

Trabajo Final Inteligencia Artificial I – año 2018: Visión Artifical

Fernández, Gonzalo Gabriel

19 de diciembre de 2018

Resumen

El presente trabajo sobre visión artificial fue realizado como trabajo final de la cátedra Inteligencia Artificial I, de la Facultad de Ingeniería de la Universidad Nacional de Cuyo.

Consiste en el diseño e implementación de un agente capaz de clasificar frutas (bananas, naranjas y limones), en base a fotografías. Se realiza un breve estudio teórico del agente. En la implementación se analiza el procesado de imágenes en sus diferentes etapas: filtrado, segmentado y extracción de características. Para clasificar las imágenes se utilizan algoritmos de aprendizaje K-means y K-nn, y se hace un estudio de desempeño comparativo entre los dos algoritmos. El lenguaje de programación utilizado a lo largo del proyecto es Python, y los diferentes *scripts* de ejecución en *Bash*.

Como objetivo final se desea obtener un agente que en un entorno con condiciones controladas se desempeñe con un rendimiento por encima de las exigencias.

1. Introducción

1.1. Visión Artificial

“La **visión artificial** o visión por ordenador es una disciplina científica que incluye métodos para adquirir, procesar, analizar y comprender las imágenes del mundo real con el fin de producir información numérica o simbólica para que puedan ser tratados por un ordenador.” Wikipedia, la Enciclopedia Libre. “En la visión por ordenador, intentamos describir el mundo que vemos en una o más imágenes e intentamos reconstruir sus propiedades, tal como la forma, iluminación y distribuciones de color.” Szeliski [3]

1.2. Problema a resolver

Se está desarrollando un **sistema de reconocimiento de frutas por visión artificial** con la intención de agilizar el proceso de cobro en las cajas de un supermercado. Para ello se pide desarrollar un agente prototípico que pueda **reconocer bananas, naranjas y limones** a partir de fotografías.

Las condiciones impuestas son que las **imágenes deben tomarse en escala de grises**, y que debe utilizarse los **algoritmos K-means y K-nn** para realizar la clasificación, y a partir de los resultados obtenidos sugerir uno de ellos para ser implementado.

Es recomendado una extracción de características mediante la aplicación de un análisis textural a través de una matriz de covarianza con un vector de características que se construya a partir de los autovalores de dicha matriz.

2. Especificación del agente

La descripción del agente y su entorno se realiza en base a la clasificación de Peter Norving [2], Capítulo 1.

2.1. Tipo de agente

El agente es **racional**: “En cada posible secuencia de percepciones, un agente racional deberá emprender aquella acción que supuestamente maximice su medida de rendimiento, basándose en las evidencias aportadas por la secuencia de percepciones y en el conocimiento que el agente mantiene almacenado.” Peter Norving [2]. Es **no omnisciente**, ya que no conoce el resultado de sus acciones, no posee total información de todo su entorno. El objetivo de utilizar algoritmos como K-means o K-nn es que el agente **aprenda** a lo largo de sus percepciones, y también que sea **autónomo**, y pueda apoyarse más en la experiencia adquirida gracias a sus propias percepciones que en el conocimiento inicial proveído por el diseñador.

El agente es un **agente que aprende**, debido al tipo de algoritmos que utiliza. La base de datos provee un “modelo” del mundo, y los algoritmo también actúan en base a una “utilidad” que se analiza más adelante.

2.2. Tabla REAS: Rendimiento, Entorno, Actuadores, Sensores

- **Tipo de Agente:**

Computadora, microcontrolador o microprocesador en el cuál se encuentran los algoritmos entrenados, y los dispositivos de *entrada y salida*, o más correcto en éste análisis, *sensores y actuadores*: cámara fotográfica, monitor, etc.

- **Medidas de Rendimiento:**

A la hora de implementar los algoritmos, se posee una base de datos con toda la información recopilada etiquetada por clase, en este caso fotografías de bananas, naranjas y limones. A esta base de datos se la divide en dos conjuntos: *entrenamiento* y *testeo*. El conjunto de *testeo* nos permitirá dar una medida de rendimiento del agente, siendo esta medida la cantidad de predicciones correctas de todos los elementos de *testeo*.

- **Entorno:**

El entorno del agente será el área donde esté ubicada la fruta a la hora de tomar la fotografía, y la fruta en sí. Al entorno también lo integran todos los factores externos que se exponen más adelante, como luminosidad, fondo, calidad de la fruta, etc.

- **Actuadores:**

El actuador del agente es sencillamente el medio por el que éste se comunicará con el usuario para darle a conocer su predicción. Puede ser un display, una consola, una señal luminosa o auditiva, o cualquier otro medio que resulte adecuado.

- **Sensores:**

El sensor del agente es la cámara fotográfica que tomará la imagen de la fruta. Esto es así, si se considera la cámara fotográfica como parte del agente. Sin embargo, la imagen puede provenir de otras fuentes externas que no se consideren parte del agente. En este caso, podría considerarse como sensor las herramientas (librerías y sus funciones) que nos permiten procesar la imagen en el formato que se reciba (*PNG, JPEG*), en un formato de información con la que el agente puede operar.

2.3. Propiedades del entorno de trabajo

- **Observable:**

El entorno es observable, ya que el agente posee todos los sensores para que sea posible el correcto funcionamiento de su lógica: preprocesamiento, algoritmos, etc.

- **Episódico:**

El entorno es episódico ya que el estado del entorno en un determinado instante de tiempo no depende de la secuencia de estados previos. La cámara fotográfica tomará imágenes de las frutas en el orden que se le presenten, y este orden no tiene ninguna inferencia en el desempeño del agente.

- **Estático:**

El entorno no cambia mientras el agente está deliberando. Una vez tomada la fotografía, comenzado el procesamiento el agente se aísla en cierto modo del entorno.

- **Discreto:**

Que el entorno sea discreto está relacionado con que también es estático. El agente no interactúa en tiempo real con el entorno, sino solo en el corto lapso en que toma la fotografía.

- **Agente individual:**

El entorno es agente individual ya que en todo el proceso de interacción agente-entorno, el único ente racional es el agente.

3. Planificación del proyecto

Se realizó un análisis de las tareas a realizar para aumentar la productividad del trabajo. Aprovechando la oportunidad, se aplicaron conceptos de planificación aprendidos en la cátedra.

