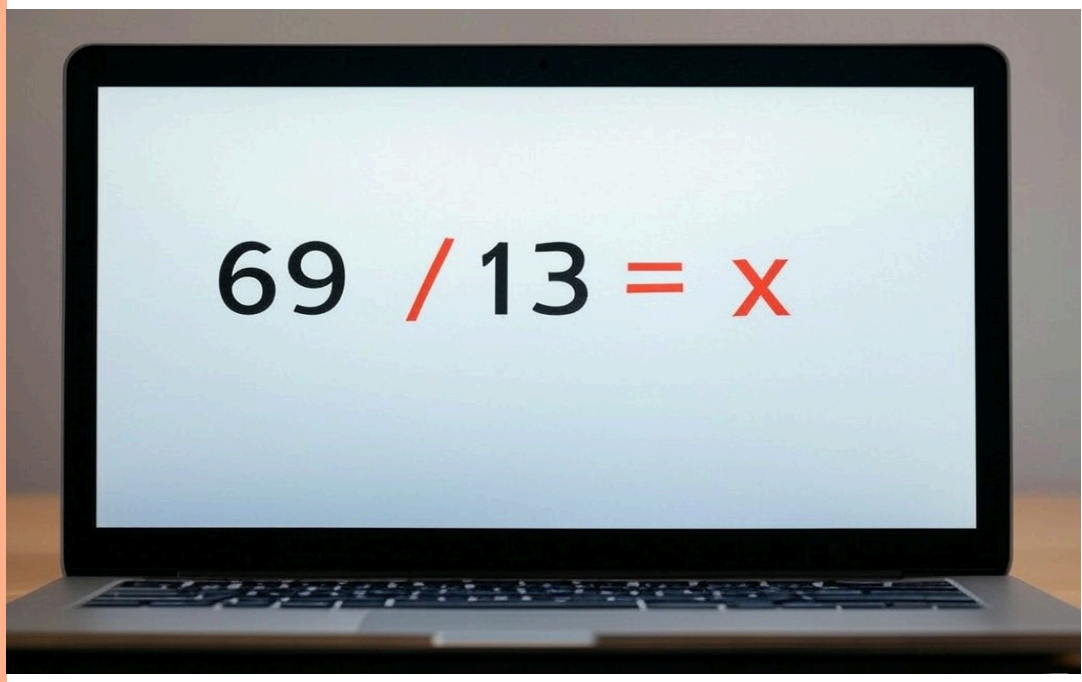
Visión por Computador



Proyecto: Math Solver

Integrantes: Juan Diego Serrato, Hugo Sevilla, Hugo López y Gabriel Segovia

Profesor: Jose Javier Valero Mas

Universitat d'Alacant

Índice

Introducción

A la hora de elegir la temática del proyecto final de esta asignatura (Visión por Computador), nos inspiramos en dos aplicaciones conocidas en el ámbito académico por resolver problemas matemáticos a partir de una imagen del mismo y, debido a su posible uso a la hora de servir de apoyo para tareas reales, decidimos desarrollar un proyecto con un funcionamiento similar. A partir de una imágen que contenga una operación o ecuación matemática, se podrá devolver su solución.

Problema planteado

Una vez ya se ha elegido la temática del proyecto, es momento de subrayar los problemas que nos vamos a encontrar.

El primer problema a resolver claramente es detectar y reconocer los dígitos. Para ello, se hará uso de funciones vistas en prácticas como **findcontours** (para detectar los dígitos) y **boundingrect** para separarlos y poder trabajar individualmente con ellos.

Seguidamente, una vez se han separado, se deben reconocer. Para ello se hará uso de un dataset para reconocer los números (del 0 al 9) y otro dataset para reconocer los símbolos (formado por +, · , x, -, /). El uso de dos datasets afectará a la hora de realizar pequeñas adaptaciones en la implementación del código.

Una vez se han reconocido, simplemente se deben pasar ordenadamente a un tipo string, que recibirá otro programa elaborado en Python para resolver el problema. Cabe destacar que el programa distingue entre operaciones matemáticas o ecuaciones de primer grado en función de si hay una igualdad en la expresión recibida.

