

**Diseño e Implementación de un Sistema de Reconocimiento de Números en Tiempo
Real Utilizando Visión por Computadora**

Isabella Ardila M. , Laura C. García G. Y Andrea C. Terraza S.

Facultad de Ciencias Exactas e Ingeniería, Universidad Sergio Arboleda

Ciencias de la Computación e Inteligencia Artificial

G01: Machine Learning

Prof. Ricardo Andres Fonseca Perdomo

12 de Junio de 2024



Índice

1.	Índice	2
2.	Introducción	4
3.	Objetivos	5
	3.1. Objetivo General	5
	3.2. Objetivos Específicos	5
4.	Aplicaciones en la Cotidianidad del Sistema de Reconocimiento de Números en Tiempo Real	6
5.	Desarrollo	7
	5.1. Preprocesamiento.....	7
	5.1.1. Módulo RoiRecognition.py.....	7
	5.1.2. Módulo TraceRecognition.py.....	8
	5.2. Selección del Algoritmo para Entrenamiento.....	8
	5.2.1. Justificación de la Selección de SVM.....	9
	5.2.2. Proceso de Ajuste de Hiperparámetros	9
	5.2.3. Resultados de la Búsqueda de Hiperparámetros	9
	5.2.4. Elección del Kernel Polinomial	10
	5.3. Cálculos	11
	5.3.1. Función de Decisión del SVM	11
	5.3.2. Función Kernel	12
	5.3.3. Parámetros del SVM	12
	5.3.4. Justificación de los Parámetros del Modelo	13
	5.3.5. Análisis Matemático	14

5.4. Límites de Decisión	14
5.5. Mejor área de aprendizaje	17
5.6. Métricas.....	18
5.7. Gráficas	20
5.8. Optimización	26
5.9. Pseudocódigo	27
5.9.1. Clase RoiRecognition	27
5.9.2. Clase TraceRecognition	29
5.9.3. Clase modelSVMpca	31
5.10. Diagrama de Flujo de Funcionamiento	32
5.11. Códigos	34
5.11.1. Clase RoiRecognition	34
5.11.2. Clase TraceRecognition	39
5.11.3. Clase modelSVMpca	42
5.12. Pruebas	43
5.12.1. Pruebas Unitarias	44
5.12.2. Pruebas Manuales	45
6. Conclusiones	49
7. Referencias	51

Introducción

El reconocimiento de números en tiempo real es un desafío relevante en el campo de la visión por computadora, con aplicaciones prácticas en diversas áreas. Este proyecto tiene como objetivo diseñar un sistema capaz de identificar números del 0 al 9 utilizando una cámara, permitiendo el reconocimiento tanto de números escritos como de aquellos dibujados en el aire con un dedo. Para ello, se emplearán técnicas avanzadas de aprendizaje automático y procesamiento de imágenes, asegurando que el sistema pueda aprender y mejorar su precisión en ambos escenarios.

El desarrollo de este sistema abarca varias etapas críticas, incluyendo la recopilación y preprocesamiento de datos, la selección del algoritmo de entrenamiento adecuado, y la optimización del rendimiento del sistema. Además, se analizarán las métricas de evaluación y se presentarán pruebas que demuestren la eficacia del sistema. Este documento ofrecerá una visión integral del proceso de creación y las aplicaciones potenciales del sistema en la vida cotidiana, reflejando el aprendizaje adquirido a lo largo del curso.

Objetivos

Objetivo General

Desarrollar un sistema de reconocimiento de números del 0 al 9 en tiempo real utilizando visión por computadora, capaz de identificar números tanto escritos como dibujados en el aire, mediante el uso de algoritmos de aprendizaje automático y procesamiento de imágenes.

Objetivos Específicos

- Diseñar e implementar un sistema que capture la imagen en tiempo real utilizando una cámara web y procese esta imagen para identificar números.
- Entrenar un modelo de Máquina de Soporte Vectorial (SVM), utilizando el dataset preprocesado y optimizar sus parámetros para mejorar la precisión y el rendimiento.
- Optimizar el modelo mediante la reducción de dimensionalidad utilizando Análisis de Componentes Principales (PCA).
- Desarrollar métodos para convertir las imágenes capturadas a escala de grises, aplicar filtros de suavizado y técnicas de binarización para mejorar la calidad de la imagen.
- Implementar el algoritmo entrenado para procesar las imágenes capturadas y reconocer los números en tiempo real.
- Definir y aplicar métricas de evaluación para medir la precisión y eficiencia del sistema.
- Realizar pruebas de precisión y robustez del sistema en condiciones variables de iluminación y representación de números.

Aplicaciones en la Cotidianidad del Sistema de Reconocimiento de Números en Tiempo Real

El sistema de reconocimiento de números en tiempo real tiene un amplio rango de aplicaciones prácticas que pueden transformar diversas áreas de la vida cotidiana. Este sistema puede integrarse en dispositivos electrónicos para mejorar la accesibilidad y la usabilidad, permitiendo a los usuarios ingresar números mediante gestos. En dispositivos móviles y sistemas de realidad aumentada, puede servir como un método alternativo de control y entrada de datos, facilitando una interacción más natural con la tecnología. Además, esta tecnología es especialmente útil para personas con discapacidades, proporcionando una forma accesible y eficiente de interactuar con dispositivos y software.

En entornos educativos, el sistema puede utilizarse para desarrollar aplicaciones interactivas que ayuden a los niños a aprender y practicar números de manera divertida y atractiva. Además, permite a los estudiantes verificar su trabajo en tiempo real, proporcionando retroalimentación instantánea en ejercicios de matemáticas y escritura de números.

El sistema también puede implementarse en mecanismos de seguridad para la autenticación por gestos, permitiendo ingresar códigos numéricos mediante movimientos en el aire. Esto mejora tanto la seguridad como la usabilidad en entornos sensibles. También puede aplicarse en sistemas de control de acceso, proporcionando una capa adicional de seguridad sin necesidad de contacto físico.

En el ámbito del entretenimiento y la realidad virtual, el sistema puede utilizarse para desarrollar juegos interactivos donde los usuarios dibujen números en el aire para interactuar con el entorno del juego. En aplicaciones de realidad virtual, permite crear experiencias más inmersivas y naturales, facilitando la interacción con elementos virtuales mediante gestos numéricos.

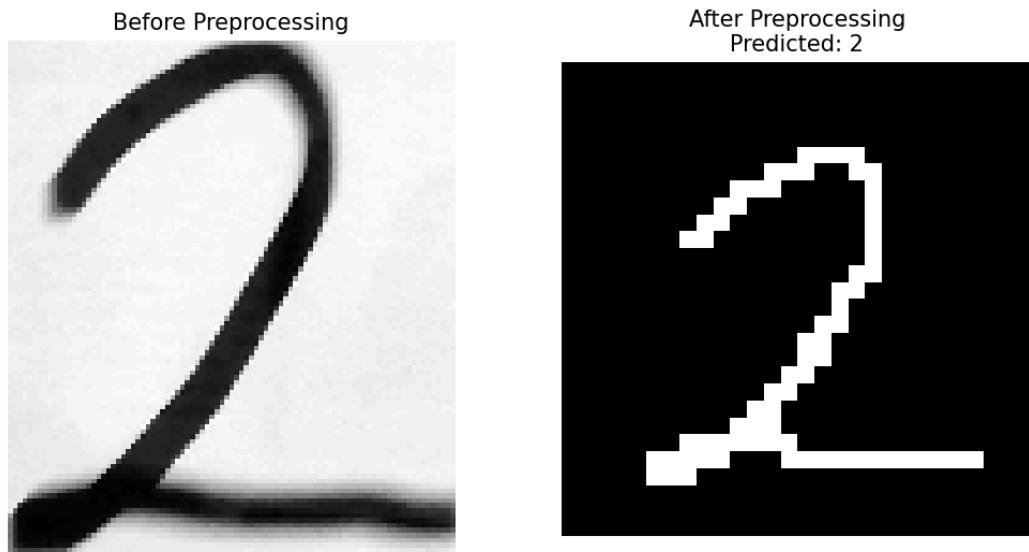
En resumen, el sistema de reconocimiento de números en tiempo real tiene aplicaciones prácticas significativas en la interacción humano-computadora, la educación, la seguridad y el entretenimiento, ofreciendo soluciones innovadoras que mejoran la accesibilidad, la seguridad y la experiencia del usuario.

Desarrollo

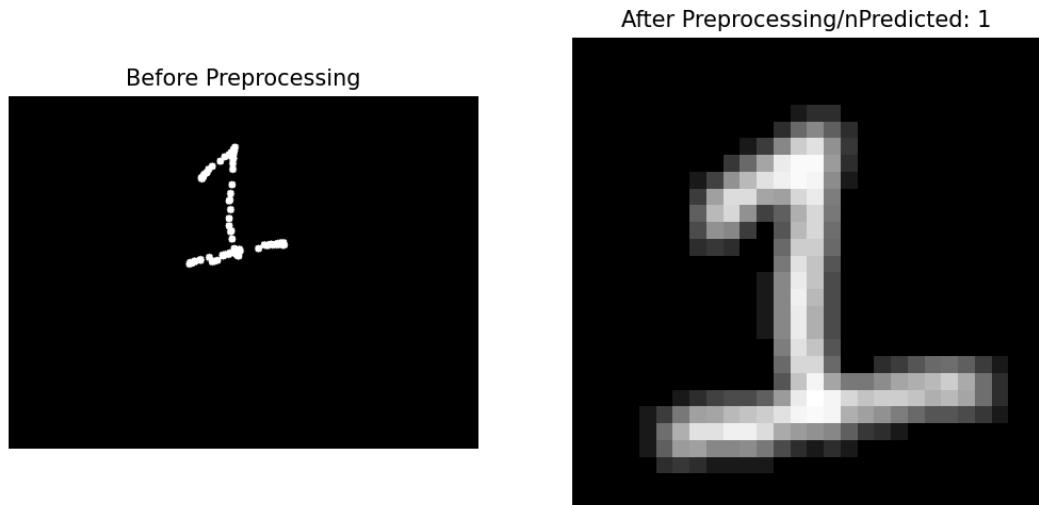
Preprocesamiento

En el proyecto de reconocimiento de números en tiempo real, se desarrollaron dos módulos principales: RoiRecognition.py y TraceRecognition.py, cada uno enfocado en el reconocimiento de números escritos y dibujados en el aire, respectivamente.

En RoiRecognition.py, el preprocesamiento se centró en la preparación de imágenes de regiones de interés (ROI) que contienen dígitos escritos a mano para su clasificación. Los pasos incluyen la conversión de las imágenes a escala de grises, suavizado y umbralización binaria para obtener imágenes claras de los dígitos. Posteriormente, se selecciona la ROI y se procesa, lo que incluye la inversión de colores, recorte y redimensionamiento. Estos pasos aseguran que las imágenes sean uniformes y adecuadas para el modelo de clasificación.



Por otro lado, en TraceRecognition.py, el enfoque está en la preparación de trayectorias de movimiento de la punta del dedo para su clasificación. Se utilizan técnicas de suavizado e interpolación sobre las coordenadas de la punta del dedo, seguidas de un proceso de preprocesamiento de imagen que incluye recorte, ajuste de tamaño, morfología y mejora del contraste. Estos métodos garantizan que las trayectorias dibujadas en el aire se conviertan en imágenes claras y coherentes para su reconocimiento.



Ambos módulos utilizan un clasificador SVM con un kernel polinomial debido a su capacidad para manejar datos de alta dimensionalidad y capturar relaciones no lineales entre las características, lo que lo hace adecuado para la clasificación de dígitos escritos a mano.

El preprocessamiento del dataset MNIST también es crucial. Se carga y preprocesa normalizando las imágenes a escala de grises y redimensionándolas a 28 x 28 píxeles. Esto asegura que las imágenes tengan una consistencia que facilite el entrenamiento del modelo. Además, se utiliza el Análisis de Componentes Principales (PCA) para reducir la dimensionalidad de los datos, mejorando la eficiencia y precisión del modelo SVM. La importancia del preprocessamiento radica en su capacidad para mejorar la precisión y eficacia del sistema de reconocimiento de números, asegurando que las imágenes sean claras y adecuadas para la clasificación.



Selección del algoritmo para entrenamiento

En el desarrollo del sistema de reconocimiento de números en tiempo real, la elección del algoritmo de aprendizaje automático adecuado es crucial para asegurar la precisión y eficiencia del modelo. Para este proyecto, se decidió utilizar Máquinas de Soporte Vectorial

(SVM) debido a su eficacia comprobada en problemas de clasificación de alta dimensionalidad y su capacidad para manejar conjuntos de datos complejos.

Justificación de la Selección de SVM

Las Máquinas de Soporte Vectorial son una opción popular para tareas de clasificación debido a su capacidad para encontrar un margen óptimo de separación entre las diferentes clases. Este margen máximo minimiza el error de clasificación en datos no vistos, haciendo que los SVM sean robustos y efectivos para diversas aplicaciones. Además, los SVM pueden trabajar con diferentes tipos de kernels que permiten transformar los datos y encontrar patrones más complejos.

Proceso de Ajuste de Hiperparámetros

Para determinar los mejores hiperparámetros y el tipo de kernel más adecuado, se utilizó GridSearchCV, una herramienta que permite realizar una búsqueda exhaustiva sobre un espacio especificado de hiperparámetros. El objetivo era encontrar la combinación óptima de parámetros que maximizan la precisión del modelo. El conjunto de hiperparámetros evaluados incluyó diferentes valores de C, gamma y tipos de kernel (linear, rbf, poly).

```
param_grid = {
    'C': [1, 10],
    'gamma': ['scale', 'auto'],
    'kernel': ['linear', 'rbf', 'poly']
}
```

Resultados de la Búsqueda de Hiperparámetros

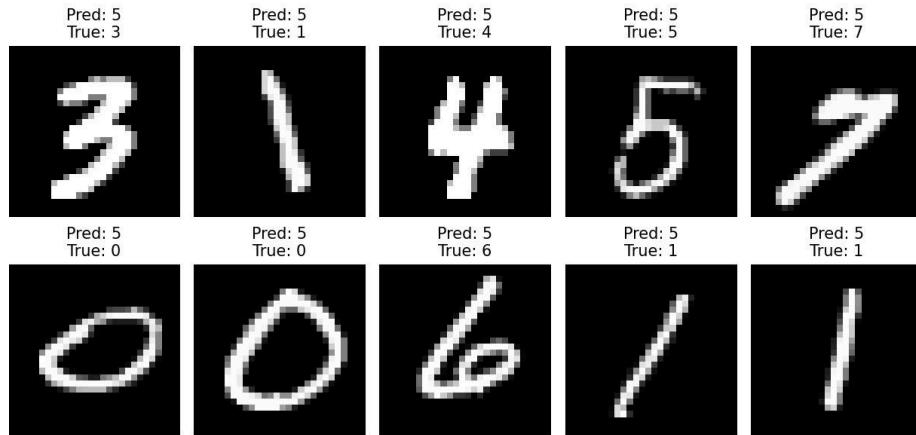
El GridSearchCV identificó que los mejores hiperparámetros para nuestro modelo SVM eran {'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}, con una puntuación de 0.98193 y una precisión de 0.9837. Sin embargo, durante las pruebas con datos reales, el modelo con kernel RBF mostró una tendencia a predecir incorrectamente el mismo valor (5) para diversas entradas, lo cual sugería un posible sobreajuste a los datos de entrenamiento.

