UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIERÍA INDUSTRIAL VALENCIA

TRABAJO FIN DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALE

DISEÑO, IMPLEMENTACIÓN Y EVALUACIÓN DE TÉCNICAS DE CLASIFICACIÓN DE PERAS MEDIANTE TÉCNICAS DE VISIÓN POR COMPUTADOR

AUTOR: JOSÉ FERNÁNDEZ NIETO

TUTOR: ANTONIO JOSÉ SÁNCHEZ SALMERÓN

Curso Académico: 2020-21

AGRADECIMIENTOS

A mi familia, sin vuestro apoyo y cariño no habría llegado hasta aquí.

A mis amigos que me han acompañado y ayudado a lo largo de este camino.

Y a mi tutor por introducirme en este asombroso campo
y por su ayuda indispensable en el trabajo.

RESUMEN

El objetivo de este trabajo consiste en el diseño, implementación y evaluación de diferentes técnicas de clasificación de imágenes basadas en visión artificial con la finalidad de poder analizar la posible automatización del proceso de control de calidad de peras en línea, durante el proceso de encajado en un almacén.

Para la realización del trabajo se utiliza un dataset de imágenes de peras, las cuales están etiquetadas en diferentes categorías proporcionadas por el Instituto de Automática e Informática Industrial (ai2) de la UPV, dependiendo de la calidad de la fruta. El dataset de partida etiquetado manualmente será editado para corregir posibles errores, se evaluarán diferentes técnicas de clasificación (Bayesiano, KNN, SVM, …) y de extracción de características analizando la posible automatización de este proceso de inspección visual.

Para la obtención de los resultados deseados, se utiliza Open CV, entre otras librerías de Python, la cual nos permite analizar imágenes y extraer las características que se utilizan para la creación de las diferentes técnicas de clasificación.

Para finalizar, se analizarán los resultados obtenidos de las diferentes técnicas, así como las características extraídas de las imágenes, con el fin de maximizar alguna métrica como por ejemplo la tasa de aciertos. También se estudiará el modo de instalación en el almacén para su correcto funcionamiento.

Palabras clave: Automatización, Visión Artificial, Control de calidad, Clasificación de peras

RESUM

L'objectiu d'aquest treball consisteix en el disseny, implementació i avaluació de diferents tècniques de classificació d'imatges basades en visió artificial amb la finalitat de poder analitzar la possible automatització del procés de control de qualitat de peres en línia, durant el procés d'encaixat en un magatzem.

Per a la realització del treball s'utilitza un dataset d'imatges de peres, les quals estan etiquetades en diferents categories proporcionades per l'Institut d'Automàtica i Informàtica Industrial (ai2) de la UPV, depenent de la qualitat de la fruita. El dataset de partida etiquetat manualment serà editat per a corregir possibles errors, s'avaluaran diferents tècniques de classificació (Bayesià, KNN, SVM, …) i d'extracció de característiques analitzant la possible automatització d'aquest procés d'inspecció visual.

Per a l'obtenció dels resultats desitjats, s'utilitza Open CV, entre altres llibreries de Python, la qual ens permet analitzar imatges i extraure les característiques que s'utilitzen per a la creació de les diferents tècniques de classificació.

Per a finalitzar, s'analitzaran els resultats obtinguts de les diferents tècniques, així com les característiques extretes de les imatges, amb la finalitat de maximitzar alguna mètrica com per exemple la taxa d'encerts. També s'estudiarà la manera d'instal·lació en el magatzem per al seu correcte funcionament.

Paraules clau: Automatització, Visió Artificial, Control de qualitat, Classificació de peres

SUMMARY

The objective of this work consists of the design, implementation and evaluation of different image classification techniques based on artificial vision in order to analyze the possible automation of the quality control process of pears in line, during the packing process in a warehouse.

To carry out the work, a dataset of images of pears is used, which are labeled in different categories provided by the Institute of Automation and Industrial Informatics (ai2) of the UPV, depending on the quality of the fruit. The manually labeled starting dataset will be edited to correct possible errors, different classification techniques (Bayesian, KNN, SVM,…) and feature extraction will be evaluated, analyzing the possible automation of this visual inspection process.

To obtain the desired results, Open CV is used, among other Python libraries, which allows us to analyze images and extract the characteristics used to create the different classification techniques.

Finally, the results obtained from the different techniques will be analyzed, as well as the characteristics extracted from the images, in order to maximize some metric such as the hit rate. The way of installation in the warehouse will also be studied for its correct operation.

Key words: Automation, Artificial Vision, Quality Control, Pear Sorting

Índice de la memoria

Listado de Figuras

Listado de Gráficas

Listado de Ecuaciones

MEMORIA DESCRIPTIVA

Capítulo 1: Introducción

1.1 Introducción general

En la actualidad, la inteligencia artificial se ha convertido en un campo muy importante en la automatización del mundo industrial. La inteligencia artificial consiste en la estimación y utilización de modelos matemáticos, implementados a través de algoritmos, con el fin de crear máquinas "inteligentes". Una de sus ramas más importantes es el Aprendizaje automático (*Machine Learning*), que consiste en permitir a las máquinas aprender de manera autónoma. Dentro del Aprendizaje automático, las técnicas basadas en visión por computador están permitiendo la automatización de muchos procesos de inspección visual.

Con el desarrollo de estas nuevas tecnologías, se está consiguiendo una gran mejora en la productividad a raíz de poder analizar los procesos de manera continua y automática. Su utilización puede suponer un aumento en la calidad del producto final, así como un ahorro económico notable.