Principales secciones del proyecto

Flujo General del Programa

1. Captura los frames desde la cámara del ordenador.
2. Dibuja un rectángulo y captura automáticamente una imagen después de 7 segundos.
3. Procesa la imagen para extraer y segmentar caracteres.
4. Clasifica los caracteres como dígitos o símbolos usando modelos de aprendizaje automático.
5. Construye una ecuación basada en los caracteres reconocidos.
6. Evalúa o resuelve la ecuación (según si es una operación matemática o una ecuación).
7. Imprime el resultado.

Obtención de los caracteres

Esta primera sección del proyecto es fundamental puesto que representa los cimientos del funcionamiento de la aplicación, si la captación de la ecuación presentada por el usuario o la separación de los elementos de la ecuación no se hiciera correctamente, previsiblemente la identificación de los dígitos y símbolos sufriría una significante reducción en su precisión.

Por esta razón, se ha seguido la siguiente estrategia para segmentar de la ecuación original los distintos caracteres presentes:

En primer lugar, se abre la cámara del portátil y se verifica que este proceso se ha realizado correctamente, luego dentro de un while capturamos los frames de la cámara constantemente durante 7 segundos, mientras se procesan (se obtienen las dimensiones) se muestra un rectángulo centrado en la pantalla. Este rectángulo de color verde sirve como guía para que el usuario alinee dentro de éste la ecuación u operación matemática que desea resolver. Además, dentro de la ventana en la que se muestra lo que se está captando con la cámara se muestra el nombre del proyecto "Math Solver" y el mensaje de "Alinea tu ecuación dentro del recuadro". Tras los 7 segundos, captura ese frame, el cual se usará como base para la separación de los caracteres.

Lo anterior explicado se puede observar en la siguiente parte del código python:

```
cap = cv2.VideoCapture(0)

    if not cap.isOpened():
```

```
        print("No se puede abrir la cámara")
        exit()

    # Tomamos la referencia de tiempo al iniciar la cámara
    start_time = time.time()
    capture_delay = 7  # Segundos antes de capturar automáticamente

while True:
        ret, frame = cap.read()
        if not ret:
            break

        # Obtener dimensiones del frame
        height, width = frame.shape[:2]

        # Definir coordenadas del rectángulo (centrado)
        rect_width = int(width * 0.6)
        rect_height = int(height * 0.3)
        rect_x = int((width - rect_width) / 2)
        rect_y = int((height - rect_height) / 2)
        rect_top_left = (rect_x, rect_y)
        rect_bottom_right = (rect_x + rect_width, rect_y + rect_height)

        # Dibujar el rectángulo en el frame (color verde, grosor 3)
        cv2.rectangle(frame, rect_top_left, rect_bottom_right, (0, 255, 0),
3)

        # Añadir texto del título y mensaje de instrucción
        cv2.putText(frame, "Math Resolver", (50, 40),
cv2.FONT_HERSHEY_DUPLEX, 1.3, (0, 255, 0), 2)
        cv2.putText(frame, "Alinea tu ecuación dentro del recuadro", (50,
80), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (61, 128, 0), 2)

        # Mostrar el frame
        cv2.imshow('Captura - Math Resolver', frame)

        # Comprobar si han pasado los 7 segundos
        if time.time() - start_time >= capture_delay:
            # Capturar la imagen dentro del rectángulo automáticamente
            roi = frame[rect_y:rect_y+rect_height,
rect_x:rect_x+rect_width]
            cap.release()
            cv2.destroyAllWindows()
            # Procesar la imagen capturada
            print("Imagen capturada y procesada automáticamente.")
            break
```

Ч.

```
        # Pulsar 'q' para abortar la captura
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    cap.release()
```

Después de haber sido capturado el frame se llama a la función **extract_characters_from_image().**

Lo primero que hace esta función es verificar que el directorio especificado (caracteres_separados) existe. Si no existe, lo crea; en caso contrario, elimina todas las imágenes anteriores en él para así garantizar que el procesamiento sea limpio. A continuación, a partir de la imagen recibida se hace una copia de ésta, la cual se convierte a escala de grises, se le aplica un redimensionamiento de 500x220 píxeles, luego se le aplica un filtro Gaussiano para suavizar la imagen y reducir el ruido existente y se le ajusta el brillo y el contraste. Tras esto, la imagen en escala de grises se binariza mediante un umbral adaptativo, esto permite que que los caracteres queden en blanco mientras el fondo en negro. Asimismo, para evitar bordes no deseados, se eliminan manualmente áreas específicas alrededor de los límites de la imagen, rellenándolas con negro.

