** Calcula la distancia euclidiana entre dos puntos dados. Devuelve -1 si alguno de los puntos no existe.

- ***find_closest_point(points_list, reference_point):*** Busca en una lista de puntos para encontrar el más cercano a un punto de referencia dado.

- ***is_nearby(moving_figure, drawing_place, magnet_slider):*** Encuentra si hay algún punto de otro polígono que está lo suficientemente cerca del polígono en movimiento en el lienzo. Utiliza el valor ***magnet_slider*** del botón de la aplicación

gráfica para determinar qué tan cerca deben estar los puntos para considerarse "cercanos".

- ***get_xy_head(_list):*** Extrae y devuelve los dos primeros elementos de una lista.
- ***tuple_to_list(_tuple):*** Convierte una tupla en una lista.
- ***capture_screenshoot(canvas):*** Toma una captura de pantalla de un área específica de la pantalla (los lienzos silueta y solución por separado) usando la biblioteca ***pyautogui*** y devuelve esta captura de pantalla.
- ***capture_to_matrix_entry(capture):*** Convierte la imagen capturada en una matriz ***NumPy*** en escala de grises y la normaliza.
- ***table_to_solution_vector(table):*** Convierte una tabla de soluciones que contiene información sobre polígonos en un vector normalizado.
- ***save_images(problem, solution, dir_path='dataset_images'):*** Guarda las imágenes de la silueta problema y solución en la carpeta dataset para tal fin, creando un nuevo nombre consecutivo para no sobre escribir archivos. Si no existiese el directorio, crea un nuevo.
- ***save_4_new_entries(new_image,new_labels,dir_path='dataset_files'):*** Al principio guardaba hasta cuatro conjuntos de imágenes y etiquetas en el disco. Intercambiando cada triangulo del mismo tipo entre ellos. Después de algunos entrenamientos se modifica para que solo guarde uno y no se crea ese intercambio, aunque el código sigue presente comentado por si se decide recuperar.

#### 11.1.5.6 Computer_vision.py

Este script se describe en detalle en el apartado de visión por ordenador y no será comentado aquí.

#### 11.1.5.7 Socket_com.py

Este script se describe en detalle en el apartado comunicación e instrucciones del robot y no será comentado aquí.

## 11.2 Diseño e implementación del modelo de aprendizaje automático

El diseño de un modelo de aprendizaje automático para resolver la configuración de piezas del rompecabezas tangram involucra varios pasos esenciales. Primero, se debe definir claramente el objetivo del proyecto, en este caso, resolver a partir de la silueta formada por las piezas, la posición y orientación de cada una de ellas. Luego, es fundamental seleccionar y preprocesar un conjunto de datos adecuado de configuraciones de piezas del tangram, esto requiere la creación, normalización y segmentación de un set de datos (dataset) de entrenamiento.

Posteriormente, se ha de elegir un tipo de modelo de aprendizaje automático adecuado para el problema, como podrían ser las redes neuronales convolucionales para el análisis de imágenes. Es esencial determinar qué características de las piezas y sus configuraciones son relevantes para la tarea y preparar los datos en consecuencia. Los datos se dividen en conjuntos de entrenamiento, validación y prueba para facilitar el proceso de entrenamiento y evaluación.

El modelo ha de entrenarse utilizando los datos de entrenamiento, ajustando sus parámetros para minimizar la función de perdida. Después del entrenamiento, el modelo se evalúa utilizando el conjunto de datos de validación, y se hacen ajustes según sea necesario para mejorar su desempeño. Finalmente, el modelo se prueba con un conjunto de datos de prueba para evaluar su eficacia en la resolución de problemas no vistos por la red antes de su implementación real.

### 11.2.1 Recopilación y preparación del dataset

Se hace uso de la aplicación grafica para generar un set de datos los suficientemente grande y variado. El proceso es manual y bastante simple, se crea una combinación de piezas en el lienzo central de la aplicación, después creamos su silueta, y por cada silueta realizamos unas tres traslaciones distintas dentro del lienzo para unos cuatro o cinco ángulos distintos. Con cada modificación que hagamos se creará la solución y se guardará los archivos de entrada y salida de la red (problema y solución).

Se crean dos datasets diferentes mediante la aplicación, y el segundo se modifica y expande procesando los datos para crear un tercero.

#### 11.2.1.1 Dataset con vector de posiciones

El primer dataset está formado por las imágenes jpeg de las siluetas que se utilizarán como entrada a nuestra red neuronal. La solución a estos problemas será un vector de posiciones de 21 valores con los posiciones y orientación del centroide de cada pieza ([x, y, θ]) que se guarda en un archivo numpy. Estos valores se normalizan para que tengan un valor entre 0 y 1 (como si se tratará de un porcentaje) para mejorar el rendimiento de la red. Las posiciones en pixeles se dividen por el ancho o alto del lienzo y los ángulos por 360º, de esta manera unos valores [0.5, 0.5, 0.5] corresponderán a la posición [300px, 300px, 180º].

Estos archivos se guardan en la carpeta "dataset_files". El proceso llevó alrededor de 6 horas separado en dos días distintos. La carpeta final contiene 14.000 archivos (7000 problemas con su solución distintos) y un tamaño de 36,5MB.

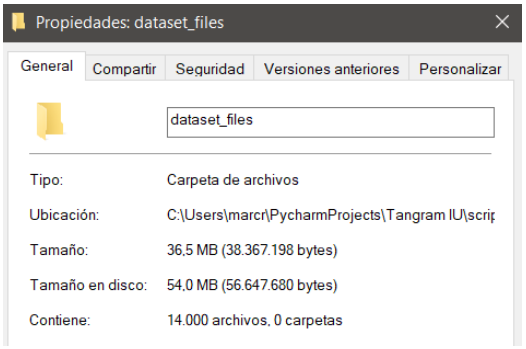



*Figura 34. Carpeta dataset con vector posiciones.*

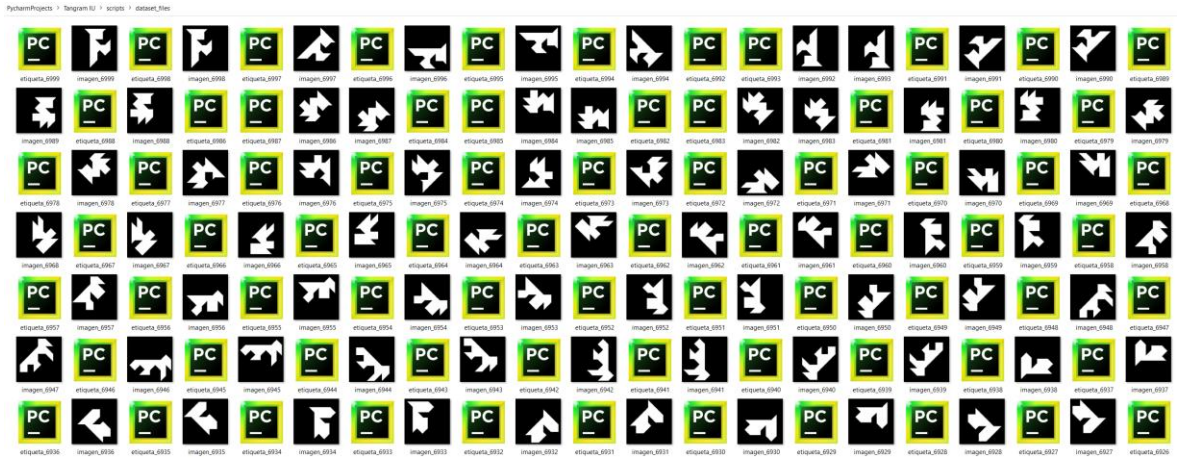


*Figura 35. Archivos del dataset con vector de posiciones.*

### 11.2.1.2 Dataset de imágenes

A la vez que se crea el dataset anterior, el aplicativo también guarda en la carpeta "dataset_images" las imágenes problema de la misma forma que antes, pero en vez del archivo de vectores guardaremos la imagen coloreada con la solución de la silueta.

Por un error en la programación de la captura del lienzo solución (solo se capturaba parte del lienzo) las primeras instancias creadas durante la creación del dataset tuvieron que ser descartadas. Por este motivo este dataset es algo menor que el anterior. La carpeta final contiene 10.000 archivos (5000 problemas junto a su solución) y un tamaño de 55,5MB.


*Figura 36. Carpeta dataset de imágenes.*

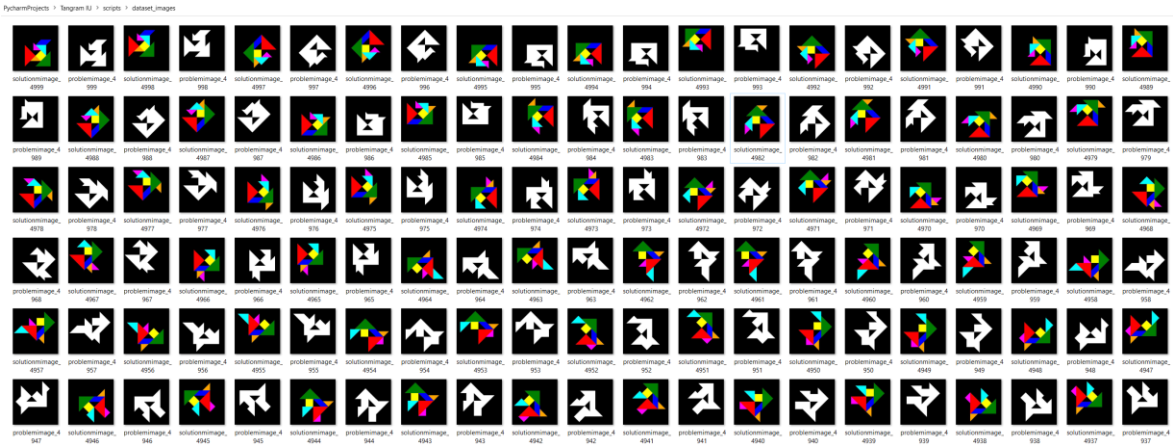


*Figura 37. Archivos del dataset de imágenes.*

### 11.2.1.3 Dataset de imágenes procesado y expandido.

Durante el largo proceso de llevar a cabo los entrenamientos de las redes neuronales, y ante la situación de que ninguna red diseñada parecía resolver el problema planteado, se decide crear un tercer dataset a partir del segundo.

El objetivo principal es agrandar el dataset de 5000 instancias a 20000, girando pareja de imágenes 90º. También se procesan las imágenes para codificar los colores a los ocho colores básicos del sistema RGB, es decir la combinación de cada canal como si fuera un sistema binario ($2^3$), y se guardan en png que no comprime los colores tanto como el formato jpeg (que añade ruido a la imagen al comprimir más). Las decisiones sobre estas medidas se explican más adelante.

La carpeta final contiene 40000 archivos y tiene un tamaño de 146MB. En este caso se separas en carpetas distintas las imágenes problema y solución.

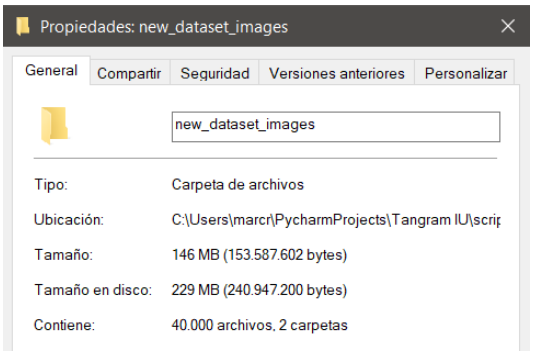



*Figura 38. Carpeta dataset de imágenes procesado.*

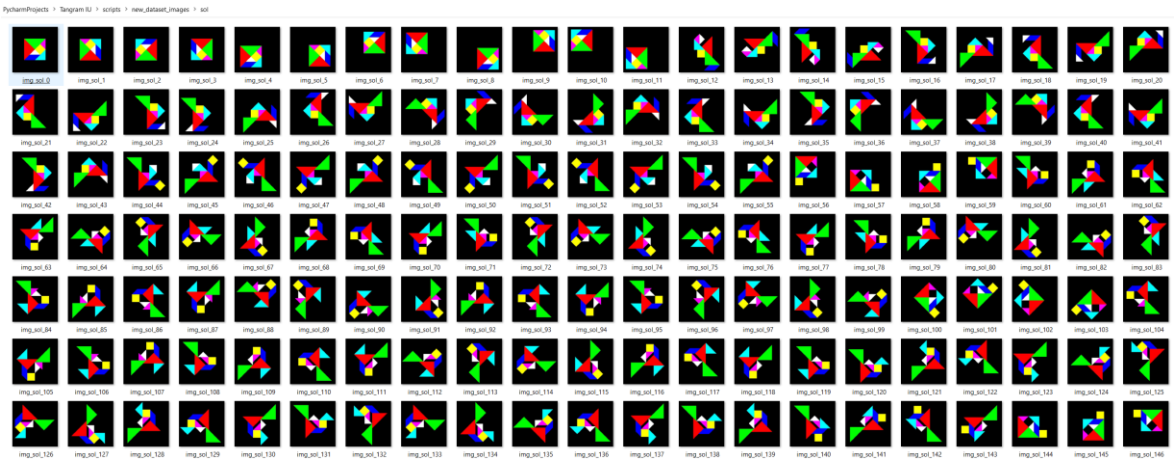


*Figura 39. Archivos solución del dataset de imágenes procesado.*

### 11.2.2 Diseño del modelo

Con la creación de nuestros dataset compuesto por las imágenes problema en blanco y negro y los vectores de etiquetas ya podemos comenzar a programar la arquitectura de nuestra red neuronal.

Se decide utilizar un Red Neuronal Convolucional (CNN) que, como ya se ha explicado anteriormente, es un tipo de red neuronal artificial diseñada para reconocer patrones a partir de datos con una estructura de cuadrícula, como una imagen. Las CNN son especialmente poderosas para tareas como la clasificación de imágenes, detección de objetos, entre otros.

Pero antes de entrar en detalle sobre las CNN es importante conocer cómo funcionan las "neuronas" dentro de cualquier red neuronal.

### 11.2.3 Neurona artificial

Las neuronas artificiales en las redes neuronales están inspiradas en las neuronas biológicas. En esencia, toman una serie de entradas, las procesan y producen una salida. La forma en que procesan estas entradas es a través de una combinación lineal de las mismas, seguida de una función de activación.

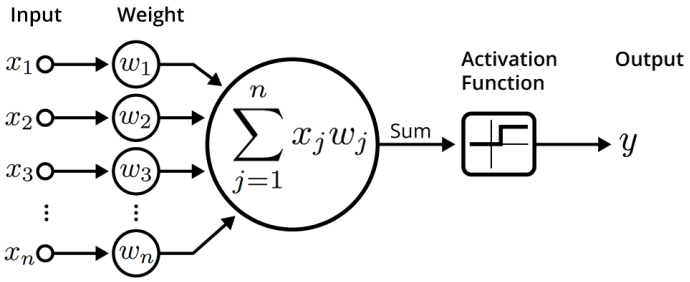



*Figura 40. Neurona artificial (sin sesgo)*

#### 11.2.3.1 Entradas y Pesos

Cada neurona recibe varias entradas. Estas entradas están asociadas con un peso. Los pesos son valores que la red ajusta durante el entrenamiento para mejorar su rendimiento.

Cada entrada (*x_i*) se multiplica por su peso asociado (*w_i*). Si tenemos "n" entradas, tendremos "n" pesos.

#### 11.2.3.2 Combinación Lineal

La neurona suma todas las entradas multiplicadas por sus respectivos pesos. Esto se conoce como una combinación lineal.

$$z = w1x1 + w2x2 + \cdots + wnxn + b$$

Donde:

***z*** *es la combinación lineal.*
***w_i*** *es el peso asociado a la entrada* ***x_i****.*
***b*** *es el sesgo.*

#### 11.2.3.3 Sesgo (Bias)

El sesgo es otro parámetro que se ajusta durante el entrenamiento. Se suma a la combinación lineal mencionada anteriormente. Actúa como un tipo de corrección o desplazamiento, permitiendo que la neurona produzca salidas diferentes incluso cuando todas las entradas son cero.

#### 11.2.3.4 Función de Activación

Después de calcular la combinación lineal, esta se pasa por una función de activación. La función de activación introduce no linealidades al modelo, lo que le permite aprender relaciones más complejas. Existen varias funciones de activación, como:

- **Sigmoide:** Produce una salida entre 0 y 1.
- **ReLU (Rectified Linear Unit):** Produce una salida que es el máximo entre 0 y la entrada.
- **Tanh:** Produce una salida entre -1 y 1.
- **Softmax:** Usada en la capa de salida de problemas de clasificación multiclase, convierte las entradas en una distribución de probabilidad.

El resultado de la función de activación es la salida de la neurona, y puede ser usado como entrada para otras neuronas en la siguiente capa de la red.

En resumen:

1. Cada entrada se multiplica por un peso.
2. Se suman todos los productos de entrada-peso.
3. Se añade un sesgo.
4. El resultado se pasa por una función de activación para obtener la salida final de la neurona.

El proceso de entrenamiento ajusta los pesos y sesgos de la red neuronal con el objetivo de minimizar el error en las predicciones del modelo. Para ello, se utiliza un algoritmo de optimización, como por ejemplo el descenso del gradiente, que ajusta iterativamente los pesos y sesgos en función del error obtenido en las predicciones previas.

### 11.2.4 Red Neuronal Convolucional (CNN)

Para entender cómo funciona una CNN se describe a continuación su estructura de capas y su propósito:

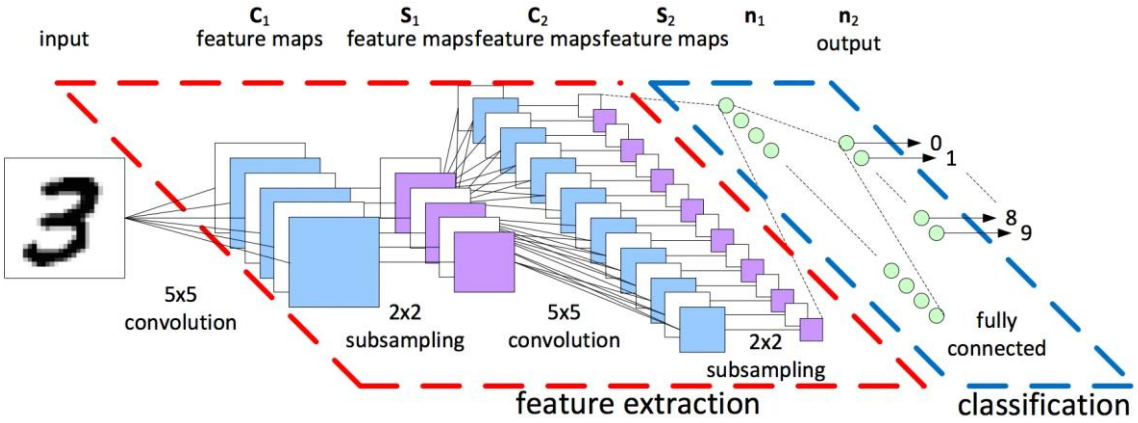



*Figura 41. Mapa de una CNN*

1. **Capa de entrada:** La entrada a una CNN generalmente es una imagen, que es simplemente una matriz con los valores de cada píxel. Una imagen a color en formato RGB tendrá una matriz de 3 canales, una para cada color (Rojo, Verde y Azul), y por lo tanto en nuestro caso, una dimensión de [300,300,3] (ya que las imágenes generadas miden 300x300 pixeles). Como estamos utilizando imágenes en blanco y negro solo necesitaremos un canal para una imagen en escala de grises, y nuestras matrices de entrada tendrán un tamaño de [300,300,1].

2. **Capas de convolución:** Aquí es donde la red utiliza un conjunto de filtros y los desliza a través de la imagen de entrada para producir mapas de características (también llamados "*feature maps*"). La idea es que cada filtro se activará con ciertas características visuales de la entrada, como bordes, texturas, colores, etc. La combinación de esos filtros nos ayuda a conseguir el objetivo de extraer características de alto nivel de la imagen, como bordes, texturas, partes de objetos, y en nuestro caso, formas.

3. **Función de activación:** Tras la operación de convolución, se aplica una función de activación, típicamente la función ReLU (Rectified Linear Unit). Esto se hace para introducir no linealidad a la red. Las funciones de activación eliminan cualquier valor negativo, reemplazándolo por cero, lo que permite a la red aprender de los errores y mejorar su precisión en las predicciones.

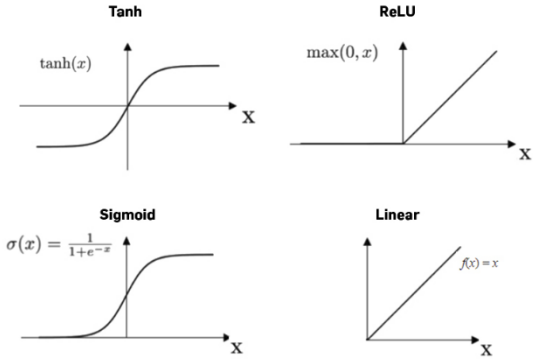



*Figura 42. Funciones de activación comunes*

4. **Capas de pooling o subsampling:** Esta etapa tiene como objetivo reducir la dimensionalidad de los mapas de características conservando la información más importante. Esto se logra tomando una ventana (por ejemplo, de tamaño 2x2) y desplazándola a través del mapa de características, y en cada paso, tomando el valor máximo (Max Pooling) o el promedio (Average Pooling) de los valores en la ventana. El Pooling ayuda a hacer la representación más robusta y resistente a variaciones menores, además de reducir el costo computacional al disminuir las dimensiones.

5. **Aplanamiento (flattening):** Después de varias etapas de convolución y pooling, los mapas de características resultantes se aplanan en un solo vector. Este vector se alimenta a una red neuronal totalmente conectada para el proceso de clasificación.

6. **Capas de red Neuronal totalmente conectada (fully connected):** Esta parte se asemeja a las redes neuronales tradicionales. El vector aplanado se conecta a una red de neuronas donde se realiza el proceso de clasificación. Se pueden tener múltiples capas ocultas en esta etapa. La última capa de esta red tiene tantas neuronas como clases de salida. Por ejemplo, para un problema de clasificación de 10 clases (como los dígitos del 0 al 9), habrá 10 neuronas en la capa de salida. En nuestro caso tendrá 21 clases, 3 por cada uno de los 7 polígonos.

7. **Función de pérdida:** Durante el entrenamiento, la CNN necesita una medida de cuán bien (o mal) está realizando sus predicciones. La función de pérdida, también llamada función de coste, proporciona esta medida al comparar la salida predicha con la salida verdadera. Una función de pérdida efectiva ayuda a guiar la optimización del modelo, proporcionando un gradiente que indica cómo cambiar los parámetros (pesos y sesgos) para mejorar las predicciones. Las funciones más comunes suelen ser el Error Cuadrático Medio (MSE), la Entropía Cruzada o el “Hinge Loss”.

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

*Figura 43. Formula MSE*

8. **Optimización:** Para ajustar iterativamente los parámetros del modelo y así minimizar la función de pérdida se utiliza un algoritmo de optimización, como el Descenso de Gradiente Estocástico (SGD) o el ADAM. Este ajustará los pesos y sesgos de la red con el objetivo de asegurar una convergencia más rápida y eficiente a una solución que minimice la pérdida. También puede ayudar a superar problemas como quedarse atascado en mínimos locales.

Las CNNs pasan por un proceso iterativo de entrenamiento, donde cada iteración mejora la precisión del modelo al ajustar los pesos y sesgos en función del error. Una vez entrenada, la CNN puede usarse para hacer predicciones en imágenes nuevas y no vistas anteriormente.

#### 11.2.4.1 Código implementado en la CNN

Se implementa el código de la red neuronal convolucional usando TensorFlow (a través de Keras). Este script define, entrena y evalúa una CNN para procesar imágenes en escala de grises. El modelo es relativamente profundo con cuatro capas convolucionales seguidas por una red neuronal densamente conectada. A través de las métricas y gráficas, podremos obtener una idea del rendimiento del modelo y cómo evoluciona durante el entrenamiento.

Los aspectos más importantes de este script son (nota: los parámetros comentados son del último modelo implementado):

- **Importaciones:** Se importan diversas librerías esenciales para el trabajo con redes neuronales y manipulación de datos, incluidos numpy, matplotlib, keras y sklearn.

- **Hiperparámetros:** Define las variables que servirán de parámetros, que son los que determinan cómo se construye y entrena el modelo. Estos incluyen el número de filtros en cada capa convolucional, el tamaño del kernel, la función de activación, el optimizador, entre otros. Al organizarlos al principio del código se nos hará más cómodo poder modificarlos durante los entrenamientos.

- **Carga de imágenes problema y etiquetas:** Utilizamos la función glob para obtener una lista de todos los archivos de imagen y etiquetas en las carpetas especificadas. Se carga cada imagen en escala de grises y se convierten, junto a sus etiquetas correspondientes, en arrays de numpy.

- **Preprocesamiento:** Las imágenes se normalizan dividiendo por 255 para que todos los valores de píxeles estén en el rango [0,1]. Se divide el conjunto de datos en conjuntos de entrenamiento, validación y prueba, a razón de un 70%, 15% y 15% respectivamente.

- **Definición del modelo de capas convolucionales y de pooling:** Se definen las primeras capas que procesan las imágenes. El modelo tiene cuatro capas convolucionales con max-pooling después de cada una.