Este Trabajo de Fin de Grado consiste en la utilización de diferentes técnicas de clasificación basadas en técnicas de visión artificial con el fin de automatizar la clasificación de peras durante el proceso de manipulación y encajado en un almacén

1.2 Objetivos

El principal objetivo del presente trabajo es diseñar e implementar diferentes técnicas de visión por computador para realizar la automatización de una línea de manipulación de peras, por lo que se debe diseñar una preselección entre peras aptas y no aptas y una posterior clasificación de las diferentes categorías de peras devoe en función de su calidad.

Para alcanzar este objetivo se han establecido los siguientes objetivos secundarios:

- Investigación y aprendizaje en el campo de la visión por ordenador, incluyendo clasificadores de aprendizaje supervisado y redes neuronales para la clasificación de objetos.
- Aprendizaje y utilización del entorno de trabajo *Google Colab*.
- Comprensión de diferentes librerías de *Python* como *OpenCV* o *PyTorch*.
- Diseño, implementación y evaluación de un método de procesado de imágenes.
- Diseño, implementación y evaluación de la etapa de extracción de características de interés.
- Diseño, implementación, entrenamiento y validación de una red neuronal para la clasificación de imágenes.
- Evaluación de los resultados conseguidos.
- Análisis de las conclusiones.

1.3 Visión por computador

La visión por computador es un campo que nos permite adquirir, procesar, analizar y comprender imágenes. Para una máquina, una imagen consiste en una matriz tridimensional de números. Para cada píxel, existe una tupla de 3 valores la cual representa la cantidad de cada uno de los tres colores primarios de la luz. La configuración RGB (rojo, verde y azul, de sus siglas en inglés), es la más habitual. Por lo tanto, el desplazamiento en horizontal o vertical en la matriz corresponde con la posición del pixel, mientras que el desplazamiento dentro de la tupla corresponde con cada uno de los colores del pixel.

Figura 1. Esquema de una imagen. Cada imagen está formada por tres matrices, una con los valores del color rojo de cada píxel, otra con los verdes y otra con los azules.
Fuente: elaboración propia

Muchas tecnologías se han visto favorecidas gracias a la visión artificial, como por ejemplo la robótica, que utiliza estas técnicas para detección y localización de objetos, o la medicina, a la cual permite detectar y diagnosticar enfermedades de manera automática. Uno de los ámbitos en los que ha supuesto una mejora mayor ha sido en los procesos de control de calidad.

La automatización de los sistemas de producción permite la creación de un gran volumen de productos, los cuales tienen que ser supervisados de manera visual para la detección de errores o para la clasificación de estos según su calidad. La supervisión manual es un proceso muy lento y caro, además de inseguro dado el factor humano que interviene. El uso de técnicas de visión por computador permite la automatización de estos procesos de inspección.

Estas técnicas necesitan la utilización de dispositivos como cámaras, sensores o escáneres. En la actualidad se está obteniendo un gran avance tecnológico en este tipo de instrumentos, lo que supone una mayor facilidad a la hora de instalarlos, así como una reducción en el coste de adquisición y mantenimiento.

1.4 Aprendizaje automático

1.4.1 Aprendizaje máquina

El aprendizaje máquina es la rama de la inteligencia artificial que consiste en el desarrollo técnicas mediante las cuales permiten a las máquinas aprender de manera autónoma. De esta manera se obtienen máquinas capaces de aprender, a partir de una serie de datos facilitados para su entrenamiento.

Para la creación de estos modelos de aprendizaje y la cuantificación de su eficiencia, se utilizan 3 conjuntos de datos:

- *Entrenamiento,* son los datos de los cuales se extrae la información necesaria para realizar el aprendizaje del modelo. De la cantidad y calidad de este conjunto de datos dependerá la calidad final de nuestro modelo.

- *Validación,* son los datos que se utilizan para aprender los hiperparámetros del proceso deaprendizaje. Es importante que este conjunto no haya sido utilizado en el entrenamiento.

- *Test,* son los datos que se usan para evaluar la calidad de nuestro modelo después de ser entrenado. Es importante que este conjunto no haya sido utilizado en el entrenamiento ni en la validación.

La proporción de la distribución de estos datos puede variar según el caso, pero se suele utilizar entorno a un 70% de los datos en el entrenamiento, un 15% en validación y otro 15% en el test.

La utilización de estos conjuntos de entrenamiento y validación permite detectar losproblemas en el entrenamiento como el sobreajuste o el subajuste.

El sobreajuste consiste en el ajuste excesivo a los datos de entrenamiento. Los modelos sobreajustados no son capaces de predecir correctamente a datos diferentes a los usados en el entrenamiento. Esto ocurre a causa de que los datos siempre presentan una cantidad de ruido, el cual el sistema debería ser capaz de ignorar, en lugar de usarlo como información de importancia y aprender de él.

Por el contrario, el subajuste ocurre cuando se utiliza un modelo demasiado simple incapaz de ajustarse correctamente a los datos.