Luego, se detectan los contornos de la imagen binarizada usando la función findcontours(), la cual nos devuelve los contornos encontrados y su jerarquía. En este punto, se guardan los contornos válidos que tengan dimensiones mayores a unos valores, ya que probablemente correspondan a ruido. Usando la jerarquía, nos quedamos solamente con aquellos contornos exteriores, puesto que para los números 0,6,8 y 9 queremos solamente el contorno y no lo que está encerrado.

Una vez identificados los contornos válidos, estos se ordenan de izquierda a derecha según su posición horizontal, asegurando de esta manera que se procesen en el orden en que aparecen visualmente.

Para poder discernir los caracteres negros y rojos, se utilliza un modelo de color HSV que permite definir rangos específicos para identificar tonos de rojo. Se establecen dos rangos de color rojo (inferior y superior) para capturar diferentes tonalidades. Para cada contorno válido se obtiene su rectángulo delimitador y con los valores de un centro y dimensiones se recorta cada contorno tanto de la imagen binarizada como de la original redimensionada, y la región correspondiente se convierte al espacio de color

HSV. A partir de esto, se genera una máscara que capta los píxeles rojos en el carácter recortado. Luego, el programa compara el número de píxeles rojos con el total de píxeles blancos del carácter, y si más del 50% son rojos, se clasifica como tal. Los resultados se almacenan en un vector que indica si cada carácter es rojo o no, se trata de un vector booleano, True si el elemento es rojo y False si es negro. Asimismo, cada una de las imágenes recortadas de los caracteres se guardan en un vector según su orden de izquierda a derecha de aparición.

Finalmente, las imágenes de los caracteres extraídos se guardan en el directorio de salida,

En el siguiente fragmento de código podemos observar la implementación completa de la función **extract_characters_from_image()**.

```
def extract_characters_from_image(img, output_dir):
    # Crear el directorio de salida si no existe
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)
    else:
        # Eliminar todos los archivos en el directorio de salida
        files = glob.glob(os.path.join(output_dir, '*'))
        for f in files:
            os.remove(f)

    # Convertir a escala de grises
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Redimensionamos
    gray = cv2.resize(gray, (500, 220))
    # tambien la imagen original
    img_resized = cv2.resize(img, (500, 220))

    gray = cv2.GaussianBlur(gray, (3, 3), 0)
    gray = cv2.convertScaleAbs(gray, alpha=1.2, beta=10)


    thresh = cv2.adaptiveThreshold(gray, 255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                   cv2.THRESH_BINARY_INV, 11, 2)

    altura, ancho = thresh.shape[0:2]
    # Limpiar bordes no deseados
    thresh[0:altura, 0:(ancho//40)] = 0
    thresh[0:altura, (ancho-ancho//40):ancho] = 0
    thresh[0:(altura//28), 0:ancho] = 0
```

```
    thresh[(altura-altura//28):altura, 0:ancho] = 0

    contours, hierarchy = cv2.findContours(thresh, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)

    clean_mask = np.zeros_like(thresh)
    min_area = 50

    for i, contour in enumerate(contours):
        area = cv2.contourArea(contour)
        if area > min_area:
            if hierarchy[0][i][3] == -1:
                cv2.drawContours(clean_mask, [contour], -1, (255),
thickness=cv2.FILLED)
            else:
                cv2.drawContours(clean_mask, [contour], -1, (0),
thickness=cv2.FILLED)

    contours, _ = cv2.findContours(clean_mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

    # Ordenar los contornos de izquierda a derecha
    sorted_contours = sorted(contours, key=lambda c:
cv2.boundingRect(c)[0])

    # Definir rango para el color rojo en HSV
# Rango inferior
    lower_red1 = np.array([0, 30, 20])
    upper_red1 = np.array([10, 255, 255])

    # Rango superior
    lower_red2 = np.array([160, 30, 20])
    upper_red2 = np.array([180, 255, 255])

    index = 0
    for c in sorted_contours:
        x, y, w, h = cv2.boundingRect(c)
        # Ignorar contornos pequeños que pueden ser ruido
        if w > 5 and h > 5:
            # Recortar el carácter de la imagen umbralizada
            char_img = clean_mask[y:y+h, x:x+w]

            # Recortamos la región correspondiente en la imagen
redimensionada original
            char_region = img_resized[y:y+h, x:x+w]
```