En la figura 1, se expone el **plan de orden parcial** realizado, con las tareas principales a resolver, tanto para el informe como el código en sí.

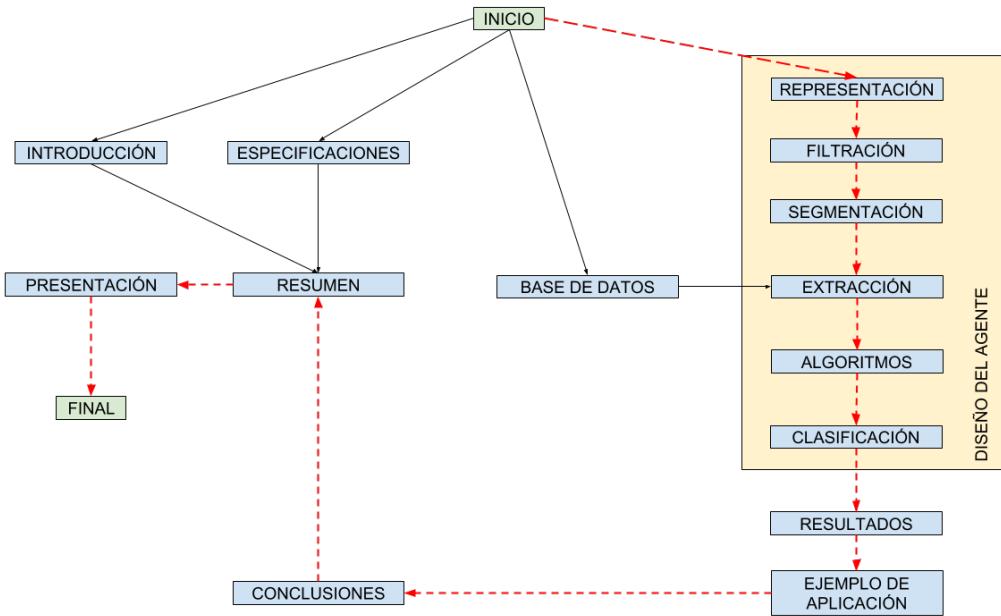


Figura 1: Plan de orden parcial para la realización del proyecto. Las flechas rojas de trazos indican el camino crítico encontrado.

Si se observa la figura 1, las flechas rojas de trazos indican el **camino crítico**, en el cual se encuentran las tareas sin tiempo de relajación, por lo que deben realizarse sin demoras. Éste camino crítico está dado básicamente por la implementación del código.

4. Diseño del agente

Se inicia el proyecto eligiendo con qué lenguaje de programación se encaráará. Se decide utilizar ***Python***, por ser tan popular en las disciplinas relacionadas con inteligencia artificial. Entre las opciones, además de *Python* se consideró utilizar *Matlab* o *R*.

Siendo la primer experiencia con análisis de imágenes, la primer interrogante que surge es cómo obtener de una imagen cruda de la fruta, un vector representativo de todas sus características que sea de utilidad como entrada a los algoritmos K-means y K-nn.

En base a distintas referencias, se decide dividir el problema con cuatro etapas principales: **representación de la imagen** en un arreglo de números fáciles de computar, **filtrar** la imagen para eliminar ruido, **segmentar** la imagen para resaltar características que puede ser relevantes para el posterior análisis y la **extracción de características**, donde como resultado se obtendría un vector representativo de la imagen capaz de ser una de las tantas entradas a los algoritmos antes mencionados, Van Der Walt [4].

Las librerías que se utilizaron para resolver el problema propuesto son *SciPy* y *OpenCV*.

SciPy es un ecosistema basado en *Python* de software *open-source* para matemática, ciencia e ingeniería. Los paquetes que se utilizaron específicamente en este trabajo son:

- **NumPy:** Es un paquete para computación científica. Contiene herramientas como arreglos n-dimensionales, funciones sofisticadas, herramientas para integrar *C/C++* y *Fortran*, álgebra lineal, transformada de Fourier, operaciones random, etc.
- **Matplotlib:** Es una librería para gráficos 2D con calidad de publicación.
- **IPython:** Provee una arquitectura para computación interactiva con una potente consola, un kernel para *Jupyter*, visualización de datos interactiva, etc.

OpenCV (Open Source Computer Vision Library), es una librería de funciones para *C++*, *Python* y otros lenguajes, donde su propósito principal es la visión artificial en tiempo real.

Para exponer el código y la investigación realizada de una forma clara, sensilla y reproducible, se utilizó *Jupyter*. *Jupyter* es una aplicación web *open-source* que permite crear y compartir documentos con bloques de código embebido, ecuaciones, gráficos y texto descriptivo.

La descripción más específica y técnica de la implementación se encuentra en los diferentes *Jupyter notebook* del proyecto. Para acceder a ellos, ir al link provisto correspondiente.

4.1. Representación adecuada de la información

Jupyter notebook: 01-representacion. En el caso de no disponer del proyecto completo: link de GitHub.

Al analizar la conversión a escala de grises se utiliza la fórmula:

$$Y'_{709} = 0,2125R' + 0,7154G' + 0,0721B' \quad (1)$$

La ecuación 1 se obtiene de Szeliski [3], capítulo 2 “Image Formation”, 2.3 “The digital camera”.

El resultado obtenido en esta etapa es la imagen en escala de grises en un tamaño normalizado, a partir de la fotografía original recibida por el agente. Se puede observar un ejemplo en la figura 2. El proceso se puede hacer por medio de la librería *skimage* o también con la librería *OpenCV* (más específicamente *cv2*).

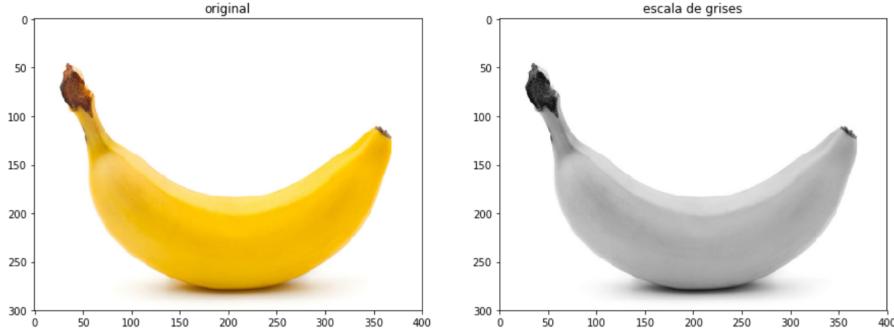


Figura 2: Resultado obtenido en *Jupyter notebook* 01-representacion.