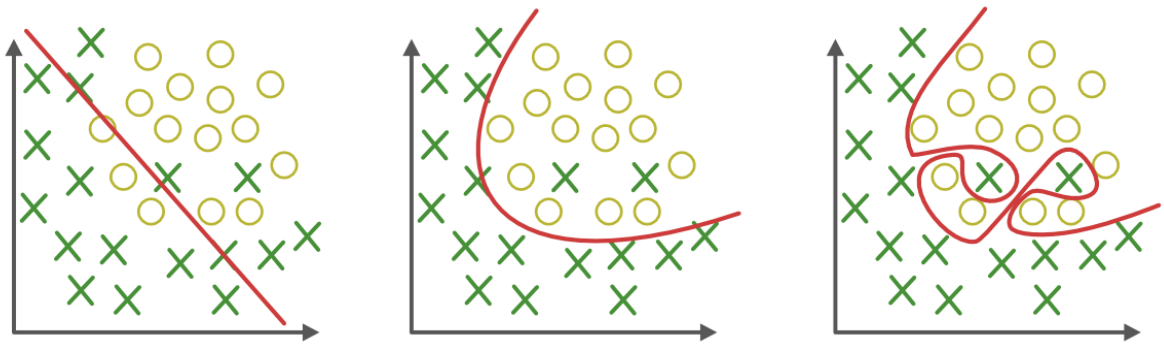


Figura 2. Ejemplo de un caso subajustado, correctamente ajustado y sobreajustado.
Fuente: https://rubialesalberto.medium.com/qu%C3%A9-es-underfitting-y-overfitting-c73d51ffd3f9

Dependiendo de la finalidad del algoritmo desarrollado se realiza un tipo de aprendizaje diferente, siendo su clasificación principal la siguiente:

- ***Aprendizaje supervisado***

El aprendizaje supervisado es un tipo de aprendizaje en el cual los datos utilizados para la generación del modelo están etiquetados, es decir, se conoce el resultado deseado.

Según el tipo de salida que se obtenga, se pueden observar dos tipos diferentes de aprendizaje supervisado:

1. *Clasificación:*

Este tipo de algoritmo se utilizan cuando el resultado que se desea obtener es una etiqueta, es decir, cuando la salida producida recae en un conjunto finito de clases.

En el entrenamiento de este tipo de sistemas, cada dato viene acompañado de la clase a la que pertenece, y está clase es la que se intenta predecir en nuevos datos.

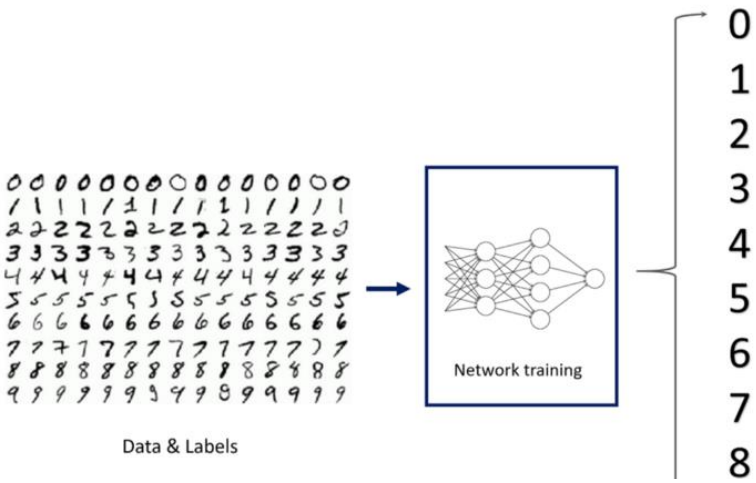


Figura 3. Esquema de una técnica de clasificación por visión artificial. Fuente: https://towardsdatascience.com/image-classification-in-10-minutes-with-mnist-dataset-54c35b77a38d

2. *Regresión*

Este tipo de algoritmo es utilizado cuando el resultado que se desea obtener es un valor numérico que está incluido en un rango de valores continuo.

El ejemplo más popular de algoritmo de regresión es la regresión lineal, la cual consiste en hallar una línea recta que mejor se adapta a los datos aportados.

Un ejemplo de regresión sería estimar la tasa de paro en función de la edad.

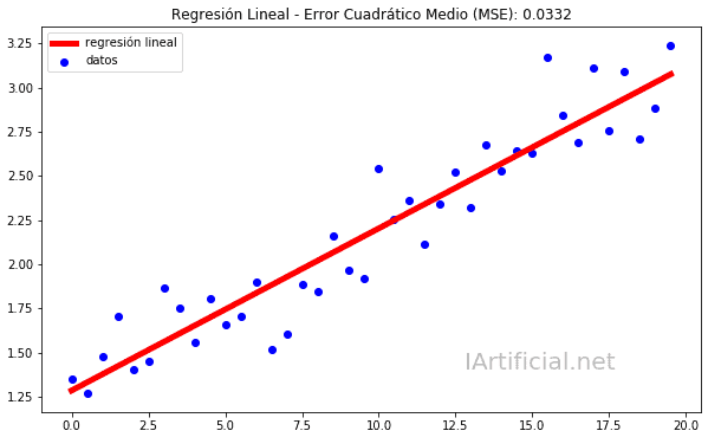


Figura 4. Esquema de una regresión lineal. Fuente: https://www.iartificial.net/error-cuadratico-medio-para-regresion/

- ***Aprendizaje no supervisado***

El aprendizaje no supervisado consiste en la utilización de un conjunto de datos (o dataset) que no está etiquetado, es decir, no se conoce el resultado que se pretende obtener. Este tipo de aprendizaje consiste en la búsqueda de información importante sin conocer previamente referencias de la salida.

Dentro del aprendizaje no supervisado, podemos encontrar dos categorías principales:

1. *Algoritmo de agrupamiento (Clustering)*

El algoritmo de agrupamiento consiste en lograr el agrupamiento de un conjunto de datos en diferentes subconjuntos. Cada subconjunto está compuesto por una colección de datos que son similares entre sí y que tienen características que les permiten diferenciarse respecto a los objetos de otros subconjuntos.

2. *Reducción dimensional*

La reducción dimensional es utilizada cuando se tiene una gran cantidad de datos o con una alta complejidad que necesitan una gran capacidad de procesamiento. Este algoritmo consiste en determinar relaciones entre las características de los conjuntos de datos, facilitando su procesamiento al disminuir la información redundante.