7.

```
            # Convertir a HSV la región del carácter en la imagen original
redimensionada
            hsv_char = cv2.cvtColor(char_region, cv2.COLOR_BGR2HSV)
            mask1 = cv2.inRange(hsv_char, lower_red1, upper_red1)
            mask2 = cv2.inRange(hsv_char, lower_red2, upper_red2)
            red_mask_char = cv2.bitwise_or(mask1, mask2)

            # Contar píxeles del carácter (blancos en char_img) y cuántos
son rojos
            char_pixels = np.sum(char_img == 255)
            red_pixels = np.sum((char_img == 255) & (red_mask_char == 255))

            #aqui determinamos si es rojo (más del 50% de píxeles del
caracter son rojos)
            is_red = (red_pixels / char_pixels > 0.5) if char_pixels > 0
else False
            vector_colors.append(is_red)

            # Guardar la imagen del carácter
            char_filename = os.path.join(output_dir, f'char_{index}.png')
            cv2.imwrite(char_filename, char_img)
            index += 1
            vector_images.append(char_img)
```

Procesado de símbolos y números

Cabe aclarar que la separación de números y símbolos ha sido una opción tomada debido a la falta de base de datos en torno a símbolos matemáticos por lo que era prácticamente imposible usar el mismo modelo de reconocimiento para símbolos y números. Aclarado esto continuamos con el procesado.

Una vez obtenida las distintas imágenes de símbolos y números se pasará al procesado de las mismas. Para ello se emplean dos funciones principales que se complementan con otras secundarias.

La separación de símbolos y números se hace midiendo el nivel de rojo en la imagen como se puede observar en el código completo pero es algo menor por lo que pasamos directamente al procesado de las partes.

Para el procesamiento de símbolos tenemos la función **“preprocess_symbol()”**:

```
def preprocess_symbol(img):
    """
    Preprocesa una imagen para que sea compatible con el modelo SVM.
```

```
    - Mantiene las proporciones del símbolo al redimensionarlo.
    - Si el símbolo es más alto que ancho, ajusta dinámicamente su ancho
para evitar que se comprima demasiado.
    - Centra el símbolo en un lienzo de 28x28 píxeles con un fondo blanco.
    """
    # Invertir colores para garantizar que el fondo sea blanco y el símbolo
negro
    _, thresh = cv2.threshold(img, 128, 255, cv2.THRESH_BINARY_INV)

    # Detectar contornos para recortar la región del símbolo
    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

    if contours:
        # Encontrar el rectángulo delimitador que rodea todos los contornos
        x_min, y_min, x_max, y_max = float('inf'), float('inf'), 0, 0
        for c in contours:
            x, y, w, h = cv2.boundingRect(c)
            x_min, y_min = min(x_min, x), min(y_min, y)
            x_max, y_max = max(x_max, x + w), max(y_max, y + h)

        cropped_img = thresh[y_min:y_max, x_min:x_max]
    else:
        cropped_img = thresh  # Si no hay contornos, usar la imagen
binarizada

    # Obtener las dimensiones originales del símbolo
    h, w = cropped_img.shape

    # Ajustar proporciones para evitar compresión excesiva
    if h > w:  # Si el símbolo es más alto que ancho
        new_h = 20
        new_w = max(int(w * (20 / h)), 10)  # Asegurarse de que el ancho
sea razonable
    else:
        new_w = 20
        new_h = max(int(h * (20 / w)), 10)  # Asegurarse de que la altura
sea razonable

    img_resized = cv2.resize(cropped_img, (new_w, new_h),
interpolation=cv2.INTER_AREA)

    # Crear un lienzo blanco de 28x28
    img_with_borders = np.full((28, 28), 255, dtype=np.uint8)

    # Calcular las coordenadas para centrar el símbolo en el lienzo
```

```
    x_offset = (28 - new_w) // 2
    y_offset = (28 - new_h) // 2
    img_with_borders[y_offset:y_offset+new_h, x_offset:x_offset+new_w] =
img_resized

    return img_with_borders
```