4.2. Filtrado de imágenes

Jupyter notebook: 02-filtracion. En el caso de no disponer del proyecto completo: link de GitHub.

De *Wikipedia, la Enciclopedia Libre*: “Es el conjunto de técnicas englobadas dentro del preprocesamiento de imágenes cuyo objetivo fundamental es obtener, a partir de una imagen origen, otra final cuyo resultado sea más adecuado para una aplicación específica mejorando ciertas características de la misma que posibilite efectuar operaciones del procesado sobre ella. Los principales objetivos que se persiguen con la aplicación de filtros son:

- Suavizar la imagen: reducir la cantidad de variaciones de intensidad entre píxeles vecinos.
- Eliminar ruido: eliminar aquellos píxeles cuyo nivel de intensidad es muy diferente al de sus vecinos y cuyo origen puede estar tanto en el proceso de adquisición de la imagen como en el de transmisión.
- Realzar bordes: destacar los bordes que se localizan en una imagen.
- Detectar bordes: detectar los píxeles donde se produce un cambio brusco en la función intensidad.

Por tanto, se consideran los filtros como operaciones que se aplican a los píxeles de una imagen digital para optimizarla, enfatizar cierta información o conseguir un efecto especial en ella.”.

En este trabajo se estudiaron los siguientes filtros:

- *Gaussian*: Altamente recomendado y utilizado ampliamente para eliminar ruido, reducir detalle de las fotografías, y como una etapa de preprocesamiento de imágenes.
- *Diferencial*: Se utiliza para obtener los cambios en la intensidad de la imagen. El resultado es el gradiente, en la dirección que se realizó el filtro, de la intensidad de la imagen. Es un primer paso al detectar bordes de la imagen.
- *Sobel*: Es un filtro muy utilizado para la detección de bordes. Es una aproximación al gradiente de la imagen. En el trabajo se realiza una comparación entre el filtro *Sobel* y la composición de dos filtros *diferenciales* ortogonales, obteniendo resultados similares. En la librería *OpenCV* a esta composición se le denomina filtro *Laplacian*, por estar relacionado con la ecuación de Laplace.
- *Median*: Este filtro a diferencia de los anteriores es no lineal. Se utiliza de una forma similar al filtro *Gaussian*, para reducir ruido en las imágenes.

Los resultados obtenidos en esta etapa es la imagen en escala de grises (o no) en un tamaño normalizado, filtrada por *Gaussian* y/o *Sobel* dependiendo la necesidad, a partir de la fotografía original recibida por el agente. Se puede observar un ejemplo en la figura 3. El proceso se puede hacer por medio de la librería *skimage* o también con la librería *OpenCV* (más específicamente *cv2*).

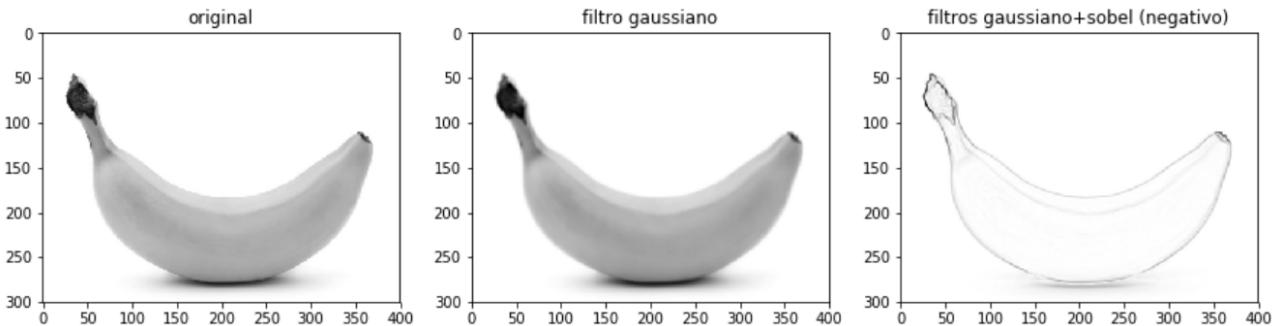


Figura 3: Resultado obtenido en *Jupyter notebook* 02-filtracion.

4.3. Segmentado de imágenes

Jupyter notebook: 03-segmentacion. En el caso de no disponer del proyecto completo: link de GitHub.

El resultado del proceso de segmentación, es una imagen binaria que contendría los bordes del objeto a analizar. Esto puede utilizarse o no, según el método de extracción de características que se utilice. Se puede observar un ejemplo en la figura 4. El proceso se puede hacer por medio de la librería *skimage* o también con la librería *OpenCV* (más específicamente *cv2*).

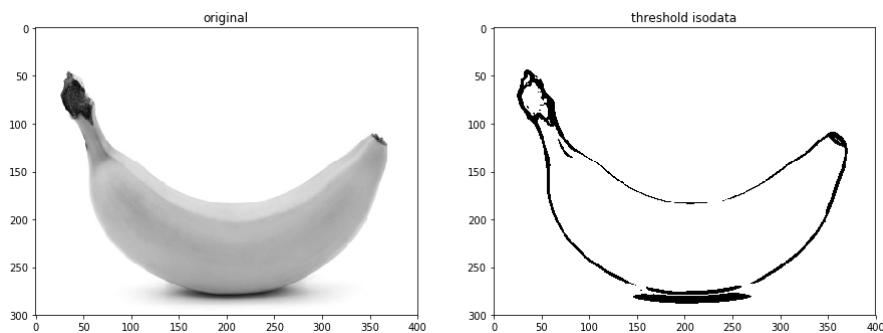


Figura 4: Resultado obtenido en *Jupyter notebook* 03-segmentacion.