- **Aplanamiento (Flatten):** Después de las capas convolucionales, los datos son aplanados (vector) para ser introducidos en una red neuronal densamente conectada.

- **Capas densamente conectadas:** Se programan tres capas densas con 512, 256 y 128 neuronas respectivamente, todas con función de activación ReLU. La capa de salida tiene 21 neuronas, que son nuestras salidas del problema de regresión con 21 clases.

- **Compilación del Modelo:** Se utiliza el optimizador "adam", una función de pérdida de "mse" (error cuadrático medio) y se rastrea el "mae" (error absoluto medio) como métrica para analizar el funcionamiento del modelo.

- **Entrenamiento del Modelo:** Se define un callback para guardar el mejor modelo y otro para detener el entrenamiento temprano si el modelo no mejora después de un cierto número de épocas (determinado por patience_val). Aquí se entrena

el modelo con los datos de entrenamiento y validación, utilizando los hiperparámetros definidos anteriormente.

- **Evaluación del Modelo:** Realizado el entrenamiento se evalúa el rendimiento del modelo con el conjunto de prueba y se imprimen las métricas en pantalla.

- **Visualización**: Se crea una gráfica que muestra cómo evolucionan el valor de perdida y el error absoluto medio (MAE) en el conjunto de entrenamiento y validación a medida que avanzan las épocas.

#### 11.2.4.2 Resultados entrenamientos de la CNN

En esta sección se describen los resultados obtenidos durante el entrenamiento y los cambios realizados para cada nuevo entrenamiento. No se detallan aquí todos los entrenamientos, solo los que se consideran más relevantes para entender como se ha llevado a cabo el desarrollo del mismo.

##### 11.2.4.2.1 Modelo CNN 1

El primer diseño es una red convolucional pequeña para realizar las primeras pruebas.

Hiperparámetros:

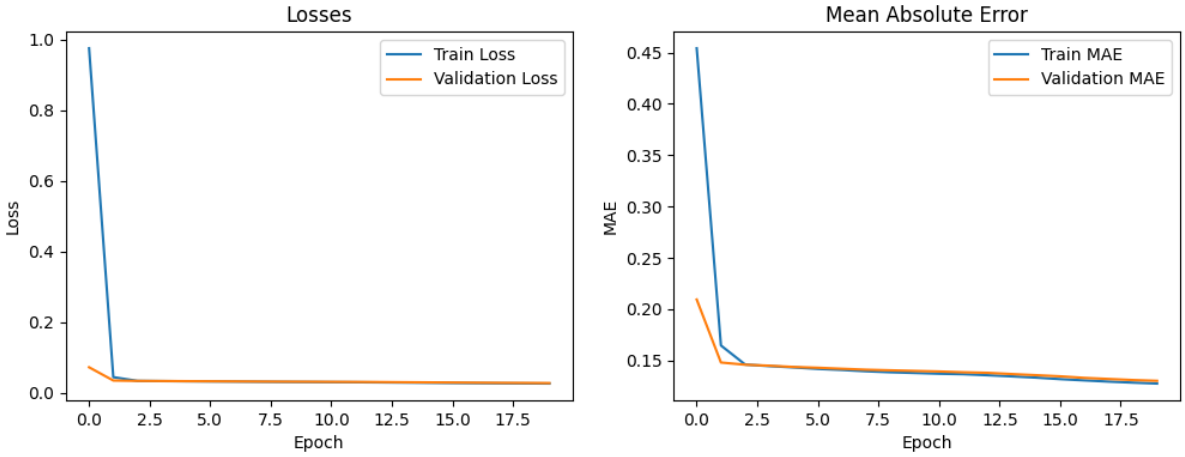
- Número de filtros en capas convolucionales: [32, 64]
- Tamaño de kernel en capas convolucionales: (3, 3)
- Función de activación en capas convolucionales: relu
- Número de capas densamente conectadas: [64, 21]
- Optimizador: adam
- Función de pérdida: mse
- Métricas: ['mae']
- Tamaño del lote (batch size): 16
- Número de épocas: 25

Tabla 3. Sumario Modelo CNN 1.

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d (Conv2D) | (None, 298, 298, 32) | 320 |
| max_pooling2d (MaxPooling2D) | (None, 149, 149, 32) | 0 |
| conv2d_1 (Conv2D) | (None, 147, 147, 64) | 18496 |
| max_pooling2d_1 (MaxPooling 2D) | (None, 73, 73, 64) | 0 |
| flatten (Flatten) | (None, 341056) | 0 |
| dense (Dense) | (None, 64) | 21.827.648 |
| dense_1 (Dense) | (None, 21) | 2709 |
| | | |
| Total params: 21.849.173 | | |
| Trainable params: 21.849.173 | | |
| Non-trainable params: 0 | | |

Rendimiento en datos de prueba:

- Test Loss (MSE): 0.04539
- Test MAE: 0.1328




*Gráfica 1. CNN 1: Funciones de perdida, MSE y MAE*

En la primera prueba podemos observar como la red no acaba de encontrar un patrón y simplemente devuelve lo que parece ser un valor medio que ha encontrado que minimiza el error. Las predicciones no parecen encontrar nada en común con las soluciones reales.

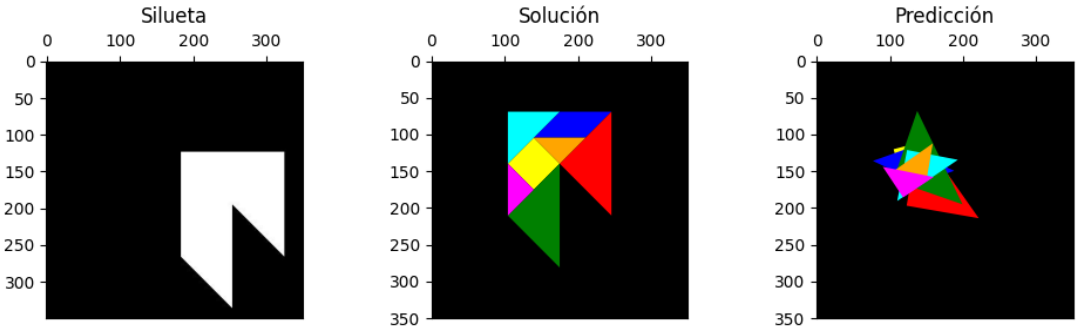



*Figura 44. CNN1: Prueba de visualización predicciones [problema nº100]*

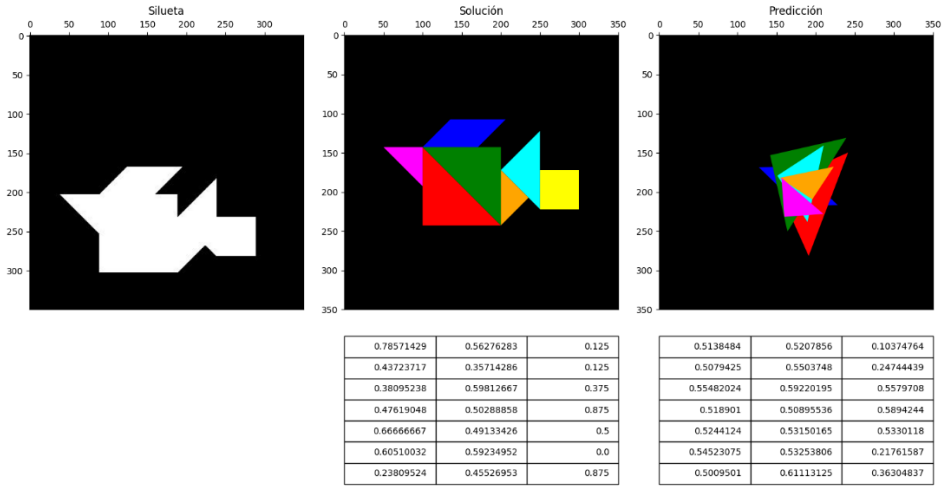
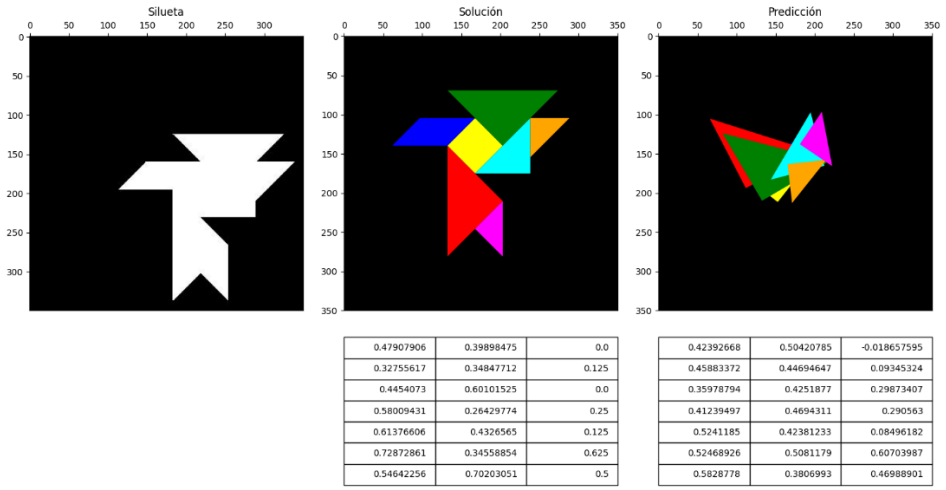


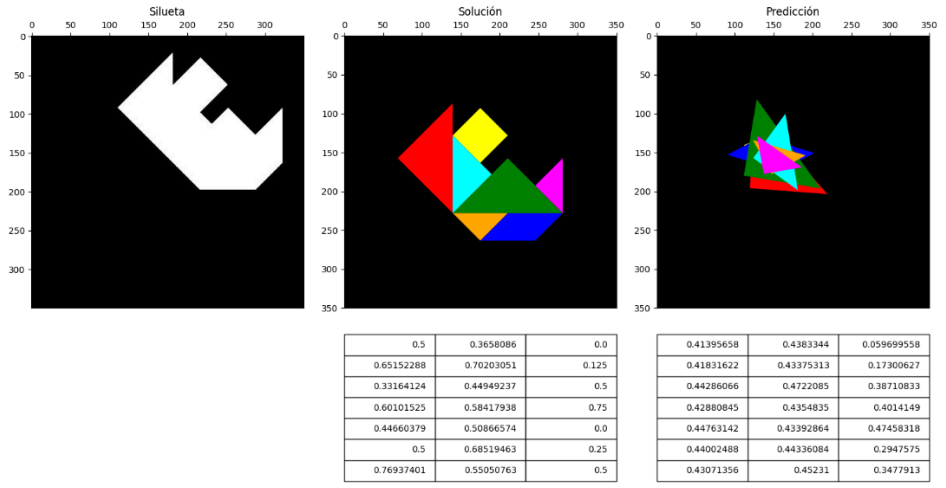

*Figura 45. CNN1: Visualización predicciones con valores de salida [problema 1200]*




| | | |
|---|---|---|
| 0.47907906 | 0.39898475 | 0.0 |
| 0.32755617 | 0.34847712 | 0.125 |
| 0.4454073 | 0.60101525 | 0.0 |
| 0.58009431 | 0.26429774 | 0.25 |
| 0.61376606 | 0.4326565 | 0.125 |
| 0.72872861 | 0.34558854 | 0.625 |
| 0.54642256 | 0.70203051 | 0.5 |

| | | |
|---|---|---|
| 0.42392668 | 0.50420785 | -0.018657595 |
| 0.45883372 | 0.44694647 | 0.09345324 |
| 0.35978794 | 0.4251877 | 0.29873407 |
| 0.41239497 | 0.4694311 | 0.290563 |
| 0.5241185 | 0.42381233 | 0.08496182 |
| 0.52468926 | 0.5081179 | 0.60703987 |
| 0.5828778 | 0.3806993 | 0.46988901 |

*Figura 46. CNN1: Visualización predicciones con valores de salida [problema 2000]*




| | | |
|---|---|---|
| 0.5 | 0.3658086 | 0.0 |
| 0.65152288 | 0.70203051 | 0.125 |
| 0.33164124 | 0.44949237 | 0.5 |
| 0.60101525 | 0.58417938 | 0.75 |
| 0.44660379 | 0.50866574 | 0.0 |
| 0.5 | 0.68519463 | 0.25 |
| 0.76937401 | 0.55050763 | 0.5 |

| | | |
|---|---|---|
| 0.41395658 | 0.4383344 | 0.059699558 |
| 0.41831622 | 0.43375313 | 0.17300627 |
| 0.44286066 | 0.4722085 | 0.38710833 |
| 0.42880845 | 0.4354835 | 0.4014149 |
| 0.44763142 | 0.43392864 | 0.47458318 |
| 0.44002488 | 0.44336084 | 0.2947575 |
| 0.43071356 | 0.45231 | 0.3477913 |

*Figura 47. CNN1: Visualización predicciones con valores de salida [problema 3000]*

#### 11.2.4.2.2 Modelo CNN 2

Se decide aumentar la capa densamente conectada a la salida de 64 a 128, así como el tamaño del lote a 32 para añadir velocidad de entrenamiento. Se reducen el número de épocas y se añade una función de “Early Stopping” que detiene el entrenamiento si durante tres épocas la función de perdida de los datos test no ha mejorado.

**Hiperparámetros:**

- Número de filtros en capas convolucionales: [32, 64]
- Tamaño de kernel en capas convolucionales: (3, 3)
- Función de activación en capas convolucionales: relu
- Número de capas densamente conectadas: [128, 21]
- Optimizador: adam
- Función de pérdida: mse
- Métricas: ['mae']

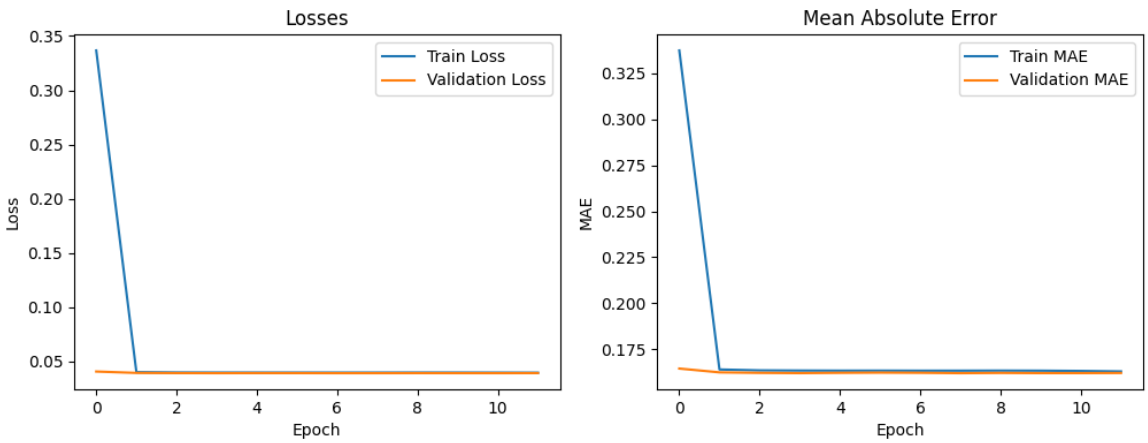
- Tamaño del lote (batch size): 32
- Número de épocas: 12
- Paciencia para Early Stopping: 3

Tabla 4. Sumario Modelo CNN 2.

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d (Conv2D) | (None, 298, 298, 32) | 320 |
| max_pooling2d (MaxPooling2D) | (None, 149, 149, 32) | 0 |
| conv2d_1 (Conv2D) | (None, 147, 147, 64) | 18496 |
| max_pooling2d_1 (MaxPooling 2D) | (None, 73, 73, 64) | 0 |
| flatten (Flatten) | (None, 341056) | 0 |
| dense (Dense) | (None, 128) | 43.655.296 |
| dense_1 (Dense) | (None, 21) | 2709 |
| | | |
| Total params: 43.676.821 | | |
| Trainable params: 43.676.821 | | |
| Non-trainable params: 0 | | |
| | | |
| Test Loss: [0.03993738070130348, 0.16340988874435425] | | |

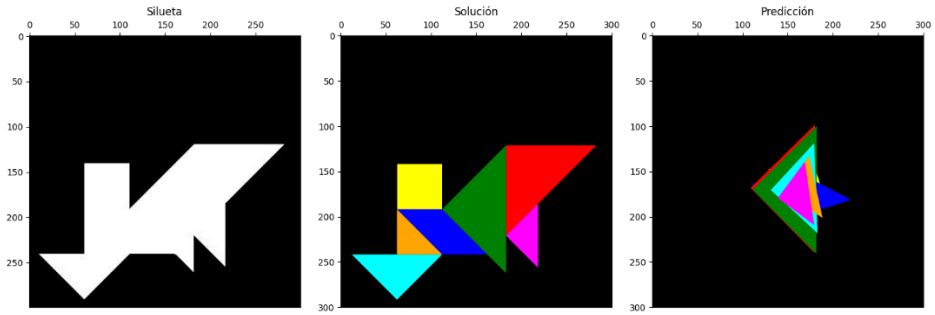
**Rendimiento en datos de prueba:**

- Test Loss (MSE): 0.0399
- Test MAE: 0.1634




*Gráfica 2. CNN 2: Funciones de perdida, MSE y MAE*

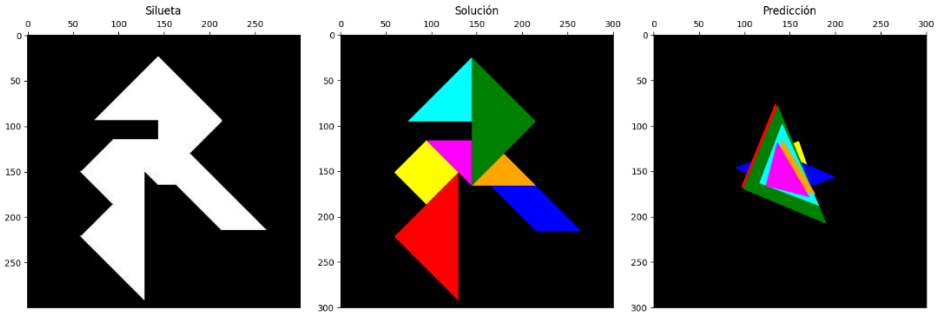
Con estas modificaciones no se obtienen mejoras en las predicciones.




| | | |
|---|---|---|
| 0.29096441 | 0.55559223 | 0.125 |
| 0.37429774 | 0.7222589 | 0.25 |
| 0.72111111 | 0.51433442 | 0.625 |
| 0.53143258 | 0.63892557 | 0.5 |
| 0.20763107 | 0.86114779 | 0.25 |
| 0.26318663 | 0.75003668 | 0.375 |
| 0.68856742 | 0.73655664 | 0.5 |

| | | |
|---|---|---|
| 0.5330733 | 0.5484316 | 0.07983334 |
| 0.54983467 | 0.5756764 | 0.20047823 |
| 0.52019 | 0.5618963 | 0.5014393 |
| 0.5277729 | 0.56700546 | 0.50069475 |
| 0.5484501 | 0.56379366 | 0.49378362 |
| 0.5651063 | 0.5645844 | 0.4657289 |
| 0.54123795 | 0.5861393 | 0.47674078 |

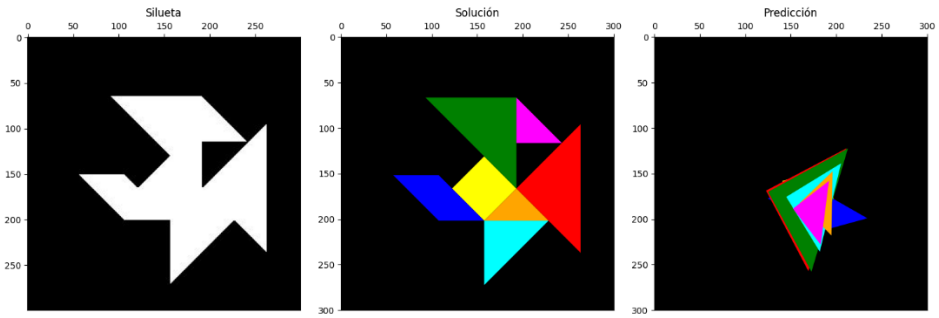
*Figura 48. CNN2: Visualización predicciones con valores de salida [problema 1000]*




| | | |
|---|---|---|
| 0.31515212 | 0.50418771 | 0.0 |
| 0.71752104 | 0.63633658 | 0.25 |
| 0.35443583 | 0.73988997 | 0.5 |
| 0.5603862 | 0.31730099 | 0.0 |
| 0.40325136 | 0.23873357 | 0.125 |
| 0.59966991 | 0.51371954 | 0.75 |
| 0.42626323 | 0.44189214 | 0.875 |

| | | |
|---|---|---|
| 0.48158124 | 0.49519783 | 0.07162777 |
| 0.48260406 | 0.5042398 | 0.19065495 |
| 0.4655984 | 0.4967211 | 0.4367001 |
| 0.47270602 | 0.50557905 | 0.4379716 |
| 0.48876804 | 0.49801677 | 0.43284258 |
| 0.49997395 | 0.5039462 | 0.4137846 |
| 0.47908747 | 0.5135092 | 0.4158001 |

*Figura 49. CNN2: Visualización predicciones con valores de salida [problema 2000]*




| | | |
|---|---|---|
| 0.52537217 | 0.55440777 | 0.0 |
| 0.35870551 | 0.58892557 | 0.25 |
| 0.80035814 | 0.55440777 | 0.5 |
| 0.53211219 | 0.33218555 | 0.875 |
| 0.60393959 | 0.75082632 | 0.625 |
| 0.6432233 | 0.63297519 | 0.75 |
| 0.69877886 | 0.33218555 | 0.375 |

| | | |
|---|---|---|
| 0.5700798 | 0.5866899 | 0.08573056 |
| 0.5981522 | 0.6270168 | 0.20753807 |
| 0.55942416 | 0.6087368 | 0.5479663 |
| 0.56734866 | 0.61115164 | 0.5457729 |
| 0.59134275 | 0.6110665 | 0.537581 |
| 0.61191595 | 0.6081641 | 0.5030605 |
| 0.58590454 | 0.6383375 | 0.52053785 |

*Figura 50. CNN2: Visualización predicciones con valores de salida [problema 3000]*

#### 11.2.4.2.3 Modelo CNN 3

Se añade una tercera capa convolucional y una capa densamente conectada a la salida aplanada de las capas convolucionales con 256 neuronas.

**Hiperparámetros:**

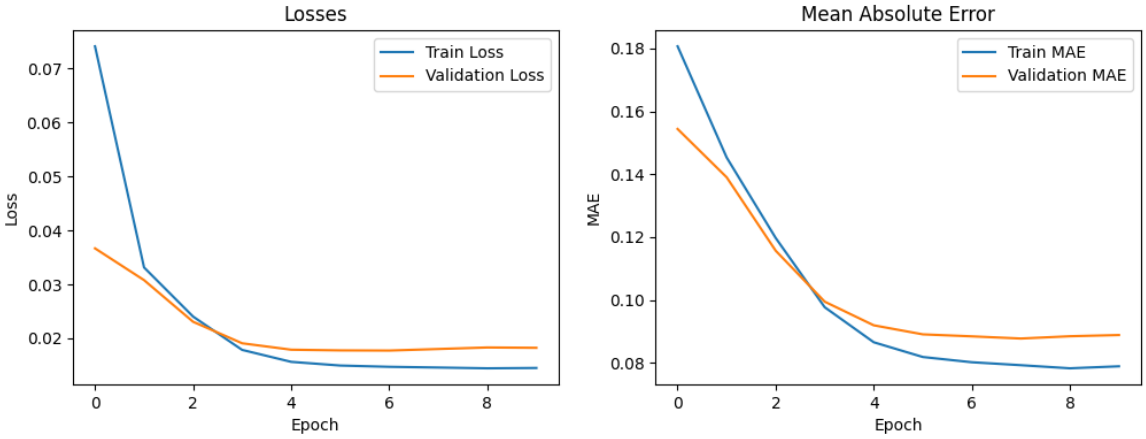
- Número de filtros en capas convolucionales: [32, 64, 128]
- Tamaño de kernel en capas convolucionales: (3, 3)
- Función de activación en capas convolucionales: relu
- Número de capas densamente conectadas: [256, 128, 21]
- Optimizador: adam
- Función de pérdida: mse
- Métricas: ['mae']
- Tamaño del lote (batch size): 32
- Número de épocas: 12
- Paciencia para Early Stopping: 3

Tabla 5. Sumario Modelo CNN 3.