Mejores hiperparámetros: {'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}

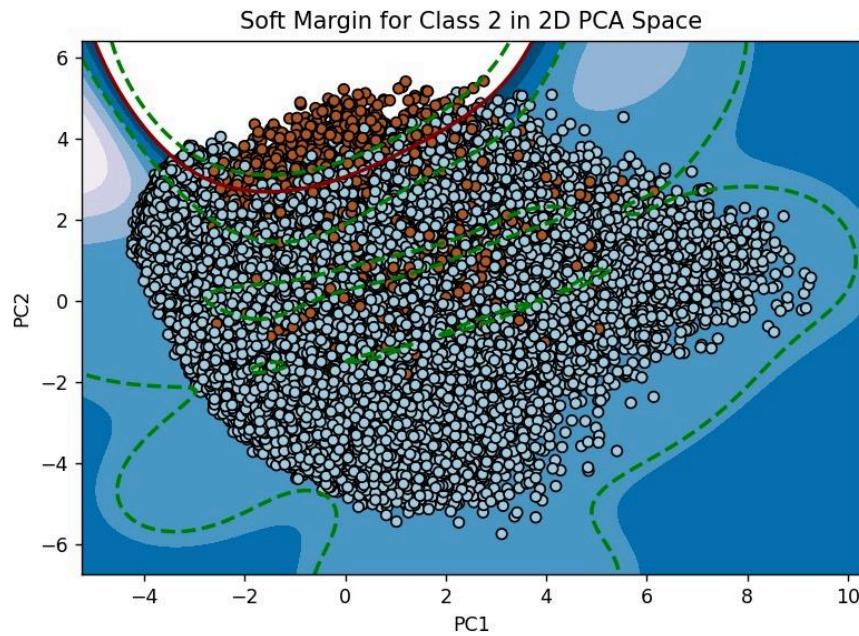
Mejor puntuación: 0.9819333333333333

Accuracy: 0.9837

Prueba de que con el modelo con Kernel rbf no clasificaba correctamente:



Prueba de que los límites de decisión estaban llegando a extremos que no deberían entrar para clasificar, lo que significa un posible sobreajuste:



Elección del Kernel Polinomial

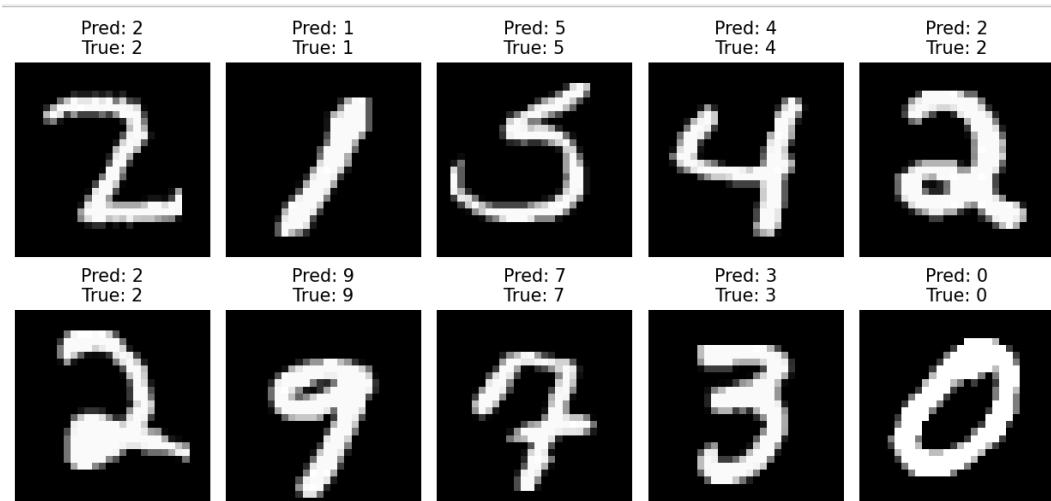
Ante la observación de que el kernel RBF no generaliza bien en pruebas con datos reales, se decidió probar el modelo con un kernel polinomial. El kernel polinomial tiene la ventaja de capturar relaciones más complejas entre las características sin sobreajustarse a los datos de entrenamiento. Al probar con este modelo, se observó una mejora significativa en la capacidad de predicción de números reales, mostrando un mejor ajuste y precisión en la identificación de los dígitos.

El modelo final se entrenó con los siguientes parámetros:

```
svm_3D = SVC(kernel='poly', C=10, gamma='scale')
svm_3D.fit(X_train_3D, y_train)
```

Este enfoque permitió que el modelo se ajustara mejor a los datos, proporcionando predicciones más precisas y confiables. La selección del kernel polinomial fue clave para lograr un equilibrio entre la complejidad del modelo y su capacidad de generalización, asegurando un rendimiento robusto y preciso en el reconocimiento de números en tiempo real.

Prueba de que con el kernel polinomial clasifica correctamente:



Cálculos

1. Función de Decisión del SVM

En un SVM, la función de decisión tiene la siguiente forma general:

$$f(x) = \text{sign} \left(\sum_{i=1}^N \alpha_i y_i K(x_i, x) + b \right)$$

Donde:

- x es el vector de características de la muestra a clasificar.

- x_i son los vectores de características de las muestras de entrenamiento.
- α_i son los multiplicadores de Lagrange.
- y_i son las etiquetas de las muestras de entrenamiento.
- $K(x_i, x)$ es la función kernel.
- b es el término de sesgo.
- N es el número de muestras de entrenamiento.

2. Función Kernel

Hemos elegido un kernel polinomial, cuya forma es:

$$K(x_i, x) = (\gamma \langle x_i, x \rangle + r)^d$$

Donde:

- γ es el parámetro de escala.
- $\langle x_i, x \rangle$ es el producto interno entre
- r es el coeficiente libre (coeff).
- d es el grado del polinomio (degree).

3. Parámetros del SVM

- C (Parámetro de Regularización):

El parámetro C controla el equilibrio entre lograr un margen lo más amplio posible y clasificar correctamente todas las muestras de entrenamiento. Matemáticamente, se puede expresar como la penalización por cada punto que queda en el margen o mal clasificado.

- Alto C: Se penalizan fuertemente las clasificaciones incorrectas, lo que puede llevar a un sobreajuste.

- Bajo C: Se permite un margen más amplio, lo que puede resultar en un modelo más generalizable pero con más errores de clasificación en el conjunto de entrenamiento.
- γ (Parámetro de Escala):

El parámetro γ define cómo influye un solo punto de entrenamiento. Valores altos significan que el alcance de la influencia de un solo punto es pequeño, lo que puede llevar a un sobreajuste. Valores bajos significan que los puntos tienen una influencia más amplia, lo que puede llevar a un subajuste.

- degree (Grado del Polinomio):

El grado del polinomio (degree) en el kernel polinomial define el grado del polinomio que se utilizará. Un valor más alto puede modelar una relación más compleja entre las características, pero también puede aumentar el riesgo de sobreajuste.

- coef0 (Coeficiente Libre):

El coeficiente libre (coef0) es un término independiente que se añade al kernel polinomial. Controla la importancia de los términos de mayor grado frente a los términos de menor grado.

Justificación de los Parámetros del Modelo

1. Parámetro C:

Elegimos $C=10$, lo que indica que el modelo clasifica correctamente las muestras de entrenamiento incluso a costa de tener un margen más estrecho. Esto puede ser adecuado si tienes suficientes datos y deseas minimizar los errores en el conjunto de entrenamiento.

2. γ :

Elegimos γ , que ajusta automáticamente el parámetro de acuerdo con la cantidad de características. Este valor es una buena opción predeterminada y asegura que el modelo tenga un buen punto de partida sin necesidad de ajustar manualmente γ .

3. Grado del Polinomio:

El grado del polinomio 3 puede capturar relaciones no lineales complejas sin sobreajustar demasiado el modelo.

Análisis Matemático

Si el kernel polinomial tiene grado 3 ($d=3$), $\gamma = \text{scale}$, y $r=0.0$, la función del kernel sería:

$$K(x_i, x) = (\text{scale} \langle x_i, x \rangle + 0.0)^3$$

```
Coef0 (r): 0.0
Degree (d): 3
Gamma: scale
C: 10
La función del SVM es: (scale * <x, x'> + 0.0)^3
```

Supongamos que después de ajustar el modelo obtenemos:

- $\alpha_i = [0.5, 0.3, 0.2, \dots]$ (Multiplicadores de Lagrange)
- $x_i = [x_1, x_2, x_3, \dots]$ (Vectores de entrenamiento)
- $y_i = [1, -1, 1, \dots]$ (Etiquetas de las muestras de entrenamiento)
- $b = 0.4$ (Término de sesgo)

La función de decisión sería:

$$f(x) = \text{sign} \left(\sum_{i=1}^N \alpha_i y_i (\text{scale} \langle x_i, x \rangle + 0.0)^3 + 0.4 \right)$$

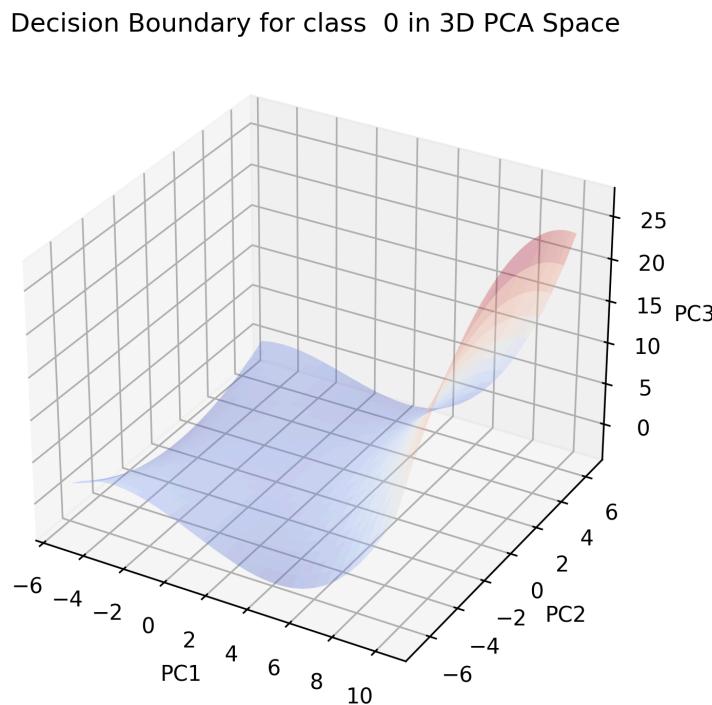
Límites de decisión

En el proyecto de reconocimiento de números en tiempo real, los límites de decisión en 3D proporcionan una visualización avanzada de cómo el modelo SVM clasifica los dígitos. Estos límites son superficies que separan las diferentes clases en el espacio de características. En este caso, el espacio tridimensional se define por tres componentes principales (PC1, PC2 y PC3), que son combinaciones lineales de las características originales reteniendo la mayor parte de la variabilidad en los datos. Estos componentes se obtuvieron aplicando PCA a las imágenes de los dígitos, permitiendo visualizar los datos en un espacio reducido de tres dimensiones y manteniendo la mayor cantidad de información posible.

La visualización tridimensional facilita la comprensión de cómo el modelo SVM maneja decisiones complejas y no lineales. En este espacio reducido, se puede observar cómo el modelo utiliza las relaciones entre los componentes principales para diferenciar entre las clases. Las áreas donde la superficie del límite de decisión cambia de color indican regiones de alta ambigüedad, señalando los puntos donde el modelo tiene más dificultad para clasificar correctamente. Estas áreas pueden ser objetivos clave para futuras mejoras del modelo.

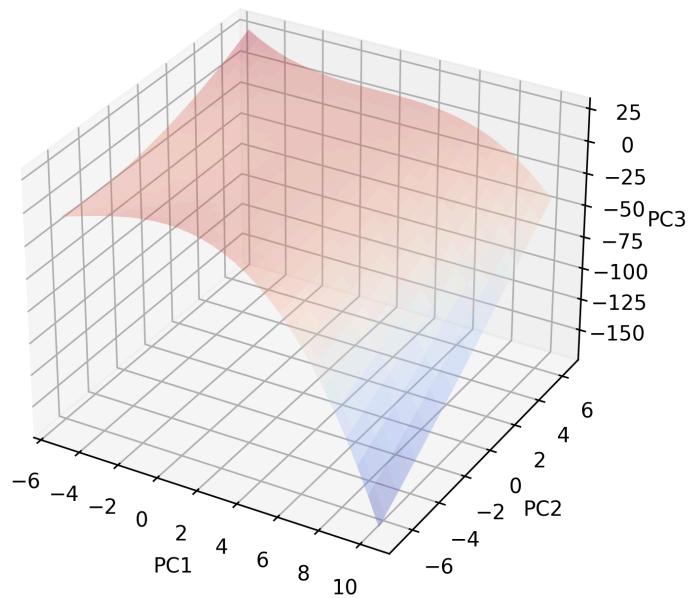
En general, cada gráfico muestra una superficie que representa la región del espacio tridimensional donde el modelo predice que la entrada pertenece a una clase específica. Las áreas de color sólido (principalmente azul) indican regiones de alta confianza donde el modelo clasifica correctamente el dígito correspondiente. Las transiciones de color de azul a rojo marcan las zonas de ambigüedad, donde el modelo cambia su predicción a otra clase. Estas transiciones son críticas ya que reflejan los límites de decisión del modelo, donde la clasificación puede ser menos segura.