4.4. Base de datos

A la hora de obtener resultados se utilizó dos bases de datos: La primera es una base de datos proveída por Kaggle: Fruit 360 Kaggle dataset. Kaggle es una comunidad online de *Data Science* y *Machine Learning* de *Google, Inc.*. Permite a los usuarios encontrar y publicar *data sets*, trabajar con otras personas expertas en el tema e inclusive participar en competencias y desafíos relacionados con el análisis de datos. El problema de éste *data set*, es que las frutas están diseñadas con herramientas de software, es decir, no son fotografías de frutas reales. Ésto hace que la diferencia con una fotografía tomada por el usuario sea muy grande. Sin embargo, fue de mucha utilidad para una primera observación de resultados y confirmar que las decisiones tomadas hasta el momento son acertadas. Luego de trabajar con este *data set* de Kaggle, se construyó uno propio a partir de fotografías de frutas reales comprobando esta gran diferencia de resultados. En el cuadro 1, puede observarse la cantidad de imágenes utilizada.

	Train			Test			Total
	Banana	Naranja	Limón	Banana	Naranja	Limón	
Kaggle	490	479	492	166	160	164	1951
Personal	132	138	130	131	138	130	799

Cuadro 1: Cantidad de imágenes en los diferentes *data sets*.

Para las imágenes de la base de datos propia, se tomaron las fotografías de las diferentes frutas individuales, sobre una hoja de papel blanca, con *flash* y buena iluminación del ambiente.

4.5. Extracción de características

Jupyter notebook: 04-extraccion. En el caso de no disponer del proyecto completo: link de GitHub.

Para la extracción de características se utilizaron diferentes métodos:

- **Histogram of oriented gradients (HOG):** Es utilizado en visión por computador y procesado de imágenes con el propósito de detección de objetos. Resultados obtenidos (reducidos a media y desviación estándar) en la figura 5.

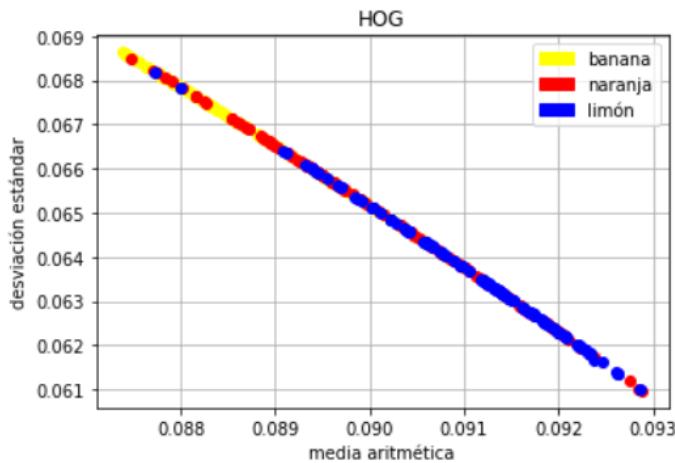


Figura 5: Histogram of Oriented Gradients (HOG) para la base de datos, reducida a media y desviación estándar.

- **Hu Moments:** Generalmente es utilizado para la extracción de la silueta o contorno del objeto en la imagen. Da una valorización de la forma de éste, que puede ser medida comparando vectores de características *Hu Moments* a través de distancias, por ejemplo. Resultados obtenidos (reducidos a media y desviación estándar) en la figura 6.

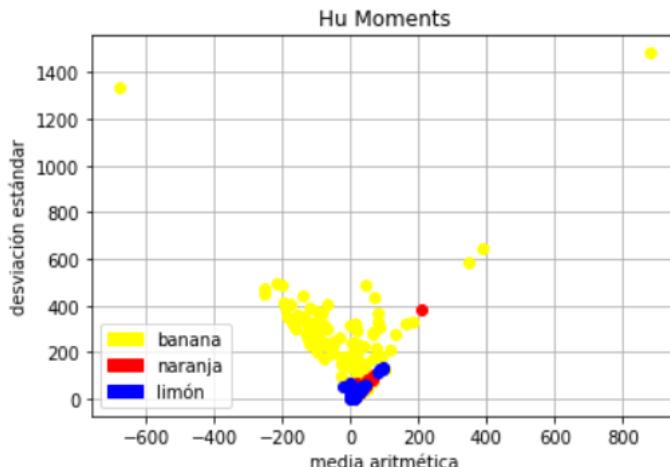


Figura 6: Hu Moments para la base de datos, reducida a media y desviación estándar.

- **Haralick Textures:** Las características de textura están basadas en estadísticas que resumen la distribución de frecuencia relativa (que describe cuán a menudo un tono gris aparece en una zona espacial específica en relación a otro tono de gris en la imagen), Haralick [1]. Resultados obtenidos (reducidos a media y desviación estándar) en la figura 7.
- **Color Histogram:** Es sencillamente una representación de la distribución de colores en la imagen. En imágenes digitales, el histograma de color representa el número de píxeles que tienen determinado color en una lista fija con el rango de colores. Resultados obtenidos (reducidos a media y desviación estándar) en la figura 8.

4.6. Implementación de los algoritmos de clasificación

Jupyter notebook: 05-algoritmos. En el caso de no disponer del proyecto completo: link de GitHub.

En esta etapa se implementan los dos algoritmos de clasificación a utilizar: **K-nn (K-nearest neighbors)** y **K-means**. Su descripción técnica se encuentra detalladamente en el *jupyter notebook* correspondiente.

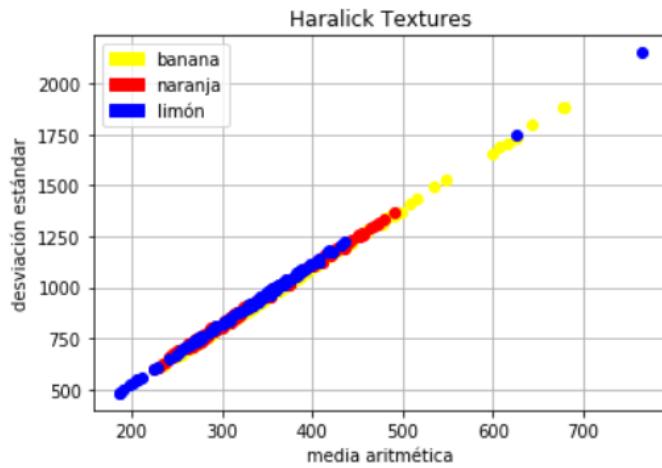


Figura 7: Haralick Textures para la base de datos, reducida a media y desviación estándar.