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d (Conv2D) | (None, 298, 298, 32) | 320 |
| max_pooling2d (MaxPooling2D) | (None, 149, 149, 32) | 0 |
| conv2d_1 (Conv2D) | (None, 147, 147, 64) | 18496 |
| max_pooling2d_1 (MaxPooling 2D) | (None, 73, 73, 64) | 0 |
| conv2d_2 (Conv2D) | (None, 71, 71, 128) | 73856 |
| max_pooling2d_2 (MaxPooling 2D) | (None, 35, 35, 128) | 0 |
| flatten (Flatten) | (None, 156800) | 0 |
| dense (Dense) | (None, 256) | 40141056 |
| dense_1 (Dense) | (None, 128) | 32896 |
| dense_2 (Dense) | (None, 21) | 2709 |
| | | |
| Total params: 40,269,333 | | |
| Trainable params: 40,269,333 | | |
| Non-trainable params: 0 | | |

**Rendimiento en datos de prueba:**

- Test Loss (MSE): 0.0183
- Test MAE: 0.0930

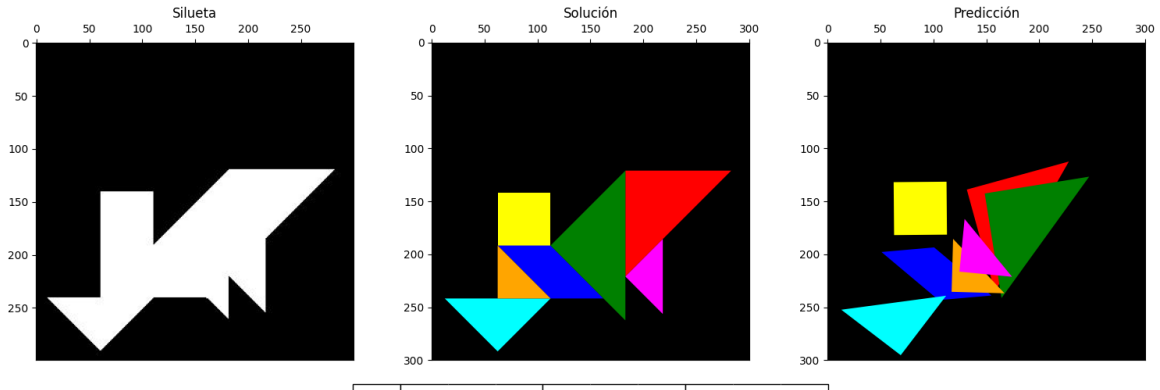



*Gráfica 3. CNN 3: Funciones de perdida, MSE y MAE*

Con este diseño de capas y configuración de hiperparametros se obtiene una mejora considerable en el error de salida. Sin embargo, si analizamos las imágenes de salida, se observa que los polígonos que minimizan el error general son el cuadrado, paralelogramo y triangulo mediano. Se analizan más de 20 predicciones, y en todas ellas estas 3 piezas están muy cerca de la solución real, mientras que los triángulos grandes y pequeños no.

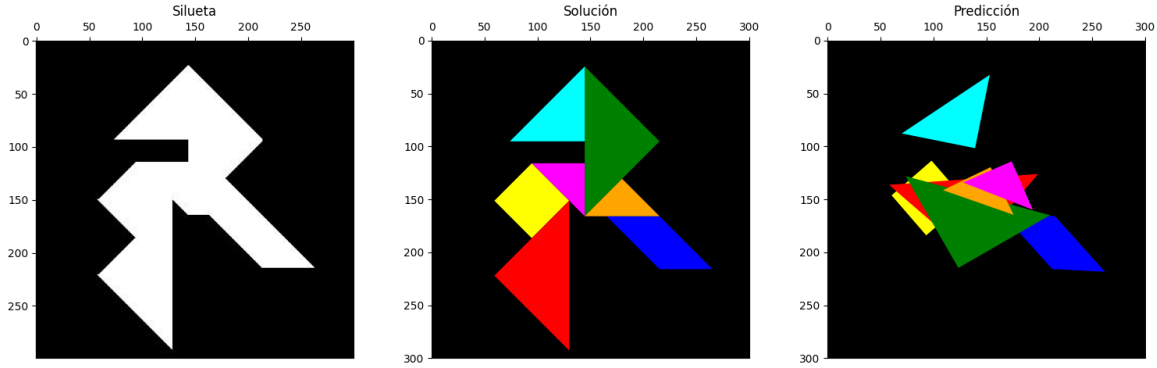
Esto nos lleva a pensar que la permutación de los triángulos que se había pensado en el diseño del dataset, pueda estar creando problemas a la red neuronal. Es decir, al entregarle la misma silueta problema con 4 soluciones distintas (intercambiando los triángulos), no es capaz de generalizar y simplemente encuentra un valor medio que minimiza el error para las 4 soluciones.

Con esto en mente, se decide eliminar las permutaciones de los triángulos del dataset, de tal manera que a cada problema le corresponda solo una solución, y que sea la red la encargada de generalizar la resolución.



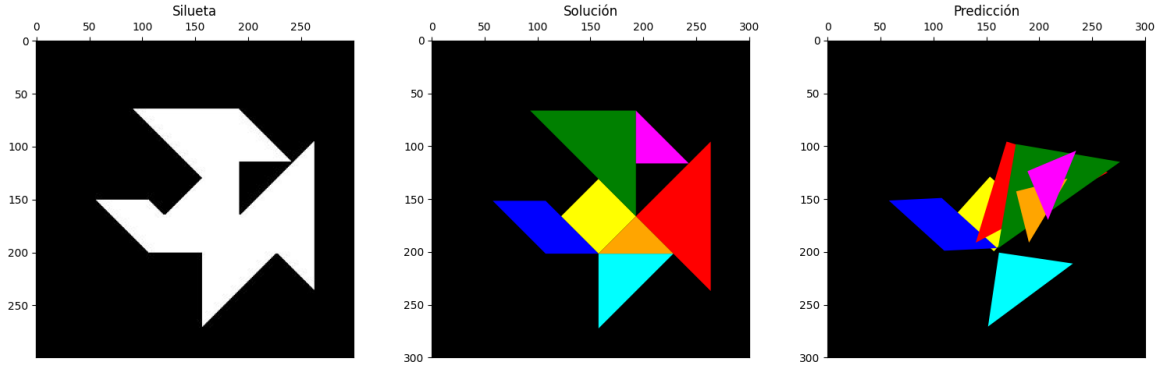

| | Center x | | | Center y | | | Orientation | | |
|---|---|---|---|---|---|---|---|---|---|
| Forma | Real | Predicción | Error | Real | Predicción | Error | Real | Predicción | Error |
| C | 0.29096448 | 0.29202768 | 0.37% | 0.55559220 | 0.52208322 | 6.03% | 0.12500000 | 0.12367682 | 1.06% |
| P | 0.37429774 | 0.34283102 | 8.41% | 0.72225890 | 0.72787482 | 0.78% | 0.25000000 | 0.23613447 | 5.55% |
| TG1 | 0.72111111 | 0.57507360 | 20.25% | 0.51433442 | 0.54051602 | 5.09% | 0.62500000 | 0.58236426 | 6.82% |
| TG2 | 0.53143258 | 0.62226337 | 17.09% | 0.63892550 | 0.56719786 | 11.23% | 0.50000000 | 0.59960864 | 19.92% |
| TM | 0.20763101 | 0.21560654 | 3.84% | 0.86114770 | 0.87407611 | 1.50% | 0.25000000 | 0.22888875 | 8.44% |
| TP1 | 0.26318663 | 0.44768143 | 70.10% | 0.75003660 | 0.73067611 | 2.58% | 0.37500000 | 0.37948400 | 1.20% |
| TP2 | 0.68856741 | 0.47593915 | 30.88% | 0.73655664 | 0.67125511 | 8.87% | 0.50000000 | 0.39057142 | 21.89% |

*Figura 51. CNN3: Visualización predicciones con valores de salida [problema 1000]*




| | Center x | | | Center y | | | Orientation | | |
|---|---|---|---|---|---|---|---|---|---|
| Forma | Real | Predicción | Error | Real | Predicción | Error | Real | Predicción | Error |
| C | 0.31515212 | 0.31906310 | 1.24% | 0.50418777 | 0.49565670 | 1.69% | 0.00000000 | 0.01142578 | 2829542675.61% |
| P | 0.71752104 | 0.71198822 | 0.77% | 0.63633650 | 0.63605618 | 0.04% | 0.25000000 | 0.25831632 | 3.33% |
| TG1 | 0.35443581 | 0.43408120 | 22.47% | 0.73988990 | 0.51554364 | 30.32% | 0.50000000 | 0.23841603 | 52.32% |
| TG2 | 0.56038620 | 0.45281168 | 19.20% | 0.31730090 | 0.56415719 | 77.80% | 0.00000000 | 0.29199207 | 29790727186.20% |
| TM | 0.40325136 | 0.40298352 | 0.07% | 0.23873350 | 0.24631622 | 3.18% | 0.12500000 | 0.15649836 | 25.20% |
| TP1 | 0.59966990 | 0.48756316 | 18.69% | 0.51371950 | 0.47272196 | 7.98% | 0.75000000 | 0.80347371 | 7.13% |
| TP2 | 0.42626320 | 0.55104047 | 29.27% | 0.44189210 | 0.45327774 | 2.58% | 0.87500000 | 0.80933189 | 7.50% |

*Figura 52. CNN3: Visualización predicciones con valores de salida [problema 2000]*




| | Center x | | | Center y | | | Orientation | | |
|---|---|---|---|---|---|---|---|---|---|
| Forma | Real | Predicción | Error | Real | Predicción | Error | Real | Predicción | Error |
| C | 0.52537217 | 0.51770043 | 1.46% | 0.55440777 | 0.54702544 | 1.33% | 0.00000000 | 0.00852667 | 4732664.23% |
| P | 0.35870552 | 0.36350960 | 1.34% | 0.58892550 | 0.57979184 | 1.55% | 0.25000000 | 0.24213867 | 3.14% |
| TG1 | 0.80035814 | 0.63764668 | 20.33% | 0.55440777 | 0.45754313 | 17.47% | 0.50000000 | 0.67208767 | 34.42% |
| TG2 | 0.53211219 | 0.68378627 | 28.50% | 0.33218555 | 0.45501399 | 36.98% | 0.87500000 | 0.65191424 | 25.50% |
| TM | 0.60393959 | 0.60605079 | 0.35% | 0.75082631 | 0.75821066 | 0.98% | 0.62500000 | 0.64836985 | 3.74% |
| TP1 | 0.64322330 | 0.66128409 | 2.81% | 0.63297510 | 0.51631492 | 18.43% | 0.75000000 | 0.58613259 | 21.85% |
| TP2 | 0.69877880 | 0.70222342 | 0.49% | 0.33218555 | 0.44191116 | 33.03% | 0.37500000 | 0.56145835 | 49.72% |

*Figura 53. CNN3: Visualización predicciones con valores de salida [problema 3000]*

#### 11.2.4.2.4 Modelo CNN 4

Se modifica el dataset para eliminar los datos con permutación de triángulos y se cran nuevo para mantener el mismo tamaño. Además, e añade otra capa densamente conectada.

**Hiperparámetros:**

- Número de filtros en capas convolucionales: [32, 64, 128]
- Tamaño de kernel en capas convolucionales: (3, 3)
- Función de activación en capas convolucionales: relu

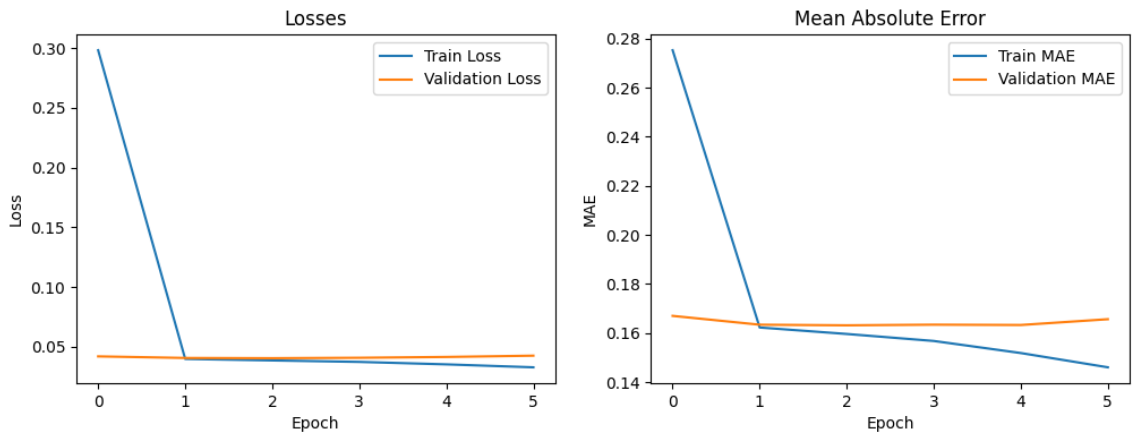
- Número de capas densamente conectadas: [256, 128, 64, 21]
- Optimizador: adam
- Función de pérdida: mse
- Métricas: ['mae']
- Tamaño del lote (batch size): 32
- Número de épocas: 12
- Número de épocas: 12
- Paciencia para Early Stopping: 3

Tabla 6. Sumario Modelo CNN 4.

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d (Conv2D) | (None, 298, 298, 32) | 320 |
| max_pooling2d (MaxPooling2D) | (None, 149, 149, 32) | 0 |
| conv2d_1 (Conv2D) | (None, 147, 147, 64) | 18496 |
| max_pooling2d_1 (MaxPooling 2D) | (None, 73, 73, 64) | 0 |
| conv2d_2 (Conv2D) | (None, 71, 71, 128) | 73856 |
| max_pooling2d_2 (MaxPooling 2D) | (None, 35, 35, 128) | 0 |
| flatten (Flatten) | (None, 156800) | 0 |
| dense (Dense) | (None, 256) | 40141056 |
| dense_1 (Dense) | (None, 128) | 32896 |
| dense_2 (Dense) | (None, 64) | 8256 |
| dense_3 (Dense) | (None, 21) | 1365 |
| | | |
| Total params: 40,276,245 | | |
| Trainable params: 40,276,245 | | |
| Non-trainable params: 0 | | |

**Rendimiento en datos de prueba:**

- Test Loss (MSE): 0.0413
- Test MAE: 0.1629

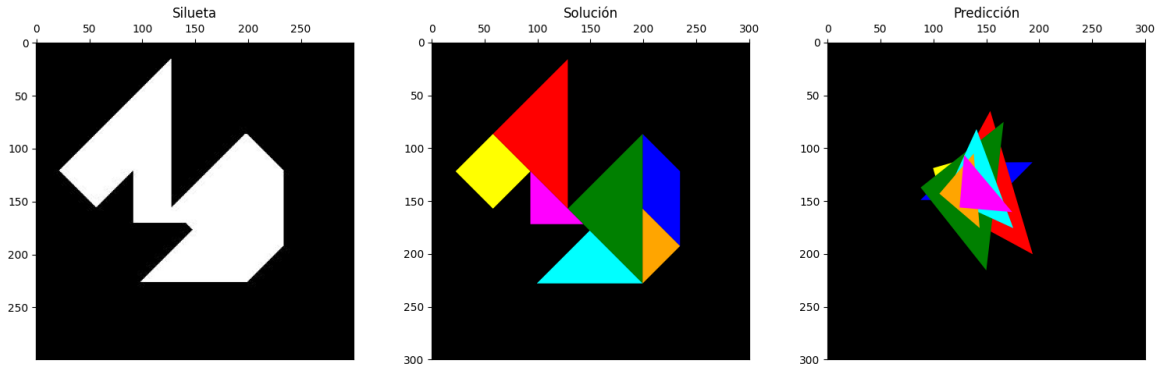



*Gráfica 4. CNN 4: Funciones de perdida, MSE y MAE*

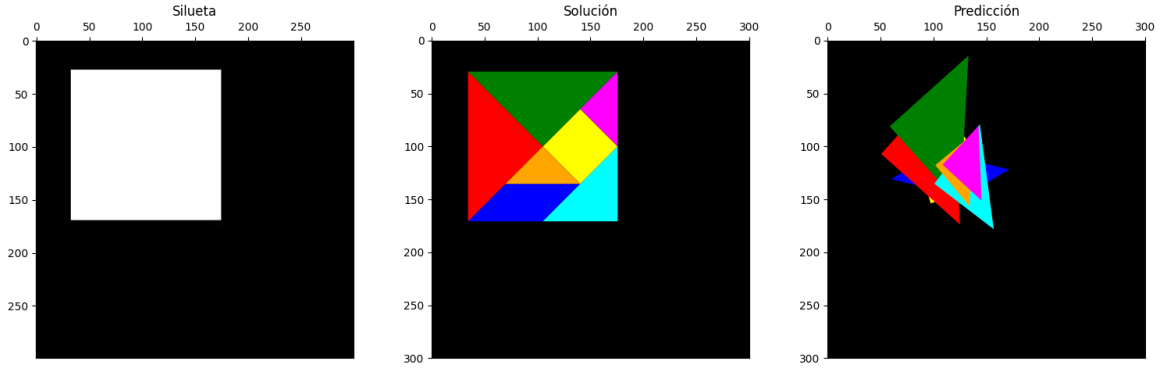
Sin las permutaciones se esperaba una mejora en el rendimiento del modelo, pero se obtiene todo lo contrario, se vuelve a una función de perdida parecida a los dos primeros entrenamientos. Después de muchas pruebas, sin tener mucha certeza de porque puede pasar esto, se llega a la conclusión que, en realidad, en el diseño anterior, la red no había aprendido a colocar las piezas cuadrado, paralelogramo y triangulo mediano, sino que simplemente había memorizado una posición similar.

Al ofrecerle siempre 4 soluciones del mismo problema, es muy probable que, al pasar un problema de test, la red ya haya visto 2 o 3 soluciones validas durante el entrenamiento, de ahí que las piezas que siempre estaban en el mismo sitio las coloque, aproximadamente bien. Porque ya ha visto el más de una solución al problema.

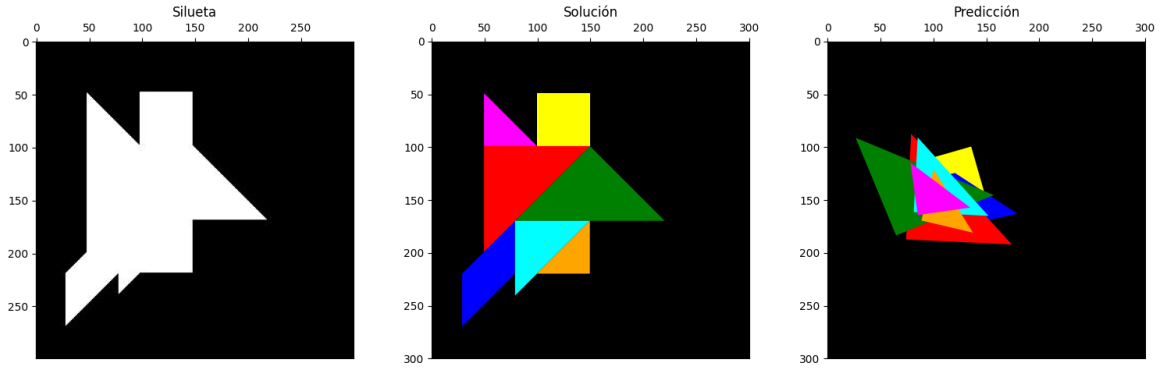
Se crean 10 problemas nuevos con la aplicación, y se vuelve a probar con el modelo anterior, comprobando como nuestras suposiciones eran ciertas, solo aproxima las piezas cuando ya ha visto varias soluciones al mismo problema, y por lo tanto la red no esta generalizando ni sabe resolver realmente el problema.




| | Center x | | | Center y | | | Orientation | | |
|---|---|---|---|---|---|---|---|---|---|
| Forma | Real | Predicción | Error | Real | Predicción | Error | Real | Predicción | Error |
| C | 0.19289321 | 0.43516624 | 125.60% | 0.40622655 | 0.45333865 | 11.60% | 0.00000000 | 0.08060455 | 4732036.59% |
| P | 0.72322330 | 0.46909019 | 35.14% | 0.46515211 | 0.43727651 | 5.99% | 0.37500000 | 0.12438191 | 66.83% |
| TG1 | 0.35002808 | 0.50395453 | 43.98% | 0.28837542 | 0.46418482 | 60.97% | 0.50000000 | 0.45417562 | 9.16% |
| TG2 | 0.58573031 | 0.44913572 | 23.32% | 0.52407760 | 0.47549361 | 9.27% | 0.50000000 | 0.51850420 | 3.70% |
| TM | 0.49763101 | 0.47480506 | 4.59% | 0.70422430 | 0.44867510 | 36.29% | 0.75000000 | 0.44345593 | 40.87% |
| TP1 | 0.70358149 | 0.43044046 | 38.82% | 0.64192880 | 0.47057596 | 26.69% | 0.00000000 | 0.48744174 | 41718578.34% |
| TP2 | 0.36629990 | 0.47576010 | 29.88% | 0.51733760 | 0.46986631 | 9.18% | 0.37500000 | 0.38883984 | 3.69% |




| | Center x | | | Center y | | | Orientation | | |
|---|---|---|---|---|---|---|---|---|---|
| Forma | Real | Predicción | Error | Real | Predicción | Error | Real | Predicción | Error |
| C | 0.46785113 | 0.37878205 | 19.04% | 0.33333333 | 0.40756485 | 22.27% | 0.00000000 | 0.07549252 | 95% |
| P | 0.29107448 | 0.38599372 | 32.61% | 0.51011100 | 0.42109707 | 17.45% | 0.12500000 | 0.16311388 | 30.49% |
| TG1 | 0.19286516 | 0.32567543 | 68.86% | 0.33333333 | 0.34825596 | 4.48% | 0.00000000 | 0.49120951 | 45% |
| TG2 | 0.35000000 | 0.35245846 | 0.70% | 0.17619840 | 0.27857468 | 58.10% | 0.25000000 | 0.50860626 | 103.44% |
| TM | 0.50713484 | 0.44642550 | 11.97% | 0.49046811 | 0.43600842 | 11.10% | 0.12500000 | 0.47904292 | 283.23% |
| TP1 | 0.35000000 | 0.41738483 | 19.25% | 0.41190079 | 0.39955008 | 3.00% | 0.75000000 | 0.51452434 | 31.40% |
| TP2 | 0.54641859 | 0.44104028 | 19.29% | 0.21548220 | 0.38764867 | 79.90% | 0.50000000 | 0.49358881 | 1.28% |




| | Center x | | | Center y | | | Orientation | | |
|---|---|---|---|---|---|---|---|---|---|
| Forma | Real | Predicción | Error | Real | Predicción | Error | Real | Predicción | Error |
| C | 0.41473789 | 0.39506853 | 4.74% | 0.24644666 | 0.43442896 | 76.28% | 0.12500000 | 0.08058959 | 35.53% |
| P | 0.17903559 | 0.41850650 | 133.76% | 0.73214880 | 0.49631071 | 32.21% | 0.00000000 | 0.21682168 | 29% |
| TG1 | 0.27584891 | 0.36351004 | 31.78% | 0.44089105 | 0.51854885 | 17.61% | 0.62500000 | 0.38254511 | 38.79% |
| TG2 | 0.49807119 | 0.27593571 | 44.60% | 0.48691470 | 0.46660200 | 4.17% | 0.75000000 | 0.31253159 | 58.33% |
| TM | 0.34093639 | 0.35394305 | 3.81% | 0.64404960 | 0.46345091 | 28.04% | 0.62500000 | 0.38322920 | 38.68% |
| TP1 | 0.44251563 | 0.36233521 | 18.12% | 0.67659330 | 0.52324533 | 22.66% | 0.12500000 | 0.41188854 | 229.51% |
| TP2 | 0.22029340 | 0.33145204 | 50.46% | 0.27422430 | 0.48447523 | 76.67% | 0.37500000 | 0.35249564 | 6.00% |

#### 11.2.4.2.5 Modelo CNN 5

Se hace un último intento para ver si se puede mejorar la red haciendo el modelo aún más grande, para ello, se añade una capa convolucional y otra capa densamente conectada.

**Hiperparámetros:**

- Número de filtros en capas convolucionales: [32, 64, 128, 256]
- Tamaño de kernel en capas convolucionales: (3, 3)
- Función de activación en capas convolucionales: relu
- Número de capas densamente conectadas: [512, 256, 128, 21]
- Optimizador: adam
- Función de pérdida: mse
- Métricas: ['mae']

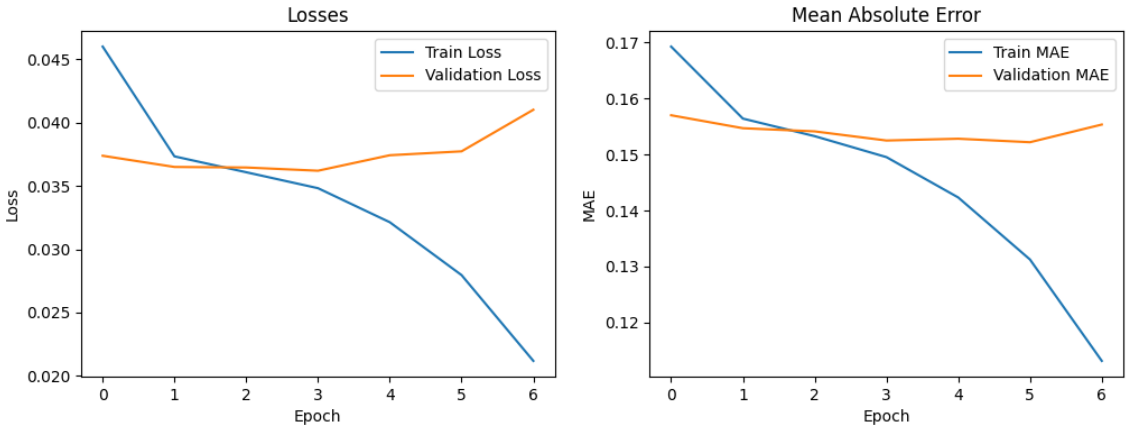
- Tamaño del lote (batch size): 32
- Número de épocas: 10
- Paciencia para Early Stopping: 3

Tabla 7. Sumario Modelo CNN 5.