Esta función recibe la imagen de los símbolos y la invierte para priorizar el fondo blanco y el símbolo negro. Posteriormente detecta los contornos del símbolo y recorta la imagen para que se ajuste a los contornos. Si no detecta contornos usa la imagen binarizada. Luego, usando las dimensiones originales del símbolo reescala la imagen para asegurarse que no haya una compresión excesiva y finalmente centra el símbolo en la imagen.

Respecto al procesamiento de dígitos tenemos la siguiente función llamada **"preprocess_digit()"**:

```
def preprocess_digit(img):

    img = centrar_digito_ajustado(img)

    # Añadir margen blanco antes de cualquier inversión de color
    img = cv2.resize(img, (20, 20))  # Escalar el dígito a 20x20
    padded_img = np.full((28, 28), 255, dtype=np.uint8)  # Crear una imagen
de 28x28 con fondo blanco
    padded_img[4:24, 4:24] = img  # Centrar el dígito en la imagen

    # Normalizar para el modelo
    img_normalized = (255 - padded_img) / 255.0  # Invertir colores para el
modelo (dígito negro sobre fondo blanco)
    img_preprocessed = img_normalized.reshape(1, 28, 28, 1)  # Ajustar las
dimensiones
    return padded_img, img_preprocessed
```

Como podemos observar es similar a la función preprocess_symbols(). En este caso para empezar se centra la imagen y se añade un margen blanco. Posteriormente se crea una imagen de 28x28 píxeles y se centra el dígito en dicha imagen. Por último se normaliza la imagen. Para ello se invierten los colores del modelo y se ajustan las dimensiones.

Una vez procesadas las imágenes es hora de identificar qué símbolo o número es en cada caso. Para ello cargamos los modelos de entrenamiento para cada caso.

```
# Cargar los modelos
digit_model = load_model('digit_recognition_model.keras')
symbol_model = joblib.load('svm_model.joblib')
```

Modelos De Reconocimiento

Para el uso del programa definitivo se emplean dos modelos cuyos programas de entrenamiento están adjuntos en el enlace a los códigos. Para poder utilizarlos en el programa previamente se deben ejecutar estos dos programas por separado, para así generar en la misma carpeta los respectivos modelos.

MODELO 1

El primero de ellos se trata de una **Red Neuronal Convolucional (CNN)**, y funciona mediante un dataset bastante conocido llamado **MNIST**. Resumidamente, este tipo de modelos funcionan con una red de neuronas conectadas de distintas formas que van interpretando las imágenes y van ajustando las ponderaciones de cada una de las uniones entre neuronas hasta obtener una buena configuración.

Para obtener una buena configuración hay que intentar ajustar lo mejor posible la cantidad de neuronas, épocas de entrenamiento, y el tipo de capas de neuronas y conexiones. En nuestro caso concreto utilizamos la siguiente configuración:

```
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28,
1)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
```

Utilizamos principalmente 3 capas (sin contar la Flatten()) :

- Capa convolucional: Se encarga de aplicar filtros en la imagen para obtener características ya sean más concretas o más generales.
- Capa MaxPooling: Se encarga de reducir el espacio de trabajo de las siguientes capas, es decir, reduce el tamaño de la imagen a la mitad (en este caso).