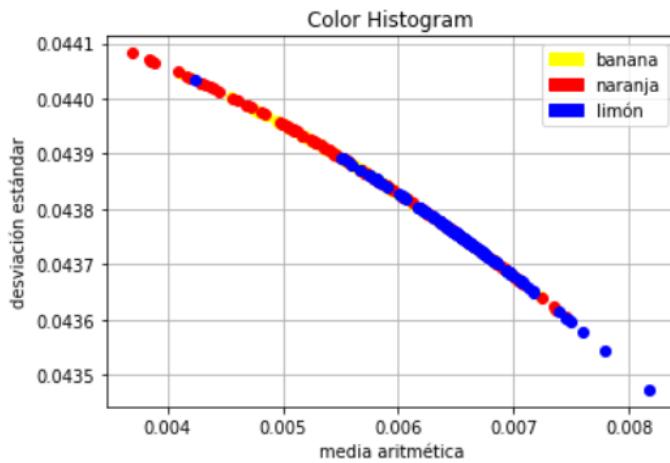


Figura 8: Color Histogram para la base de datos, reducida a media y desviación estándar.

4.7. Clasificación de la información

Jupyter notebook: 06-clasificacion. En el caso de no disponer del proyecto completo: link de GitHub.

En este último paso, a diferencia de la implementación inicial, no se trabaja con una imagen ejemplo, sino con toda la base de datos. Se divide toda la información en dos subconjuntos, uno para el entrenamiento de los algoritmos y otro de testeo para calificar su desempeño.

Se ejecutó numerosas veces con el objetivo de comparar resultados utilizando filtros, segmentación y diferentes métodos de extracción de características. Los resultados obtenidos se exponen en la sección 6.

5. Código

El código de todo lo previamente explicado se encuentra expuesto junto al análisis teórico, en los diferentes *Jupyter notebook*. En las figuras 9 y 10, se muestra en diagrama de flujo las etapas generales de los algoritmos *K-nn* y *K-means*.

El código final obtenido posterior a la etapa de diseño se encuentra en la carpeta src del proyecto. Consiste en un conjunto de programas *python* que ejecutan las diferentes etapas. Sin embargo, para simplicidad de ejecución, se escribió dos *scripts* en *bash* nombrados *run-test* y *run-project*, que ejecutan los programas en *python* en la secuencia correcta junto con otras acciones necesarias para una ejecución limpia.

- **run-test:** Ejecutar el *script* para entrenar la base de datos y proceder con el testeo del proyecto. Una barra de progreso provee de forma gráfica los resultados finales obtenidos. Se ejecutan ambos algoritmos *K-means* y *K-nn*. El comando es, desde la carpeta principal del proyecto donde se encuentra *run-test*, como el que sigue:

```
./run-test
```

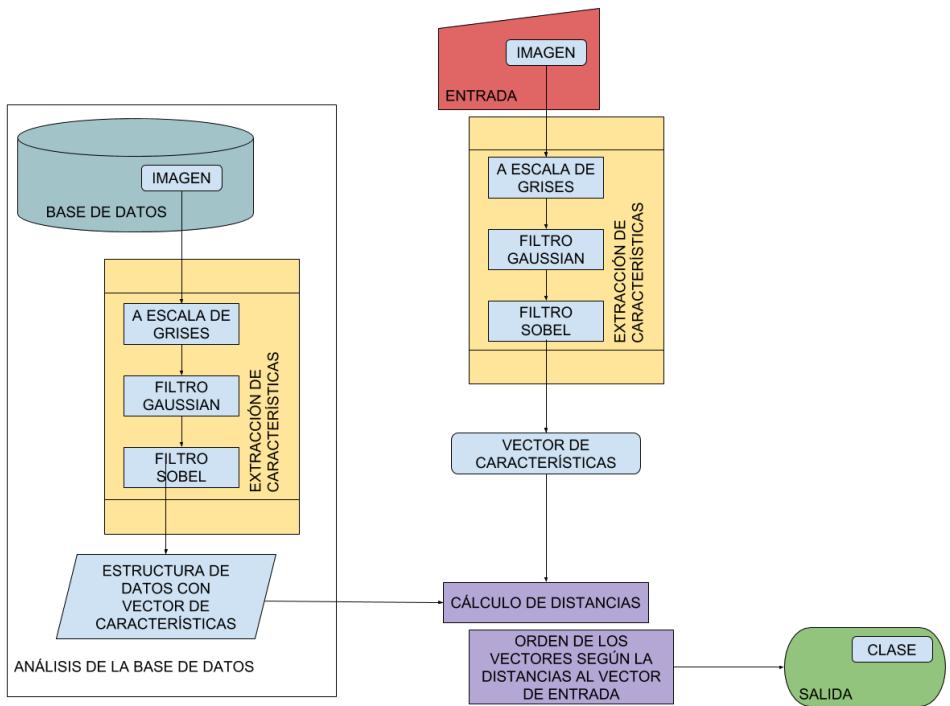


Figura 9: Diagrama de flujo del algoritmo *K-nn*.

- **run-knn:** Ejecutar el *script* para clasificar una determinada imagen con algoritmo *K-nn*. La ruta donde se encuentra esta imagen es un parámetro del programa. El comando es, desde la carpeta principal del proyecto donde se encuentra *run-knn*, como el que sigue:

```
./run-knn <ruta de la imagen a clasificar>
```

Las **dependencias**, es decir, las librerías necesarias de *Python* para poder ejecutar el proyecto son: *Numpy*, *Skimage*, *cv2* (OpenCV), *Mahotas*, *Pickle*, *sys* y *random*.

6. Resultados

Los resultados obtenidos son en referencia a la base de datos utilizada. Como se mencionó previamente, se divide en dos subconjuntos de imágenes: uno de entrenamiento y otro de testeо. Los resultados se dan en **porcentaje de aciertos en predicciones del conjunto de testeо**.

El objetivo del conjunto de testeо es dar una noción de que métodos de extracción de características, filtros y segmentados, dan mejores resultados y tomar una decisión en el diseño final del agente en base a ello.