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d (Conv2D) | (None, 298, 298, 32) | 320 |
| max_pooling2d (MaxPooling2D) | (None, 149, 149, 32) | 0 |
| conv2d_1 (Conv2D) | (None, 147, 147, 64) | 18496 |
| max_pooling2d_1 (MaxPooling 2D) | (None, 73, 73, 64) | 0 |
| conv2d_2 (Conv2D) | (None, 71, 71, 128) | 73856 |
| max_pooling2d_2 (MaxPooling 2D) | (None, 35, 35, 128) | 0 |
| conv2d_3 (Conv2D) | (None, 33, 33, 256) | 295168 |
| max_pooling2d_3 (MaxPooling 2D) | (None, 16, 16, 256) | 0 |
| flatten (Flatten) | (None, 65536) | 0 |
| dense (Dense) | (None, 512) | 33554944 |
| dense_1 (Dense) | (None, 256) | 131328 |
| dense_2 (Dense) | (None, 128) | 32896 |
| dense_3 (Dense) | (None, 21) | 2709 |
| | | |
| Total params: 34,109,717 | | |
| Trainable params: 34,109,717 | | |
| Non-trainable params: 0 | | |

**Rendimiento en datos de prueba:**

- Test Loss (MSE): 0.0409
- Test MAE: 0.1558




*Gráfica 5. CNN 5: Funciones de perdida, MSE y MAE*

Se comprueba al crear un modelo tan grande la red empieza mejorar el error para los datos de entrenamiento y no para los de prueba, lo que quiere decir que está cayendo en el overfiting (se sobreajuste a los datos vistos porque los "memoriza") y no generaliza bien, lo que confirma las suposiciones del entrenamiento anterior.

#### 11.2.4.3 Conclusiones entrenamiento CNN

Después de muchas horas de entrenamientos, documentación y programación se decide que quizá una CNN no sea el mejor modelo para nuestra tarea, ya que el punto fuerte de las CNN son la regresión y la clasificación, y no tanto la generación de datos nuevos (en nuestro caso, soluciones al rompecabezas).

Se decide probar con el dataset de imágenes y diseñar un autoencoder convolucional.

### 11.2.5 Red Neuronal Convolucional Autoencoder (CAE)

Un autoencoder convolucional es una variante del autoencoder tradicional diseñado específicamente para trabajar con imágenes. Aprovecha las características de las redes neuronales convolucionales (CNN) para manejar eficientemente la estructura espacial inherente en las imágenes.

Entre las utilidades de un CAE para imágenes encontramos:

- **Reducción de Dimensionalidad:** Al igual que los autoencoders tradicionales, los CAE pueden reducir la dimensionalidad de imágenes, lo que es útil para la visualización o el procesamiento posterior.
- **Detección de Anomalías:** Al entrenar un CAE con imágenes normales (por ejemplo, imágenes de productos defectuosos), puede detectar anomalías al intentar reconstruir imágenes que no siguen el patrón normal.
- **Denoising:** Los CAE son útiles para eliminar el ruido de las imágenes. Se entrenan con imágenes ruidosas como entrada y las imágenes originales (limpias) como objetivo.
- **Generación de Imágenes:** Variaciones como el Variational Convolutional Autoencoder pueden generar nuevas imágenes basadas en las características aprendidas.

Estructura de un Autoencoder Convolucional:

- **Encoder (Codificador) Convolucional:** La entrada es una imagen que se pasa a través de varias capas convolucionales. Estas capas extraen características espaciales jerárquicas de la imagen. Después de cada capa convolucional, a menudo se utiliza una capa de pooling para reducir la dimensionalidad espacial. El resultado final es una representación compacta y densamente informativa de la imagen original.
- **Decoder (Decodificador) Convolucional:** Toma la representación compacta del encoder y busca "descomprimirla" o "upsamplearla" de nuevo a la dimensión original de la imagen. Esto se logra utilizando capas de deconvolución o capas convolucionales transpuestas. La idea es aumentar gradualmente la resolución espacial a través de cada capa hasta alcanzar el tamaño original de la imagen.

El entrenamiento implica la minimización de la diferencia entre la imagen original y su reconstrucción, generalmente usando una función de pérdida como el error cuadrático medio.

#### 11.2.5.1 Código implementado en el CAE

El modelo sigue una arquitectura de autoencoder con capas de Conv2D, MaxPooling2D, Dropout para el encoder y Conv2D, UpSampling2D, Dropout para el decoder.

La capa de salida tiene 3 canales para producir imágenes en color y usa una función de activación sigmoide, que es adecuada para imágenes normalizadas en el rango [0, 1].

Se entrena el modelo utilizando el optimizador Adam y la pérdida de error cuadrático medio (MSE). El entrenamiento se detiene temprano si no se observan mejoras en la pérdida de validación después de un número específico de épocas.

A continuación se describen los aspectos más importantes del programa de diseño del CAE:

- **Importaciones:** Se importan varias bibliotecas necesarias para la manipulación de datos y la construcción y entrenamiento del modelo, incluidas numpy, matplotlib, tensorflow, sklearn, os y glob.
- **Hiperparámetros:** Se definen varios hiperparámetros para la red y el entrenamiento, incluido el número de filtros en cada capa convolucional, el tamaño del kernel, la función de activación, el optimizador, la función de pérdida, las métricas, entre otros.
- **Función para visualizar imágenes view_images:** Esta función toma imágenes de problema y solución y un índice, y muestra las imágenes de problema y solución lado a lado. Utiliza matplotlib para crear las gráficas.
- **Carga de imágenes:** Se cargan y procesan las imágenes de problema (en blanco y negro) y solución (a color) desde el disco utilizando keras.preprocessing.image.load_img y keras.preprocessing.image.img_to_array. Las imágenes se normalizan y binarizan.
- **Visualización de una muestra:** Se visualiza una muestra de las imágenes cargadas utilizando la función view_images.
- **División del conjunto de datos:** Se divide el conjunto de datos en conjuntos de entrenamiento, validación y prueba utilizando train_test_split de sklearn (70%,15%, 15% respectivamente).
- **Construcción del modelo:** Se construye un modelo secuencial utilizando la API de Keras. El modelo consiste en un encoder (tres capas convolucionales seguidas de MaxPooling y Dropout) y un decoder (tres capas convolucionales seguidas de UpSampling y Dropout). La capa de salida tiene 3 filtros (correspondientes a los canales de color RGB) y usa la función de activación 'sigmoid'.
- **Compilación y entrenamiento:** Se compila el modelo con los hiperparámetros definidos anteriormente y se entrena utilizando el conjunto de entrenamiento, mientras se valida en el conjunto de validación. Se utilizan dos callbacks: uno para guardar el mejor modelo basado en la pérdida de validación (ModelCheckpoint) y otro para detener el entrenamiento temprano si la pérdida de validación no mejora después de un cierto número de épocas (EarlyStopping).

- **Evaluación:** Se evalúa el modelo en el conjunto de prueba y se imprimen la pérdida y el error absoluto medio (MAE).
- **Resumen del modelo y hiperparámetros**: Se imprime un resumen del modelo y los hiperparámetros utilizados.
- **Gráficos de las curvas de aprendizaje:** Se grafican las curvas de la pérdida y el MAE durante el entrenamiento para los conjuntos de entrenamiento y validación.

#### 11.2.5.2 Resultados entrenamientos del CAE

En esta sección se describen los resultados obtenidos durante el entrenamiento del CAE y los cambios. Como anteriormente, no se detallan aquí todos los entrenamientos.

##### 11.2.5.2.1 Modelo CAE 1

Se diseña el modelo con 3 capas convolucionales para el encoder y otras 3 para el dencoder, con una salida de 3 canales RGB con función de activación sigmoide.

**Hiperparámetros:**

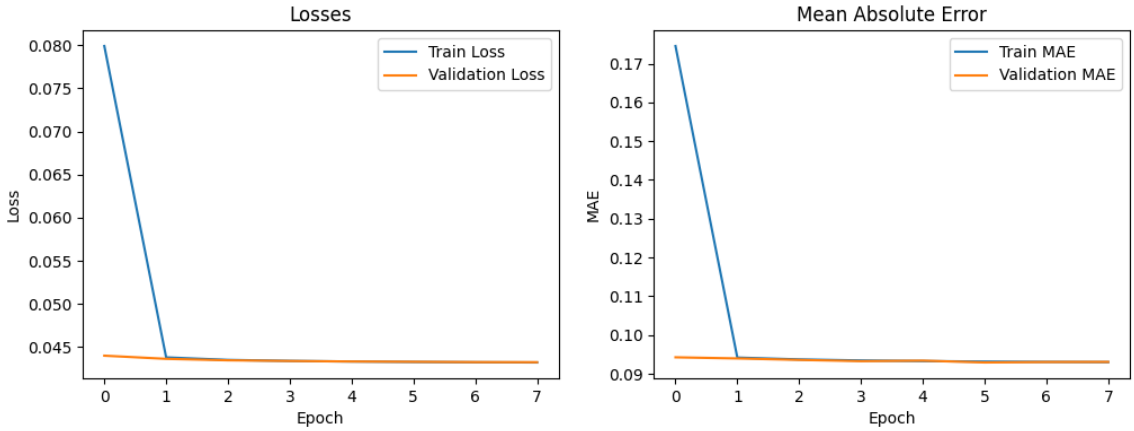
- Número de filtros en capas convolucionales: [32, 64, 128, 128, 64, 3]
- Tamaño de kernel en capas convolucionales: (3, 3)
- Función de activación en capas convolucionales: relu
- Optimizador: adam
- Función de pérdida: mse
- Métricas: ['mae']
- Tamaño del lote (batch size): 32
- Número de épocas: 12
- Paciencia para Early Stopping: 3

Tabla 8. Sumario Modelo CAE 1.

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d (Conv2D) | (None, 300, 300, 32) | 320 |
| max_pooling2d (MaxPooling2D) | (None, 150, 150, 32) | 0 |
| conv2d_1 (Conv2D) | (None, 150, 150, 64) | 18496 |
| max_pooling2d_1 (MaxPooling 2D) | (None, 75, 75, 64) | 0 |
| conv2d_2 (Conv2D) | (None, 75, 75, 128) | 73856 |
| up_sampling2d (UpSampling2D) | (None, 150, 150, 128) | 0 |
| conv2d_3 (Conv2D) | (None, 150, 150, 64) | 73792 |
| up_sampling2d_1 (UpSampling 2D) | (None, 300, 300, 64) | 0 |
| conv2d_4 (Conv2D) | (None, 300, 300, 3) | 1731 |
| | | |
| Total params: 168,195 | | |
| Trainable params: 168,195 | | |
| Non-trainable params: 0 | | |

Rendimiento en datos de prueba:

- Test Loss (MSE): 0.0431
- Test MAE: 0.0925




*Gráfica 6. CAE 1: Función perdida y MAE*

Con los resultados de este nuevo modelo observamos un factor en común con la red anterior, el modelo no generaliza y solo encuentra un valor medio donde se minimiza el error para todos los colores, en este caso un verde marrón de tono pastel.

También se observa que el modelo si parece encontrar cierto patrón en los ángulos del polígono, coloreando los vértices del mismo según el ángulo la abertura de su angulo:

- Amarillo para los ángulos de 90º
- Azul para los ángulos de 45º
- Verde oscuro para los ángulos de 270º

Tiene cierto sentido que los ángulos de 90º los clasifique de color amarillo, ya que el cuadrado es de ese color, y sea el polígono predominante en las formaciones con esos ángulos, por probabilidad la red habrá encontrado que ese color, en general, minimiza el error. El resto de ángulos no se acaba de entender el patrón de color que ha encontrado la red.

En cualquier caso, esto no se aproxima bien poco a la solución que se busca. Así que se decide probar con otros hiperparámetros.

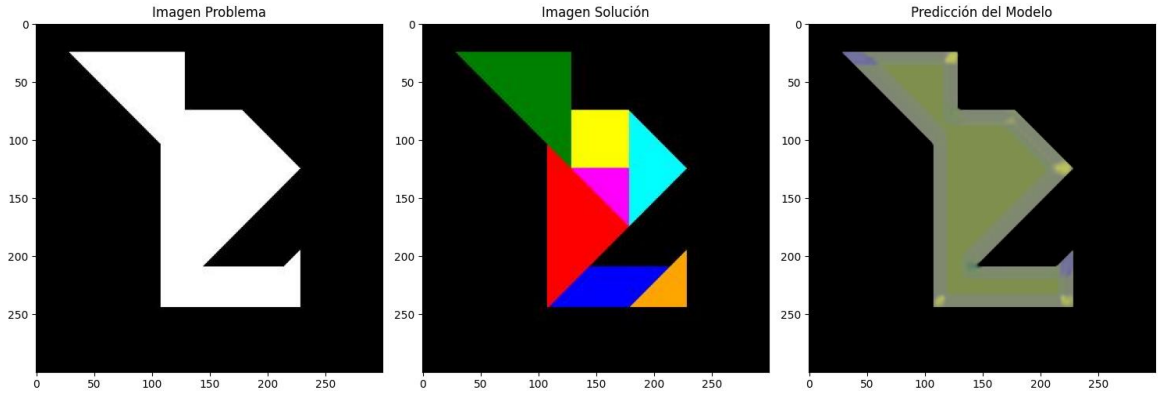



*Figura 54. CAE1: Visualización predicciones con valores de salida [problema 1500]*

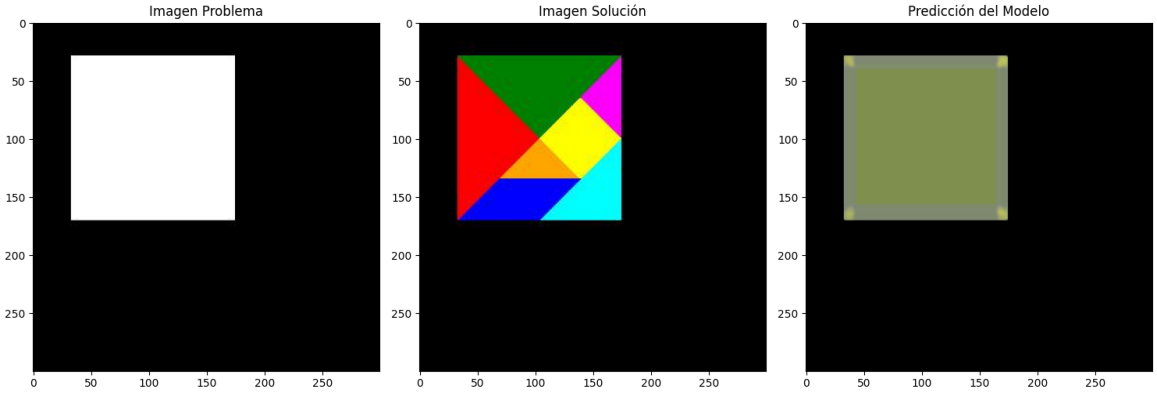



*Figura 55. CAE1: Visualización predicciones con valores de salida [problema 2500]*

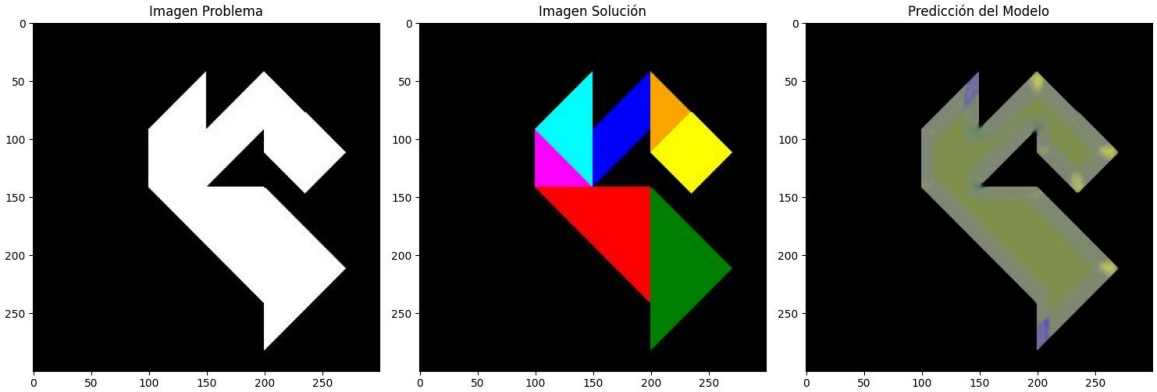



*Figura 56. CAE1: Visualización predicciones con valores de salida [problema 3500]*

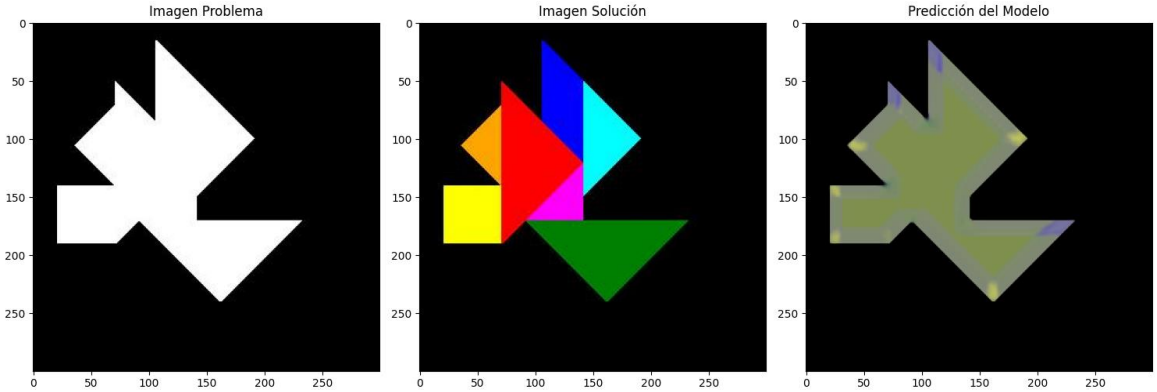
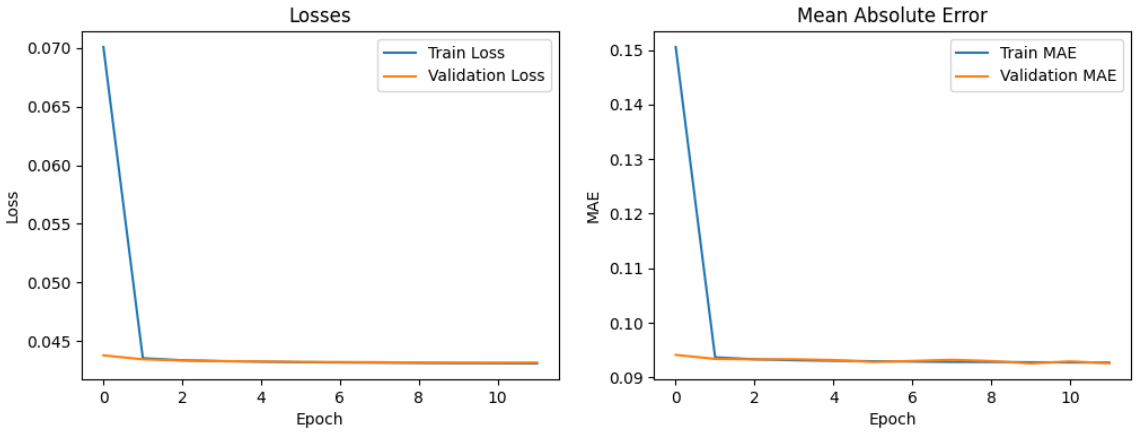



*Figura 57. CAE1: Visualización predicciones con valores de salida [problema 4000]*

#### 11.2.5.2.2 Modelo AE 2 y más.

Se realizan hasta 6 pruebas distintas, con diferentes hiperparámteros, en todos ellos no se logra ningún tipo de mejora, y todos acaban llegando a una función de perdida parecida la siguiente. No se añaden más imágenes porque no se encuentran apenas diferencias.




*Gráfica 7. CAE 2: Función perdida y MAE.*

#### 11.2.5.3 Conclusiones entrenamiento AE

Viendo el resultado de los dos modelos, y tras más trabajo de investigación, se llega a la conclusión que estas redes no son las más indicadas para generar soluciones. En un principio se pensó que con una generalización a la hora de clasificar las siluetas estas redes (indicadas para estas tareas) podrían aproximar una solución que se pudiera adaptar al proyecto.

Al tiempo dedicado a esta tarea de entrenamiento, que sobrepasa el doble de tiempo que se había previsto inicialmente se decide investigar y probar nuevas redes neuronales como el **Variational Autoencoder (VAE)** y la **Generative Adversary Networks (GAN)**.

### 11.2.6 Red Neuronal Variational Autoencoder (VAE)

Un Autoencoder Variacional (VAE, por sus siglas en inglés: Variational Autoencoder) es una clase de autoencoders que tiene una formulación probabilística y es especialmente interesante para generar nuevos datos que se asemejen a los datos de entrada, en lugar de simplemente aprender a replicar sus entradas. A diferencia de un autoencoder tradicional, que simplemente se esfuerza por reconstruir la entrada, un VAE modela la distribución de probabilidad de los datos de entrada. La introducción de un componente probabilístico y una pérdida basada en la divergencia KL son las principales diferencias entre un VAE y un CAE tradicional.

Descripción detallada del VAE:

- **Codificador (Encoder) Probabilístico:** A diferencia de un autoencoder tradicional, donde el encoder produce una única representación vectorial (o código) para una entrada dada, en un VAE, el encoder produce parámetros para una distribución de probabilidad, típicamente una distribución gaussiana. Estos parámetros son usualmente la media y la varianza (o su logaritmo) de esta distribución.

- **Muestreo:** Una vez que se obtienen los parámetros de la distribución (media y log-varianza), se realiza un muestreo de esta distribución para generar un punto en el espacio latente. Este punto se utilizará para reconstruir la entrada.

- **Decodificador (Decoder) Probabilístico:** Este punto muestreado se pasa al decodificador, que intenta reconstruir la entrada original. Al igual que en el codificador, la reconstrucción no es determinista, sino que se modela como una distribución de probabilidad.

- **Función de Pérdida:** La función de pérdida de un VAE tiene dos componentes:
  - Reconstrucción de la Pérdida: Mide cuán bien el decodificador ha reconstruido la entrada.
  - Pérdida de KL: Es la divergencia KL (Kullback-Leibler) que mide cuán diferente es la distribución codificada de la distribución objetivo (que es típicamente una distribución gaussiana estándar). Esta parte de la pérdida asegura que las representaciones latentes se distribuyan de una manera que facilite el muestreo y la generación.

- **Generación de Nuevos Datos:** Una vez que el VAE ha sido entrenado, puede generar nuevos datos muestreando un punto del espacio latente y pasándolo a través del decodificador.

#### 11.2.6.1 Código implementado en el VAE

El siguiente código implementa un Autoencoder Variacional (VAE por sus siglas en inglés) para aprender representaciones latentes de imágenes y generar nuevas imágenes que sean similares a las de entrenamiento. A continuación, se describe el funcionamiento de cada parte del código:

**Carga y Preprocesamiento de Datos**

- **Carga de imágenes:** Se cargan imágenes de problemas y soluciones de un directorio especificado. Las imágenes se cargan en modo RGB.
- **Normalización:** Las imágenes se normalizan dividiendo cada píxel por 255, de modo que todos los valores estén en el rango [0, 1].

**Creación del Modelo VAE**

- **Encoder:** Toma una imagen como entrada, reduce su dimensión utilizando capas convolucionales y se mapea a una representación latente a través de una capa densa. Se obtienen dos valores: la media y la varianza logarítmica.
- **Layer de Muestreo:** Toma la media y la varianza logarítmica del encoder. Utiliza estos valores para muestrear un punto en el espacio latente.
- **Decoder:** Toma un punto muestreado del espacio latente y lo decodifica para producir una imagen.
- **Modelo VAE:** Combina el encoder y el decoder. Calcula la pérdida de reconstrucción y la divergencia Kullback-Leibler (KL) y las suma para formar la pérdida total.

**Compilación y Entrenamiento**

- **Compilación del Modelo:** Se compila el modelo VAE con un optimizador Adam y una tasa de aprendizaje especificada.

- **Entrenamiento:** Se entrena el modelo con los datos de entrenamiento y se valida el modelo con un conjunto de datos de validación

**Resultados**

- **Visualización:** Se grafican las pérdidas de entrenamiento y validación en función de las épocas.
- **Guardar el Modelo:** Se guarda el modelo entrenado para uso futuro.
- **Evaluación:** Se evalúa el modelo en el conjunto de datos de validación y se imprime la pérdida de validación.
- **Sumario del Modelo y Hiperparámetros:** Se imprime un sumario del modelo y los hiperparámetros utilizados.

**Clases y metodos clave del código**

- **class VAE:** Define la arquitectura del VAE, construyendo tanto el encoder como el decoder.
- **class Sampling:** Define la capa de muestreo que muestrea un punto en el espacio latente dada la media y la varianza logarítmica.
- **vae.model.fit():** Entrena el modelo.
- **vae.model.save():** Guarda el modelo entrenado.
- **vae.model.evaluate():** Evalúa el modelo en el conjunto de validación.
- **vae.model.summary():** Imprime un sumario del modelo.