Algunos ejemplos de como se ve el límite de decisión en algunos números:



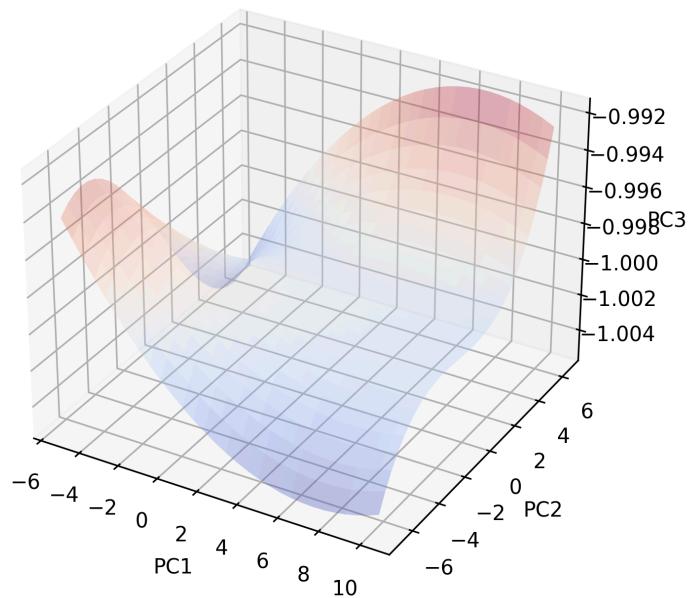
La superficie azul en el gráfico del número '0' indica que el modelo tiene alta confianza en clasificar correctamente este dígito. Las áreas de transición suaves sugieren que el modelo maneja bien las variaciones en los datos de entrada para el '0'.

Decision Boundary for class 1 in 3D PCA Space



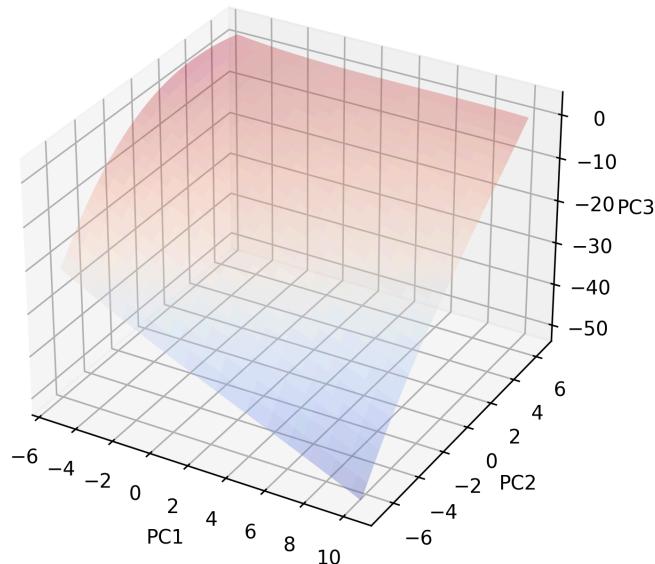
El gráfico del número '1' muestra una superficie de decisión más compleja con transiciones más pronunciadas entre azul y rojo. Esto sugiere que el modelo puede enfrentar más desafíos al clasificar el '1' en comparación con otros dígitos, posiblemente debido a la simplicidad y similitud del '1' con otras formas.

Decision Boundary for class 2 in 3D PCA Space



Para el número '2', la superficie de decisión es notablemente bien definida, con claras áreas de alta confianza en azul. Las transiciones indican una buena generalización del modelo, aunque existen algunas áreas donde la clasificación podría ser mejorada.

Decision Boundary for class 3 in 3D PCA Space



La superficie de decisión para el número '3' muestra una transición suave entre las regiones de confianza. Esto indica que el modelo está bien ajustado para clasificar el '3', aunque existen algunas áreas de ambigüedad que podrían beneficiarse de un ajuste adicional del modelo.

Mejor área de aprendizaje

En el contexto de la clasificación de dígitos, es crucial entender cómo y dónde el modelo toma sus decisiones. Las gráficas de margen suave y los límites de decisión nos brindan una representación visual de estas áreas clave.

En las gráficas, las zonas de alta confianza se identifican por las áreas de colores sólidos (azul oscuro para las clases negativas y marrón claro para las clases positivas). Estas áreas indican donde el modelo está seguro de sus decisiones. Los puntos dentro de estas zonas son clasificados con alta precisión, ya que sus características están claramente definidas y separadas de otras clases.

Las zonas de baja confianza se encuentran cerca de las líneas verdes discontinuas, que representan los márgenes del modelo. Estas regiones son donde el modelo tiene más

dificultad para decidir a qué clase pertenecen los puntos. La ambigüedad en estas zonas es mayor, lo que lleva a una mayor probabilidad de errores de clasificación. Estas áreas reflejan puntos donde las características de los datos son similares entre clases, lo que complica la toma de decisiones del modelo.

El límite de decisión, representado por la línea roja sólida, es la frontera donde el modelo cambia de decidir que un punto pertenece a una clase a decidir que pertenece a otra. Este límite es crucial para la efectividad general del modelo, ya que traza la frontera entre diferentes clases basado en los datos de entrenamiento. La precisión del modelo en esta frontera determina su capacidad para manejar casos de borde, donde las características de los datos son menos claras.

La mejor área de aprendizaje se encuentra dentro de los márgenes, representados por las líneas verdes discontinuas. En esta región, el modelo está "aprendiendo" más activamente porque los puntos de datos están más cercanos entre sí en términos de sus características. Los puntos cerca de estos márgenes son cruciales porque ayudan al modelo a definir mejor dónde colocar el límite de decisión. Entrenar el modelo con datos que se encuentran dentro de estos márgenes permite que el modelo refine su capacidad de discriminación entre clases, mejorando así su precisión general.

Métricas

Reporte de clasificación:				
	precision	recall	f1-score	support
0	0.98	0.99	0.98	980
1	0.98	0.99	0.99	1135
2	0.98	0.97	0.98	1032
3	0.98	0.98	0.98	1010
4	0.98	0.98	0.98	982
5	0.98	0.97	0.97	892
6	0.98	0.98	0.98	958
7	0.98	0.97	0.98	1028
8	0.98	0.98	0.98	974
9	0.98	0.97	0.97	1009
accuracy			0.98	10000
macro avg	0.98	0.98	0.98	10000
weighted avg	0.98	0.98	0.98	10000

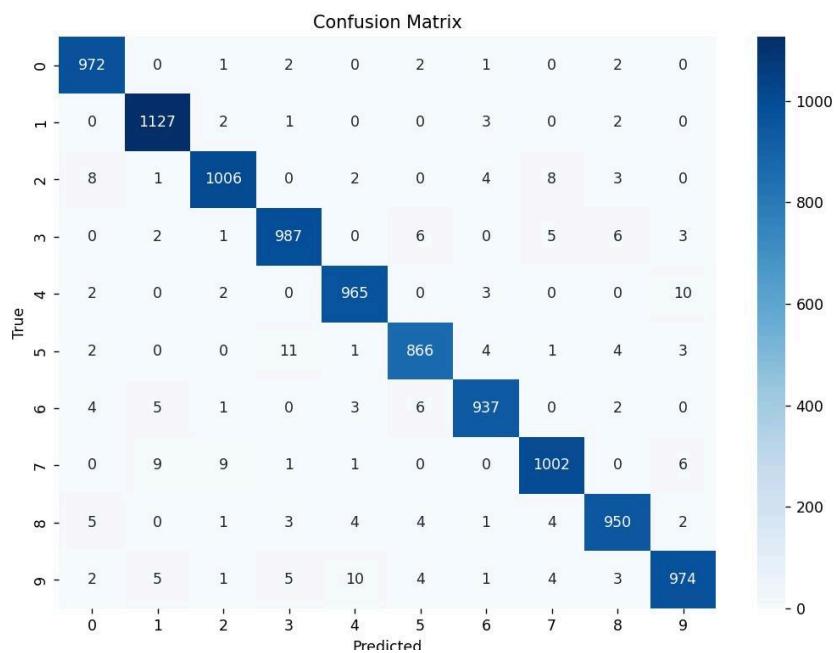
Los resultados del modelo SVM para el reconocimiento de números en tiempo real reflejan un desempeño sobresaliente. La precisión global del modelo es del 98%, lo que indica que el sistema es altamente efectivo en la identificación de los números del 0 al 9. Esta alta

precisión es un indicativo claro de la efectividad de las técnicas de preprocesamiento y la optimización del modelo SVM mediante el Análisis de Componentes Principales (PCA).

Al observar el desempeño por clase, se puede apreciar que para la clase '0', el modelo muestra una precisión del 98%, un recall del 99% y un F1-score de 0.98. Esto significa que el modelo identifica correctamente los ceros con una tasa muy baja de falsos negativos, lo que es crucial para aplicaciones donde la exactitud es vital. Similarmente, la clase '1' presenta una precisión del 98%, un recall del 99% y un F1-score de 0.99, lo que demuestra que el modelo es muy eficaz en la identificación de unos.

El desempeño se mantiene consistentemente alto para otras clases también. El promedio ponderado de la precisión, recall y F1-score es de 0.98 para todas las clases, lo cual indica una consistencia alta en la capacidad del modelo para clasificar correctamente todas las categorías de números. Este nivel de consistencia es esencial para garantizar que el sistema funcione de manera confiable en condiciones diversas y con distintos tipos de datos de entrada.

El soporte para cada clase es equilibrado, con alrededor de 1000 ejemplos por clase en promedio, lo que proporciona una evaluación robusta del rendimiento del modelo. Este amplio soporte ayuda a confirmar la generalización del modelo y su capacidad para mantener un alto desempeño en distintas situaciones.



La matriz de confusión del modelo SVM muestra una alta precisión y recall en la mayoría de las clases, lo que indica que identifica correctamente la mayoría de los números del 0 al 9. La diagonal principal de la matriz, que representa las predicciones correctas, está dominada por valores altos, lo cual es un indicativo positivo del rendimiento del modelo.

La clase '0' tiene una alta tasa de aciertos con 972 ejemplos correctamente clasificados y solo unos pocos errores. De manera similar, la clase '1' muestra un excelente desempeño con 1127 aciertos y un número muy bajo de falsos positivos y falsos negativos. Este patrón de alta precisión se mantiene en las clases '2', '3', '4', '6', '7', '8', y '9', donde los valores correctos son consistentemente altos y las confusiones son mínimas.

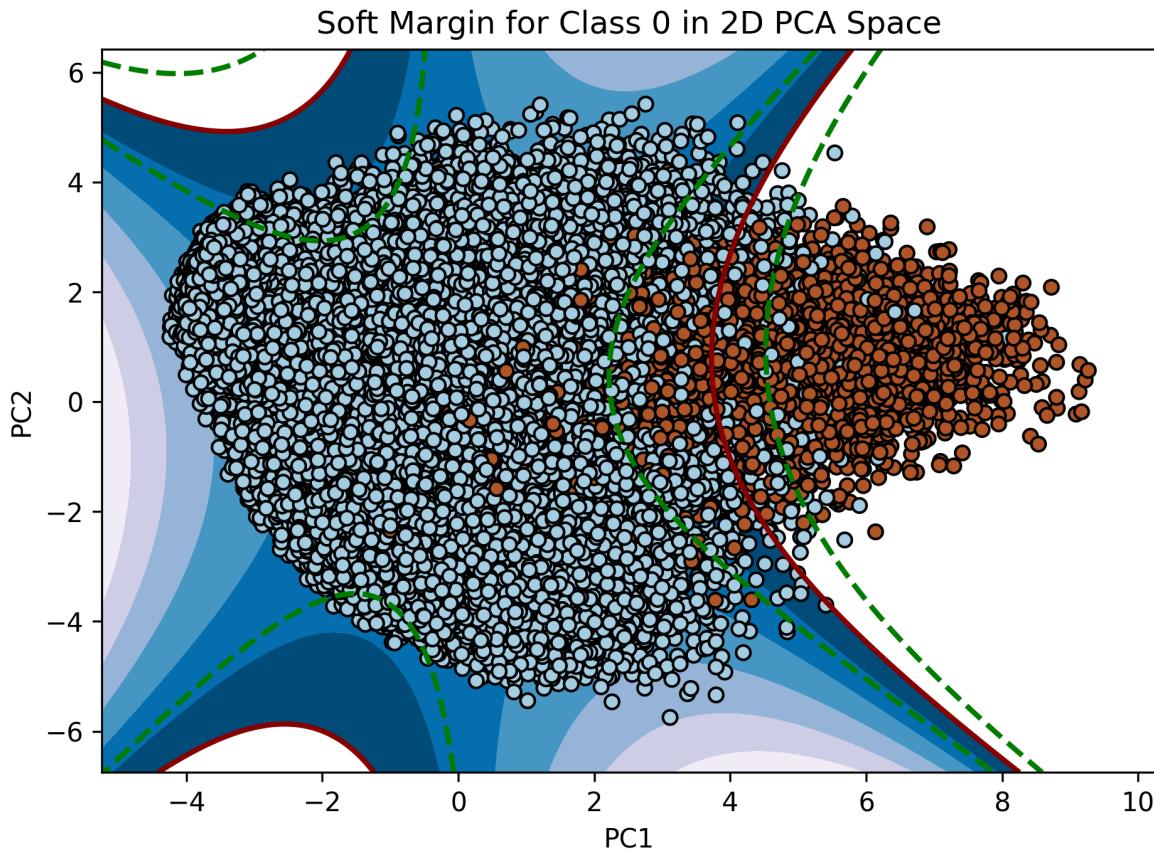
La clase '5' presenta un mayor número de falsos negativos y algunos falsos positivos en comparación con otras clases. Esto sugiere que el modelo tiene más dificultades para clasificar correctamente los dígitos '5', lo que podría requerir ajustes adicionales en el preprocesamiento o en la optimización del modelo para mejorar su rendimiento en esta categoría.

En conjunto, la matriz de confusión muestra que el modelo SVM es altamente efectivo en la clasificación de números, con una precisión general de 98%. Las bajas tasas de error en la mayoría de las clases indican que el modelo es capaz de generalizar bien y mantener un alto rendimiento en distintas situaciones y con diferentes tipos de datos de entrada.