- ***Aprendizaje reforzado***

El aprendizaje reforzado consiste en la toma de decisiones del software en un entorno dinámico, compuesto por agentes. Para llevar a cabo esto, se utiliza un sistema de recompensas (o *rewards*) que se intenta maximizar. Para ello, el modelo (denominado agente) recibirá como entrada el entorno en el que se encuentra (denominado ambiente). El agente decidirá la siguiente acción a tomar en base a los indicadores de estado del ambiente. Como consecuencia de esta acción, se genera una recompensa la cual premiará o penalizará al modelo en base a su desempeño.

Figura 5. Esquema de un aprendizaje reforzado. Fuente: https://www.aprendemachinelearning.com/aprendizaje-por-refuerzo/

1.5 Clasificadores

El presente trabajo desarrolla un aprendizaje supervisado, ya que se dispone de una conjunto de datos previamente etiquetados. Concretamente, se trata de una clasificación que agrupan las entradas en diferentes clases según su calidad.

Dentro del conjunto de clasificadores se encuentran diferentes tipos según el modelo de clasificación que se utilice. Algunos de los más famosos son los siguientes:

1.5.1 Clasificador los K-vecinos más próximos (KNN)

El Clasificador KNN (del inglés *K-Nearest-Neighbor*) es un método que clasifica las entradas en función de la mayoría de datos que le rodean. En el KNN se calcula la distancia entre las características elegidas del objeto a clasificar y las del resto de datos del dataset usado en el entrenamiento, se selecciona el número "k" de vecinos que se vaya a tener en cuenta (elementos que estén a una menor distancia) y se escoge como clase la más predominante en los vecinos.

El número de vecinos óptimo a elegir será diferente en cada caso en función de diversos factores. Por ejemplo, si disponemos de un número par de clases se prioriza la elección de un "k" impar para evitar empates.

El correcto ajuste del parámetro "k" es muy importante ya que no solo tiene influencia en la precisión del clasificador si no que afecta directamente al tiempo de ejecución.

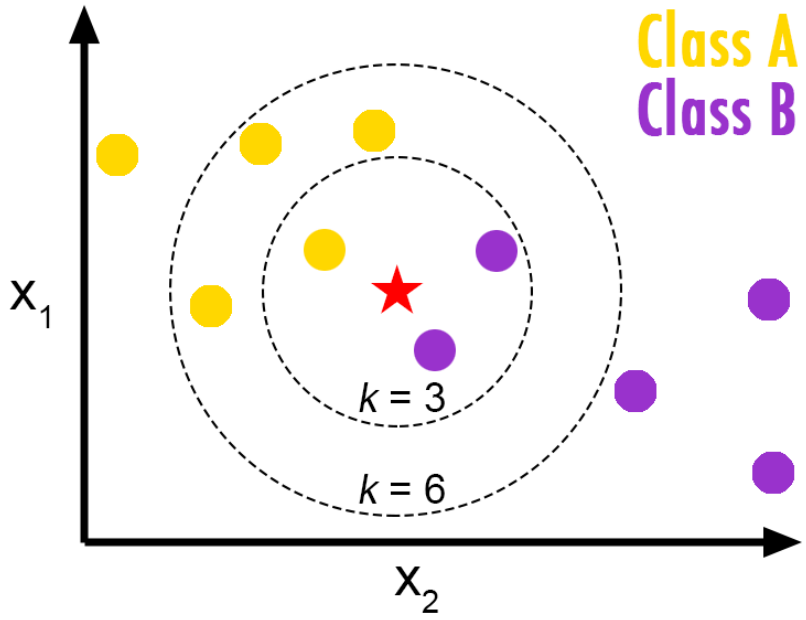


Figura 6. Esquema de un clasificador KNN. Fuente: https://pythondiario.com/2018/01/introduccion-al-machine-learning-9-k.html

1.5.2 Clasificador Bayesiano (Naive Bayes)

Los clasificadores Naive Bayes se basan en el teorema de Bayes, para calcular la probabilidad de que un dato pertenezca a una clase en concreto. El Teorema de Bayes, también conocido como el teorema de la probabilidad condicionada, permite calcular la probabilidad de que se dé un evento "h" habiendo ocurrido un evento "D" en función de la probabilidad de que ocurra "D" dado "h" y la propia probabilidad marginal de "h".

Para la implementación de este clasificador se efectúa la suposición de que una característica en una clase es independiente del resto de características. De esta manera, se calculan las probabilidades de que una entrada pertenezca a cada una de las clases y se toma como correcta la de mayor valor.

$$P(h \mid D) = \frac{P(D \mid h)P(h)}{P(D)}$$

Ecuación 1. Teorema de Bayes

1.5.3 Máquinas de Vectores de Soporte (SVM)

Las máquinas de vectores de soporte son un conjunto de transformaciones de los datos de entrada utilizadas en el entrenamiento en un espacio de mayor dimensión que permita la separación de éstas en dos espacios de la manera más amplia posible mediando un hiperplano.

El correcto ajuste de este hiperplano es el responsable de una buena precisión a la hora de calcular nuevas predicciones. Para un ajuste óptimo, se busca maximizar el margen entre los puntos que se encuentren más cerca del hiperplano. El vector que une ambos puntos se llama vector de apoyo o de soporte.