#### 11.2.6.2 Resultados entrenamientos del VAE

En esta sección se describen los resultados obtenidos durante el entrenamiento del VAE y los cambios realizados. Como anteriormente, no se detallan aquí todos los entrenamientos, que en este caso han sido 3 diferentes.

##### 11.2.6.2.1 Modelo VAE 1

Se implemente el primer modelo con el siguiente diseño.

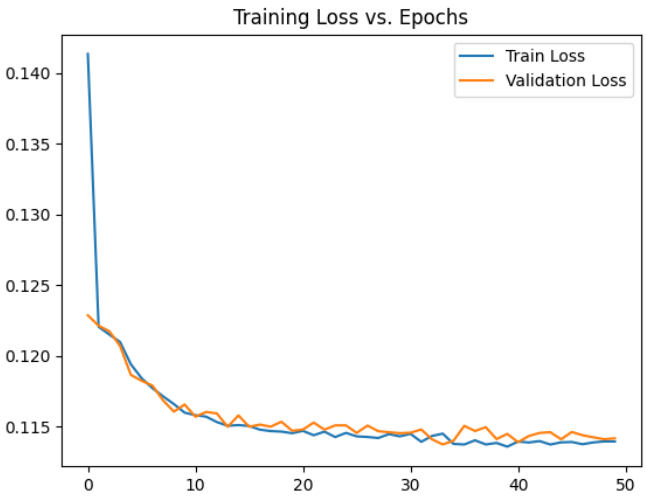
**Hiperparametros:**

- IMG_SHAPE: (300, 300, 3)
- BATCH_SIZE: 32
- EPOCHS: 50
- LATENT_DIM: 128
- LEARNING_RATE: 0.001

Tabla 9. Sumario Modelo VAE 1.

| Layer (type) | Output Shape | Param # | Connected to |
|---|---|---|---|
| input_3 (InputLayer) | [(None, 300, 300, 3)] | 0 | [] |
| model (Functional) | [(None, 128), (None, 128), (None, 128)] | 46132544 | ['input_3[0][0]'] |
| model_1 (Functional) | (None, 300, 300, 3) | 46512771 | ['model[0][2]'] |
| tf.__operators__.add (TFOpLambda) | (None, 128) | 0 | ['model[0][1]'] |
| tf.math.square (TFOpLambda) | (None, 128) | 0 | ['model[0][0]'] |
| tf.math.subtract (TFOpLambda) | (None, 128) | 0 | ['tf.__operators__.add[0][0]', 'tf.math.square[0][0]'] |
| tf.math.exp (TFOpLambda) | (None, 128) | 0 | ['model[0][1]'] |
| tf.convert_to_tensor (TFOpLambda) | (None, 300, 300, 3) | 0 | ['model_1[0][0]'] |
| tf.cast (TFOpLambda) | (None, 300, 300, 3) | 0 | ['input_3[0][0]'] |
| tf.math.subtract_1 (TFOpLambda) | (None, 128) | 0 | ['tf.math.subtract[0][0]', 'tf.math.exp[0][0]'] |
| tf.math.squared_difference (TFOpLambda) | (None, 300, 300, 3) | 0 | ['tf.convert_to_tensor[0][0]', 'tf.cast[0][0]'] |
| tf.math.reduce_mean_1 (TFOpLambda) | () | 0 | ['tf.math.subtract_1[0][0]'] |
| tf.math.reduce_mean (TFOpLambda) | (None, 300, 300) | 0 | ['tf.math.squared_difference[0][0]'] |
| tf.math.multiply (TFOpLambda) | () | 0 | ['tf.math.reduce_mean_1[0][0]'] |
| tf.__operators__.add_1 (TFOpLambda) | (None, 300, 300) | 0 | ['tf.math.reduce_mean[0][0]', 'tf.math.multiply[0][0]'] |
| tf.math.reduce_mean_2 (TFOpLambda) | () | 0 | ['tf.__operators__.add_1[0][0]'] |
| add_loss (AddLoss) | () | 0 | ['tf.math.reduce_mean_2[0][0]'] |

| |
|---|
| Total params: 92,645,315 |
| Trainable params: 92,645,315 |
| Non-trainable params: 0 |




*Gráfica 8. VAE 1: Función perdida*

Se comprueba como este modelo de red es mucho más difícil de programar que los anteriores. Los resultados obtenidos quedan muy lejos de lo esperado, con unas imágenes de salida sin color, ni ningún tipo de patrón respecto a sus problemas.

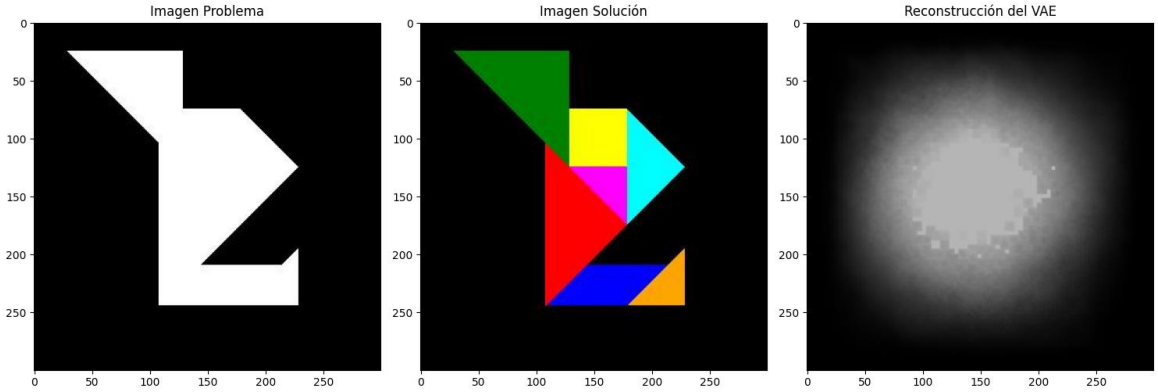



*Figura 58. VAE1: Visualización predicciones con valores de salida [problema 1000]*

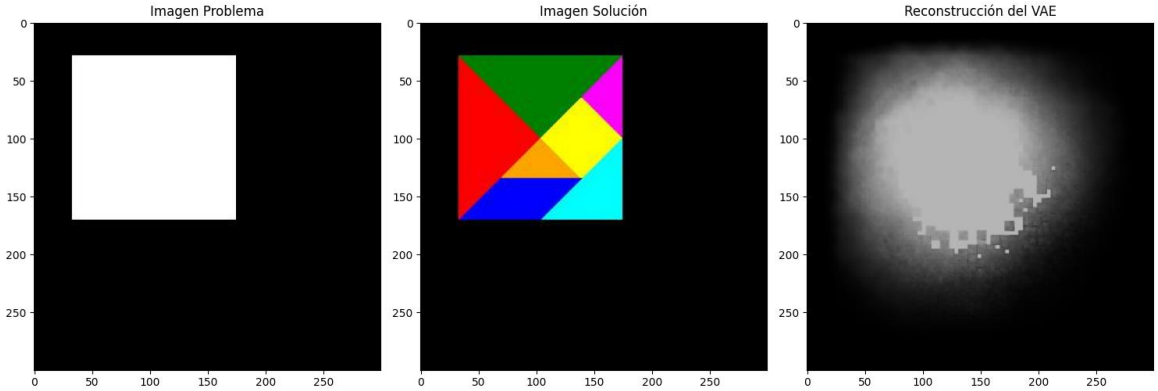



*Figura 59. VAE1: Visualización predicciones con valores de salida [problema 2000]*

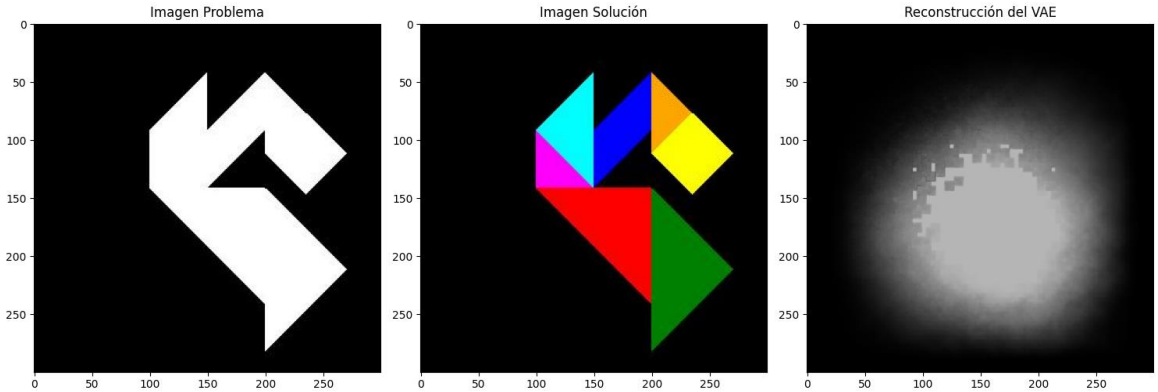



*Figura 60. VAE1: Visualización predicciones con valores de salida [problema 3000]*

### 11.2.6.3 Conclusiones entrenamiento VAE

Vistos los resultados obtenidos en los pocos entrenamientos realizados se decide que este tipo de red requiere de mucha más documentación para ser implementada correctamente y se pasa a probar como última opción un modelo GAN.

### 11.2.7 Red Neuronal Generativa Adversaria (GAN)

Una Red Neuronal Generativa Adversaria (GAN, por sus siglas en inglés) es un tipo de red neuronal artificial diseñada para generar datos nuevos que son similares a un conjunto de datos de entrada dado. Estas redes se componen de dos partes principales: un generador y un discriminador, que están entrenados simultáneamente a través de un proceso de confrontación, de ahí el término "adversario". Se compone de:

**Generador**

- Objetivo: Crear datos nuevos que sean similares a los datos de entrada.
- Funcionamiento: Toma un ruido aleatorio como entrada y genera datos.
- Entrenamiento: Se entrena para maximizar la probabilidad de que el discriminador cometa un error.

**Discriminador**

- Objetivo: Diferenciar entre los datos reales (entrada) y los datos falsos (generados).
- Funcionamiento: Toma datos (reales o generados) como entrada y decide si son reales o falsos.
- Entrenamiento: Se entrena para minimizar la probabilidad de error al clasificar los datos.

**Proceso de Entrenamiento**

Fase de Discriminador:

- El generador crea datos a partir de ruido aleatorio.
- Estos datos generados, junto con los datos reales, se alimentan al discriminador.
- El discriminador intenta clasificar los datos como reales o falsos.

Fase de Generador:

- El generador crea datos a partir de ruido aleatorio.
- Estos datos generados se alimentan al discriminador.
- El generador se ajusta para maximizar la probabilidad de que el discriminador clasifique incorrectamente estos datos generados como reales.

**Iteración**

Este proceso se repite muchas veces, con el objetivo de que el generador mejore su capacidad para generar datos falsos, y el discriminador mejore su capacidad para distinguir los datos falsos de los reales.

**Resultado**

Al final del entrenamiento, el generador es capaz de producir datos que son indistinguibles de los datos reales para el discriminador. Las GANs son útiles en una variedad de aplicaciones, incluida la generación de imágenes, videos, y texto, entre otros.

**Desafío**

Las GANs pueden ser difíciles de entrenar, ya que se requiere un equilibrio entre el generador y el discriminador. Un componente no debe superar al otro durante el entrenamiento para evitar que el proceso de aprendizaje se estanque.

#### 11.2.7.1 Código implementado en el GAN

El siguiente código desarrolla y entrena una GAN para convertir las imágenes en blanco y negro de las siluetas a sus soluciones a color. Durante el entrenamiento, el generador intenta producir imágenes a color lo más parecidas a la solución posible, mientras que el discriminador intenta distinguir entre imágenes solución reales y generadas. El entrenamiento se detiene después de un número predefinido de épocas, pero se podría implementar una lógica de parada temprana (Early Stopping) usando el hiperparámetro de "paciencia" para detener el entrenamiento si el modelo no mejora durante un número determinado de épocas consecutivas.

- **Importaciones:** Librerías necesarias para la manipulación de datos, visualización y construcción del modelo.
- **Hiperparámetros:** Definimos los parámetros como el número de filtros para las capas convolucionales, tamaño de kernel, función de activación, optimizador, función de pérdida, entre otros.
- **Carga de Imágenes:** Carga las imágenes en blanco y negro (problema) y a color (solución) desde el directorio "dataset_images". Las imágenes se cargan en listas y posteriormente se convierten a arreglos numpy y se normalizan dividiendo por 255.
- **División de Datos:** Las imágenes se dividen en conjuntos de entrenamiento, validación y prueba usando la función train_test_split de scikit-learn.
- **Construcción del Generador:** Se construye una arquitectura secuencial con capas de codificación (encoder) que reducen las dimensiones de las imágenes y capas de decodificación (decoder) que las aumentan. Esta arquitectura genera una imagen a color a partir de una imagen en blanco y negro.
- **Construcción del Discriminador:** Se construye una arquitectura que toma imágenes a color y decide si son reales o generadas por el generador.
- **Compilación:** Se compila el discriminador con una función de pérdida de entropía cruzada binaria y el optimizador elegido. El generador es integrado dentro de la GAN combinada, donde se utiliza para generar imágenes que luego son evaluadas por el discriminador (cuyo entrenamiento se desactiva para este propósito).
- **Entrenamiento de la GAN:** Durante cada época, se entrena el discriminador con imágenes reales y generadas etiquetadas como 1s y 0s respectivamente. Se entrena el generador mediante la GAN combinada, tratando de engañar al discriminador para que crea que las imágenes generadas son reales. Se registran las pérdidas del generador y del discriminador y la precisión del discriminador.

- **Evaluación:** Se evalúa el desempeño del generador usando MSE (Error Cuadrático Medio) entre las imágenes generadas y las imágenes reales del conjunto de prueba.
- **Guardado de Modelos:** Se guardan los modelos del generador y del discriminador para su uso posterior.
- **Impresión y Visualización:** Se imprimen los resúmenes de los modelos y los hiperparámetros. Se visualizan las curvas de aprendizaje del generador y del discriminador a lo largo del entrenamiento, mostrando la pérdida y la precisión del discriminador.

#### 11.2.7.2 Resultados entrenamiento GAN

A continuación, se describen los resultados de los entrenamientos, solo se añaden a esta memoria los datos de uno de ellos ya que para los tres realizados apenas se encuentran diferencias.

##### 11.2.7.2.1 Modelo GAN 1

**Hiperparámetros:**

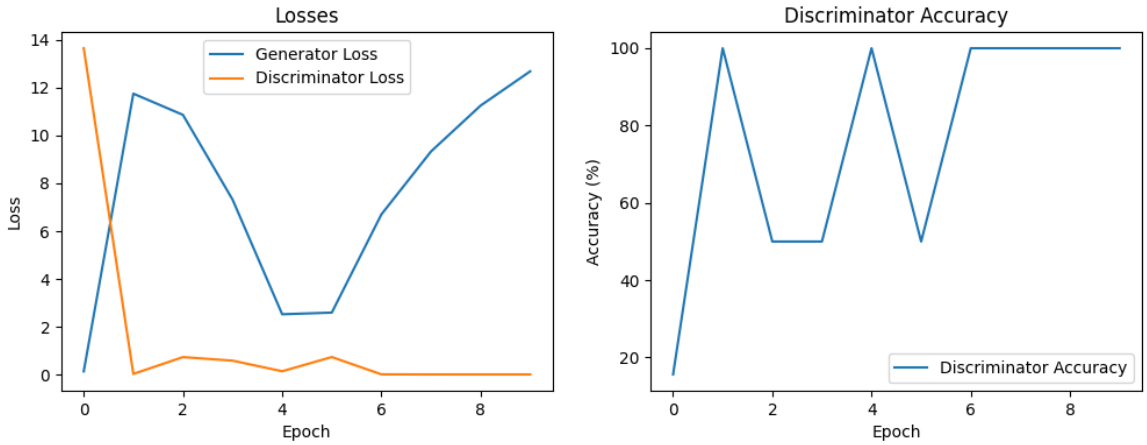
- Número de filtros en capas convolucionales: [32, 64, 128, 128, 64, 32, 3]
- Tamaño de kernel en capas convolucionales: (3, 3)
- Función de activación en capas convolucionales: relu
- Optimizador: adam
- Función de pérdida: mse
- Métricas: ['mae']
- Tamaño del lote (batch size): 16
- Número de épocas: 10

Tabla 10. Sumario Modelo GAN 1.

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d_2 (Conv2D) | (None, 300, 300, 32) | 320 |
| max_pooling2d (MaxPooling2D) | (None, 150, 150, 32) | 0 |
| dropout_2 (Dropout) | (None, 150, 150, 32) | 0 |
| conv2d_3 (Conv2D) | (None, 150, 150, 64) | 18496 |
| max_pooling2d_1 (MaxPooling 2D) | (None, 75, 75, 64) | 0 |
| dropout_3 (Dropout) | (None, 75, 75, 64) | 0 |
| conv2d_4 (Conv2D) | (None, 75, 75, 128) | 73856 |
| up_sampling2d (UpSampling2D) | (None, 150, 150, 128) | 0 |
| dropout_4 (Dropout) | (None, 150, 150, 128) | 0 |
| conv2d_5 (Conv2D) | (None, 150, 150, 64) | 73792 |
| up_sampling2d_1 (UpSampling 2D) | (None, 300, 300, 64) | 0 |
| dropout_5 (Dropout) | (None, 300, 300, 64) | 0 |
| conv2d_6 (Conv2D) | (None, 300, 300, 32) | 18464 |
| conv2d_7 (Conv2D) | (None, 300, 300, 3) | 867 |

| |
|---|
| Total params: 185,795 |
| Trainable params: 185,795 |
| Non-trainable params: 0 |

**Perdida (MSE) del Generador:** 0.10521557




*Gráfica 9. Perdida del Generador y Discriminador GAN 1.*

Una vez más, con este modelo tampoco se logra que la red generaliza y acaba encontrando solo valores medios para todos los problemas. En este caso no se acaba de entender porque la red acaba aplicando un fondo azul oscuro en vez de negro, negro a las siluetas y una especie de reborde redondeado a todo el perímetro de la silueta.

Muy probablemente se deba a que la programación de esta red no está bien diseñada.

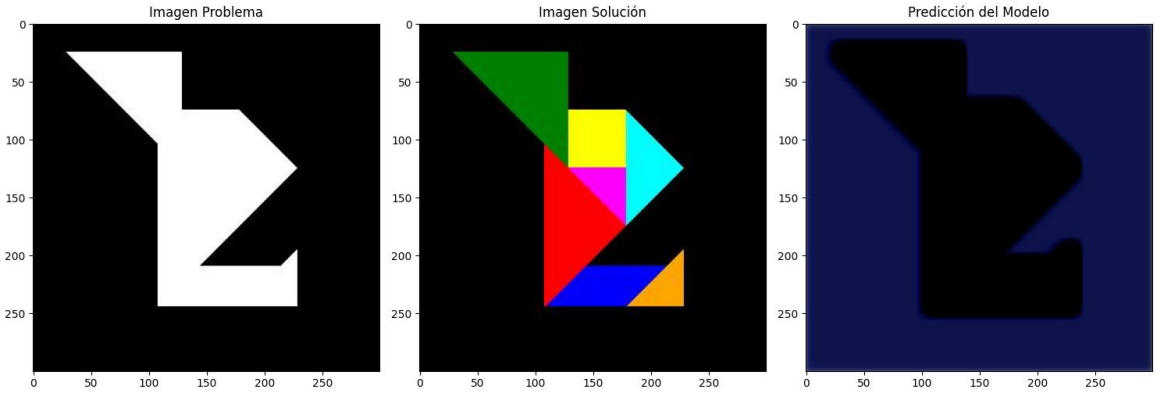



*Figura 61. GAN1: Visualización predicciones con valores de salida [problema 1000]*

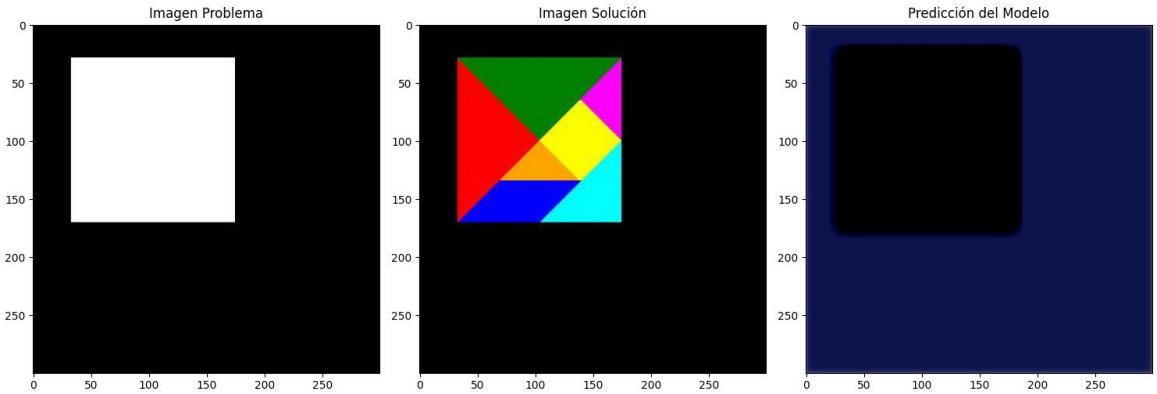



*Figura 62. GAN1: Visualización predicciones con valores de salida [problema 1000]*

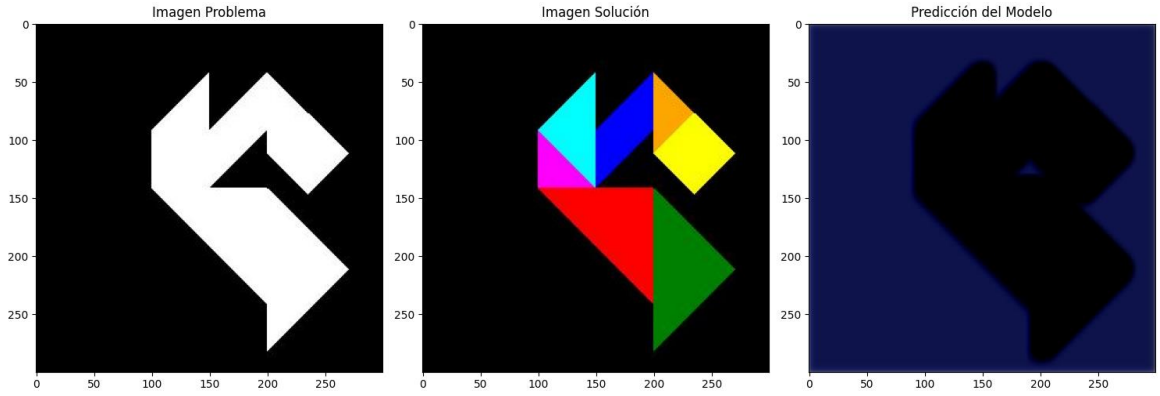



*Figura 63. GAN1: Visualización predicciones con valores de salida [problema 1000]*

#### 11.2.7.3 Conclusiones entrenamiento GAN

Se encuentra una dificultad a la hora de diseñar y programar este tipo de redes muy superior a las capacidades actuales del estudiante, y del tiempo disponible para poder dedicarle el estudio y documentación necesaria para llevar a cabo tal desafío.

Sin embargo, se concluye que se han dado los pasos correctos para la su resolución pero que no se ha sabido encarar y programar correctamente.

### 11.2.8 Conclusiones finales

Después de dedicarle más de la mitad del tiempo disponible durante el proyecto al entrenamiento de las redes, y vistos los resultados obtenidos y la dificultad encontrada a la hora de lograr una solución satisfactoria se decide detener esta parte del trabajo y no completarla.

Lamentablemente, esta memoria no puede reflejar todas las pequeñas soluciones, ideas y variaciones durante el transcurso del proyecto.

Se decide continuar con el resto del proyecto utilizando la solución generada por el aplicativo para, como mínimo, completar con éxito las otras tareas del proyecto.

Respecto al desarrollo de redes neuronales, se asume que se ha trabajado en una dirección correcta pero que quizás, tal desafío, requiere de más tiempo y/o formación para llevarla a cabo.

Se espera y se anima a que este trabajo pueda servir como base y apoyo para posteriores proyectos relacionados, o incluso retomar el desarrollo del código desde donde se ha detenido aquí.

## 11.3 Programa de visión por ordenador con OpenCV

Con el objetivo de encontrar la posición inicial aleatoria de las piezas en el tablero (que deberán estar todas presentes en el recuadro) se diseña un script en Python que, utilizando la webcam conectada al portátil vía USB, realice una captura del tablero para luego ser analizada mediante la librería OpenCV.