En el cuadro 2 se exponen los resultados obtenidos con los diferentes *data sets* para el algoritmo *K-nn*.

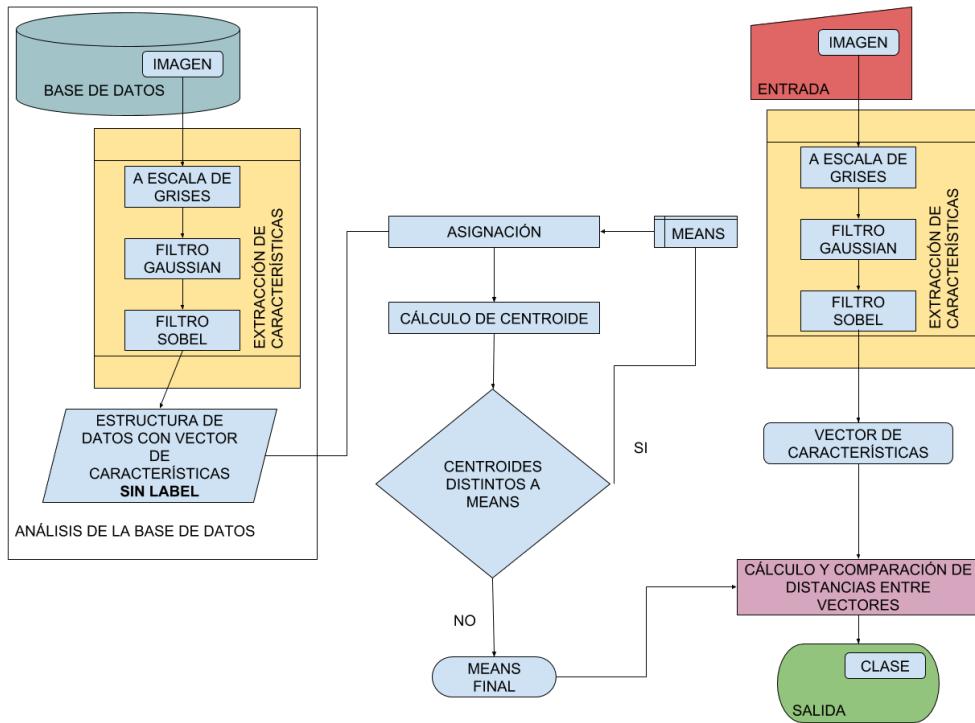


Figura 10: Diagrama de flujo del algoritmo *K-means*.

Análisis de imagen		Kaggle		Personal	
Método de extracción de características	Filtro	$K = 1$	$K = 10$	$K = 1$	$K = 10$
Haralick Textures	Gaussian	83.5 %	65.7 %	62.2 %	49.1 %
	Ninguno	74.9 %	63.3 %	62.7 %	52.4 %
Hu Moments	Gaussian + Sobel	63.3 %	65.9 %	76.9 %	76.7 %
	Sobel	93.5 %	93.5 %	59.9 %	71.7 %
Color Histogram	Gaussian	94.7 %	100.0 %	68.4 %	72.4 %
	Ninguno	94.5 %	100.0 %	67.9 %	72.4 %
Histogram of Oriented Gradients	Gaussian	72.4 %	72.0 %	47.4 %	53.9 %
	Ninguno	72.4 %	77.3 %	46.4 %	51.4 %

Cuadro 2: Resultados obtenidos para algoritmo *K-nn*, para diferentes *data sets*; utilizando diferentes métodos de extracción de características y filtros. En porcentaje, la cantidad de predicciones acertadas sobre el total de imágenes en el conjunto de test.

De los resultados puede observarse como aplicar un filtro en el *data set* brindado por Kaggle disminuye la efectividad. Se concluye que es debido a que las imágenes son diseñadas por software, no hay ruido y el filtro lo único que logra es hacer la imagen más borrosa. También puede observarse lo mucho que disminuyen los aciertos al mudarse de un *data set* como el de Kaggle a uno con fotografías propias. Aún así, la diferencia relativa entre métodos de extracción de características y filtros, se mantiene aproximadamente constante. Esto permite poder elegir una combinación entre todas las opciones y estimar que el resultado se puede extrapolar si hay un crecimiento en la cantidad de imágenes en el *data set*, por ejemplo. Otra observación, es que es conveniente utilizar $K = 10$ por sobre $K = 1$ cuando el *data set* es el de fotografías propias.

Para el algoritmo *K-means* se obtuvo los resultados del cuadro 3.

Puede observarse como la cantidad de aciertos para este algoritmo es menor que para el algoritmo *K-nn*.

Uno de los motivos por los que el algoritmo *K-means* no tiene tan buen rendimiento, y que también afecta al algoritmo *K-nn*, es que las diferentes clases no forman conjuntos de puntos con límites definidos. Esto puede observarse en las figuras 5, 6, 7 y 8. Los subconjuntos de puntos se superponen, entonces al ejecutarse el algoritmo, por cómo está diseñado, una de las clases tiene predominancia sobre otra (esto puede observarse bien en las distribuciones finales obtenidas en el cuadro 3). Esta superposición de conjuntos lógicamente afecta al

Método de extracción de características	Filtro	Aciertos / Distribución final
Haralick Textures	Gaussian	48.6 % (209, 138, 53)
	Ninguno	43.4 % (36, 144, 220)
Hu Moments	Gaussian + Sobel	41.9 % (4, 353, 43)
	Sobel	40.1 % (4, 45, 351)
	Gaussian	69.4 % (157, 112, 131)
	Ninguno	63.2 % (157, 131, 112)
Color Histogram	Gaussian	49.6 % (60, 142, 198)
	Ninguno	52.4 % (67, 133, 200)
Histogram of Oriented Gradients	Gaussian	59.4 % (129, 128, 143)
	Ninguno	26.8 % (139, 137, 124)

Cuadro 3: Resultados obtenidos para el algoritmo *K-means* sobre el *data set* con fotografías propias, utilizando diferentes métodos de extracción de características y filtros. En porcentaje, la cantidad de predicciones acertadas sobre el total de imágenes en el conjunto de test. Debajo del porcentaje, la cantidad de imágenes en cada clase al finalizar el algoritmo.

algoritmo *K-nn* al encerrar los vecinos más cercanos en una circunferencia.