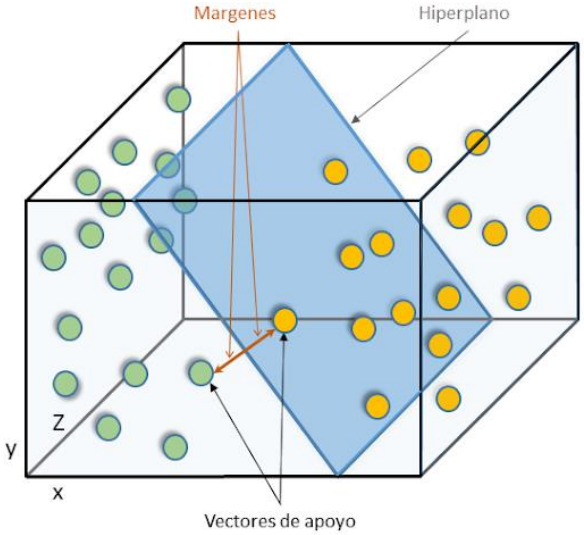


Figura 7. Esquema de un clasificador SVM.
Fuente: https://numerentur.org/svm/

Para realizar el ajuste del hiperplano, existe un parámetro conocido como "*C*", que permite definir el límite entre clases. Valores altos de C fijan una frontera capaz de generalizar entre las clases pese a disponer de errores en los datos de entrenamiento. Por el contrario, valores muy pequeños de C separan perfectamente los datos de cada clase. El cálculo de este parámetro se realiza de manera empírica, de manera que se intente evitar el sobreajuste con valores muy pequeños pero que no generalice de manera incorrecta y se produzca mucho error.

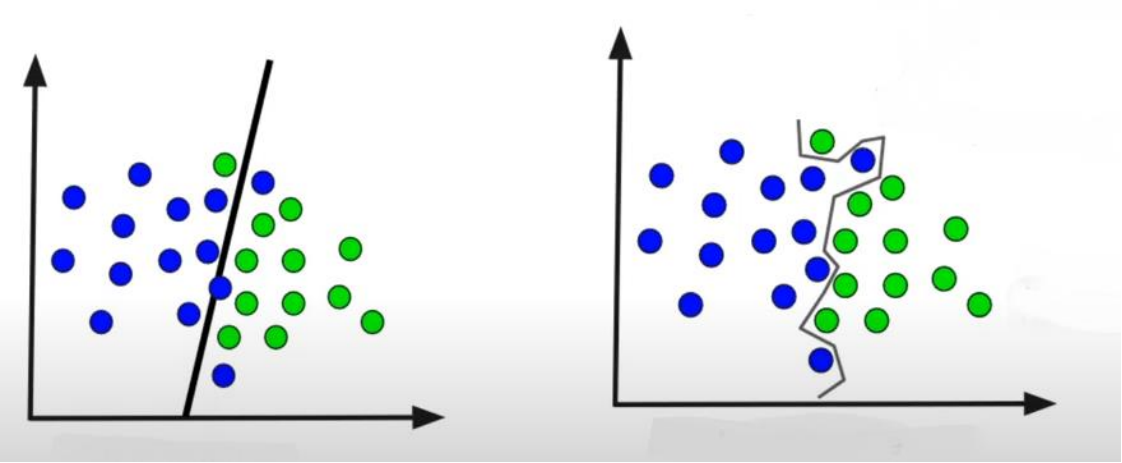


Figura 8. Ejemplo de SVM con C alta (izquierda) y C baja (derecha). Fuente: elaboración propia

1.6 Aprendizaje Profundo

En la actualidad el campo de la inteligencia artificial está logrando grandes avances, consiguiendo automatizar tareas muy específicas y complejas como la conducción automática de un vehículo o la detección de enfermedades a partir de fotografías celulares.

Estos grandes avances se están consiguiendo en gran parte gracias a la rama del aprendizaje profundo (en inglés *Deep Learning*). Este campo consiste en la creación de modelos de millones de parámetros que se asemejan al modo de funcionamiento de las neuronas de un cerebro.

Estos modelos, conocidos como redes neuronales artificiales, consisten es una cantidad de neuronas artificiales conectadas entre sí que son capaces de generar unos valores de salida relacionados con los valores de entrada que se le aporte. La principal ventaja de estos sistemas es la capacidad de aprendizaje que tienen, puesto que variando los valores de estas neuronas es posible obtener la salida deseada.

1.6.1 Neurona artificial o Perceptrón

Las neuronas artificiales nombradas anteriormente, también conocidas como perceptrones, son la unidad básica de las redes neuronales. Un perceptrón es capaz de generar una salida que varía en función de los datos que reciba. Esta salida está condicionada a una serie de parámetros que se pueden ajustar y que permiten la capacidad de aprendizaje de estas.

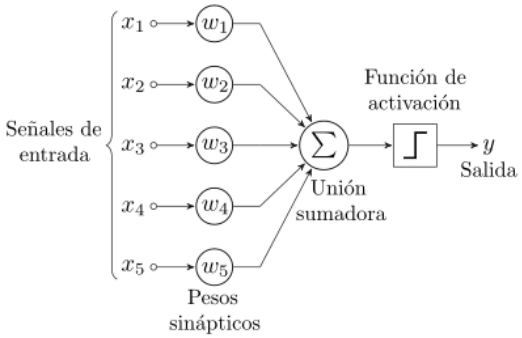


Figura 9. Esquema de un Perceptrón.
Fuente: https://es.wikipedia.org/wiki/Perceptr%C3%B3n

- *Pesos sinápticos:*

Los pesos sinápticos son los valores que multiplican a cada uno de los datos de entrada del perceptrón. En el entrenamiento de una red neuronal, estos son los parámetros que cambian su valor en cada iteración en función de un valor llamado tasa de aprendizaje (*learning rate*).

- *Umbral o Bias*:

El umbral es un valor constante que permite que, sumándose a la multiplicación de todas las entradas con sus pesos sinápticos, se pueda tomar una decisión mediante la función de activación.