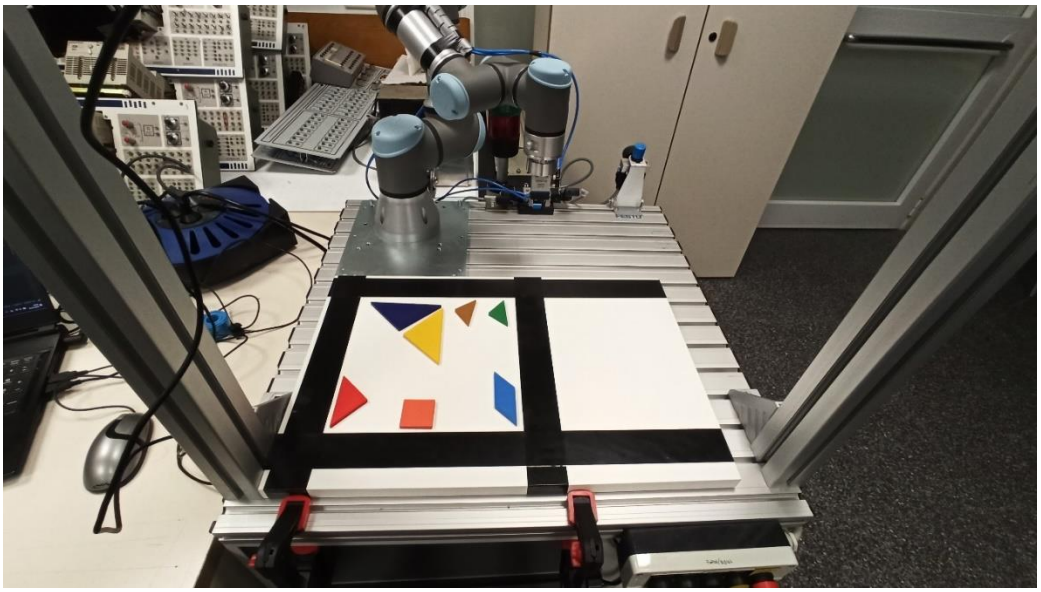


*Figura 64. Tablero donde capturar la posición inicial aleatoria de las piezas*

Con este objetivo en mente, se decide colocar la webcam justo encima del centro del tablero donde deben ser manipuladas las piezas para minimizar posibles distorsiones. Gracias al soporte vertical ajustable de la estación robótica y utilizando un pequeño sargento se fija la webcam al soporte para, más tarde, utilizar las articulaciones de la misma webcam para acabar de ajustar y alinear correctamente la cámara.

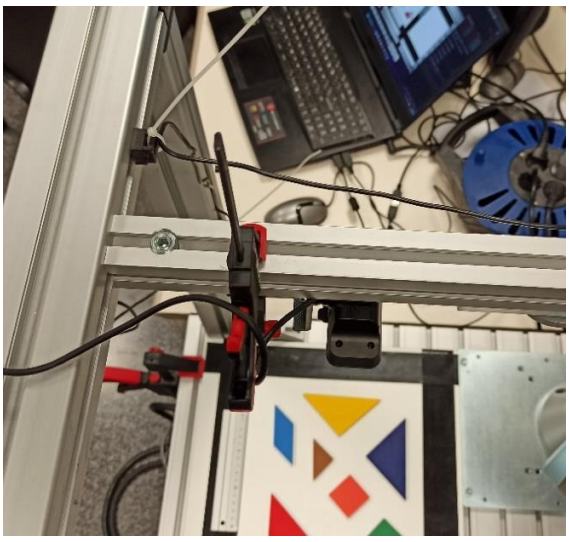


*Figura 65. Detalle de la fijación de la cámara.*

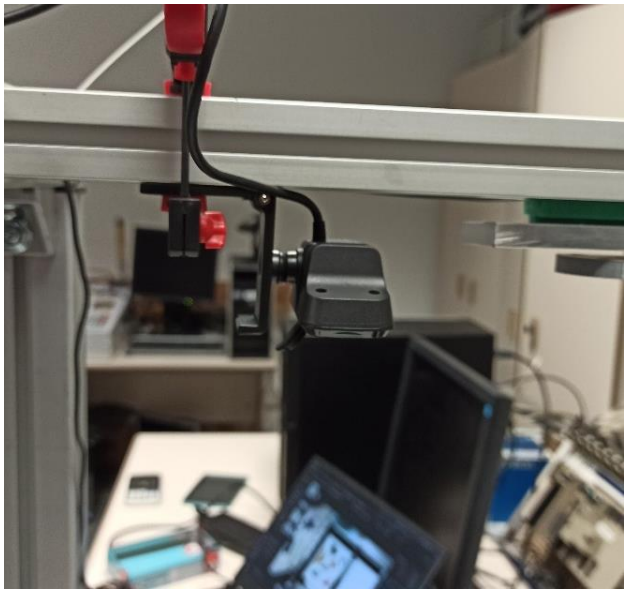


*Figura 66. Detalle lateral de la fijación de la cámara.*

Una vez la cámara está fijada y utilizando a la aplicación de **"Cámara"** de Windows, acabaremos de alinearla correctamente ayudándonos de la función "cruceta en cámara" (función que crea unas guías verticales y horizontales en el centro de la cámara) para servirnos de guía junto a una hoja de calibración diseñada para tal fin.

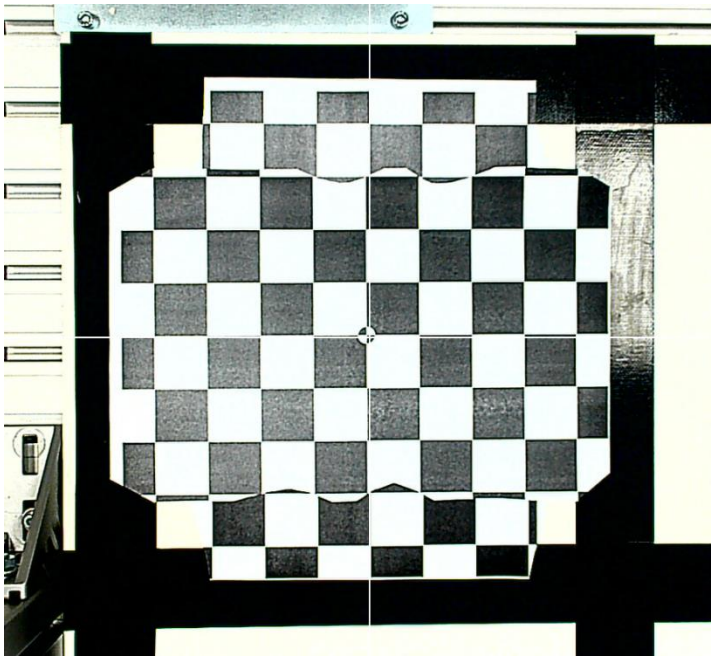


*Figura 67. Alineación de cámara mediante cruceta.*

Desde la misma aplicación ajustamos también los valores de brillo, contraste, nitidez, y saturación para obtener una imagen de las piezas lo mas limpia de ruido y sombras posible.

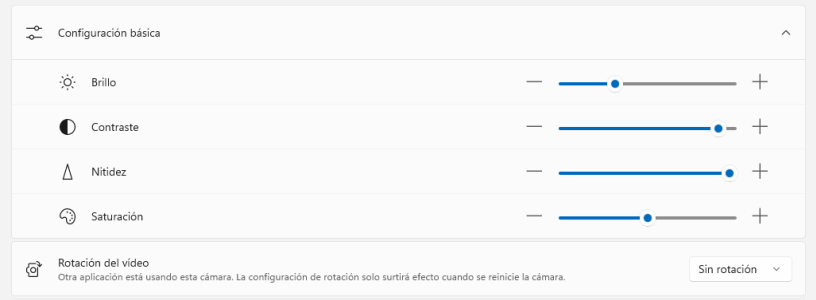



*Figura 68. Ajustes de cámara.*

### 11.3.1 Calibración de la cámara

La calibración de la cámara en sistemas de visión por ordenador es un paso crítico. Esta se divide en calibración óptica y geométrica. La calibración óptica se encarga de corregir distorsiones en la imagen causadas por la lente de la cámara. Estas distorsiones pueden ser radiales, donde la imagen puede parecer como si estuviera dentro de un barril o un cojín, y tangenciales, donde los objetos en la imagen aparecen inclinados o sesgados debido a que la lente no está alineada perfectamente con el plano de la imagen. Estas distorsiones son descritas por cinco coeficientes únicos para cada cámara, que, una vez calculados, no necesitan ser recalculados para esa cámara en particular. Estos coeficientes se obtienen capturando múltiples imágenes de un patrón conocido, como un tablero de ajedrez, desde diferentes perspectivas. Por otro lado, la calibración geométrica nos da la relación entre la posición de un píxel en la imagen y su posición real, lo que es crucial para obtener mediciones precisas.

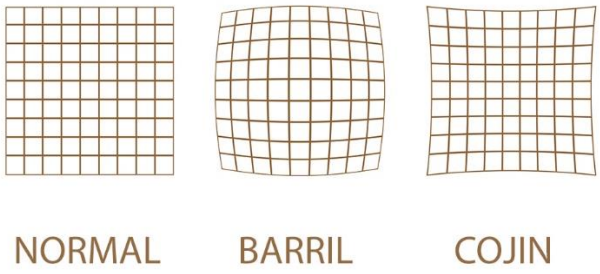



*Figura 69. Distorsiones radiales.*

En nuestro caso, vamos a verificar si desde la posición fija de la cámara se pueden observar distorsiones significativas que puedan afectar a los cálculos de posición. Para ello utilizamos dos hojas cruzadas DIN A4 ajedreadas diseñadas para tal fin. Las casillas tienen un tamaño de 31,25 mm y se alinean perfectamente con nuestro recuadro de 250 mm x 250 mm en un set de 8x8 casillas.

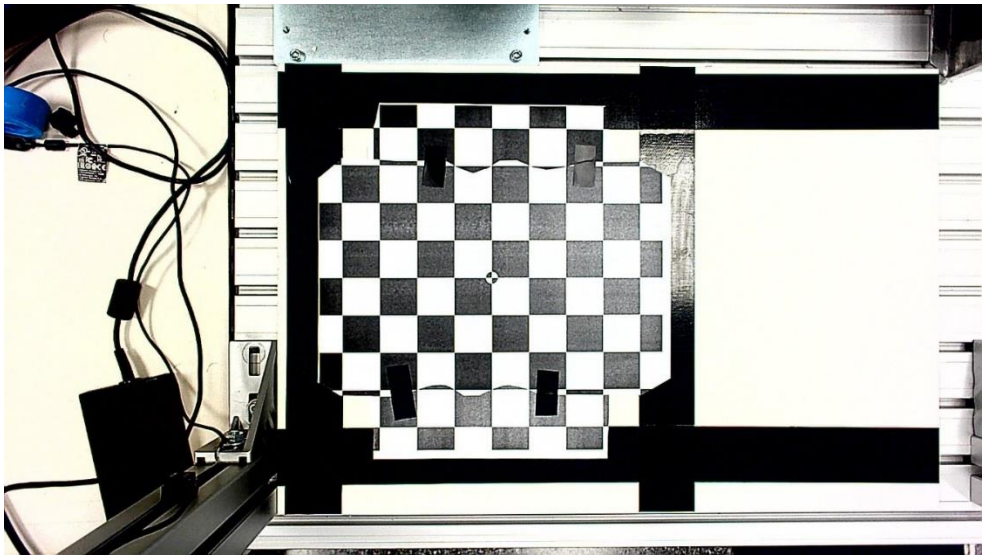


*Figura 70. Calibración cámara.*

Utilizando el software **Adobe Ilustrator** (programa para la edición grafica de vectores) se realizan mediante la herramienta de reglas las mediciones de tamaño de cada casilla para ver si existen distorsiones.

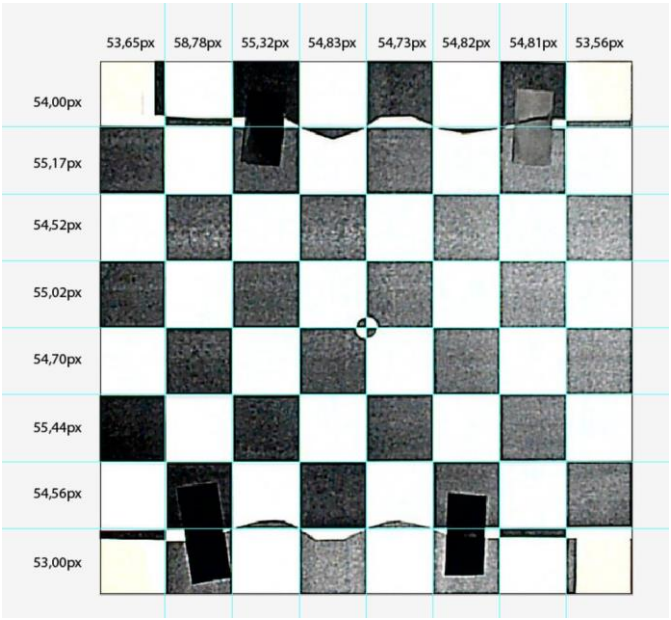



*Figura 71. Comprobación de distorsiones.*

No se observan distorsiones significativas en la imagen, solo pequeñas variaciones inconsistentes en algunas casillas. Tampoco parece crear la lente ningun tipo de distorsión radial o tangencial, las líneas se observan perfectamente verticales y horizontales.

Con estos datos **se decide no realizar ningún tipo de corrección** ya que no parece ser necesario.

### 11.3.2 Desarrollo de las capturas y detecciones

Antes de empezar a comentar el código final implementado se describe en esta sección el proceso de desarrollo de las primeras hasta las últimas capturas, y los pasos dados para llegar al programa final.

Despues de preparar el código en el ordenador de escritorio se transfiere al portatil para hacer las primeras pruebas. Inmediatamente se hace evidente la necesidad de utilizar un tablero blanco para hacer de fondo, porque si bien es cierto que algunas piezas las detecta correctamente, los railes generan sombras que provocan problemas de detección.

*Figura 72. Primeras pruebas visión por ordenador, captura original.*

En estas pruebas aún no está implementado la detección de color de los 7 colores (solo su código BGR), ni el cálculo del ángulo, ni se diferencia entre cuadrado y paralelogramo.

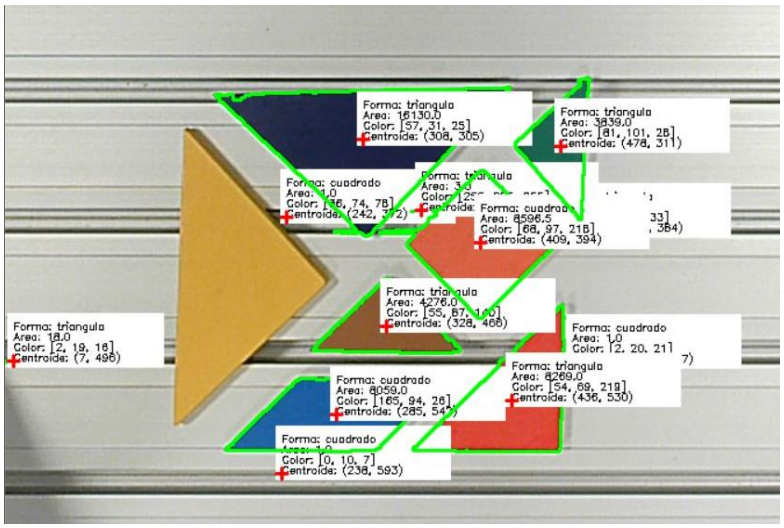



*Figura 73. Primeras pruebas visión por ordenador, detecciones.*

Se refina un poco el código, añadiendo la detección de los 7 colores, el ángulo de orientación y la diferenciación entre cuadrado y paralelogramo. Se ajustan también los parámetros de la cámara, y con la ayuda de una balda blanca (balda que por dimensiones no encajaba horizontalmente en la estación) se realizan las detecciones de forma satisfactoria.

*Figura 74. Segundas pruebas visión por ordenador, captura original.*

Por un problema en el orden en las líneas de código, el color se detecta después de añadir la cruceta del centroide, de ahí que en estas pruebas todas las piezas se detecten como “naranja”. El resto parece funcionar correctamente.

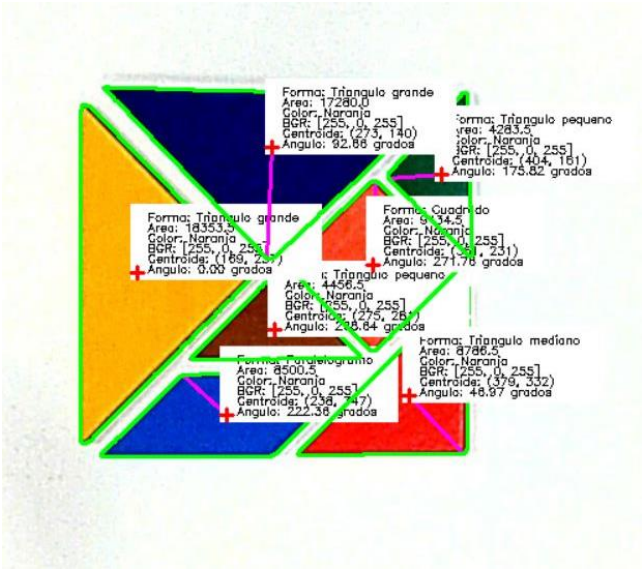



*Figura 75. Segundas pruebas visión por ordenador, detecciones.*

Para las terceras pruebas se prepara una balda que por dimensiones pueda colocarse horizontalmente en la estación, y se marca un recuadro con cintas negras como la zona de configuración de piezas. Se elige esta zona como área de manipulación porque es donde el robot tiene más libertad de movimientos. La zona tiene un tamaño de 250 x 250 milímetros.

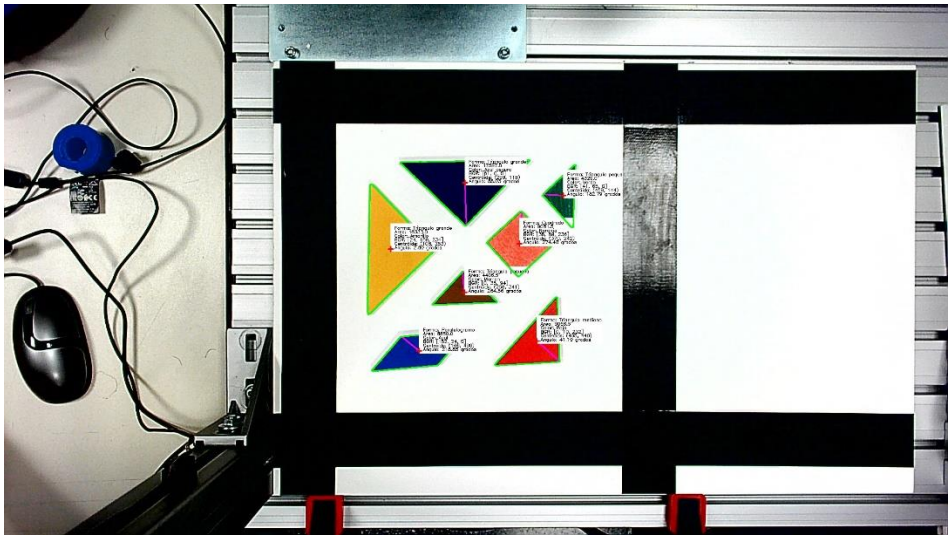


*Figura 76. Terceras pruebas visión por ordenador, captura original con detecciones.*

En estas pruebas todo parece correcto y todo parece estar listo para empezar la siguiente fase.

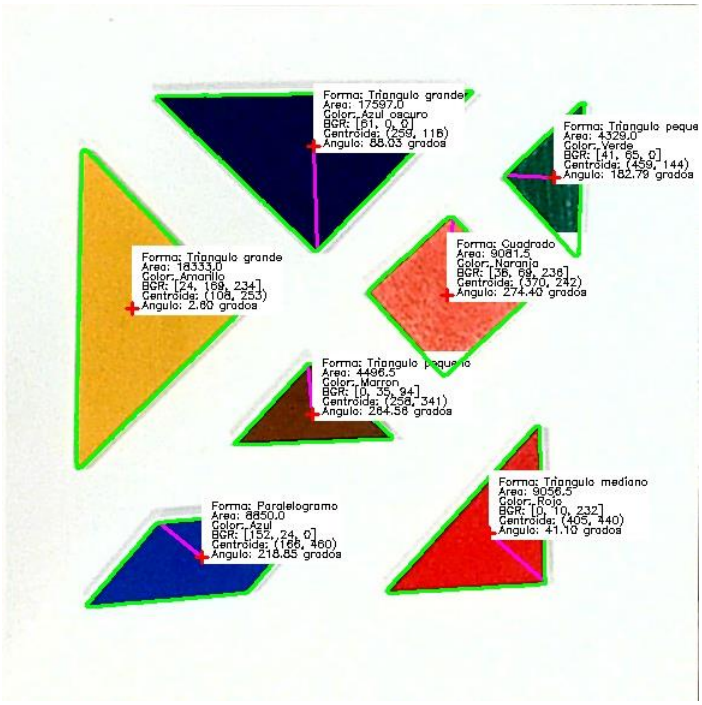



*Figura 77. Terceras pruebas visión por ordenador, detecciones*

En el cuarto día de pruebas se utilizan las piezas ya con su agarre para las pinzas del robot fijado. En las primeras capturas se detecta que estos bloques blancos en los centros de las piezas generan problemas en nuestro código anterior, aunque solo en algunos casos.

*Figura 78. Cuartas pruebas visión por ordenador, captura original con detecciones.*

Se puede comprobar como en algunas piezas la adición del agarre en blanco provoca un error de lectura importante, o directamente no genera ninguna detección.

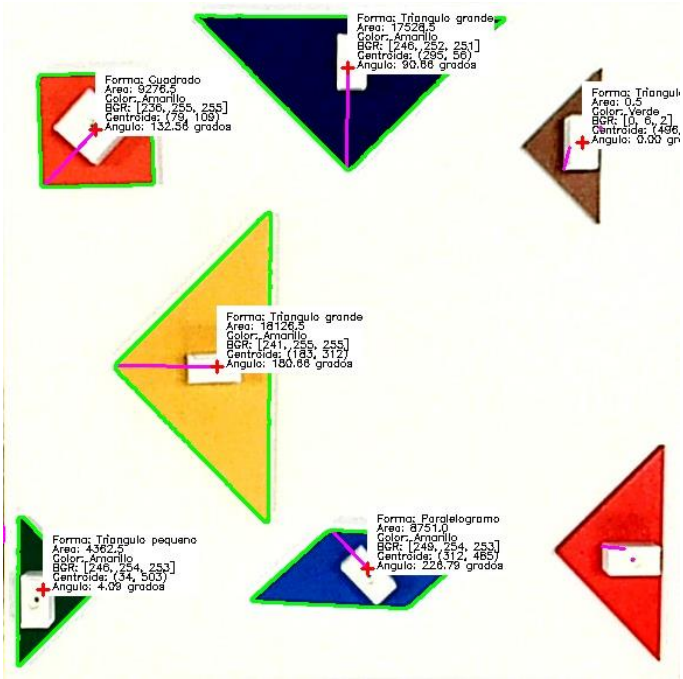



*Figura 79. Cuartas pruebas visión por ordenador, detecciones.*

Para solucionarlo se decide girar los agarres de tal manera que su altura no pueda alterar la percepción de la forma de la pieza (sobre todo en los triángulos pequeños, donde la perspectiva de la cámara en las esquinas es más pronunciada). También se pintan los agarres de los colores del mismo color de la pieza antes de rectificar el código.

*Figura 80. Quintas pruebas visión por ordenador, captura original con detecciones.*

Con las piezas pintadas todo vuelve a funcionar correctamente, en estas últimas pruebas se ha quitado de las anotaciones la información que ya no parecía relevante mostrar.

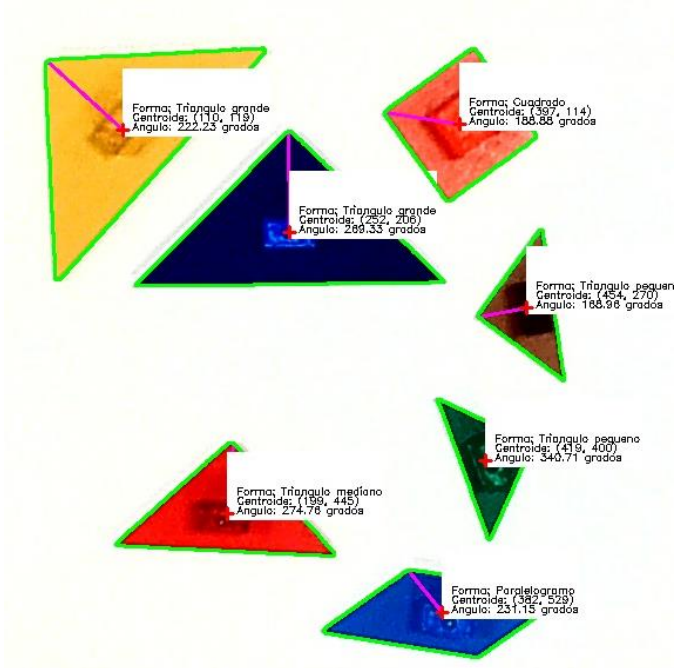



*Figura 81. Quintas pruebas visión por ordenador, detecciones.*

### 11.3.3 Conversión pixeles de la cámara a milímetros del robot

Una vez tenemos los datos de las posiciones y orientaciones de cada pieza debemos convertir esa información útil para ser utilizada por el robot. En nuestro caso debemos encontrar la relación entre los pixeles detectados por la cámara y las coordenadas en milímetros en el espacio físico que utiliza el robot.

Para ello, primero obtendremos en los valores que utiliza el robot en su plano de referencia Base del espacio de trabajo que captura la cámara. Después, con los datos de los pixeles de captura, extraeremos una función de conversión.