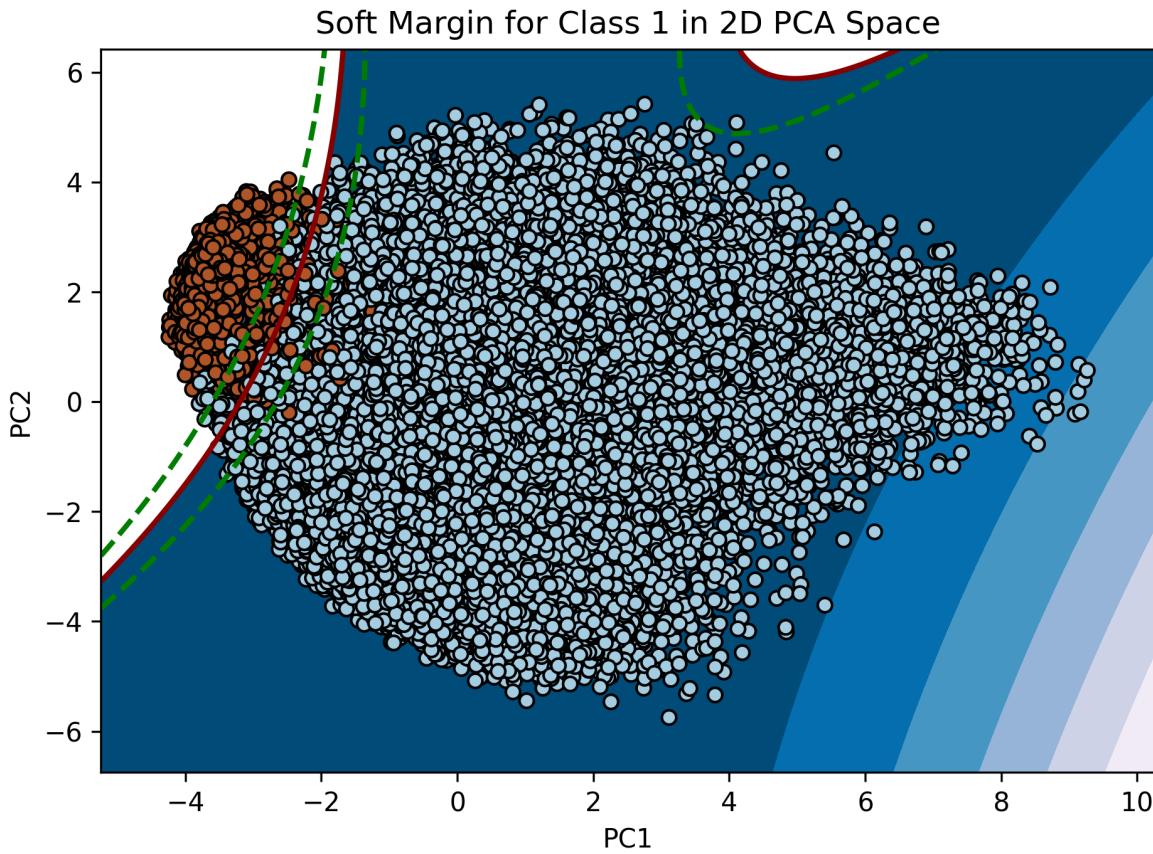
Gráficas

El análisis de los límites de decisión y los márgenes suaves en el espacio 2D, obtenidos mediante PCA, proporciona una visión detallada de cómo el modelo SVM clasifica los diferentes dígitos. Estas visualizaciones ayudan a entender mejor las decisiones del modelo y su comportamiento ante los datos.

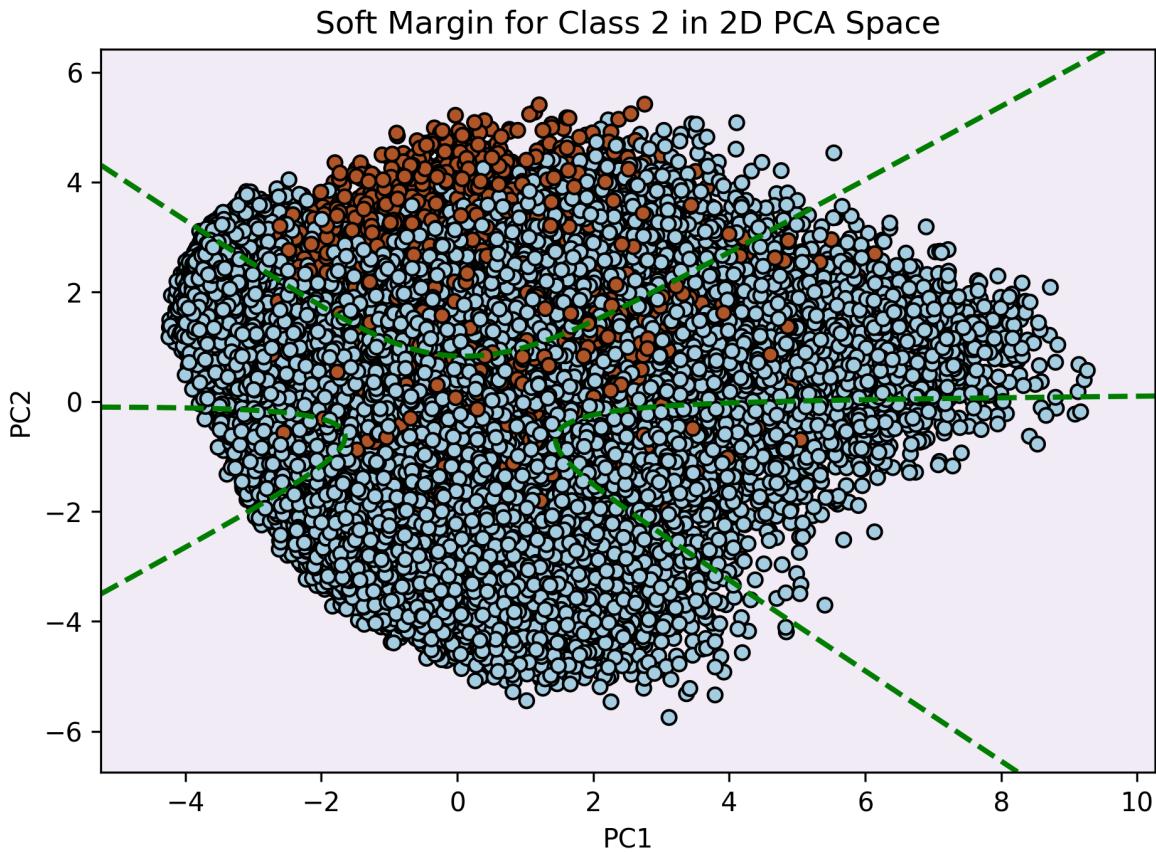
En general, cada gráfica muestra la separación entre dos clases de dígitos en el espacio definido por los dos primeros componentes principales. Estos componentes son combinaciones lineales de las características originales que retienen la mayor parte de la variabilidad de los datos. En las gráficas, los puntos azules y marrones representan las dos clases distintas, mientras que las líneas verdes y rojas marcan los márgenes suaves y los límites de decisión del clasificador SVM.



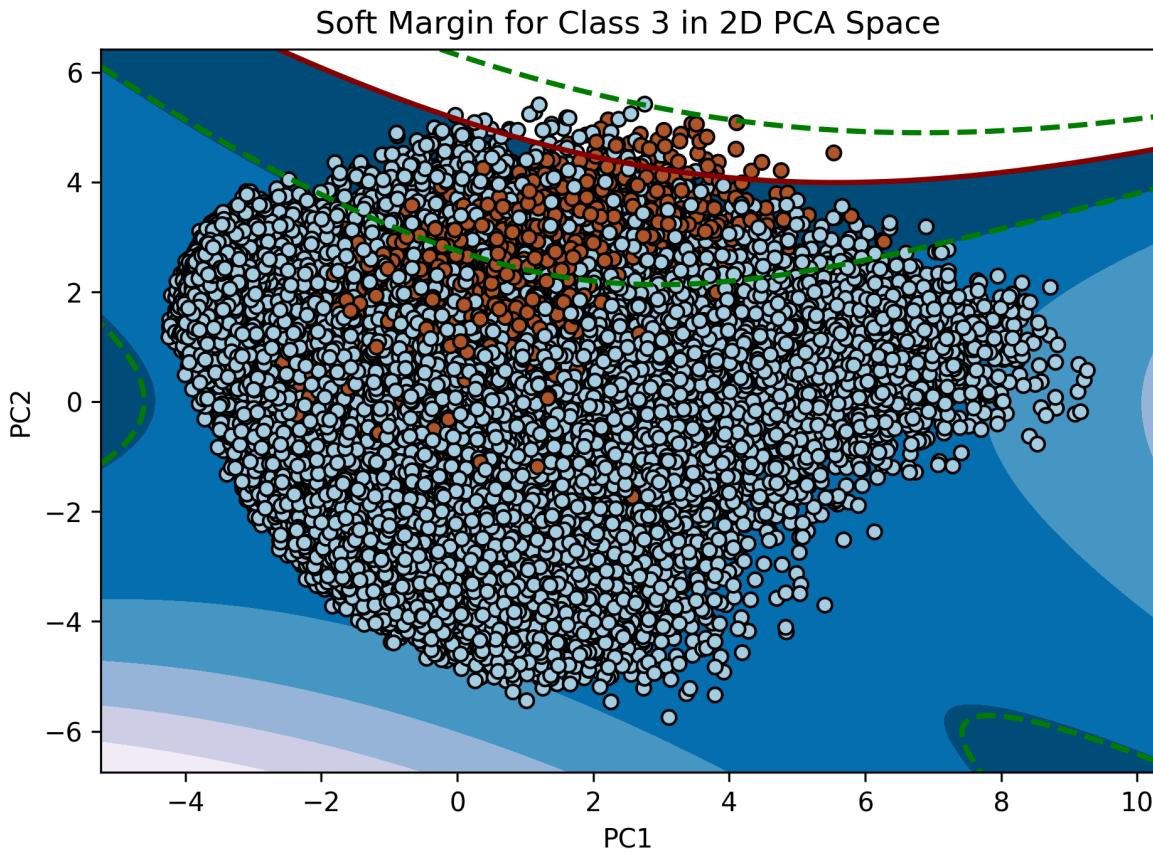
Para la clase 0, la gráfica indica una buena separación entre los dígitos 0 y los otros dígitos. La mayoría de los puntos se encuentran claramente en uno de los lados del hiperplano de decisión, lo que sugiere que el modelo tiene un alto nivel de confianza en sus predicciones para esta clase. Sin embargo, hay una pequeña cantidad de puntos marrones en la región de los puntos azules, lo que indica algunos errores de clasificación.



En el caso de la clase 1, la gráfica también muestra una separación clara, aunque con una mayor densidad de puntos marrones cercanos al margen de decisión. Esto sugiere que, aunque el modelo es bastante efectivo en la clasificación de unos, hay más ambigüedad en comparación con la clase 0. Los puntos que se encuentran cerca del margen indican casos donde el modelo tiene más dificultad para decidir.

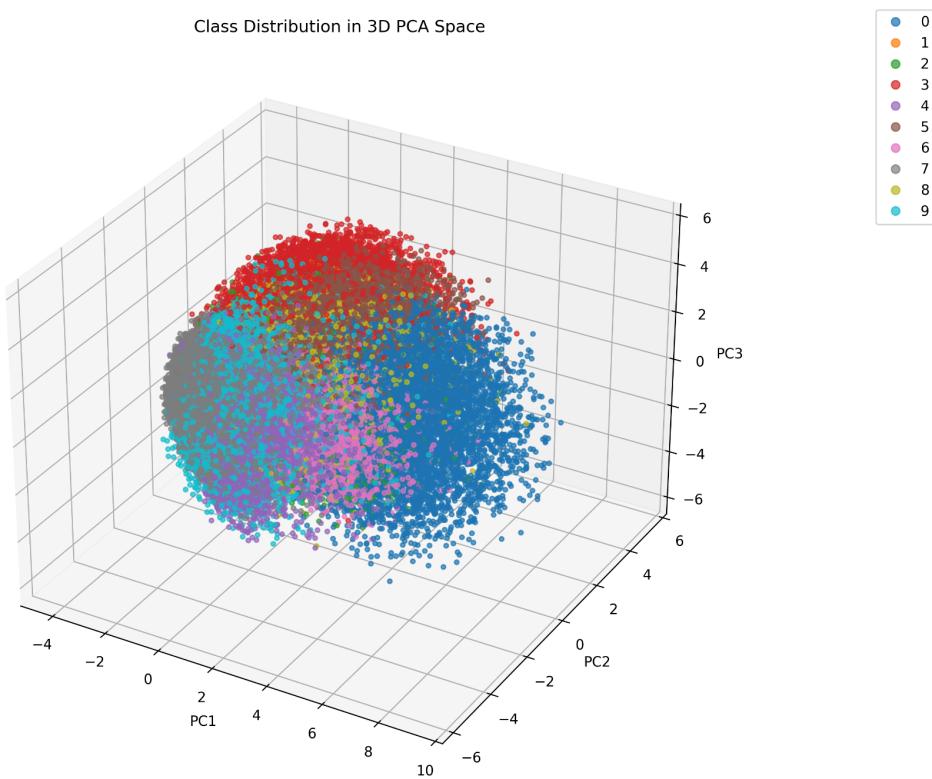


La gráfica para la clase 2 muestra una dispersión más amplia de los puntos, con algunas intersecciones entre las clases. Los márgenes suaves aquí indican que el modelo permite más flexibilidad en la clasificación, lo que puede ser necesario debido a la variabilidad en la representación de los dígitos 2. Las áreas donde las clases se superponen sugieren que el modelo tiene más dificultades para distinguir entre estas clases, lo que podría ser un área para futuras mejoras.



Finalmente, para la clase 3, la gráfica destaca una clara separación con algunas intersecciones en los márgenes. Esto sugiere que, aunque el modelo SVM logra separar eficazmente las clases en la mayoría de los casos, aún hay espacio para mejorar, especialmente en las áreas donde los puntos de diferentes clases están más cerca unos de otros. Las intersecciones en los márgenes indican ambigüedad en las predicciones del modelo.

En conjunto, estas gráficas y el análisis de los límites de decisión y márgenes suaves revelan la efectividad del modelo SVM en la clasificación de dígitos. Aunque el modelo generalmente realiza un buen trabajo separando las clases, las áreas de superposición y ambigüedad resaltan posibles oportunidades para optimizar y mejorar aún más el rendimiento del sistema.



La gráfica de distribución de clases en el espacio 3D PCA proporciona una visualización clara de cómo se distribuyen las diferentes clases de dígitos en el espacio tridimensional reducido. Los tres componentes principales (PC1, PC2 y PC3) se utilizaron para transformar el espacio original de características de las imágenes de dígitos a un espacio tridimensional que conserva la mayor parte de la variabilidad de los datos.

En esta gráfica, cada punto representa una muestra del conjunto de datos y está coloreado según su clase correspondiente (dígitos del 0 al 9). Esta visualización permite observar la distribución y la separación entre las diferentes clases en el espacio de componentes principales.

La distribución muestra que ciertas clases están más concentradas en regiones específicas del espacio PCA. Por ejemplo, los puntos azules (dígitos 0) tienden a agruparse más hacia un lado del gráfico, mientras que los puntos rojos (dígitos 1) se agrupan en otra región. Esto sugiere que los dígitos 0 y 1 tienen representaciones en el espacio PCA que los hacen fácilmente distinguibles unos de otros.