- *Función de activación:*

La función de activación es una función matemática que se encarga de permitir o denegar el paso de información en la salida en función del resultado de la suma de la multiplicación de los pesos sinápticos y las entradas y del umbral.

$$f(x) = \begin{cases} 1 & \text{si } w \cdot x - u > 0 \\ 0 & \text{en otro caso} \end{cases}$$

Como se puede observar, un solo perceptrón es capaz de resolver un problema lineal, pero existen una gran cantidad de casos en las que un solo perceptrón no es suficiente para ajustar el modelo a los datos, por ejemplo el caso de la función XOR.

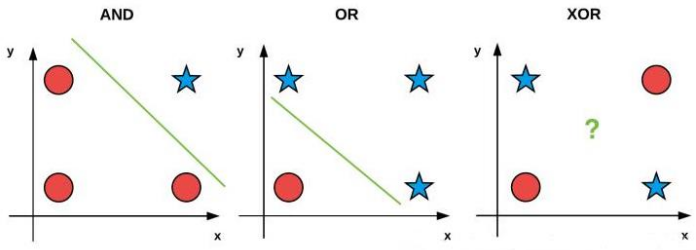


Figura 10. Esquema de problemas posibles e imposibles de resolver con un Perceptrón. Fuente: https://www.pyimagesearch.com/2021/05/06/implementing-the-perceptron-neural-network-with-python/

1.6.2 Perceptrón multicapa

El perceptrón multicapa surge como solución al problema mencionado anteriormente. El perceptrón multicapa es un tipo de red neuronal artificial formado por múltiples perceptrones individuales interconectados entre sí.

Los perceptrones se agrupan en conjuntos llamados capas de manera que las salidas de una capa son las entradas de la siguiente capa.

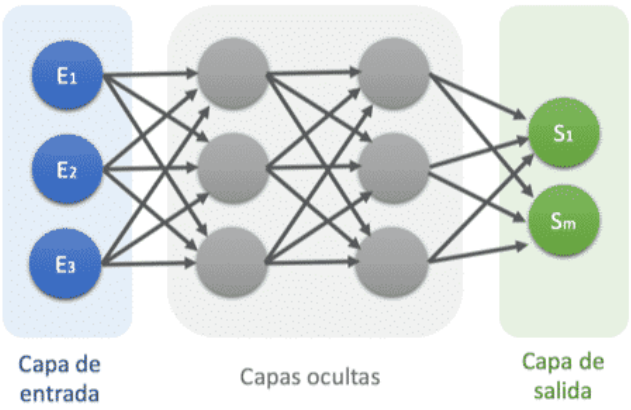


Figura 11. Esquema de un Perceptrón multicapa. Fuente: https://www.diegocalvo.es/perceptron-multicapa/

La primera capa, la cual recibe las entradas se llama capa de entrada; la última capa, cuyos valores finales corresponden con los valores de salida se llama capa de salida y el resto de las capas intermedias reciben el nombre de capas ocultas.

La capa de entrada debe estar compuesta del mismo número de neuronas como entradas haya en el modelo.

El número de capas ocultas que formen parte del perceptrón multicapa definirá la complejidad de este, y por lo tanto está relacionado con su capacidad de aprendizaje y con su coste de computación.

La capa de salida debe de tener el mismo número de neuronas como salidas se desee tener.

1.6.3 Redes neuronales convolucionales

Las redes neuronales convolucionales son un tipo de red neuronal artificial el cual recibe como datos de entrada matrices. Como se comentó en el apartado 1.3, las imágenes son un conjunto de matrices por lo que las redes neuronales convolucionales son la herramienta adecuada para trabajar con imágenes. El funcionamiento de las redes neuronales convolucionales es similar a los perceptrones multicapa, adaptando sus parámetros a las operaciones de convolución que nos permiten procesar las imágenes.

1.6.4 Entrenamiento de redes neuronales

En apartados anteriores se ha hablado de la capacidad de aprendizaje de las redes neuronales, cambiando los pesos sinápticos para conseguir un ajuste de la salida deseada, pero no se ha explicado el procedimiento de entrenamiento.

Para entrenar redes neuronales artificiales mediante un proceso de aprendizaje supervisado existe un método de cálculo llamado la propagación hacia atrás (en inglés *backpropagation*). La propagación hacia atrás o retropropagación se realiza en dos fases. La primera fase consiste en el un proceso similar al de un perceptrón multicapa, en el que se inicializan los pesos sinápticos en valores aleatorios y se multiplican a los valores de entrada con el fin de pasarlos por la función de activación. La principal diferencia con un perceptrón consiste en que es necesario calcular la derivada de la función de activación para la segunda fase, por lo que es necesario utilizar funciones de activación diferentes al escalón.

La segunda fase consiste en la comparación de las salidas con las etiquetas que disponemos de nuestros datos, con el fin de calcular una función llamada función de coste. La red neuronal calcula el gradiente de esta función respecto a cada uno de los pesos sinápticos y, se calculan nuevos valores en función de esta derivada y de un hiperparámetro conocido como la tasa de aprendizaje, que está relacionado con el porcentaje de cambio que realizamos en los pesos.

1.7 Visión artificial en las líneas de manipulación de peras

Las técnicas de clasificación mediante visión artificial que se desarrollan en este trabajo están enfocadas a su uso en una línea de manipulación postcosecha. Con el fin de entender las posibilidades de estos sistemas en una línea real, se explica el proceso de postcosecha en un almacén de peras.

1.7.1 Líneas de manipulación de peras

Una línea de manipulación de fruta es una línea que está formada por un conjunto de equipos que trata la fruta para poder ser comercializada. Los procesos que realiza una de estas líneas son: la entrada de la fruta a la línea, un tratamiento de limpieza y un proceso de encajado.