Luego de elegir el algoritmo *K-nn* como el algoritmo con el que se obtienen mejores resultados, y el método *Hu Moments* para extracción de características; se comparó el vector de características crudo, de 7 componentes, con el vector de dos componentes que son la media aritmética y la desviación estándar. El vector de características de 7 componentes da por resultado un **85.0 %** de predicciones correctas en el *data set* de fotografías propias, frente al 76.7 % de predicciones correctas con el vector de 2 componentes. Luego de comprobar esto, con el objetivo de obtener conjuntos con límites aún más marcados, se compararon combinaciones de componentes del vector *Hu Moments*, obteniendo resultados de 91,2 % con componentes 0 y 1; y 95,0 % con **tres componentes 0, 1 y 3** como se observa en la figura 11.

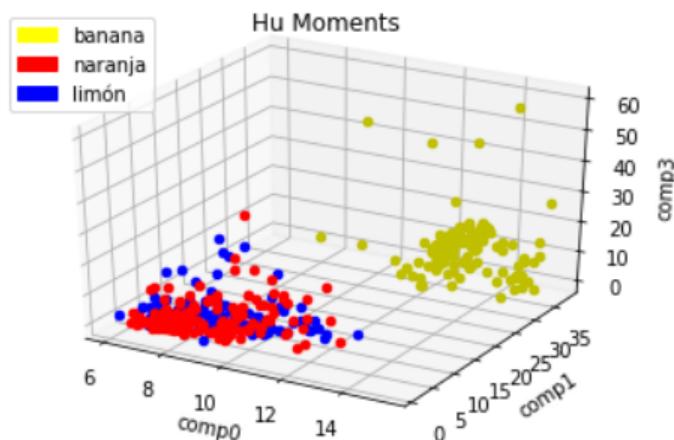


Figura 11: Vector de características formado por componentes 0, 1 y 3 del vector *Hu Moments*.

Para la elección del *K* del algoritmo, se ejecuto repetidas veces variando este parámetro desde 1 a 25, obteniendo los valores de la figura 12.

Como puede observarse, los valores más altos se encuentran entre $K = 20$ y $K = 30$, por lo que se termina adoptando un valor de **K igual a 24**.

Con ésta última forma de extraer características de las imágenes, el algoritmo *K-means* tiene un porcentaje

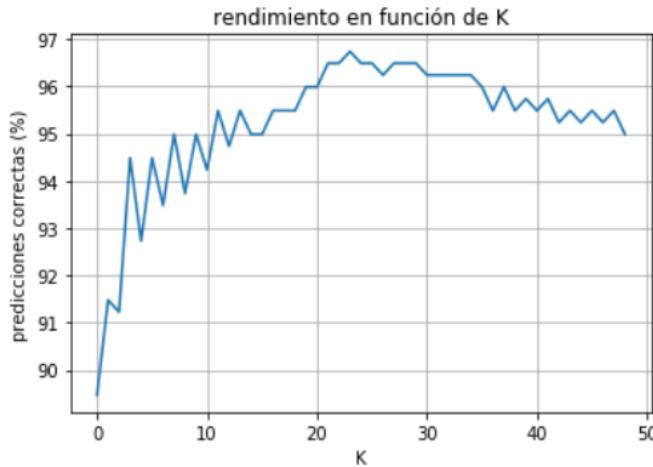


Figura 12: Porcentaje de predicciones correctas en función de K para el algoritmo $K\text{-}nn$.

de aciertos del 95,0 %, por lo que todavía no se descarta su uso. Dado el gran parecido con los resultados de $K\text{-}nn$, se procede a observar casos particulares para decidir que algoritmo utilizar.

En el cuadro 4, se puede observar la gran diferencia de tiempo de ejecución entre los algoritmos, la cuál es una ventaja importante del algoritmo $K\text{-means}$.

K-nn			Clasificación: 0m 6.387s (Variable con el tamaño de la base de datos)
K-means	Análisis de base de datos: 1m 34.254s	Entrenamiento del algoritmo: 0m 1.279s (Variable con el tamaño de la base de datos)	Clasificación: 0m 0.551s

Cuadro 4: Comparación del tiempo de ejecución de algoritmos $K\text{-}nn$ y $K\text{-means}$, en igualdad de condiciones.

7. Ejemplo de aplicación

A continuación, se muestra como ejemplo la ejecución del proyecto para clasificar una fruta de cada clase, figura 13.



(a) Banana



(b) Naranja



(c) Limón

Figura 13: Fotografías de frutas a clasificar.

Para algoritmo $K\text{-}nn$:

- Banana: Se obtiene en consola la figura 14a. Como puede observarse, por ser la primera ejecución, el archivo *dataset.pkl* que contiene la base de datos analizada no existe. Entonces, el programa procede a analizar la base de datos y luego ejecutar el algoritmo. La penúltima línea da la clase final, en este caso "banana", y la última línea indica en porcentajes la distribución de los K vecinos más cercanos.
- Naranja: Se obtiene en consola la figura 14b. Ahora como es la segunda ejecución, y el archivo *dataset.pkl* ya existe, el programa directamente ejecuta el algoritmo utilizando la base de datos ya analizada, y da la

respuesta. Puede apreciarse como entre naranjas y limones la diferencia de características es menor, y la respuesta final ahora es con un 87% de seguridad.

- Limón: Se obtiene en consola la figura 14c, donde nuevamente no es necesario analizar la base de datos porque el archivo *dataset.pkl* con esta información ya existe.

```
$ ./run-knn examples/test-banana.jpg
Data not found. Feature extraction from data.
Loading banana data: |██████████| 100.0% Complete
Banana data is ready
Loading orange data: |██████████| 100.0% Complete
Orange data is ready
Loading lemon data: |██████████| 100.0% Complete
Lemon data is ready
Data analysis completed
banana
100 banana 0 orange 0 lemon
```

(a) Análisis con algoritmo *K-nn*, de la fotografía 13a.

```
$ ./run-knn examples/test-orange.jpg
orange
0 banana 87 orange 12 lemon
```

(b) Análisis con algoritmo *K-nn*, de la fotografía 13b.

```
$ ./run-knn examples/test-lemon.jpg
lemon
0 banana 0 orange 100 lemon
```

(c) Análisis con algoritmo *K-nn*, de la fotografía 13c.