Sin embargo, también hay regiones donde las clases se superponen. Por ejemplo, los puntos correspondientes a los dígitos 5 y 8 (representados en rosa y marrón claro, respectivamente)

muestran una mayor superposición, lo que indica que estos dígitos comparten características similares y pueden ser más difíciles de distinguir para el modelo. La superposición de las clases en el espacio tridimensional revela las posibles ambigüedades que el modelo SVM debe manejar al clasificar los dígitos.

Esta visualización también pone de manifiesto la efectividad del PCA para reducir la dimensionalidad, conservando la estructura y la separabilidad de los datos en un espacio de menor dimensión. A pesar de la reducción a tres dimensiones, la mayor parte de la variabilidad y las relaciones entre las clases se mantienen intactas, permitiendo que el modelo SVM funcione eficientemente.

Optimización

Para asegurar un rendimiento óptimo del sistema de reconocimiento de números en tiempo real, se implementaron varias técnicas de optimización, centradas principalmente en la selección de los hiperparámetros adecuados para las Máquinas de Soporte Vectorial (SVM) y en la reducción de dimensionalidad mediante el Análisis de Componentes Principales (PCA).

La elección de los hiperparámetros adecuados es crucial para maximizar la precisión y eficiencia del modelo SVM. Utilizamos GridSearchCV para realizar una búsqueda exhaustiva de los mejores hiperparámetros, evaluando diferentes valores de C, gamma y tipos de kernel (linear, rbf, poly). Inicialmente, los mejores hiperparámetros encontrados fueron {'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}. Sin embargo, en pruebas con datos reales, el kernel RBF mostró una tendencia a sobreajustarse, prediciendo incorrectamente el mismo valor (5) para diversas entradas. Esta observación nos llevó a optar por un kernel polinomial, que demostró un mejor ajuste y precisión en la identificación de los dígitos reales.

Además de ajustar los hiperparámetros del modelo SVM, se implementó PCA para reducir la dimensionalidad de los datos. PCA transforma los datos a un espacio de menor dimensionalidad, reteniendo la mayor parte de la variabilidad de los datos originales. Esta técnica no solo facilita el entrenamiento del modelo, sino que también mejora su rendimiento al eliminar el ruido y las características redundantes. En este proyecto, PCA permitió reducir la dimensionalidad a un espacio tridimensional, facilitando un entrenamiento más eficiente del modelo SVM y mejorando su capacidad de generalización.

El preprocessamiento de datos también jugó un papel crucial en la optimización del modelo. Para el módulo RoiRecognition.py, se aplicaron varias técnicas: las imágenes de las regiones de interés (ROI) se convirtieron a escala de grises, se suavizaron y se convirtieron a imágenes binarias. Estas imágenes fueron luego normalizadas, recortadas y redimensionadas para asegurar uniformidad y mejorar la precisión del modelo. En el módulo TraceRecognition.py, las trayectorias de movimiento de la punta del dedo se suavizaron e interpolaron, seguidas de un preprocessamiento que incluyó recorte, ajuste de tamaño y operaciones morfológicas para mejorar el contraste y la calidad de las imágenes generadas a partir de las trayectorias.

Pseudocódigo

Clase RoiRecognition

```

Proceso RoiRecognition
    Definir clf Como Entero
    clf = cargar_modelo("model_archivos/svm_digit_classifier.pkl")

    # Función para preprocessar la imagen ROI
    Función preprocess_image(image)
        Definir inverted_roi, normalized_image, blurred_image, binary_image Como Enteros

        # Invertir los colores de la imagen
        inverted_roi = invertir_colores(image)
        # Normalizar la imagen
        normalized_image = normalizar(inverted_roi, 0, 255)
        # Aplicar desenfoque Gaussiano
        blurred_image = aplicar_desenfoque(normalized_image, (3, 3), 0)
        # Convertir la imagen a binaria
        binary_image = convertir_a_binaria(blurred_image, 128, 255)
        # Preprocessar la imagen binaria-
        retornar preprocess_binary_image(binary_image)

    FinFuncion

    # Función para preprocessar la imagen binaria
    Función preprocess_binary_image(image)
        Definir x_coords, y_coords, x_min, x_max, y_min, y_max, margin, cropped_image, resized_image, final_image, kernel Como Enteros
        # Encontrar coordenadas de pixeles blancos
        (x_coords, y_coords) = encontrar_coordenadas(image, 0)

        Si x_coords está vacío o y_coords está vacío Entonces
            retornar Ninguno
        FinSi

        # Determinar límites de la región blanca
        x_min = minimo(x_coords)
        x_max = máximo(x_coords)
        y_min = minimo(y_coords)
        y_max = máximo(y_coords)

        Si x_min == x_max o y_min == y_max Entonces
            retornar Ninguno
        FinSi

        # Agregar margen alrededor de la región blanca
        margin = 50
        x_min = máximo(x_min - margin, 0)
        x_max = mínimo(x_max + margin, tamaño_imagen[1])
        y_min = máximo(y_min - margin, 0)
        y_max = mínimo(y_max + margin, tamaño_imagen[0])

        # Recortar la imagen a la región blanca con margen
        cropped_image = recortar_imagen(image, y_min, y_max, x_min, x_max)

```

```

# Redimensionar la imagen a 20x20 pixeles
resized_image = redimensionar(cropped_image, (20, 20))
# Colocar la imagen redimensionada en el centro de una imagen 28x28
final_image = crear_imagen_blanco(28, 28)
final_image[4:24, 4:24] = resized_image

# Aplicar operaciones morfológicas
kernel = crear_kernel(2, 2)
final_image = aplicar_morfologia(final_image, kernel)
# Equalizar histograma
final_image = equalizar_histograma(final_image)
# Convertir la imagen a binaria
final_image = convertir_a_binaria(final_image, 128, 255)
retornar final_image
FinFuncion

# Función para predecir el dígito
Función predict(image)
    Definir processed_image Como Entero
    processed_image = preprocess_image(image)

    Si processed_image no es Ninguno Entonces
        processed_image = redimensionar(processed_image, (1, -1))
        retornar predecir_con_modelo(clf, processed_image)
    FinSi

    retornar Ninguno
FinFuncion

# Función para iniciar la captura de video y reconocimiento en tiempo real
Función run()
    Definir cap, ret, frame, gray, blurred, thresh, contours, h, w, center_x, center_y, offset, center_region, best_contour, best_area, cnt, x, y, w, h, area, roi, processed_digit, roi_digits
    cap = iniciar_captura_video(0)

    Mientras verdadero Hacer
        ret, frame = capturar_frame(cap)

        Si no ret Entonces
            romper
        FinSi

        # Convertir la imagen a escala de grises
        gray = convertir_a_grises(frame)
        # Aplicar desenfoque Gaussiano
        blurred = aplicar_desenfoque(gray, (5, 5), 0)
        # Convertir la imagen a binaria
        thresh = convertir_a_binaria(blurred, 128, 255, inv)

        # Convertir la imagen a binaria
        thresh = convertir_a_binaria(blurred, 128, 255, inv)
        # Encontrar contornos en la imagen binaria
        contours = encontrar_contornos(thresh)

        (h, w) = tamaño_imagen(frame)
        center_x = w / 2
        center_y = h / 2
        offset = 300
        center_region = definir_region_central(center_x, center_y, offset)

        best_contour = Ninguno
        best_area = 0

        Para cada cnt en contours Hacer
            (x, y, w, h) = obtener_rectángulo_del_contorno(cnt)
            area = calcular_area(w, h)

            Si contorno_en_region_central(center_region, x, y, w, h) y cumple_criterios_tamaño(w, h, area, best_area) Entonces
                best_contour = cnt
                best_area = area
            FinSi
        FinPara

        Si best_contour no es Ninguno Entonces
            (x, y, w, h) = obtener_rectángulo_del_contorno(best_contour)
            roi = recortar_imagen(gray, y, y+h, x, x+w)
            processed_digit = preprocess_image(roi)

            Si processed_digit no es Ninguno Entonces
                mostrar_imagen("Segmented and Preprocessed Image", redimensionar(processed_digit, (280, 280)))
                roi_digits = redimensionar(processed_digit, (1, -1))
                number_poly = predecir_con_modelo(clf, roi_digits)
                dibujar_rectángulo(frame, x, y, w, h, (0, 255, 0))
                dibujar_texto(frame, number_poly, x, y - 10, (0, 255, 0))
            FinSi
        FinSi

        mostrar_imagen("Frame with Digits and Trajectory", frame)

        key = esperar_tecla(1)
        Si key == 'q' Entonces
            romper
        FinSi
    FinMientras

    liberar_captura(cap)
    destruir_ventanas()
FinFuncion
FinProyecto

```

Clase TraceRecognition

```

# Pseudocódigo para la clase TraceRecognition

Proceso TraceRecognition
    Definir clf, trajectory, mp_hands, hands, cap Como Entero
    clf = cargar_modelo("model_archivos/svm_digit_classifier.pkl")
    trajectory = []
    mp_hands = inicializar_mediapipe_hands()
    hands = configurar_manos(mp_hands, 1, 0.7, 0.7)
    cap = iniciar_captura_video(0)

    # Función para suavizar la trayectoria
    Función smooth_trajectory(trajectory, alpha)
        Definir smoothed, prev, curr Como Enteros
        alpha = 0.75
        smoothed = []

        Si longitud(trajectory) > 1 Entonces
            agregar_a_lista(smoothed, trajectory[0])
            Para i = 1 Hasta longitud(trajectory) - 1 Hacer
                prev = smoothed[longitud(smoothed) - 1]
                curr = trajectory[i]
                agregar_a_lista(smoothed, (int(prev[0] * alpha + curr[0] * (1 - alpha)), int(prev[1] * alpha + curr[1] * (1 - alpha))))
            FinPara
        Sino
            smoothed = trajectory
        FinSi
        retornar smoothed
    FinFunción

    # Función para interpolar la trayectoria
    Función interpolate_trajectory(trajectory)
        Definir interpolated, prev, curr, t Como Enteros
        interpolated = []

        Para i = 1 Hasta longitud(trajectory) - 1 Hacer
            prev = trajectory[i - 1]
            curr = trajectory[i]
            agregar_a_lista(interpolated, prev)
            Para t = 0 Hasta 1 Con Paso 0.1 Hacer
                agregar_a_lista(interpolated, (int(prev[0] * (1 - t) + curr[0] * t), int(prev[1] * (1 - t) + curr[1] * t)))
            FinPara
        FinPara
        agregar_a_lista(interpolated, trajectory[longitud(trajectory) - 1])
        retornar interpolated
    FinFunción

```

```

Funcion preprocess_trajectory(trajectory, frame_shape)
    Si longitud(trajectory) == 0 Entonces
        Escribir "Trajectory is empty"
        retornar Ninguno
    FinSi

    Definir image, x_coords, y_coords, x_min, x_max, y_min, y_max, margin, cropped_image, resized_image, kernel Como Enteros

    # Crear una imagen en blanco
    image = crear_imagen_blanca(480, 640)
    # Dibujar la trayectoria interpolada en la imagen
    Para cada (x, y) en interpolate_trajectory(trajectory) Hacer
        dibujar_circulo(image, (x, y), 5, 255, -1)
    FinPara

    # Encontrar los límites de la trayectoria
    (x_coords, y_coords) = obtener_coordenadas(trajectory)
    x_min = minimo(x_coords)
    x_max = maximo(x_coords)
    y_min = minimo(y_coords)
    y_max = maximo(y_coords)

    Si x_min == x_max o y_min == y_max Entonces
        retornar Ninguno
    FinSi

    # Recortar la imagen al tamaño de la trayectoria con un margen
    margin = 10
    x_min = maximo(x_min - margin, 0)
    x_max = minimo(x_max + margin, tamaño_imagen[1])
    y_min = maximo(y_min - margin, 0)
    y_max = minimo(y_max + margin, tamaño_imagen[0])
    cropped_image = recortar_imagen(image, y_min, y_max, x_min, x_max)

    # Redimensionar la imagen a 20x20 pixeles
    resized_image = redimensionar_imagen(cropped_image, (20, 20))

    # Colocar la imagen redimensionada en el centro de una imagen en blanco de 28x28 pixeles
    finalImage = crear_imagen_blanca(28, 28)
    finalImage[4:24, 4:24] = resized_image

    # Aplicar operaciones morfológicas y normalización
    kernel = crear_kernel((2, 2))
    final_image = aplicar_morfologia(finalImage, kernel)
    final_image = aplicar_desenfoque(final_image, (3, 3), 0)
    final_image = equalizar_histograma(final_image)
    retornar final_image
FinFuncion

# Función para iniciar la captura de video y reconocimiento de trazos en tiempo real
Funcion run()
    Definir ret, frame, rgb_frame, result, hand_landmarks, index_finger_tip, h, w, cx, cy, smoothed_trajectory, i, key, input_img_for_model, digit, trajectory_img, x, y Como Enteros

    Mientras verdadero Hacer
        (ret, frame) = capturar_frame(cap)
        Si no ret Entonces
            romper
        FinSi

        frame = invertir_imagen_horizontalmente(frame)
        rgb_frame = convertir_a_rgb(frame)
        result = procesar_imagen_para_manos(hands, rgb_frame)

        Si result.multi_hand_landmarks Entonces
            Para cada hand_landmarks en result.multi_hand_landmarks Hacer
                index_finger_tip = obtener_punta_dedo(hand_landmarks, mp_hands)
                (h, w, _) = tamaño_imagen(frame)
                cx = int(index_finger_tip.x * w)
                cy = int(index_finger_tip.y * h)
                dibujar_circulo(frame, (cx, cy), 8, (0, 255, 0), -1)
                agregar_a_lista(trajetoria, (cx, cy))
                smoothed_trajectory = smoothen_trajectory(trajetoria)

            Para i = 1 Hasta longitud(smoothed_trajectory) - 1 Hacer
                Si smoothed_trajectory[i - 1] es Ninguno o smoothed_trajectory[i] es Ninguno Entonces
                    continuar
                FinSi
                dibujar_linea(frame, smoothed_trajectory[i - 1], smoothed_trajectory[i], (0, 255, 0), 2)
            FinPara
        FinSi

        mostrar_imagen("Frame", frame)
        key = esperar_tecla()

        Si key == 'p' Entonces
            Si longitud(trajetoria) > 0 Entonces
                input_img = preprocess_trajectory(trajetoria, tamaño_imagen(frame))
                Si input_img no es Ninguno Entonces
                    input_img_for_model = redimensionar(input_img, (1, -1))
                    digit = predecir_con_modelo(clf, input_img_for_model)
                    Escribir "Predicted Digit: ", digit[0]

                    trajectory_img = crear_imagen_blanca(480, 640)
                    Para cada (x, y) en trajetoria Hacer
                        dibujar_circulo(trajectory_img, (x, y), 5, 255, -1)
                    FinPara
            FinSi
        FinSi
    FinMientras