Tras realizar el desencajado y volcado de las peras en la línea transportadora, se realiza una preselección de las frutas aptas y no aptas. Este proceso se efectúa al comienzo de la línea con la finalidad de desechar lo más temprano posible las frutas que estén podridas y pudieran acabar contaminando el resto de la línea.

Tras la preselección, las peras se lavan con detergente para eliminar la suciedad procedente del campo como pesticidas, polvo o insectos. Es necesario enjuagar la fruta para eliminar los restos de jabón. Posteriormente, las peras se secan con chorros de aire.

Después del lavado, las frutas son clasificadas en función de su calidad. Finalmente, las peras son calibradas por peso y/o tamaño y se encajan para su expedición.

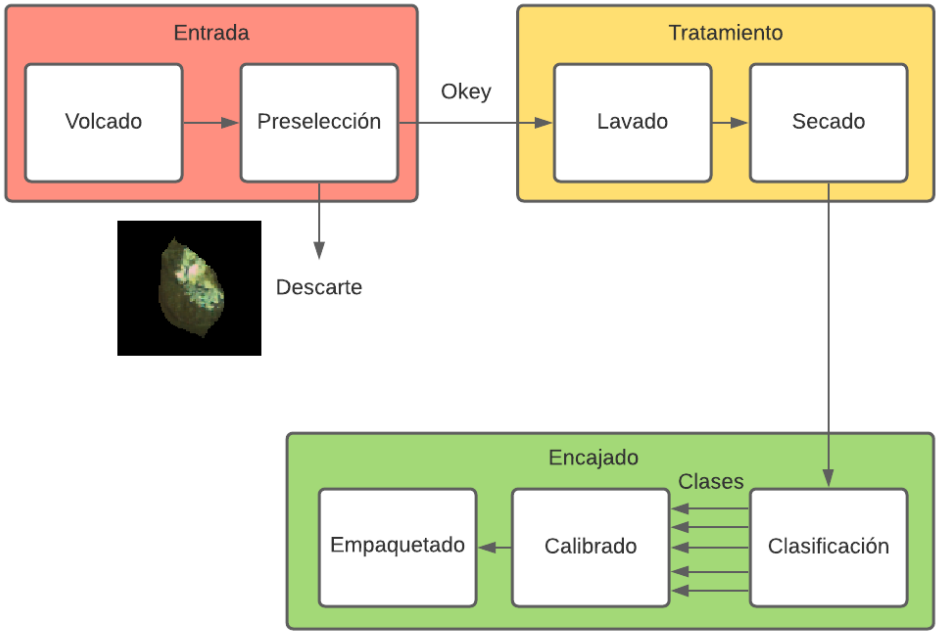


Figura 12. Esquema de una línea de manipulación de peras. Fuente: elaboración propia

1.7.2 Selección y preselección por visión artificial

En el estado del arte se utilizan sistemas de visión artificial para realizar la selección y preselección de la fruta. Estos procesos se realizan mediante técnicas aplicadas a imágenes capturadas de la fruta que se encuentra en la línea, que proporcionan valores a características como el color, textura y la forma. Además de la captura de imágenes a color, se está comenzando a incorporar cámaras infrarrojas con el fin de la detección de errores internos de las peras como podridos blandos.

Para la aplicación de técnicas de visión artificial en una línea de manipulación tan solo es necesario un sistema de captura de imágenes que sea capaz de adaptarse a la velocidad de la línea, un ordenador que realice la ejecución del modelo de predicción y un sistema de separación electrónico que sea capaz de reaccionar en función de las entradas que se le aporte.

La utilización de una red neuronal convolucional en las líneas de manipulación podría proporcionar la capacidad de realizar una clasificación de las peras en función de su calidad de manera autónoma y eficiente. De esta manera, se podría aumentar la precisión en proceso de clasificación ya que las redes neuronales, tras un entrenamiento adecuado, son capaces de aprender qué características de interés se ajustan mejor de cada categoría. Además, se podría obtener una alta velocidad de clasificación con la utilización de GPUs.

La principal función que se realiza en la preselección de la fruta es eliminar lo más pronto posible las peras que estén rotas o podridas y que pudieran potencialmente contaminar el resto de la línea. En el caso de la clasificación por su calidad, el interés es tanto normativo como comercial. La UE impone una normativa de calidad para comercializar la fruta, pero también es un requisito de los clientes. En función de los resultados de la clasificación, la empresa puede realizar estudios de la cosecha, analizando la calidad y cantidad, así como plantear estrategias de comerciales en base a la situación del mercado.

1.8 Antecedentes de la visión artificial

El aumento de la competitividad es un requerimiento primordial en la situación económica actual. Un factor clave para mejorar la competitividad es aumentar la productividad incorporando nuevos sistemas de inspección que permitan la automatización de diferentes procesos. Los sistemas de inspección visual automática, basados en visión por computador, han demostrado ser una herramienta fundamental para mejorar los procesos. Estos sistemas de visión permiten la inspección continua, evitando fatigas y distracciones, y facilitando la cuantificación de las variables de calidad en prácticamente el 100% de la producción. Esto se traduce, no sólo en una mejora de la calidad final de los productos, sino también en un ahorro en términos económicos y medioambientales.

Durante las últimas décadas se han podido resolver multitud de aplicaciones mediante la implantación de sistemas de inspección 2D en la industria utilizando técnicas de visión por computador tradicionales. El principal problema a resolver en estos sistemas ha sido seleccionar un conjunto mínimo de características discriminante para clasificar los objetos de interés, además de la gran variabilidad de las imágenes que puede dificultar la segmentación de estos objetos de interés. Para resolver este problema se han propuesto algunas técnicas robustas de segmentación que intentan absorber dicha variabilidad [1] [2]. Además, también se han propuesto infinidad de soluciones de seguimiento de objetos, como por ejemplo la utilización de splines cúbicos para predecir trayectorias [3].