#### 11.3.3.1 Obtención de los puntos del tablero según los ejes del robot

Una vez terminado el proceso de captura de la piezas se obtienen los puntos físcos que delimitan nuestro recuadro del tablero respecto al eje de coordenadas **Base** de nuestro robot.

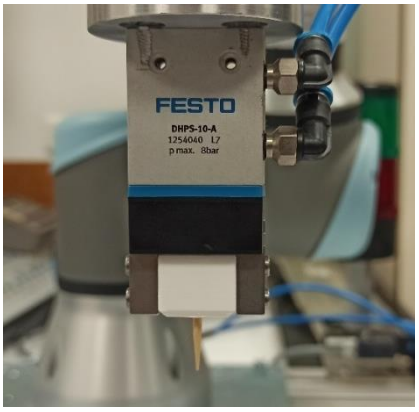



*Figura 82. Detalle pieza para medir puntos del tablero.*


*Figura 83. Toma del punto central.*

Mediante el uso de una de las piezas sobrantes de agarre para los tans (junto a un palillo insertado lo mas centrado posible) y moviendo el robot a cada punto, anotaremos la posición x e y del robot en cada punto.

De esta manera obtenemos los siguientes puntos **aproximados y en milímetros** (x, y):

- P1 = [-48.5, -233]
- P2 = [201,5, -233]
- P3 = [-48,5, -483]
- P4 = [201,5, -483]

Nuestra área hace 250 x 250 milímetros, que coincide con el tablero diseñado. Con el punto central situado en [76,5, -358].

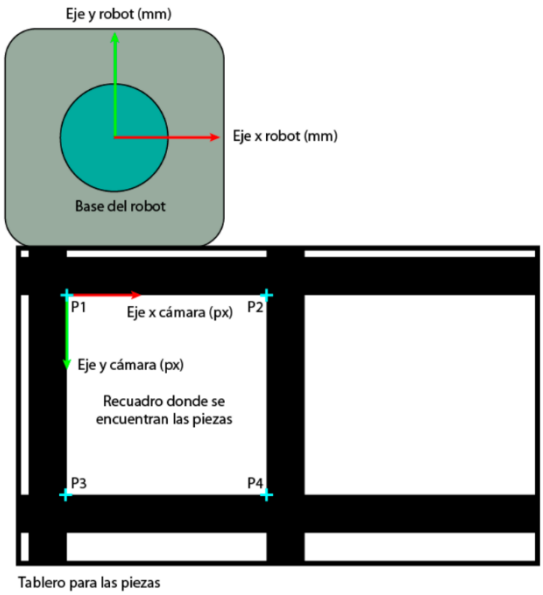



*Figura 84. Diagrama del tablero en la estación.*

#### 11.3.3.2 Función de conversión

Con los datos obtenidos anteriormente ya podemos extraer la relación. De momento, solo necesitamos convertir los valores del eje x e y de pixeles a milímetros, y los ángulos de grados a radianes.

En el eje x:

- El rango en pixeles es [0, 580]
- El rango en milímetros es [-48.5, 201.5]

Utilizando la fórmula de la pendiente de una línea:

$$y = mx + b$$

Donde:

$$m = \frac{201{,}5 - (-48{,}5)}{580 - 0}$$

$$m = \frac{25}{58}$$

Y:

$$b = -48{,}5 - \left(\frac{25}{58}\right) * (0)$$

$$b = -48{,}5$$

Por lo tanto, la ecuación completa es:

$$y = \left(\frac{25}{58}\right) x - 48{,}5$$

En el eje y:

- El rango en pixeles es [0, 580]
- El rango en milímetros es [-233, -483]

Siguiendo el mismo procedimiento, pero invirtiendo la variable ya que los ejes y de la cámara y el robot son contrarios, la ecuación completa es:

$$y = -\left(\frac{25}{58}\right) x - 233$$

Utilizaremos estas conversiones, junto a la conversión de grados a radianes, a la hora de guardar nuestros datos obtenidos en la matriz **captured_pos** (explicado en detalle en el próximo apartado).

### 11.3.4 Código implementado para la detección de piezas.

El código se inicia al presionar el botón "**Enviar al ROBOT**" en la pantalla del aplicativo cuando existe una solución creada. Este botón llama al método estático ***capture_and_send()*** donde, al inicio de esta parte del código, utiliza OpenCV para:

1. Abrir la cámara con índice 1 (ya que el índice 0 es la cámara del propio portátil).
2. Ajustar su resolución a 1920x1080 (la resolución de la cámara).
3. Capturar una imagen (frame) en ese instante.

Si la captura es exitosa, se llama a la función ***detect_shapes(frame)*** del archivo **computer_vision.py** para procesar la imagen y detectar las posiciones, más adelante se detalla este código. Al terminar esta función se libera la cámara y se cierran todas las ventanas de visualización.

#### 11.3.4.1 Computer_vision.py

Este script se utiliza para almacenar las funciones que detectan las posiciones y orientaciones de las piezas en el tablero utilizando las librerías **socket** y **time** que forman parte de Python.

Al inicio del programa se declara un diccionario de colores que define los nombres de los colores para los valores BGR (Blue, Green, Red) detectados en las primeras capturas. Este diccionario se utiliza para establecer los colores más cercanos al color detectado.

Funciones:

- **find_nearest_color:** esta función recibe un valor BGR y devuelve el color más cercano del diccionario de colores basado en la distancia euclidiana en el espacio de color. De esta manera, si no se detecta el mismo color, simplemente por proximidad sabremos qué color es.

- **detect_shapes:** esta función es el núcleo del programa. Esta función recibe una imagen y realiza varias operaciones para detectar y anotar formas en la imagen:

  - **Recorte:** La imagen se recorta a un área específica con el objetivo de solo detectar las piezas del tablero. El área se delimita por el recuadro marcado en el tablero, de manera que no se vea ningún pixel de las líneas negras, intentando ajustarlo al máximo. Bien ajustada la cámara obtenemos siempre un área de recorte de 580x580 pixeles, esta área será utilizada para convertir nuestros pixeles a milímetros.
  - **Conversión y desenfoque:** La imagen recortada se convierte a espacio de color HSV y luego se desenfoca usando el filtro Gaussiano.
  - **Umbralización:** Se umbraliza la saturación de la imagen para detectar colores vivos.
  - **Detección de contornos:** se utiliza la función ***findContours*** de la biblioteca OpenCV, que es una función ampliamente utilizada en visión por ordenador para detectar y describir formas en una imagen. La función retorna contornos encontrados en una imagen binaria.

    Utilizando ***RETR_EXTERNAL*** retorna solo los contornos exteriores, es decir, no busca contornos dentro de otros contornos. Por ejemplo, en una imagen donde tendríamos un círculo dentro de otro círculo, solo detectaría el contorno del círculo más grande. Con ***CHAIN_APPROX_SIMPLE*** comprimimos segmentos horizontales, verticales y diagonales y dejamos solo sus puntos finales. Por ejemplo, si tenemos un contorno con forma de rectángulo, en lugar

de almacenar todos los puntos que forman los lados del rectángulo, solo almacenará los cuatro vértices (esto nos ahorra memoria).

El resultado de esta función es una lista de contornos encontrados en la imagen que se guardan en la lista variable **contours.**

- **Detección de formas:** Por cada contorno detectado y almacenado en la lista, y de forma iterativa, se calculan:
  - El perímetro del contorno.
  - El área del contorno.
  - Su centroide utilizando la función **moments** de OpenCV
  - El color llamando a la función **find_nearest_color.**
  - Se encuentra el vértice más cercano al centroide
  - Se calcula el ángulo de orientación del contorno desde el centroide hasta el vértice más cercano.

  Obtenidos estos datos, determinamos que tipo de pieza es basándonos en la cantidad de vértices, su área y su ratio de aspecto (este último para diferenciar cuadrado de paralelogramo).
- **Creación de matriz con los datos obtenidos:** Se genera una matriz donde guardar los datos que necesitamos para enviar al robot. Para cada pieza detectada (máximo siete) se guarda su forma, el valor x e y (por separado) de su centroide convertido a mm, el ángulo convertido a radianes y el área en píxeles. En este paso también se rotan los ángulos 90º ya que los agarres han acabado a perpendiculares al ángulo calculado. Esta matriz es ordenada después, primero por tipo de pieza y, para el mismo tipo de pieza, por área para así mantener siempre el mismo orden y poder procesar correctamente por separado cada tipo de pieza.
- **Imagen con anotaciones:** Se genera en pantalla la imagen recortada y se dibujan los contornos encontrados junto con las anotaciones de cada pieza encontrada, incluyendo el nombre de la forma, la ubicación del centroide en pixeles y su ángulo de orientación en grados.
- **Guardado de imágenes:** Se guarda dos imágenes con la fecha actual, la captura original a 1920x1080 y la recortada a 580x580, cada una con las anotaciones añadidas.
- Al finalizar, la función devuelve la matriz de los datos obtenidos ordenados, que en el código donde se llama a esta función (***Canva.py***), se guardan en la matriz ***captured_pos***. Estos datos se utilizarán a posteriori para enviar las instrucciones al robot.

## 11.4 Interfaz de comunicación e instrucciones pick and place del cobot

Para llevar a cabo los movimientos del robot se decide establecer una conexión TCP/IP cliente servidor con el robot. Desde el programa en Python se enviarán las instrucciones de movimientos, separando movimientos de “pick” (punto de recogida) y movimientos de “place” (punto de colocación).

En total son 14 movimientos para cada tipo (28 en total), ya que recogeremos cada una de las 7 piezas detectadas por visión por ordenador, las depositaremos en un espacio de reserva en el lateral del tablero, para luego volver a recogerlas y depositarlas en la configuración final establecida en la aplicación. Esto se ha diseñado así por ser una forma sencilla de asegurarse que nunca solapamos una pieza sobre otra durante el “pick&place”.

Durante los movimientos iremos ejecutando, alternativamente, una instrucción de pick y una de place, hasta realizar las 28 instrucciones.

A continuación, se detalla la información necesaria para cada tipo de movimiento:

- **Movimientos Pick:** se creará una matriz con las siete posiciones obtenidas por la visión por ordenador, que corresponden a la posición inicial de las piezas en el tablero. A estas 7 posiciones se le añadirán en orden, las 7 posiciones de reserva, ya que, al acabar de recoger todas las piezas del recuadro, empezará de nuevo por la primera en la reserva, para colocarla de nuevo en el recuadro, pero en su posición final.

- **Movimientos Place:** se creará otra matriz con las siete posiciones reserva al lado del tablero primero. A estas 7 posiciones se le añadirán las 7 posiciones finales, extraídas de la información del aplicativo, que corresponden al problema que hemos configurado virtualmente.

Sin embargo, esto no puede hacerse directamente y debemos realizar una conversión de datos para que el robot pueda moverse correctamente.

### 11.4.1 Conversión de datos y creación de matriz de instrucciones

Realizada la detección de la posición de las piezas necesitamos convertir esa información y la de la posición final obtenida de la aplicación grafica.

Con una conexión TCP/IP, y debido a cómo funciona URScript, necesitamos convertir cada posición que enviemos al robot en un “string” con el siguiente formato concreto:

**“(x, y, z, rx, ry, rz)”**

Donde **x**, **y**, **z** representan las coordenadas cartesianas en el espacio 3D del robot. Estos datos deben pasarse a metros ya que así funciona en los scripts de URScripts, a diferencia de la interfaz grafica Polyscope de la flexpendant del robot, donde insertamos directamente milímetros.

Los valores **rx**, **ry**, **rz** representan los ángulos de rotación alrededor de los ejes cartesianos, es decir, la orientación del objeto o punto en el espacio. Estos son ángulos de Euler y son muy comunes en robótica y gráficos por computador, y en nuestro caso, deben estar en

radianes. Deberemos convertir nuestros ángulos de inclinación en una configuración válida de estos tres valores, y para ello es necesario el siguiente calculo.

#### 11.4.1.1 Conversión de ángulos en radianes a ángulos de rotación de los ejes cartesianos

En nuestro caso buscamos que la inclinación de la herramienta del robot sea siempre perpendicular al plano donde descansan las piezas, que coincide con el plano Base del robot. Por lo tanto, solo necesitamos girar la muñeca de la herramienta para conseguir coger cada pieza en su orientación exacta.

Para ello, partiremos de la información que nos muestra la FlexPendant sobre la posición del robot. Se posiciona el brazo robot manualmente en una posición perpendicular al plano de trabajo y con la herramienta girada en la posición que nosotros decidimos como ángulo cero (como en la fig. 86, con el logo de la herramienta mirando al frente). Anotamos los valores de los 3 valores **rx**, **ry** y **rz** y empezamos a girar solamente la herramienta. Vamos anotando los valores cada 45º hasta realizar un giro completo, siempre con el brazo perpendicular.

Al finalizar tenemos los siguientes datos (los valores **rz** se han aproximado a 0 porque eran todos prácticamente 0):

Tabla 11. Valores de giro rx, ry, rz obtenidos manualmente.

| Grados | Rx (rad) | Ry (rad) | Rz (rad) |
|---|---|---|---|
| 0º | 1,3142 | 2,8785 | 0,0000 |
| 45º | 2,2278 | 2,2278 | 0,0000 |
| 90º | 2,9494 | 1,0606 | 0,0000 |
| 135º | 3,1178 | -0,1245 | 0,0000 |
| 180º | 2,8256 | -1,2978 | 0,0000 |
| 225º | 2,1830 | -2,2058 | 0,0000 |
| 270º | 1,0638 | -2,9147 | 0,0000 |
| 315º | 0,1025 | 3,1735 | 0,0000 |
| 360º | 1,3142 | 2,8785 | 0,0000 |

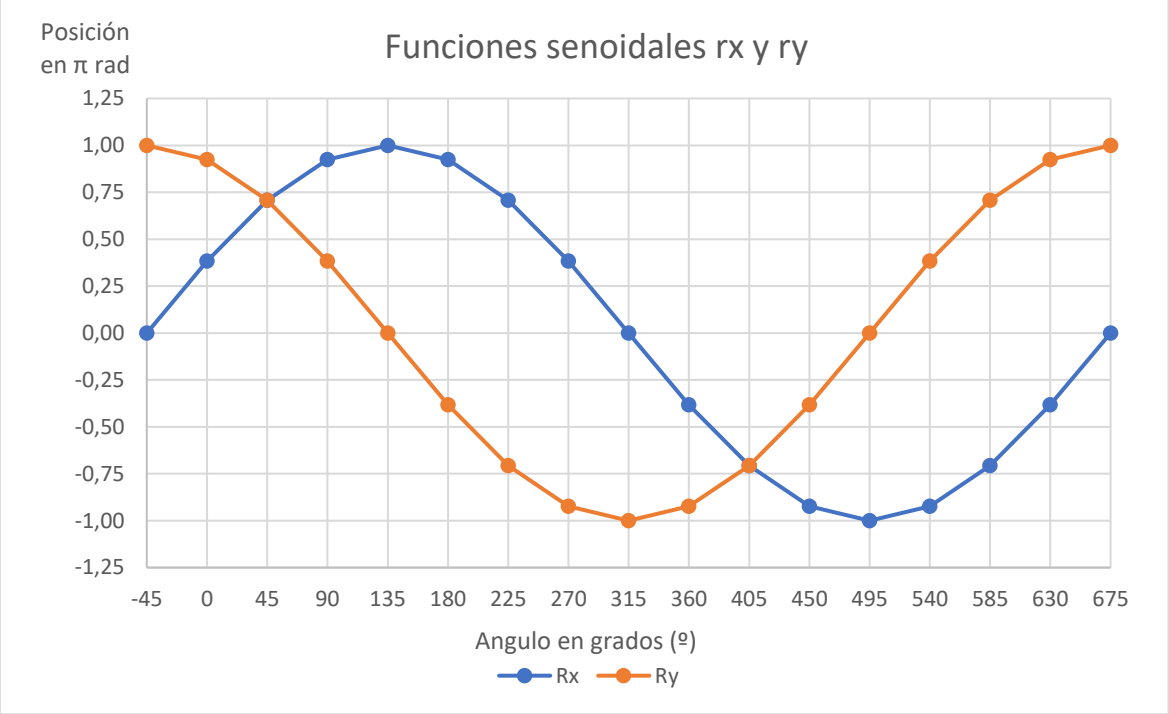
Se observa que el valor **rz** siempre vale 0 cuando la herramienta apunta perpendicularmente al plano y que los valores **rx** y **ry** toman valores entre $\pi$ y - $\pi$ alternativamente, probablemente en una función senoidal. Por ejemplo, en los ángulos 135º y 315º los valores son alternativamente 0 y $\pi$ aproximadamente (ya que la posición establecida manualmente no es exacta), esto puede indicar un desfase de $\pi$ entre sus funciones.

Utilizando como referencia la información de la tabla, se crean dos funciones senoidales que se adapten a los datos, obteniendo las siguientes funciones:

$$rx = \pi * \sin\left(\frac{x}{2} + \frac{\pi}{8}\right)$$

$$ry = \pi * \sin\left(\frac{x}{2} + \frac{5\pi}{8}\right)$$

Donde $x$ es nuestro ángulo de inclinación en radianes. Estas funciones senoidales tienen una amplitud de $\pi$ (oscila entre $-\pi$ y $\pi$), han sido estiradas horizontalmente por un factor de 2 y tienen un desfase entre ellas de $\frac{\pi}{2}$.




*Gráfica 10. Funciones senoidales rx y ry.*

Con estas funciones podemos obtener una la tabla parecida a la anterior, ahora rectificada:

Tabla 12. Valores de giro rx, ry, rz calculados mediante función de conversión.

| Grados | Radianes | Rx (rad) | Ry (rad) |
|---|---|---|---|
| 0º | 0 | 1,2022 | 2,9025 |
| 45º | 0,7854 | 2,2214 | 2,2214 |
| 90º | 1,5708 | 2,9025 | 1,2022 |
| 135º | 2,3562 | 3,1416 | 0 |
| 180º | 3,1416 | 2,9025 | -1,2022 |
| 225º | 3,9270 | 2,2214 | -2,2214 |
| 270º | 4,7129 | 1,2022 | -2,9025 |
| 315º | 5,4978 | 0 | -3,1416 |
| 360º | 6,2832 | -1,2022 | -2,9025 |

Probamos estas posiciones manualmente en el robot y generan exactamente la posición deseada, más exacta que las que habíamos generado manualmente. También se prueban por si acaso ángulos intermedios a partir del cálculo de las funciones y se comprueba que todo funciona correctamente.

#### 11.4.1.2 Conversión de las posiciones detectadas por visión por ordenador

Nuestros valores almacenados en la matriz ***captured_pos*** son convertidos utilizando el método ***get_pick_matrix(captured_pos)*** del archivo **Utils.py** y guardados en la nueva matriz **pick_matrix**. En este pequeño código primero se acaban de ajustar en 45º los ángulos de orientación del cuadrado y el paralelogramo (ya que sus agarres se configuran así). Después se crea la matriz **pick_matrix** con la conversión de los valores **x**, **y**, **z** de milímetros a metros. Establecemos el valor **z** a 60 milímetros (0.060 m) de altura para todos los puntos, ya que será nuestra altura de aproximación al punto final de recogida. Los valores **rx**, **ry**, **rz** se extraen a partir del ángulo en radianes con la función descrita en el apartado anterior.

Esta matriz se extiende con la matriz **WAIT_POS**, matriz con las siguientes posiciones de espera en el lateral derecho del tablero (posiciones extraídas manualmente con el brazo robot en la estación):

```
WAIT_POS = [
[0.270, -0.260, 0.060, 1.2022, 2.9025, 0], # Cuadrado
[0.380, -0.317, 0.060, 1.2022, 2.9025, 0], # Paralelogramo
[0.360, -0.250, 0.060, 1.2022, 2.9025, 0], # Triangulo grande 1
[0.312, -0.375, 0.060, 1.2022, 2.9025, 0], # Triangulo grande 2
[0.295, -0.317, 0.060, 1.2022, 2.9025, 0], # Triangulo mediano
[0.430, -0.275, 0.060, 1.2022, 2.9025, 0], # Triangulo pequeño 1
[0.282, -0.425, 0.060, 1.2022, 2.9025, 0], # Triangulo pequeño 2
]
```

#### 11.4.1.3 Conversión de las posiciones solución creadas en la aplicación gráfica

Los valores de las posiciones de la formación de piezas creadas en la aplicación se encuentran almacenadas en la variable **solution_table** (matriz que utilizábamos para crear también la información del dataset). Se aprovecha la función ***table_to_solution_vector(solution_table)*** del archivo **Utils.py** para extraer solo las posiciones **x**, **y**, **ángulo** (guardándose en ***solution_vector***). Con estos datos llamamos a la función en el mismo archivo ***get_place_matrix(solution_vector)***.

Antes de realizar la conversión es importante detallar que para evitar que las piezas puedan solaparse unas con otras al colocarse muy cerca unas de otras se ha sobredimensionado el recuadro de trabajo. Se decide así porque se prevé que, por aproximaciones, pequeños errores de captura o distorsiones, será prácticamente imposible ajustar milimétricamente las piezas físicas como en el lienzo de la aplicación. En la aplicación gráfica, donde el lienzo hace 600 pixeles y el cuadrado hace 60 pixeles, tenemos una relación 6:1. Nuestro pieza cuadrado física mide 40 mm por lado, si quisiéramos tener la misma relación el recuadro de trabajo debería hacer 240 mm (40 x 6).

En su lugar se decide redimensionar la zona de trabajo a 250 mm, de esta manera tendremos una diferencia de aproximadamente 4% entre las medidas digitales y las físicas, creando así un pequeño margen entre las piezas al ser colocadas.

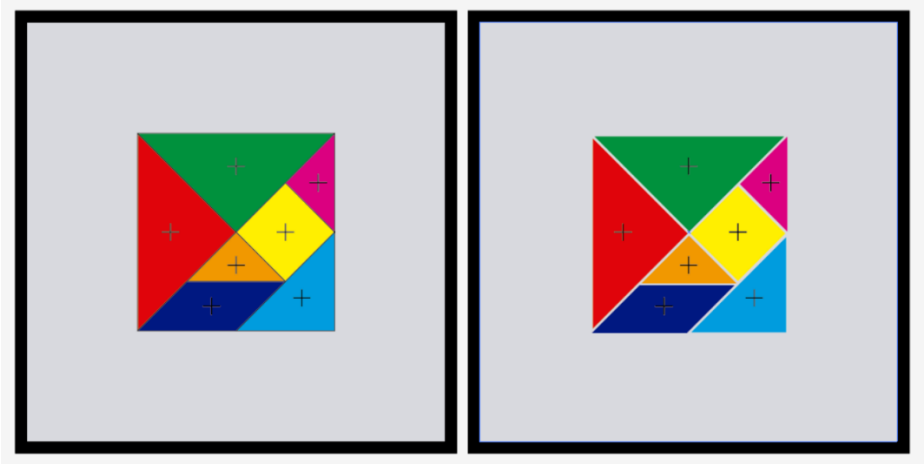


*Figura 85. Diagrama diferencia márgenes entre posición piezas digitales y físicas.*

Siguiendo con el código, en esta función se convierten los pixeles del lienzo a milímetros del recuadro del tablero. De forma similar a como se calculó en el apartado de visión por ordenador tenemos:

En el eje x:

- El rango en pixeles es [0, 1] (recordemos que estos datos se normalizaron dividiendo entre 300, 1 corresponde a 300 pixeles del lienzo solución).
- El rango en milímetros es [-48.5, 201.5]

Aplicando la fórmula de la pendiente de una línea queda:

$$y = 250x - 48{,}5$$

En el eje y:

- El rango en pixeles es [0, 1]
- El rango en milímetros es [-233, -483]

La interfaz gráfica tiene los ejes mismos ejes que la cámara (fig. 87) y por lo tanto también debe invertirse el eje y. Aplicando la fórmula de la pendiente de una línea queda:

$$y = -250x - 233$$

Convertidos los valores de los ejes **x** e **y**, procedemos a convertir los ángulos.

Se ajustan los ángulos para que coincidan los agarres de cada pieza (90º para los triángulos, 45º para el resto) y se convierten a radianes.

Se crea una matriz ***place_matrix*** con las primeras 7 posiciones con las posiciones de espera **WAIT_POS** (ya que primero se depositarán en esta zona).

Se extiende esta matriz con los datos **x**, **y**, **z** convertidos de milímetros a metros (todas las posiciones del eje z se establecen de nuevo a 60 mm) y los valores **rx**, **ry** y **rz** utilizando las mismas funciones obtenidas en la conversión de las imágenes capturadas por la cámara.

#### 11.4.1.4 Convertir matrices posición cadenas de "string"

Por último, creadas las matrices **pick_matrix** y **place_matrix** necesitamos convertir sus valores en cadenas string que podamos enviar mediante mensajes con la conexión cliente servidor TCP/IP.

Llamaremos a la función ***get_robot_coords(matrix)*** del archivo **Utils.py** para convertir cada lista de posiciones [x, y, z, rx, ry, rz] en un string con formato "(x, y, z, rx, ry, rz)".

Hecho esto las matrices quedan listas para ser utilizadas por el código de envío de instrucciones descrito más adelante.