- Capa Densa: Esta capa trabaja con vectores unidimensionales, por tanto antes de esta tiene que haber una flatten() para modificar las imágenes 2D a 1D. Estas capas están ampliamente conectadas y ya trabajan con características y descriptores avanzados, por tanto sirven para clasificar cosas más específicas.

Finalmente, la última capa tiene solo 10 neuronas que corresponden a las salidas. El resultado de la predicción del modelo corresponderá a la neurona con más alta ponderación de estas 10.

MODELO 2

El segundo modelo utilizado se trata de un **SVM (Support Vector Machine)**, el cuál trabaja de manera distinta a la **CNN** empleada anteriormente. Inicialmente se iba a utilizar también una CNN para el reconocimiento de símbolos y así poder juntarlo todo en la misma, pero debido a la carencia de un amplio dataset de estos, se decidió finalmente usar un **SVM** que fuera capaz de distinguir mejor los símbolos sin necesidad de tener un aprendizaje con un gran dataset.

Básicamente, un **SVM** trata de aproximar una frontera espacial (en este caso solo superficial) entre las distintas clases que se desean separar. Obtiene patrones y datos más estadísticos, a diferencia de la red neuronal que es más aleatoriedad y prueba y error.

Finalmente, cuando se terminó de entrenar, era capaz de distinguir correctamente la mayoría de símbolos, dando la mayoría de problemas el símbolo del **“+”** y la **“x”**.

Resolución de la expresión

La última etapa es la resolución de la expresión obtenida, para ello, se ha elaborado el siguiente código, haciendo uso de la librería sympy:

```
import re
import sympy as sp
import math as mp

def preprocesar_ecuacion(ecuacion):     # 3x debe pasarse como 3*x, con los
parentesis igual

    i = 0
    aux = ""
    while(i < len(ecuacion)):
        if (ecuacion[i] in ('x(') and ecuacion[i-1].isdigit()):
            aux += '*'
```

```
        aux += ecuacion[i]
        i += 1
    return ''.join(aux)


def cambiar_signos_der (ecuacion):

    changed_signos = ""
    in_parentesis = False
    i = 0

    while(i < len(ecuacion)):

        if (ecuacion[i] == '('):    # Se abre parentesis
            in_parentesis = True
            changed_signos += '('

        elif (ecuacion[i] == ')'):    # Se cierra el parentesis
            in_parentesis = False
            changed_signos += ')'

        else:
            if (not in_parentesis):
                if (ecuacion[i] == '+'):
                    changed_signos += '-'
                elif (ecuacion[i] == '-'):
                    changed_signos += '+'
                else:
                    changed_signos += ecuacion[i]
            else:
                changed_signos += ecuacion[i]

        i += 1
    return ''.join(changed_signos)


def ecuacion_izq(expresion):

    # Primero sustituimos los espacios (si los hubiera) en blanco de ambos
lados (esto no deberia pasar pq daria problemas entre otras cosas)
    expresion = expresion.replace(" ","")

    # Preprocesamos las multiplicaciones de la ecuacion
    expresion = preprocesar_ecuacion(expresion)
    #print(f'Ecuacion con multiplicaciones: {expresion}')
```

```
    # Dividimos la ecuación en dos partes separadas por el '='

    # Si no hay un igual hace la operacion obtenida
    if '=' not in expresion:
        print("OPCION DETECTADA: OPERACION MATEMÁTICA")
        sol = eval(expresion)
        print(f"Solucion = {sol}")

    else:
        print("OPCION DETECTADA: ECUACION")

        izq, der = expresion.split('=')            # Se separa la ecuacion
por el = y se trabaja independientemente con ambas

        # Comprobamos si el primer número de la der es positivo o negativo
para ponerle el '+' si es positivo
        if der and not der[0] in ('+', '-'):
            der = '+' + der

        # Cambiamos los signos de la derecha ('+' --> '-' y viceversa)
        der = cambiar_signos_der(der)
        #print(f'Derecha cambiada de signo: {der}')

        # Juntamos la ecuacion como si estuviera igualada a 0
        eq = f"{izq}{der}"
        print("Ecuacion inicial igualada a 0:")
        print(f"{eq} = 0")
        sol = res_eq(eq)
        print(f"Solucion = {sol}")


def res_eq (ecuacion):
    # se define la variable simbólica
    x = sp.symbols('x')

    # convertir el string en una expresion simbolcia
    ecuacion = sp.sympify(ecuacion)

    # Se resuelve
    sol = sp.solve(ecuacion, x)

    if sol:
        return sol
    else:
        return "NO SOL"
```

```
expresion = " "

ecuacion_izq(expresion)
```