```

```

7
8     Si key == 'p' Entonces
9         Si longitud(trajectory) > 0 Entonces|
10            input_img = preprocess_trajectory(trajectory, tamaño_imagen(frame))
11            Si input_img no es Ninguno Entonces|
12                input_img_for_model = redimensionar(input_img, (1, -1))
13                digit = predecir_con_modelo(clf, input_img_for_model)
14                Escribir "Predicted Digit: ", digit[0]
15
16                trajectory_img = crear_imagen_blanca(480, 640)
17                Para cada (x, y) en trajectory Hacer|
18                    dibujar_circulo(trajectory_img, (x, y), 5, 255, -1)
19                FinPara
20                guardar_imagen("trajectory.png", trajectory_img)
21                mostrar_imagen("Trace Image", redimensionar(trajectory_img, (280, 280)))
22                mostrar_imagen("Model Input Image", convertir_a_uint8(input_img * 255))
23
24                visualizar_preprocesamiento(input_img, digit[0])
25
26                trajectory = []
27            Sino
28                Escribir "Invalid trajectory, try again."
29            FinSi
30        FinSi
31    Sino Si key == 'c' Entonces
32        trajectory = []
33        Escribir "Trazo eliminado. Empieza de nuevo."
34    Sino Si key == 'q' Entonces|
35        romper
36    FinSi
37 FinMientras
38
39 liberar_captura(cap)
40 destruir_ventanas()
41
42 FinFuncion
43 FinProceso
44

```

Clase modelSVMpca

```

# Pseudocódigo para cargar los datos MNIST, reducir la dimensionalidad y entrenar un SVM

Proceso Principal
    # Función para cargar los datos MNIST
    Funcion load_mnist_data()
        Definir X_train, y_train, X_test, y_test Como Arreglo
        (X_train, y_train), (X_test, y_test) = cargar_datos_MNIST()

        # Aplanar las imágenes
        X_train = aplanar_imagenes(X_train, 28*28)
        X_test = aplanar_imagenes(X_test, 28*28)

        # Normalizar las imágenes
        X_train = normalizar_imagenes(X_train, 255.0)
        X_test = normalizar_imagenes(X_test, 255.0)

        retornar X_train, X_test, y_train, y_test
    FinFuncion

    Definir X_train, X_test, y_train, y_test Como Arreglo
    (X_train, X_test, y_train, y_test) = load_mnist_data()

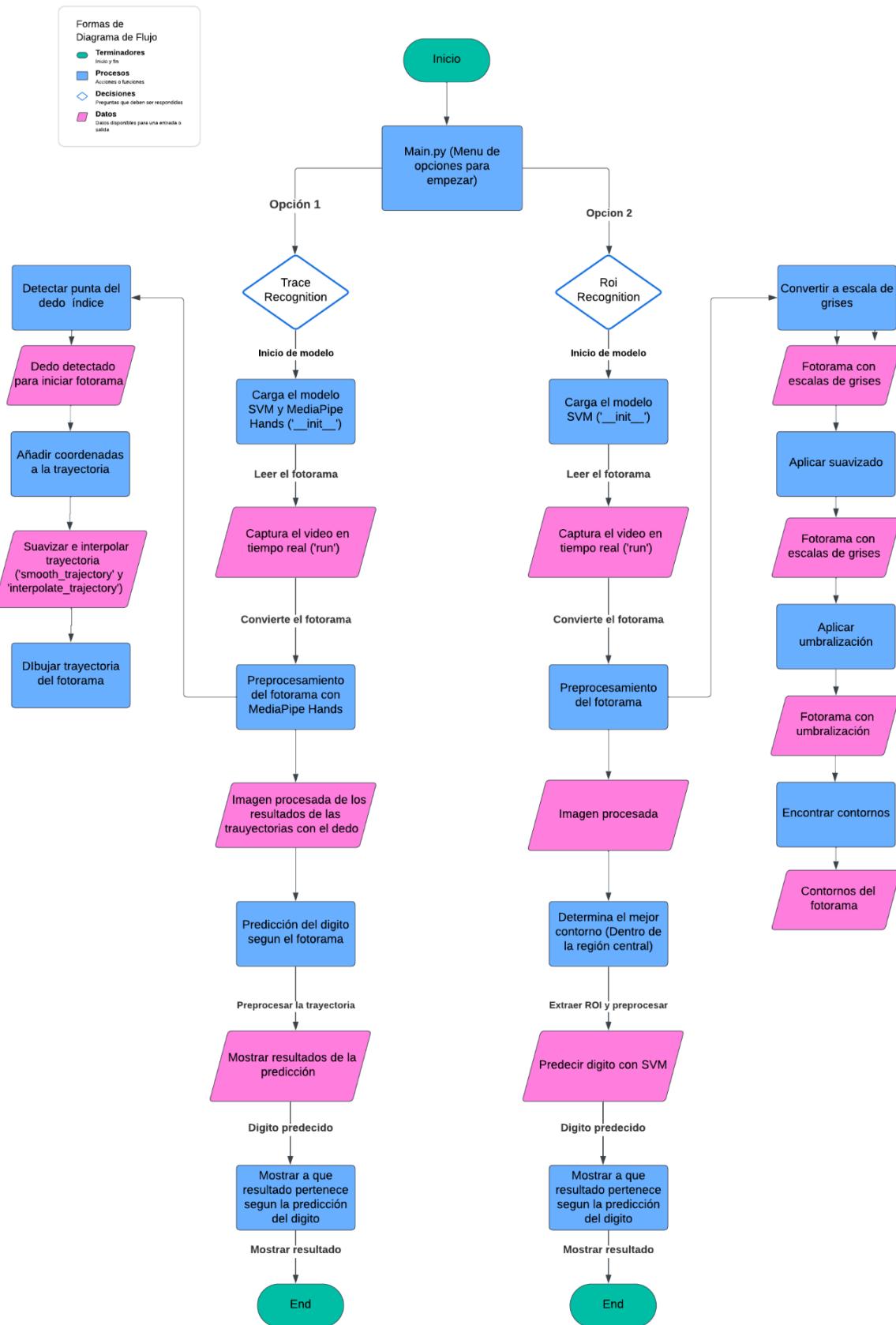
    # Reducir la dimensionalidad a 3D usando PCA
    Definir pca, X_train_3D, X_test_3D Como Arreglo
    pca = inicializar_PCA(3)
    X_train_3D = ajustar_transformar_PCA(pca, X_train)
    X_test_3D = transformar_PCA(pca, X_test)

    # Entrenar un SVM en el espacio 3D reducido
    Definir svm_3D Como Entero
    svm_3D = inicializar_SVM('poly', 10, 'scale')
    entrenar_SVM(svm_3D, X_train_3D, y_train)

    # Guardar el modelo entrenado
    guardar_modelo(svm_3D, "model_archivos/svm_digit_classifier_PCA3.pkl")
FinProceso

```

Diagrama de flujo de funcionamiento



Códigos (cada línea comentada)

Clase TraceRecognition

```

import cv2
import numpy as np
import joblib
import mediapipe as mp
import matplotlib.pyplot as plt

class TraceRecognition:
    def __init__(self):
        """
        Inicializa la clase TraceRecognition cargando el modelo SVM,
        configurando MediaPipe Hands y la captura de video.
        """
        self.clf = joblib.load("model_archivos/svm_digit_classifier.pkl")
        self.trajectory = []
        self.mp_hands = mp.solutions.hands
        self.hands = self.mp_hands.Hands(max_num_hands=1, min_detection_confidence=0.7, min_tracking_confidence=0.7)
        self.cap = cv2.VideoCapture(0)

    def smooth_trajectory(self, trajectory, alpha=0.75):
        """
        Suaviza la trayectoria utilizando un filtro de media móvil exponencial.

        Args:
            trajectory (list): Lista de coordenadas (x, y) de la trayectoria.
            alpha (float): Factor de suavizado.

        Returns:
            list: Lista de coordenadas suavizadas.
        """
        smoothed = []
        if len(trajectory) > 1:
            smoothed.append(trajectory[0])
            for i in range(1, len(trajectory)):
                prev = smoothed[-1]
                curr = trajectory[i]
                smoothed.append((int(prev[0] * alpha + curr[0] * (1 - alpha))), int(prev[1] * alpha + curr[1] * (1 - alpha))))
        else:
            smoothed = trajectory
        return smoothed

```

```
def interpolate_trajectory(self, trajectory):
    """
    Interpolates the trajectory to add additional points between coordinates.

    Args:
        trajectory (list): List of coordinates (x, y) of the trajectory.

    Returns:
        list: List of interpolated coordinates.
    """
    interpolated = []
    for i in range(1, len(trajectory)):
        prev = trajectory[i - 1]
        curr = trajectory[i]
        interpolated.append(prev)
        # Adds additional points between coordinates to smooth the trajectory
        for t in np.linspace(0, 1, num=10):
            interpolated.append((int(prev[0] * (1 - t) + curr[0] * t), int(prev[1] * (1 - t) + curr[1] * t)))
    interpolated.append(trajectory[-1])
    return interpolated

def preprocess_trajectory(self, trajectory, frame_shape):
    """
    Preprocesses the trajectory to create an appropriate image for the SVM classifier.

    Args:
        trajectory (list): List of coordinates (x, y) of the trajectory.
        frame_shape (tuple): Shape of the captured image frame.

    Returns:
        numpy.ndarray: Preprocessed image of 28x28 pixels.
    """
    if not trajectory:
        print("Trajectory is empty")
        return None
```

```
# Crear una imagen en blanco
image = np.zeros((480, 640), dtype=np.uint8)
# Dibujar la trayectoria interpolada en la imagen
for (x, y) in self.interpolate_trajectory(trajectory):
    cv2.circle(image, (x, y), 5, 255, -1)

# Encontrar los límites de la trayectoria
x_coords, y_coords = zip(*trajectory)
x_min, x_max = min(x_coords), max(x_coords)
y_min, y_max = min(y_coords), max(y_coords)

# Asegurarse de que los límites son válidos
if x_min == x_max or y_min == y_max:
    return None

# Recortar la imagen al tamaño de la trayectoria con un margen
margin = 10
x_min = max(x_min - margin, 0)
x_max = min(x_max + margin, image.shape[1])
y_min = max(y_min - margin, 0)
y_max = min(y_max + margin, image.shape[0])
cropped_image = image[y_min:y_max, x_min:x_max]

# Redimensionar la imagen a 20x20 píxeles
resized_image = cv2.resize(cropped_image, (20, 20), interpolation=cv2.INTER_AREA)

# Colocar la imagen redimensionada en el centro de una imagen en blanco de 28x28 píxeles
final_image = np.zeros((28, 28), dtype=np.uint8)
final_image[4:24, 4:24] = resized_image

# Aplicar operaciones morfológicas y normalización
kernel = np.ones((2, 2), np.uint8)
final_image = cv2.morphologyEx(final_image, cv2.MORPH_CLOSE, kernel)
final_image = cv2.GaussianBlur(final_image, (3, 3), 0)
final_image = cv2.equalizeHist(final_image)
return final_image
```

```
def predict(self, trajectory, frame_shape):
    """
    Predice el dígito a partir de la trayectoria preprocesada.

    Args:
        trajectory (list): Lista de coordenadas (x, y) de la trayectoria.
        frame_shape (tuple): Forma del marco de la imagen capturada.

    Returns:
        int: Dígito predicho.
    """
    input_img = self.preprocess_trajectory(trajectory, frame_shape)
    if input_img is not None:
        input_img_for_model = input_img.reshape(1, -1)
        digit = self.clf.predict(input_img_for_model)
        return digit[0]
    return None

def run(self):
    """
    Inicia la captura de video y el reconocimiento de trazos en tiempo real.
    """
    while True:
        ret, frame = self.cap.read()
        if not ret:
            break

        # Invertir la imagen horizontalmente para una mejor experiencia del usuario
        frame = cv2.flip(frame, 1)

        # Convertir la imagen a RGB para MediaPipe
        rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

        # Procesar la imagen para encontrar las manos
        result = self.hands.process(rgb_frame)
```

```

if result.multi_hand_landmarks:
    for hand_landmarks in result.multi_hand_landmarks:
        # Obtener las coordenadas del índice del dedo
        index_finger_tip = hand_landmarks.landmark[self.mp_hands.HandLandmark.INDEX_FINGER_TIP]
        h, w, _ = frame.shape
        cx, cy = int(index_finger_tip.x * w), int(index_finger_tip.y * h)

        # Dibujar el círculo en la punta del dedo índice
        cv2.circle(frame, (cx, cy), 8, (0, 255, 0), -1)

        # Agregar la coordenada a la trayectoria
        self.trajectory.append((cx, cy))

        # Suavizar la trayectoria y dibujarla en la imagen
        smoothed_trajectory = self.smooth_trajectory(self.trajectory)
        for i in range(1, len(smoothed_trajectory)):
            if smoothed_trajectory[i - 1] is None or smoothed_trajectory[i] is None:
                continue
            cv2.line(frame, smoothed_trajectory[i - 1], smoothed_trajectory[i], (0, 255, 0), 2)

# Mostrar la imagen con la trayectoria
cv2.imshow('Frame', frame)
key = cv2.waitKey(1) & 0xFF

# Si el usuario presiona 'p', procesar la trayectoria y predecir el dígito
if key == ord('p'):
    if self.trajectory:
        input_img = self.preprocess_trajectory(self.trajectory, frame.shape)
        if input_img is not None:
            input_img_for_model = input_img.reshape(1, -1)
            digit = self.clf.predict(input_img_for_model)
            print(f'Predicted Digit: {digit[0]}')

            # Guardar la imagen del rastro
            trajectory_img = np.zeros((480, 640), dtype=np.uint8)
            for (x, y) in self.trajectory:
                cv2.circle(trajectory_img, (x, y), 5, 255, -1)
            cv2.imwrite('trajectory.png', trajectory_img)

```

```

# Mostrar la imagen del trazo y la imagen que se envía al modelo
cv2.imshow('Trace Image', cv2.resize(trajectory_img, (280, 280), interpolation=cv2.INTER_AREA))
cv2.imshow('Model Input Image', (input_img * 255).astype(np.uint8))

# Visualizar el preprocessamiento detallado
plt.figure(figsize=(5, 5))
plt.imshow(input_img.reshape(28, 28), cmap='gray')
plt.title(f'Predicted: {digit[0]}')
plt.axis('off')
plt.show()

# Limpiar la trayectoria
self.trajectory = []
else:
    print("Invalid trajectory, try again.")

elif key == ord('c'):
    # Limpiar la trayectoria
    self.trajectory = []
    print("Trazo eliminado. Empieza de nuevo.")

elif key == ord('q'):
    break

# Liberar la captura de video y cerrar las ventanas
self.cap.release()
cv2.destroyAllWindows()