### 11.4.2 Conexión TCP/IP cliente servidor entre el portátil y el robot

La técnica de comunicación utilizando sockets TCP/IP opera bajo una estructura cliente-servidor. TCP/IP es un conjunto de protocolos de comunicación entre ordenadores en una red. TCP (Protocolo de Control de Transmisión) y IP (Protocolo de Internet) son los dos protocolos principales en este conjunto. Cliente-Servidor es un modelo en el cual un dispositivo (cliente) solicita y recibe información de otro dispositivo (servidor).

En este escenario específico, el ordenador portátil utilizando Python tiene el rol de servidor, mientras que el robot juega el papel del cliente.

Se conectan los dos dispositivos a través de un cable ethernet, y se establece la dirección IP del ordenador (desde la configuración de red IPv4 de Windows) en el rango IP del robot. Junto con la puerta de enlace.

Hecho esto revisamos desde la consola de Windows (cmd) que podemos comunicarnos con el robot enviando un ping a la IP del robot. Al recibir respuesta positiva sabemos que podremos establecer comunicación con la controladora del robot.

#### 11.4.2.1 Establecer conexión

Realizada las configuraciones iniciales, se realizara la conexión con el robot cuando, habiendo ejecutado la parte inicial del código al accionar el botón "Enviar al robot" (la captura de posiciones física, las posiciones solución, y la conversión de todas ellas a cadenas *string)* se llame al método ***connection_to_robot(pick_coords, place_coords)*** del archivo **socket_com.py**.

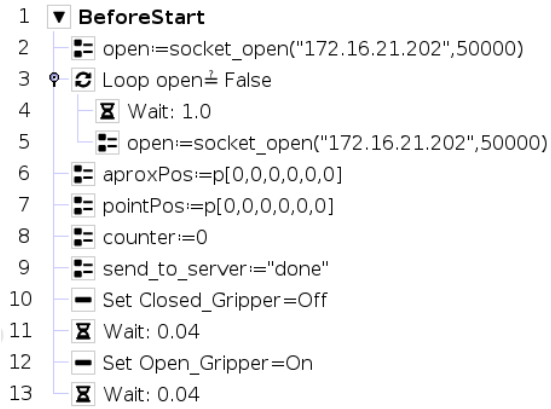
En este script hacemos uso de la biblioteca socket, parte integral de Python, que facilita la creación y gestión de conexiones socket. Para aprovechar sus funciones, basta con importarla al comienzo del código con ***import socket.***

Después de la importación, podemos proceder a la creación del socket, definiendo su tipo y características, y finalmente, vinculándolo a la dirección IP y un puerto específico.

Con el servidor en marcha, el ordenador está listo, y pasa a esperar y aceptar conexiones entrantes, en este caso, desde el robot.

Por parte del robot, en su programa en URScript, configuraremos en el apartado de código antes del programa normal de ejecución (llamado **BeforeStart** en URScript) un bucle donde el programa hará uso de la función ***socket_open(dirección, puerto)*** para inicializar la comunicación socket con el ordenador. Dentro de dicha función se ha de incluir la misma dirección IP del ordenador y el mismo puerto utilizado. La función se incluye dentro de un

bucle para asegurarse que el programa no comienza hasta haber establecido la comunicación con el ordenador.




Normalmente, esperaremos a que el servidor entre en espera para inicializar el programa del robot. Una vez que está establecida la conexión comienza el envío de instrucciones por parte del servidor al robot.

### 11.4.3 Envío de instrucciones al robot

Establecida la conexión con el robot el programa entra en un bucle recorriendo las matrices **pick_matrix** y **place_matrix**.

Por cada valor string en **pick_place** lo enviaremos al robot y esperaremos a que acabe los movimientos esperando una respuesta mediante el siguiente código:

```
# Enviamos la posición PICK:
connection.sendall(bytes(pick_coords[coord]+"\n", 'utf8'))
print("Posicion enviada: ", pick_coords[coord])

# Esperamos a la señal de fin del movimiento PICK del robot
resp = ""
while len(resp) == 0:
    data = connection.recv(1024)
    resp = str(data, 'utf8')
    print('Response: ' + resp)
    time.sleep(0.5)  # Retraso de 500ms
```

Esta línea enviará el string de la posición actual (***[coord]*** se utiliza como índice del bucle) junto al número de posiciones del vector (6).

Por parte del programa del robot, después de establecer conexión se moverá a un punto guardado justo encima del centro del recuadro del tablero, y dentro del programa estructurado en la carpeta “Pick” llamará a la función **Recieve_Pos** (recibir posición). Este pequeño script ejecuta los comandos necesarios para leer correctamente el mensaje enviado por el servidor:

Mediante este subprograma el robot recibe el mensaje del servidor y va asignando cada valor a la variable **aproxPos** (inicializada en **BeforeStart**). Una vez se hayan añadido los

seis valores, copiaremos la posición en **pointPos** y rectificaremos la altura (eje **z**) a 25 mm (que es la altura a la que el robot debe recoger/soltar las piezas).

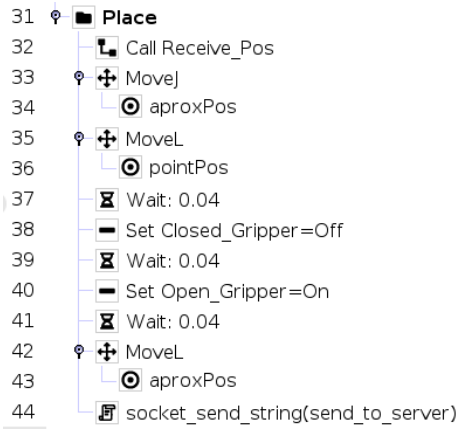
```
42 Receive_Pos
43   recPos:=socket_read_ascii_float(6)
44   Loop recPos≠6
45     recPos:=socket_read_ascii_float(6)
46   Loop counter<6
47     aproxPos[counter]=recPos[counter+1]
48     counter:=counter+1
49   counter:=0
50   pointPos:=aproxPos
51   pointPos:=aproxPos
52   pointPos[3]=0.025
```

Con las posiciones de aproximación y de recogida establecidas continua el programa contenido en la carpeta "Pick", el robot se moverá libremente hasta el punto de aproximación, bajará linealmente y más despacio al punto de recogida, activará y desactivará las variables digitales asociadas a las pinzas paralelas y volverá a subir al punto de aproximación (ya con la pieza agarrada). Para que el servidor sepa que el robot ha acabado de realizar los movimientos.

```
14 Robot Program
15   MoveJ
16     aproxBoard
17   Pick
18     Call Receive_Pos
19     MoveJ
20       aproxPos
21     MoveL
22       pointPos
23     Wait: 0.04
24     Set Open_Gripper=Off
25     Wait: 0.04
26     Set Closed_Gripper=On
27     Wait: 0.04
28     MoveL
29       aproxPos
30     socket_send_string(send_to_server)
```

De vuelta al código Python del ordenador, al recibir la respuesta de que el robot a finalizado su proceso, enviará esta vez la posición "place" de su matriz correspondiente, y volverá a esperar respuesta.

Por parte del robot, habrá comenzado el código de la carpeta "Place", que funciona exactamente igual que la anterior con la diferencia que, en vez de agarrar la pieza, la deposita.



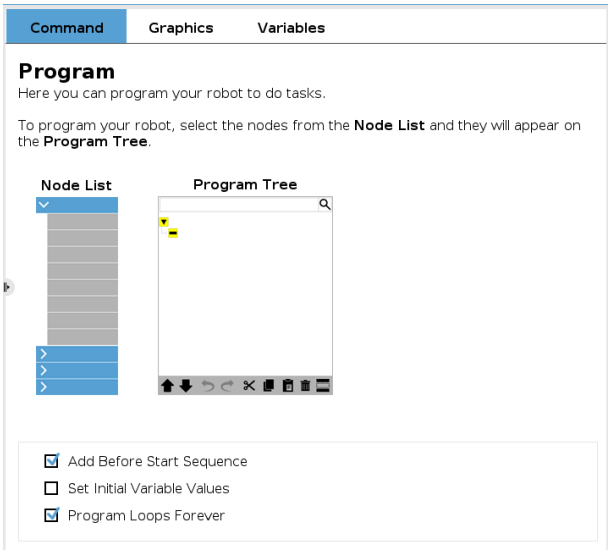

Depositada la pieza, el código del robot vuelve a comenzar el programa desde el principio (excepto **BeforeStart**) porque así se configurado en las opciones de programa.

Por su parte el ordenador mediante el bucle seguirá enviando todas las posiciones alternativamente hasta que haya recorrido todas las matrices, momento en el cual se cerrará la conexión socket y podremos seguir utilizando la aplicación gráfica.

Con estas instrucciones queda resuelta satisfactoriamente la tarea de montar la solución del tangram físicamente en la estación.

### 11.4.4 Código completo del robot en URScript

```
1  BeforeStart
2      open:=socket_open("172.16.21.202",50000)
3      Loop open≟ False
4          Wait: 1.0
5          open:=socket_open("172.16.21.202",50000)
6      aproxPos:=p[0,0,0,0,0,0]
7      pointPos:=p[0,0,0,0,0,0]
8      counter:=0
9      send_to_server:="done"
10     Set Closed_Gripper=Off
11     Wait: 0.04
12     Set Open_Gripper=On
13     Wait: 0.04
14 Robot Program
15     MoveJ
16         aproxBoard
17     Pick
18         Call Receive_Pos
19         MoveJ
20             aproxPos
21         MoveL
22             pointPos
23         Wait: 0.04
24         Set Open_Gripper=Off
25         Wait: 0.04
26         Set Closed_Gripper=On
27         Wait: 0.04
28         MoveL
29             aproxPos
30         socket_send_string(send_to_server)
31     Place
32         Call Receive_Pos
33         MoveJ
34             aproxPos
35         MoveL
36             pointPos
37         Wait: 0.04
38         Set Closed_Gripper=Off
39         Wait: 0.04
40         Set Open_Gripper=On
41         Wait: 0.04
42         MoveL
43             aproxPos
44         socket_send_string(send_to_server)
42 Receive_Pos
43     recPos:=socket_read_ascii_float(6)
44     Loop recPos≠6
45         recPos:=socket_read_ascii_float(6)
46     Loop counter<6
47         aproxPos[counter]=recPos[counter+1]
48         counter:=counter+1
49     counter:=0
50     pointPos:=aproxPos
51     pointPos:=aproxPos
52     pointPos[3]=0.025
```



# 12 Soluciones alternativas

En esta sección se describen algunas soluciones alternativas a la planteadas y desarrolladas en el proyecto.

**Creación de dataset**

En el presente proyecto se decidió realizar el dataset mediante una aplicación gráfica par poder tener un entorno controlado, pero sobre todo para servir como base de programación de todo el proyecto ya que se deseaba familiarizarse con el lenguaje de programación Python.

Como alternativa se podría haber creado el dataset directamente formando configuraciones de las piezas físicamente y realizando sus capturas. A posteriori estas capturas podrían ser procesadas para convertirlas en siluetas. Si bien este método hace mucho más lento la tarea de crear el dataset (cada configuración de piezas manualmente llevaría más tiempo) el ahorro a la hora de crear toda la programación de los lienzos y movimientos de polígonos en la aplicación no es menospreciable.

**Resolución del Tangram**

Durante este proyecto no fue posible crear una red neuronal que fuese capaz de resolver el rompecabezas. Una alternativa más clásica es crear a partir de una programación por objetos un sistema jerárquico de pruebas donde matemáticamente se pueda comprobar la viabilidad de las posiciones de las piezas en la silueta hasta encontrar las combinaciones posibles correctas. Sin embargo, no parece un desafío sencillo.

**Comunicación ordenador robot**

Una alternativa obvia para la comunicación TCP/IP escogida en el proyecto es crear una conexión OPC UA entre el ordenador y el robot, estableciendo una comunicación segura y eficiente, permitiendo el control y monitoreo del robot de manera remota.

# 13 Presupuesto

En el siguiente apartado, se ofrece un análisis pormenorizado de todos los costes que han estado involucrados en la ejecución del trabajo en cuestión. Estos costes se clasificarán en dos categorías principales.

**Costes de recursos humanos:** Para calcular el gasto en este ámbito, consideraremos el total de horas que se han dedicado exclusivamente a este proyecto. Es importante mencionar que este trabajo se llevó a cabo paralelamente bajo un acuerdo de prácticas entre el alumno, la facultad de l'ESEIAT y la farmacéutica Siegfried SL, sin estar relacionados de ninguna manera prácticas y proyecto. Aun así, es lógico aplicar una la tarifa similar establecida en ese convenio de prácticas, siendo esta de 8€ por cada hora trabajada.

Tabla 13. Costes de recursos humanos.

| Concepto | Horas | Precio/hora (€/h) | Precio total (€) |
|---|---|---|---|
| Investigación | 90 | 8 | 720,00 € |
| Programación de la aplicación Python | 150 | 8 | 1.200,00 € |
| Programación y entrenamiento redes neuronales | 220 | 8 | 1.760,00 € |
| Programación y testeo del sistema de visión por ordenador | 80 | 8 | 640,00 € |
| Programación del cálculo de posiciones | 20 | 8 | 160,00 € |
| Programación del sistema de comunicaciones | 15 | 8 | 120,00 € |
| Programación y testeo instrucciones robot | 40 | 8 | 320,00 € |
| Otras pruebas en laboratorio | 15 | 8 | 120,00 € |
| Elaboración de la documentación | 160 | 8 | 1.280,00 € |
| **Total horas** | **790** | **Subtotal** | **6.320,00 €** |

**Costes de materiales y desplazamientos:** En esta categoría se incorporan todos los insumos físicos necesarios para el desarrollo y ejecución del proyecto, así como los gastos de desplazamiento al laboratorio.

No se incluyen aquí los dispositivos que no han sido un gasto directo para la realización del proyecto. Los ordenadores, la cámara, la impresora 3D, algunas herramientas, el robot del laboratorio, no se consideran gastos ya que se ha dispuesto de ellos gratuitamente, ya sea porque se disponía de ellos de antemano o por ser un servicio de la misma universidad.

Tabla 14. Costes de materiales y desplazamientos.

| Concepto | Precio | Unidades | Precio total (€) |
|---|---|---|---|
| Piezas tangram (marca: Schmidt Spiele) | 6,99 € | 1 | 6,99 € |
| PLA piezas impresas de agarre (marca: Eleego) | 0,20 € | 12 | 2,40 € |
| Paquete de tornillos tirafondos | 1,90 € | 1 | 1,90 € |
| Pintura acrílica (marca: LIDL) | 5,99 € | 1 | 5,99 € |
| Balda para tablero (marca: IKEA) | 7,90 € | 1 | 7,90 € |
| Impresión hojas de calibración | 0,60 € | 2 | 1,20 € |
| Tickets T10 (1 zona) RENFE | 11,35 € | 4 | 45,40 € |
| | | **Subtotal** | **71,78 €** |

Al sumar los gastos derivados de estas dos categorías, podremos obtener una estimación del coste total del proyecto.

Tabla 15. Costes totales.

| Tipo de coste | Precio total (€) |
|---|---|
| Recursos humanos | 6.320,00 € |
| Materiales y desplazamientos | 71,78 € |
| **TOTAL** | **6391,78 €** |

# 14 Estimación del impacto medioambiental

El proyecto en cuestión ha sido diseñado y ejecutado con una preocupación ambiental en mente, manteniendo su consumo energético a un mínimo. El equipamiento utilizado para su desarrollo ha sido, en esencia, un ordenador de sobremesa, un portátil y las bombillas LED de escritorio que se utilizan para una correcta iluminación de la zona de trabajo.

Especificaciones del consumo energético de los equipos:

- Ordenador de sobremesa + monitor:
  En uso normal de ofimática el ordenador y el monitor consumen aproximadamente unos 350Wh.
  A plena carga, durante los entrenamientos de las redes neuronales, este ordenador tiene un consumo elevado de unos 600Wh (durante los entrenamientos el monitor se mantiene apagado).
- Ordenador portátil: En uso normal el dispositivo tiene un consumo de 140Wh.
- Bombillas LED: Estos dispositivos son de bajo consumo y conjuntamente consumen 16Wh.

Tabla 16. Consumo energético en kWh

| Concepto | Horas | Consumo (Wh) | Consumo Total (kWh) |
|---|---|---|---|
| Ordenador de escritorio + monitor (uso normal) | 520 | 350 | 182 |
| Ordenador de escritorio (plena carga) | 120 | 600 | 72 |
| Ordenador portátil (uso normal) | 120 | 140 | 16,8 |
| Bombillas LED | 400 | 16 | 6,4 |
| | | **Total** | **277,2 kW/h** |

Teniendo en cuenta las horas invertidas durante los 6 meses de duración del proyecto, el consumo total, y que los desplazamientos se han efectuado todos con transporte público, se considera que el impacto ambiental es bajo, similar a un cualquier trabajo de oficina.

# 15 Conclusiones

**Resultados**

Cada componente del proyecto fue finalizado con éxito, con la excepción notable de la resolución de problemas Tangram mediante técnicas de aprendizaje automático. A pesar de la considerable cantidad de tiempo y esfuerzo invertidos en esta tarea, la misma se comprobó ser mucho más compleja de lo que se había anticipado originalmente.

Durante esta fase del proyecto, se encontraron diversos desafíos inesperados que obstaculizaron el progreso y pusieron a prueba nuestras habilidades y conocimientos sobre aprendizaje automático. Problemas como la incompatibilidad de la tarjeta gráfica con la librería tensorflow para realizar entrenamientos más veloces o un equipo más potente para los últimos entrenamientos. No obstante, los desafíos enfrentados en la resolución automática de problemas Tangram no opacaron los logros conseguidos en otras áreas del proyecto.

La aplicación gráfica que se desarrolló resultó ser altamente eficaz, facilitando la generación de datos para el entrenamiento del algoritmo de manera eficiente y la transmisión fluida de instrucciones al robot.

Además, se implementó exitosamente las técnicas de visión por ordenador con las que el robot pudo identificar y manejar con gran precisión las piezas físicas presentes en la estación de trabajo, asegurando que cada pieza fuera manipulada de manera adecuada.

Finalmente, el robot pudo realizar la configuración de piezas recibidas por las instrucciones de la aplicación en la estación de trabajo de una manera segura y consistente.

**Conclusiones**

Aunque este proyecto no ha cumplido con la totalidad de cada uno de sus objetivos planificados, sí ha alcanzado un amplio porcentaje de ellos. Más allá de los logros tangibles, el estudio del mismo puede servir como camino sólido y estructurado para un estudio posterior y ha establecido unas bases robustas para abordar la tarea de programar una red neuronal con el propósito específico de resolver el complejo rompecabezas Tangram.

A lo largo del desarrollo de este proyecto, el estudiante ha adquirido nuevas y valiosas competencias en diversas áreas de la robótica. El conocimiento adquirido en campos tales como el aprendizaje automático, la visión por ordenador, la automatización de robots y la programación en Python, ha enriquecido de manera significativa la base de competencias del alumno, preparándolo para enfrentar desafíos más complejos en el futuro.

El proyecto ha servido también como un campo de pruebas importante para las ideas y metodologías aplicadas, permitiendo un análisis detallado de su efectividad y áreas de mejora potencial. Este aspecto es crucial, ya que proporciona “insights” valiosos que pueden guiar la optimización y refinamiento de futuros proyectos similares.

Aunque la meta de programar una red neuronal para resolver el rompecabezas Tangram no se ha podido alcanzar, se desea y espera que este proyecto pueda servir como precedente alentador para futuras investigaciones y desarrollos.

# 16 Agradecimientos

Quisiera expresar mi gratitud hacia aquellas personas que me han ayudado y apoyado durante la realización de este proyecto.

Agradecerle a mi director de proyecto, Marc Flor, el darme la oportunidad de matricular el proyecto rápidamente para facilitarme el inicio del convenio de prácticas. Así como la libertad que me ha dado con la elaboración del mismo y su apoyo a pesar de la falta de tiempo del que disponía.

Agradecerle a mi responsable de prácticas, Sergio Masegosa, la oportunidad de realizar unas prácticas de gran importancia para mí, cerca de casa y con un horario flexible que me permitieran finalizar este proyecto de forma satisfactoria.

Agradecerle a mi hermano, Aleix Roca, el prestarme su portátil de forma incondicional siempre que lo he necesitado durante todas las pruebas de laboratorio.

Agradecerle a todos los familiares y amigos que me han apoyado y animado durante estos duros meses de trabajo en los que, en ocasiones, he sentido que me faltaban las fuerzas.

A todos y cada uno, gracias de corazón.

# 17 Referencias

[1] Tangram [en línea]. Wikipedia. [Consulta: 6 abril 2023]. Disponible en: https://en.wikipedia.org/wiki/Tangram

[2] Introduction to Python [en línea]. W3schools. [Consulta: 6 abril 2023]. Disponible en: https://www.w3schools.com/python/python_intro.asp

[3] What is PyCharm? [en línea]. Intellipaat Blog. [Consulta: 6 abril 2023]. Disponible en: https://intellipaat.com/blog/what-is-pycharm/

[4] What is Python? [en línea]. Intellipaat Blog. [Consulta: 6 abril 2023]. Disponible en: https://www.python.org/doc/essays/blurb/

[5] tkinter — Interface de Python para Tcl/Tk [en línea]. Python. [Consulta: 15 abril 2023]. Disponible en: https://docs.python.org/es/3/library/tkinter.html

[6] The Shapely User Manual [en línea]. Shapely. [Consulta: 15 abril 2023]. Disponible en: https://shapely.readthedocs.io/en/stable/manual.html

[7] Machine Learning with Python [en línea]. FreeCodeCamp. [Consulta: 5 junio 2023]. Disponible en: https://www.freecodecamp.org/learn/machine-learning-with-python/

[8] CS231n Convolutional Neural Networks for Visual Recognition Course Website [en línea]. Stanford University. [Consulta: 5 junio 2023]. Disponible en: https://cs231n.github.io/

[9] Deep Learning [en línea]. MIT Press. [Consulta: 12 junio 2023]. Disponible en: https://www.deeplearningbook.org/

[10] La guía definitiva de las redes neuronales convolucionales [en línea]. Frogames. [Consulta: 18 junio 2023]. Disponible en: https://frogames.es/la-guia-definitiva-de-las-redes-neuronales-convolucionales/

[11] Camera Modeling: Exploring Distortion and Distortion Models, Part I [en línea]. Tangram Vision. [Consulta: 20 julio 2023]. Disponible en: https://www.tangramvision.com/blog/camera-modeling-exploring-distortion-and-distortion-models-part-i

[12] How to Detect Shapes in Images in Python using OpenCV? [en línea]. GeeksForGeeks. [Consulta: 20 julio 2023]. Disponible en: https://www.geeksforgeeks.org/how-to-detect-shapes-in-images-in-python-using-opencv/

[13] OpenCV - Python Tutorials [en línea]. OpenCV. [Consulta: 22 julio 2023]. Disponible en: https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html

[14] Universal Robots e-Series User Manual [en línea]. Universal Robots SL. [Consulta: 19 agosto 2023]. Disponible en: https://s3-eu-west-1.amazonaws.com/ur-support site/181319/99403_UR3e_User_Manual_en_Global.pdf

[15] Universal Robots programming feature point orientation by Matt Bush [en línea] Hirebotics [Consulta: 19 agosto 2023] Disponible en: https://blog.hirebotics.com/engineering/universal-robots-programming-feature-point-orientation

[16] TCP/IP socket communication via URScript [en línea] Universal Robots SL [Consulta: 24 agosto 2023] Disponible en: https://www.universal-robots.com/articles/ur/interface-communication/tcpip-socket-communication-via-urscript/

[17] Moving to a position calculated from user input [en línea] Universal Robots SL [Consulta: 24 agosto 2023] Disponible en: https://www.universal-robots.com/articles/ur/programming/moving-to-a-position-calculated-from-user-input/

[18] Overview of client interfaces [en línea] Universal Robots SL. [Consulta: 02 septiembre 2023] Disponible en: https://www.universal-robots.com/articles/ur/interface-communication/overview-of-client-interfaces/

[19] Joan, T. R. SISTEMA DE VISIÓN POR COMPUTADOR DE BAJO COSTE PARA ROBOTS UR [en línea]. Trabajo final de grado, UPC, Escola d'Enginyeria de Barcelona Est. 2023. Disponible en: https://upcommons.upc.edu/bitstream/handle/2117/388273/TFG_JoanTurRuiz.pdf?sequence=2&isAllowed=y

[20] Ferriol, A.G. DISSENY I PROTOTIPAT D'UNA PINÇA ELÈCTRICA PER ROBOT COL·LABORATIU [en línea]. Trabajo final de grado, UPC, Escola d'Enginyeria de Barcelona Est. 2022. Disponible en: https://upcommons.upc.edu/bitstream/handle/2117/374650/TFE_Antoni_Ferriol_2022.pdf?sequence=1&isAllowed=y

[21] Xavier, M.G. SISTEMA AUTOMÀTIC DE CLASSIFICACIÓ DE PECES BASAT EN VISIÓ ARTIFICIAL I ROBÒTICA [en línea]. Trabajo final de grado, UPC, Escola d'Enginyeria de Barcelona Est. 2021. Disponible en: https://upcommons.upc.edu/bitstream/handle/2117/356299/TFG_XMJ_Sistema_autom%c3%a0tic_de_classificaci%c3%b3_de_peces_basat_en_visi%c3%b3_artificial_i_rob%c3%b2tica.pdf?sequence=1&isAllowed=y

# 18 Referencias de figuras