Figura 14: Ejemplos de aplicación del algoritmo *K-nn*

Para algoritmo *K-means*:

- Banana: Se obtiene en consola la figura 15a. Como puede observarse, por ser la primera ejecución, los archivos *dataset.pkl* que contiene la base de datos analizada y *means.pkl* con los tres vectores *means* finales, no existen. Entonces, el programa procede a analizar la base de datos y entrenar el algoritmo para conseguir los *means*. Luego, ejecuta el algoritmo. La última línea es la clase asignada a la imagen de entrada.
- Naranja: Se obtiene en consola la figura 15b. Ahora como es la segunda ejecución, y los archivos *dataset.pkl* y *means.pkl* ya existen, el programa directamente ejecuta la clasificación utilizando los vectores en el archivo *means.pkl*, y da la respuesta.
- Limón: Se obtiene en consola la figura 15c, donde nuevamente no es necesario analizar la base de datos porque el archivo *dataset.pkl* con esta información ya existe, ni entrenar el algoritmo porque los vectores a utilizar se encuentran en *means.pkl*.

```
$ ./run-kmeans examples/test-banana.jpg
Data not found. Feature extraction from data.
Loading banana data: |██████████| 100.0% Complete
Banana data is ready
Loading orange data: |██████████| 100.0% Complete
Orange data is ready
Loading lemon data: |██████████| 100.0% Complete
Lemon data is ready
Data analysis completed
Means not found. Computing means.
banana
```

(a) Análisis con algoritmo *K-means*, de la fotografía 13a.

```
$ ./run-kmeans examples/test-orange.jpg
orange
```

(b) Análisis con algoritmo *K-means*, de la fotografía 13b.

```
$ ./run-kmeans examples/test-lemon.jpg
lemon
```

(c) Análisis con algoritmo *K-means*, de la fotografía 13c.

Figura 15: Ejemplos de aplicación del algoritmo *K-means*

8. Conclusiones

Con todo lo estudiado, puede concluirse que el algoritmo a implementar es el **K-nn**, con **K igual a 24**; utilizando **filtro Gaussian** con σ igual a 1 para la eliminación de ruido y **filtro Sobel** para la detección de bordes; y así luego utilizar las **componentes 0, 1 y 3** del vector obtenido con **Hu Moments** para la extracción de características.

El algoritmo *K-nn* dio mejores resultados, sin embargo una de sus mayores desventajas al compararlo con el algoritmo *K-means* es el **tiempo de ejecución**, ver cuadro 4. Si recordamos el algoritmo *K-nn*, compara el vector de características de la imagen a clasificar con el vector de características de cada una de las imágenes en el *data set*, mientras que *K-means* solo calcula la distancia a los tres diferentes means a la hora de realizar la clasificación, separando el proceso de clasificación del de entrenamiento. Por lo tanto, si la velocidad necesaria a la hora de clasificar una fruta es determinante, y el algoritmo *K-nn* no cumple con este requerimiento, será necesario seguir estudiando el algoritmo *K-means* y todas las formas posibles de optimizarlo.

Respecto al punto anterior, un trabajo posterior al proyecto es ampliar y mejorar la base de datos utilizada, tanto en extensión como en calidad de fotografías. Una base de datos más extensa mejorará el nivel de desempeño de los algoritmos, pero también hay que tener en cuenta que aumentará el tiempo de ejecución del algoritmo *K-nn*. La **relación de compromiso entre tamaño de la base de datos y tiempo de ejecución** también determinará la elección de uno de los dos algoritmos.

Dentro del algoritmo *K-nn* hay un algoritmo de ordenamiento de los vectores de características según la distancia euclíadiana al vector de características de la imagen a clasificar. A la fecha de presentación, el algoritmo de ordenamiento implementado es *bubblesort*, y hay que contemplar que este algoritmo es de complejidad n^2 . Ésto afectará directamente sobre el tiempo de ejecución a la hora de decidir el tamaño de la base de datos. Por lo tanto es necesario estudiar el problema, y elegir un algoritmo de ordenamiento con una complejidad que se adecue a los requerimientos.

Además del tiempo de ejecución, es necesario estudiar de una forma más detallada los métodos de extracción de características analizados e incorporar nuevos. Los métodos de extracción de características fue la principal dificultad presente en el proyecto, y es determinante ya que define los subconjuntos de puntos de entrada de los algoritmos de clasificación. Es fundamental la buena elección de uno o más métodos (en el proyecto se utilizó un método, pero perfectamente se puede trabajar con más de una característica) y también procesar la imagen con los filtros y/o segmentaciones correctas e indicadas para dicho método.

Otro aspecto a mejorar en el proyecto es el modo en que se toma las fotografías. Es muy probable que las condiciones mejoren mucho, por ejemplo, si se las toma sobre fondo negro.

Referencias

- [1] Robert M. Haralick. "Textural Features for Image Classification". En: (1973).
- [2] Stuart Russell y Peter Norving. *Inteligencia Artificial: Un Enfoque Moderno*. Pearson Educación S. A., 2004.
- [3] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer, 2010.
- [4] Stefan Van Der Walt. *Image Analysis in Python with SciPy and scikit-image — SciPy 2018 Tutorial — Stefan van der Walt*. 2018. URL: <https://www.youtube.com/watch?v=arXiv-TM7DY&t=4343s&list=LLbi4i-j4bBUGaeJvDIn96aw&index=27> (visitado 12-07-2018).

Índice

1. Introducción	1
1.1. Visión Artificial	1
1.2. Problema a resolver	1
2. Especificación del agente	1
2.1. Tipo de agente	1
2.2. Tabla REAS: Rendimiento, Entorno, Actuadores, Sensores	2
2.3. Propiedades del entorno de trabajo	2
3. Planificación del proyecto	3
4. Diseño del agente	3
4.1. Representación adecuada de la información	4
4.2. Filtrado de imágenes	4
4.3. Segmentado de imágenes	5
4.4. Base de datos	5
4.5. Extracción de características	6
4.6. Implementación de los algoritmos de clasificación	7
4.7. Clasificación de la información	7
5. Código	8
6. Resultados	8
7. Ejemplo de aplicación	11
8. Conclusiones	12