```

Clase RoiRecognition

```
import cv2
import numpy as np
import joblib
import mediapipe as mp
import matplotlib.pyplot as plt

class RoiRecognition:
    def __init__(self):
        """
        Inicializa la clase RoiRecognition cargando el modelo SVM.
        """
        self.clf = joblib.load("model_archivos/svm_digit_classifier.pkl")

    def preprocess_image(self, image):
        """
        Preprocesa la imagen ROI para invertir los colores, normalizar,
        aplicar un desenfoque Gaussiano y convertirla a binaria.

        Args:
            image (numpy.ndarray): Imagen de la región de interés (ROI).

        Returns:
            numpy.ndarray: Imagen preprocesada de 28x28 píxeles.
        """
        # Invertir los colores de la imagen (blanco a negro y viceversa)
        inverted_roi = cv2.bitwise_not(image)
        # Normalizar la imagen para que los valores de los píxeles estén entre 0 y 255
        normalized_image = cv2.normalize(inverted_roi, None, 0, 255, cv2.NORM_MINMAX)
        # Aplicar un desenfoque Gaussiano para suavizar la imagen
        blurred_image = cv2.GaussianBlur(normalized_image, (3, 3), 0)
        # Convertir la imagen a binaria utilizando un umbral de 128
        _, binary_image = cv2.threshold(blurred_image, 128, 255, cv2.THRESH_BINARY)
        # Preprocesar la imagen binaria
        return self.preprocess_binary_image(binary_image)
```

```
def preprocess_binary_image(self, image):
    """
    Preprocesa la imagen binaria recortando, redimensionando y aplicando
    operaciones morfológicas para preparar la imagen para el modelo SVM.

    Args:
        image (numpy.ndarray): Imagen binaria.

    Returns:
        numpy.ndarray: Imagen preprocesada de 28x28 píxeles.

    # Encontrar las coordenadas de los píxeles blancos en la imagen binaria
    x_coords, y_coords = np.where(image > 0)
    if len(x_coords) == 0 or len(y_coords) == 0:
        return None
    # Determinar los límites de la región blanca en la imagen
    x_min, x_max = min(x_coords), max(x_coords)
    y_min, y_max = min(y_coords), max(y_coords)
    if x_min == x_max or y_min == y_max:
        return None
    # Agregar un margen alrededor de la región blanca
    margin = 50
    x_min = max(x_min - margin, 0)
    x_max = min(x_max + margin, image.shape[1])
    y_min = max(y_min - margin, 0)
    y_max = min(y_max + margin, image.shape[0])
    # Recortar la imagen a la región blanca con el margen añadido
    cropped_image = image[y_min:y_max, x_min:x_max]
    # Redimensionar la imagen a 20x20 píxeles
    resized_image = cv2.resize(cropped_image, (20, 20), interpolation=cv2.INTER_AREA)
    # Colocar la imagen redimensionada en el centro de una imagen en blanco de 28x28
    final_image = np.zeros((28, 28), dtype=np.uint8)
    final_image[4:24, 4:24] = resized_image
    # Aplicar operaciones morfológicas para cerrar los pequeños agujeros en la imagen
    kernel = np.ones((2, 2), np.uint8)
    final_image = cv2.morphologyEx(final_image, cv2.MORPH_CLOSE, kernel)
    # Ecualizar el histograma de la imagen para mejorar el contraste
    final_image = cv2.equalizeHist(final_image)
```

```

    # Equalizar el histograma de la imagen para mejorar el contraste
    final_image = cv2.equalizeHist(final_image)
    # Convertir la imagen a binaria nuevamente utilizando un umbral de 128
    _, final_image = cv2.threshold(final_image, 128, 255, cv2.THRESH_BINARY)
    return final_image

def predict(self, image):
    """
    Predice el dígito a partir de la imagen ROI preprocesada.

    Args:
        image (numpy.ndarray): Imagen de la región de interés (ROI).

    Returns:
        int: Dígito predicho.
    """

    processed_image = self.preprocess_image(image)
    if processed_image is not None:
        processed_image = processed_image.reshape(1, -1)
        return self.clf.predict(processed_image)[0]
    return None

```

```

def run(self):
    """
    Inicia la captura de video y el reconocimiento de dígitos en la región de interés (ROI) en tiempo real.
    """

    cap = cv2.VideoCapture(0)
    while True:
        ret, frame = cap.read()
        if not ret:
            break
        # Convertir la imagen a escala de grises
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        # Aplicar un desenfóque Gaussiano para suavizar la imagen
        blurred = cv2.GaussianBlur(gray, (5, 5), 0)
        # Convertir la imagen a binaria utilizando un umbral de 128
        _, thresh = cv2.threshold(blurred, 128, 255, cv2.THRESH_BINARY_INV)
        # Encontrar los contornos en la imagen binaria
        contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        h, w = frame.shape[:2]
        center_x, center_y = w // 2, h // 2
        offset = 300 # Aumenta el tamaño del offset
        center_region = (center_x - offset, center_y - offset, center_x + offset, center_y + offset)
        best_contour = None
        best_area = 0
        for cnt in contours:
            x, y, w, h = cv2.boundingRect(cnt)
            area = w * h
            # Verificar si el contorno está dentro de la región central y si cumple con los criterios de tamaño
            if (center_region[0] < x < center_region[2] and center_region[1] < y < center_region[3] and
                30 < w < 200 and 30 < h < 200 and area > best_area):
                best_contour = cnt
                best_area = area
        if best_contour is not None:
            x, y, w, h = cv2.boundingRect(best_contour)
            roi = gray[y:y+h, x:x+w]
            processed_digit = self.preprocess_image(roi)
            if processed_digit is not None:
                # Mostrar la imagen segmentada y preprocesada
                cv2.imshow('Segmented and Preprocessed Image', cv2.resize(processed_digit, (280, 280), interpolation=cv2.INTER_AREA))

```

```
x, y, w, h = cv2.boundingRect(cnt)
area = w * h
# Verificar si el contorno está dentro de la región central y si cumple con los criterios de tamaño
if (center_region[0] < x < center_region[2] and center_region[1] < y < center_region[3] and
    30 < w < 200 and 30 < h < 200 and area > best_area):
    best_contour = cnt
    best_area = area
if best_contour is not None:
    x, y, w, h = cv2.boundingRect(best_contour)
    roi = gray[y:y+h, x:x+w]
    processed_digit = self.preprocess_image(roi)
    if processed_digit is not None:
        # Mostrar la imagen segmentada y procesada
        cv2.imshow('Segmented and Preprocessed Image', cv2.resize(processed_digit, (280, 280), interpolation=cv2.INTER_AREA))
        roi_digits = processed_digit.reshape((1, -1))
        number_poly = self.clf.predict(roi_digits)
        # Dibujar un rectángulo alrededor de la región de interés y mostrar el dígito predicho
        cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)
        cv2.putText(frame, f'{int(number_poly)}', (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)
    # Mostrar la imagen con los dígitos y la trayectoria
    cv2.imshow('Frame with Digits and Trajectory', frame)
    key = cv2.waitKey(1) & 0xFF
    if key == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

Clase modelSVMpca

```

    ✓ import joblib
    import matplotlib.pyplot as plt
    import numpy as np
    import tensorflow as tf
    from mpl_toolkits.mplot3d import Axes3D
    from sklearn.decomposition import PCA
    from sklearn.svm import SVC
    from tensorflow.keras.datasets import mnist
    import tensorflow as tf

    # Cargar los datos MNIST
    ✓ def load_mnist_data():
        mnist = tf.keras.datasets.mnist
        (X_train, y_train), (X_test, y_test) = mnist.load_data()
        # Aplanar las imágenes
        X_train = X_train.reshape(-1, 28*28).astype('float32')/255.0
        X_test = X_test.reshape(-1, 28*28).astype('float32')/255.0
        return X_train, X_test, y_train, y_test

    X_train, X_test, y_train, y_test = load_mnist_data()

    # Reducir la dimensionalidad a 3D usando PCA
    pca = PCA(n_components=3)
    X_train_3D = pca.fit_transform(X_train)
    X_test_3D = pca.transform(X_test)

    # Entrenar un SVM en el espacio 3D reducido
    svm_3D = SVC(kernel='poly', C=10, gamma='scale')
    svm_3D.fit(X_train_3D, y_train)

    joblib.dump(svm_3D, "model_archivos/svm_digit_classifier_PCA3.pkl")

```

Pruebas

Para garantizar la precisión y robustez del sistema de reconocimiento de números en tiempo real, se llevaron a cabo dos tipos de pruebas: pruebas unitarias y pruebas manuales. Estas pruebas son esenciales para validar el desempeño del modelo en distintos escenarios y asegurar que el sistema funcione correctamente en condiciones reales.

Pruebas Unitarias

Las pruebas unitarias se diseñaron para evaluar individualmente cada componente del sistema de reconocimiento. Utilizando el marco de trabajo unittest de Python, se implementaron varias pruebas que cubren diferentes aspectos del preprocesamiento y la predicción de números. En el módulo RoiRecognition.py, las pruebas se centraron en la preparación de imágenes de regiones de interés (ROI) que contienen dígitos escritos a mano. Se crearon imágenes de prueba con dígitos grandes y claros, que luego fueron procesadas y clasificadas utilizando el modelo entrenado. Las pruebas verificaron que los dígitos predichos coincidieran con los dígitos esperados, asegurando la funcionalidad correcta del módulo.

Imágenes que se generan para predecir:



```
Predicted digit (ROI) for 7: 7
Predicted digit (ROI) for 0: 0
```

En el módulo TraceRecognition.py, las pruebas unitarias se enfocaron en la preparación y procesamiento de trayectorias de movimiento de la punta del dedo, dibujadas en el aire. Se generaron trayectorias de prueba que simulan el dibujo de números y se verificó que el procesamiento y la predicción se realizarán correctamente. Las pruebas evaluaron la suavización e interpolación de trayectorias, así como el preprocesamiento final de la imagen resultante.

Imágenes que se generan para predecir:



```
Predicted digit (Trace): 7
```

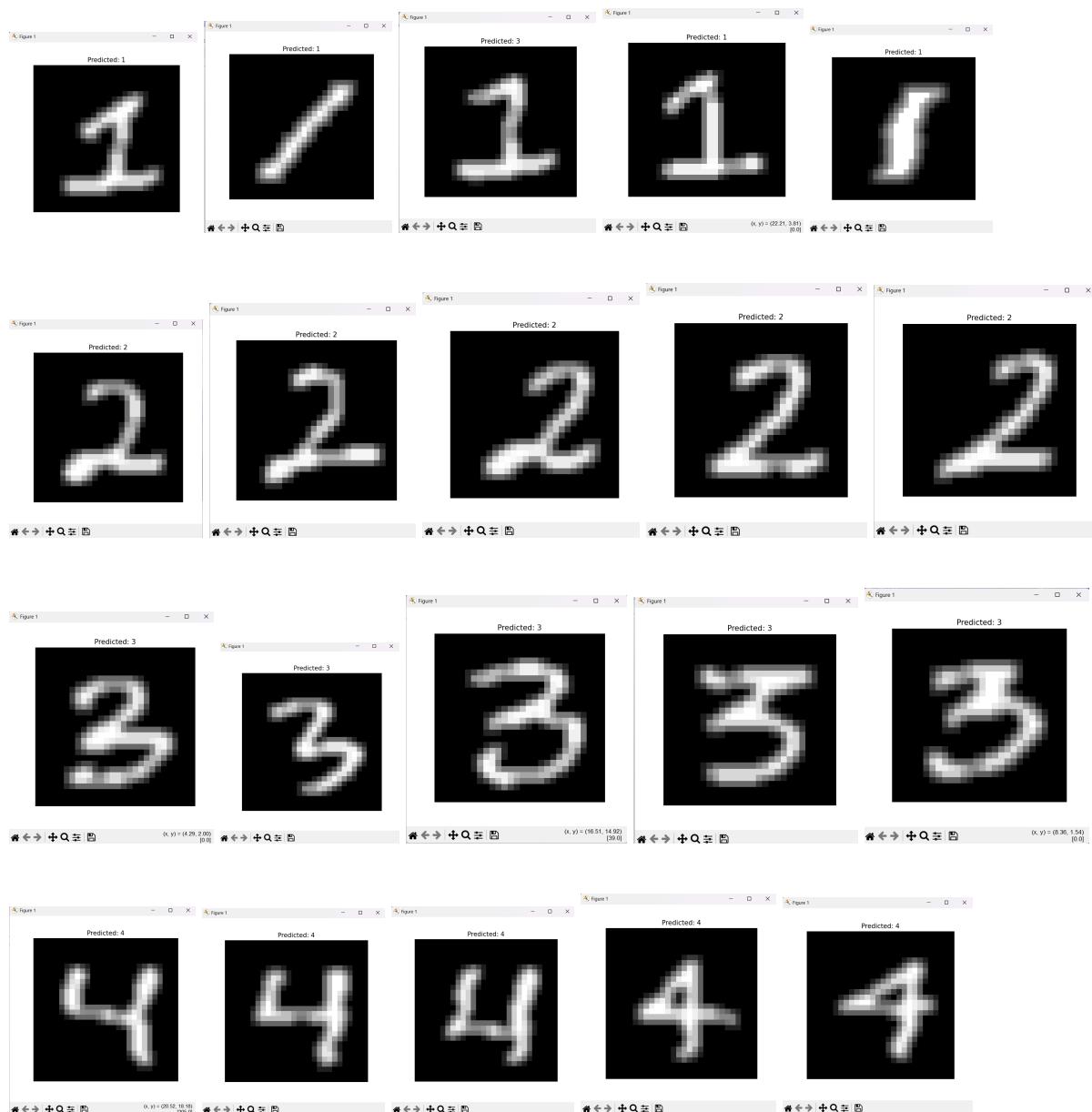
```
.
```

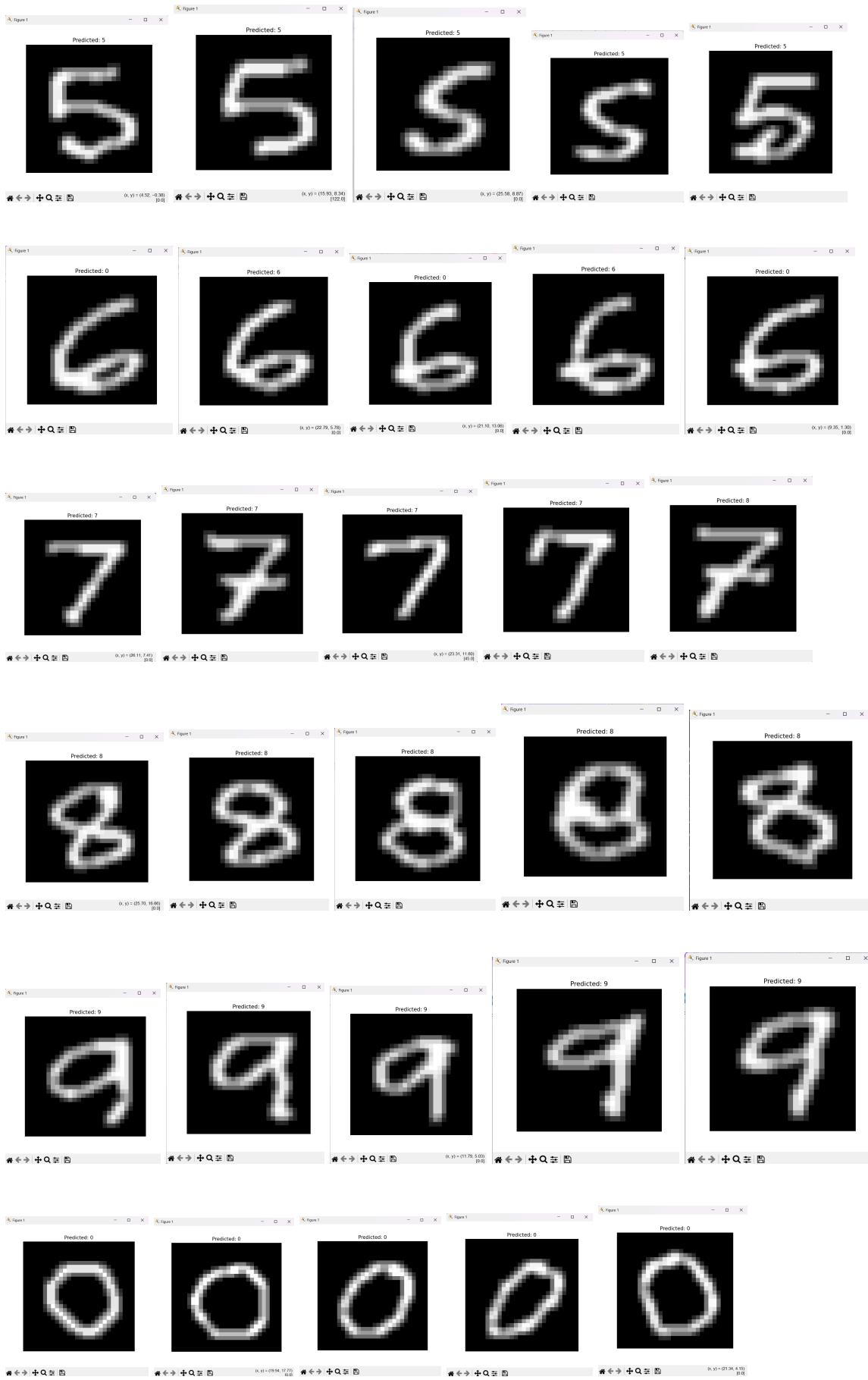
```
Ran 2 tests in 102.514s
```

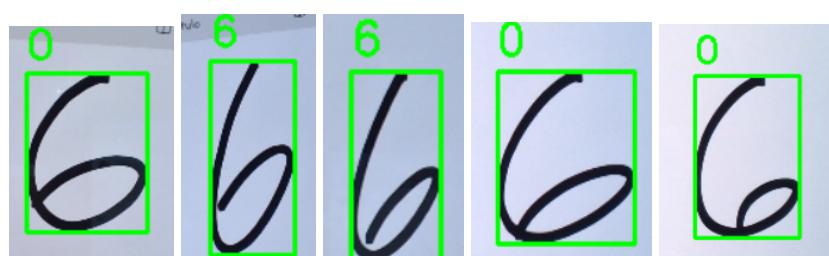
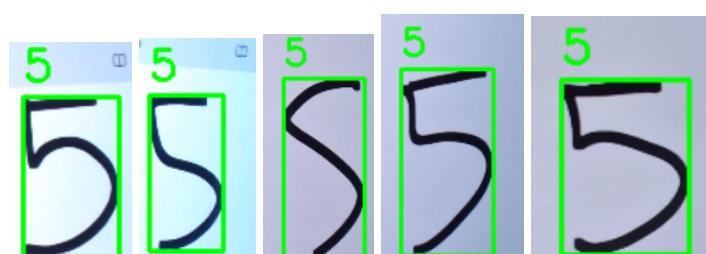
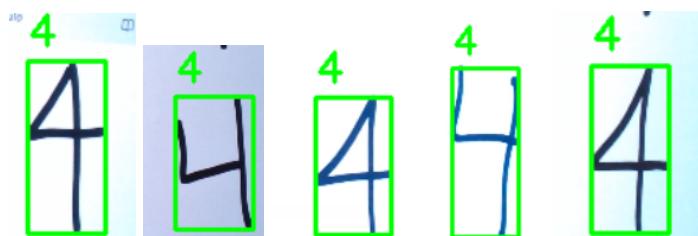
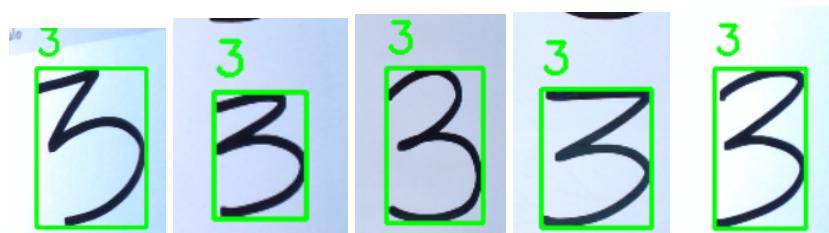
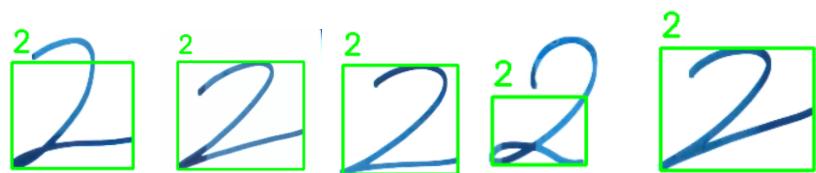
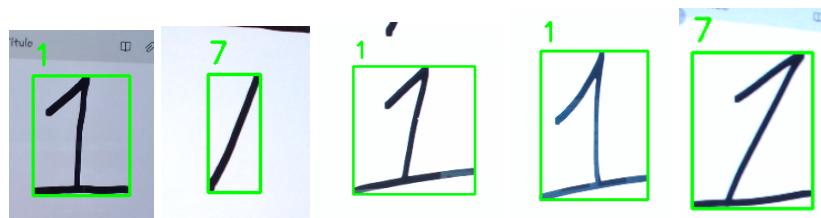
```
OK
```

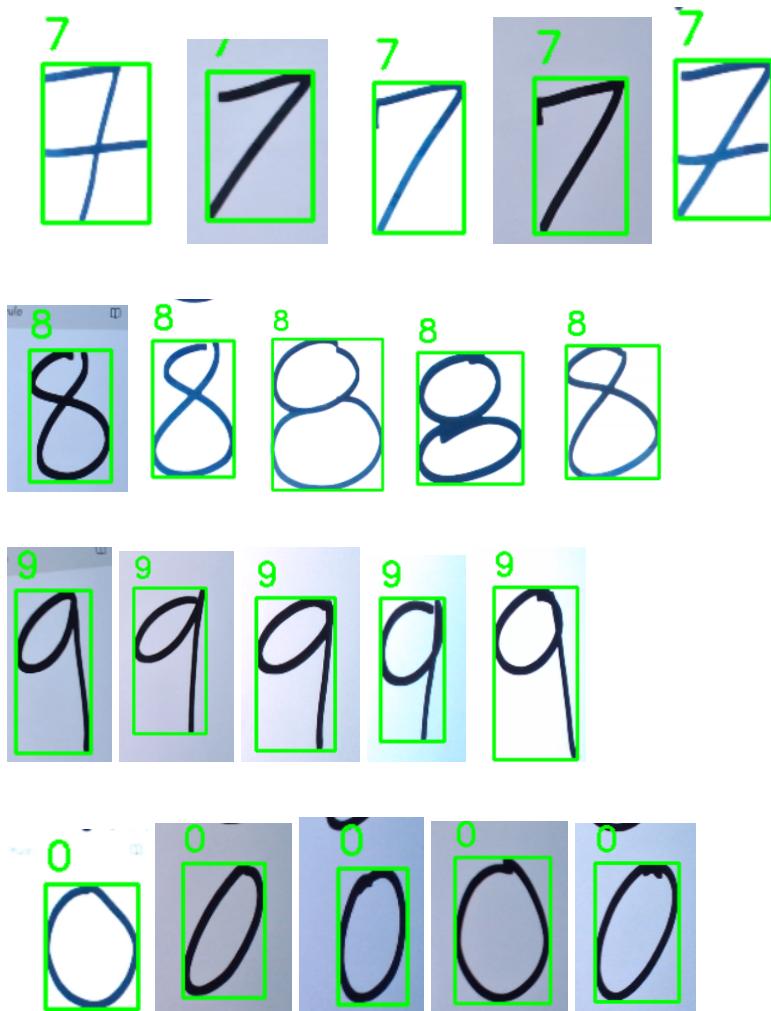
Pruebas Manuales

Además de las pruebas unitarias, se realizaron pruebas manuales para evaluar el desempeño del sistema en condiciones reales. Se llevaron a cabo un total de cinco pruebas por número y por modelo (ROI y TRACE), donde se verificó visualmente si el sistema predecía correctamente los dígitos. Estas pruebas manuales permitieron observar directamente el comportamiento del sistema y confirmar su precisión en la identificación de números escritos a mano y dibujados en el aire.









Cada prueba manual implicó la captura de imágenes de números escritos y trayectorias dibujadas en tiempo real, seguidas de la predicción y visualización de los resultados. Los resultados de estas pruebas fueron satisfactorios, ya que la mayoría de las predicciones fueron correctas. De un total de 100 pruebas realizadas, solo 10 resultaron en predicciones incorrectas, lo que indica que el modelo tiene un alto grado de precisión. Estas pruebas no solo ayudaron a identificar cualquier discrepancia, sino que también proporcionaron retroalimentación valiosa para mejorar aún más el sistema. La baja tasa de error sugiere que el modelo es robusto y confiable para su uso en aplicaciones prácticas, ofreciendo una base sólida para futuras mejoras y optimizaciones.

Conclusiones

Durante el desarrollo del proyecto de reconocimiento de números en tiempo real mediante visión por computadora y SVM, se destacan varias conclusiones significativas. En primer lugar, se logra una impresionante precisión general del 98%, lo que valida la efectividad de las técnicas de preprocesamiento aplicadas y la calidad del modelo SVM optimizado. Sin embargo, se observa una variabilidad en la precisión por clase, lo que sugiere áreas específicas que podrían beneficiarse de mejoras futuras. Además, se destaca la eficiencia del preprocesamiento de imágenes y trayectorias, donde técnicas como la conversión a escala de grises, suavizado y umbralización binaria contribuyen significativamente a la calidad de los datos. Esta uniformidad en el preprocesamiento es crucial para garantizar que las imágenes sean adecuadas para la clasificación mediante SVM, resaltando la importancia de esta etapa en el proceso.

Por otro lado, la versatilidad del algoritmo SVM, especialmente con el kernel polinómico, se revela como una herramienta eficaz en la clasificación de dígitos escritos a mano y trazados en el aire. Esta capacidad para manejar relaciones no lineales y datos de alta dimensionalidad es esencial para aplicaciones que requieren reconocimiento de patrones en diversos contextos y entornos. Asimismo, se reconoce la amplia gama de aplicaciones prácticas del sistema, desde mejorar la accesibilidad hasta el entretenimiento y la educación, con un enfoque especial en su utilidad para personas con discapacidades, facilitando su interacción con dispositivos y software de manera eficiente.

La robustez del sistema se evidencia en su capacidad para operar en condiciones variables de iluminación y representación de números, asegurando su viabilidad en escenarios del mundo real. La implementación conjunta de SVM y PCA permite un rendimiento eficiente y adaptable, sin comprometer la precisión, adaptándose a diversas situaciones y usuarios. Además, se destaca el alto desempeño y la generalización del modelo, respaldados por un promedio ponderado de la precisión, recall y F1-score de 0.98 para todas las clases. Esto demuestra una consistente capacidad del modelo para clasificar correctamente todos los números, manteniendo un alto rendimiento en diferentes contextos y condiciones.

Finalmente, se identifican oportunidades para mejoras futuras, especialmente en la reducción de falsos negativos y positivos en clases específicas. Se sugiere el uso de técnicas avanzadas

de ajuste de hiperparámetros y enriquecimiento del conjunto de datos para abordar estas áreas de mejora. Además, se plantea la exploración de la integración de redes neuronales profundas para mejorar aún más la precisión y robustez del sistema, abriendo nuevas vías para el desarrollo y la innovación en el reconocimiento de números en tiempo real.

Referencias

- Szeliski, R. (2010). **Computer Vision: Algorithms and Applications**. Springer.
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). **Deep Learning**. MIT Press.
- Dix, A., Finlay, J., Abowd, G., & Beale, R. (2004). **Human-Computer Interaction**. Pearson Education.
- Nielsen, J. (1993). **Usability Engineering**. Academic Press.
- Mayer, R. E. (2009). **Multimedia Learning**. Cambridge University Press.
- Papert, S. (1980). **Mindstorms: Children, Computers, and Powerful Ideas**. Basic Books.
- Siciliano, B., & Khatib, O. (2016). **Springer Handbook of Robotics**. Springer.
- Craig, J. J. (2004). **Introduction to Robotics: Mechanics and Control**. Pearson/Prentice Hall.
- Burdea, G. C., & Coiffet, P. (2003). **Virtual Reality Technology**. Wiley-Interscience.
- Sherman, W. R., & Craig, A. B. (2003). **Understanding Virtual Reality: Interface, Application, and Design**. Morgan Kaufmann.
- Rizzo, A. S., & Kim, G. J. (2005). A SWOT analysis of the field of virtual reality rehabilitation and therapy. **Presence: Teleoperators and Virtual Environments**, 14(2), 119-146.
- Weiss, P. L., Keshner, E. A., & Levin, M. F. (Eds.). (2014). **Virtual Reality for Physical and Motor Rehabilitation**. Springer.