En primer lugar, se distingue si la expresión es una ecuación o una operación básica, comprobando si hay un signo de igualdad en ella.

Si no hay una igualdad, simplemente se resuelve la operación con la función de python `eval()`, que resuelve la operación recibida como parámetro y muestra la solución por pantalla.

Si la expresión recibida es una ecuación (es decir, si hay una igualdad en ella) se resuelve dicha ecuación con la función , `sp.solve()`. No obstante, dicha función debe recibir la ecuación igualada a 0, y, además, todas las multiplicaciones deben tener su respectivo signo (nx debe representarse como n*x, y con los paréntesis igual), por lo que se debe preparar la expresión antes de resolverla.

Para ello, primero se hace uso de la función `preprocesar_ecuación()` para prepararla en el formato correcto, seguidamente se separa la ecuación en dos (antes y después del igual), `cambiar_signos_der()` para cambiar los signos de la derecha y unirlos a la izquierda, igualándola a 0 y, finalmente, se resuelve y muestra la solución por pantalla.
.

Posibles usos de Math Solver

El objetivo principal del programa está enfocado a la docencia ya que puede ayudar a estudiantes a verificar y comprender la solución de ecuaciones y operaciones matemáticas simples. Además que también puede ser útil para profesores en la corrección de exámenes de forma inmediata.

También, modificando ligeramente el código y ampliando la base de datos del programa se podría usar para el reconocimiento de escritura a mano. Esto es posible ya que, gracias al uso de inteligencia artificial y machine learning el programa es capaz de detectar y aislar tanto símbolos como números, por lo que sería posible ampliarlo con letras y signos de escritura.
Por último se puede incluir en la digitalización de textos o libros matemáticos ya que se especializa en el reconocimiento de símbolos matemáticos y números.

Capacidad de adaptación y mejora

El uso de una base de datos para el entrenamiento de los modelos hace que la capacidad de mejora sea una opción factible y no demasiado costosa. Para ello habría

que ampliar esta base con más imágenes de números y símbolos. Además se pueden incluir nuevos elementos en la base como por ejemplo símbolos como integrales o referentes a conjuntos etc.

También se puede mejorar la detección de bloques matemáticos en diversos entornos usando algoritmos como YOLO o Detectron para detectar regiones específicas de ecuaciones en imágenes más grandes como una pizarra o un documento.

Otra pequeña mejora posible sería el reconocimiento de unidades físicas como metros, segundos, kg etc. Esto haría posible la especificación dentro de problemas y la detección de errores en los mismos.

Conclusión

Habiendo visto el funcionamiento del código y de los modelos usados para el reconocimiento de números y símbolos podemos sentenciar que el programa funciona correctamente y devuelve los resultados esperados para ecuaciones simples incluyendo números negativos y fracciones.

Sin embargo aún es un proyecto “sencillo” comparado con aplicaciones como Photomath debido a las posibilidades de operaciones que se pueden realizar y la falta de interfaz para el usuario.

Por lo tanto podemos decir que, aunque ahora mismo el programa no es óptimo, es completamente funcional y tiene una gran capacidad de mejora. También cabe recalcar que mediante pequeñas variaciones este puede ser usado para diversas aplicaciones en varios ámbitos. Estamos seguros de que si se obtuviera más datasets de calidad con más símbolos matemáticos, probablemente nuestra aplicación podría realizar muchas más operaciones.

En conclusión el proyecto ha servido tanto como aprendizaje como reto. Surgiendo varios problemas internos en el código en torno a la detección de las operaciones y siendo superados con satisfacción aprendiendo en el proceso.