Por otro lado, los enormes avances tecnológicos producidos en las cámaras lineales, los escáneres 3D [4] [5] [6] [7] [8] y los sensores hiperespectrales [9] [10] [11] están permitiendo resolver algunas aplicaciones complejas en el sector industrial que hace unos años eran impensables. Estas nuevas tecnologías aplicadas sobre todo al sector agroalimentario y cosmético, se están empezando a introducir en la industria.

Además, los últimos avances producidos en la aplicación de técnicas de detección y seguimiento de objetos deformables [12] [13] [14] [15] [16] [17] basadas en redes neuronales convolucionales; están permitiendo empezar a automatizar algunas aplicaciones de detección complejas.

La clasificación de fruta es un problema complejo debido a la gran variabilidad de defectos que se deben detectar, y a los altos requerimientos temporales de una línea de calibrado dentro de un almacén. En este trabajo se plantean técnicas tradicionales para resolver el problema de preselección y nuevas técnicas de imagen basadas en redes neuronales convolucionales para clasificación de fruta durante el proceso de encajado en un almacén.

Capítulo 2: Tecnologías y herramientas

En este apartado se va a presentar las tecnologías y herramientas utilizadas durante el desarrollo del presente trabajo.

2.1 Conjunto de datos (Dataset)

Se conoce como Dataset al conjunto de datos utilizados en alguna función, en este caso se han utilizado con el fin de clasificar las diferentes calidades de las peras devoe de un almacén.

Para la realización de este proyecto se ha utilizado un Dataset de peras devoe proporcionadas por el Instituto de Automática e Informática Industrial (ai2) de la UPV, las cuales están etiquetadas en diferentes categorías dependiendo de la calidad de la fruta.

El Dataset utilizado consta de 22782 imágenes de peras a color y la misma cantidad de capturas en blanco y negro del infrarrojo de éstas.

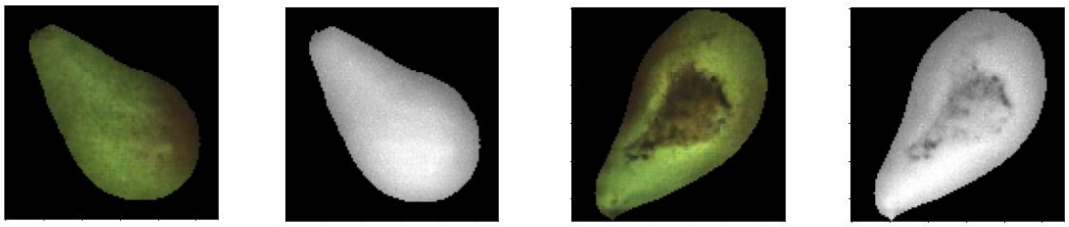


Figura 13. Ejemplo de imágenes del Dataset.
Se observa una pera de calidad 1 (izquierda) y una pera de calidad 5 (derecha).
Fuente: elaboración propia

2.2 Lenguaje de programación

El lenguaje de programación utilizado en la realización del proyecto ha sido Python. La elección de este lenguaje sobre otros ha sido principalmente porque Python es el principal lenguaje utilizado en los campos de inteligencia artificial y machine learning.

Python ofrece una amplia variedad de librerías que permiten realizar diferentes funciones, algunos ejemplos de las librerías utilizadas en este trabajo son:

- *OpenCV*, una librería que permite trabajar con imágenes y videos de manera sencilla.
- *PyTorch,* utilizada para la creación de modelos de Aprendizaje Profundo.
- *Numpy* es una librería que posibilita la creación de vectores y matrices, así como una gran cantidad de operaciones matemáticas para operar con estas.
- *Os* es un módulo de la librería estándar de Python que da acceso a diferentes funcionalidades del sistema operativo, destacando las que permiten manipular las carpetas, así como leer y escribir archivos en éstas.
- *Matplotlib* es la librería que permite crear y modificar gráficos de todo tipo.

2.3 OpenCV

Como ya se ha mencionado anteriormente, OpenCV es una librería destinada al campo de la visión artificial.

OpenCV fue desarrollado por Intel en 1999 y en el 2000 se liberó su código al público.

Esta librería no solo permite la modificación de imágenes o videos, también permite la extracción de contornos que se encuentren en estos para posibilitar la extracción de características de interés de los mismo. Además, OpenCV permite la creación y ejecución de múltiples clasificadores mencionados anteriormente.

2.4 PyTorch

PyTorch es una librería desarrollada por Facebook AI en 2016 que está enfocada en la realización de operaciones matemáticas mediante el uso de tensores. Los tensores son matrices multidimensionales que contienen elementos de un único tipo.

Esta librería está enfocada en el campo del *Machine Learning*, principalmente en el tema de Aprendizaje Profundo para la creación y optimización de redes neuronales. Esta librería permite el uso de GPU en su ejecución, lo que reduce de una manera considerable los tiempos de ejecución.

Otras librerías con funciones similares a PyTorch son TensorFlow, Keras, Caffe y Theano.

2.5 Google Colaboratory

Google Colaboratory (o Google Colab) es un entorno gratuito de programación y ejecución de código en Python. La principal ventaja de Google Colab frente a otros entornos de ejecución es que permite el uso de una GPU en la nube.

El cálculo iterativo que se realiza en el entrenamiento de una red neuronal es muy costoso de manera computacional, pudiendo tardar desde horas a días en un entorno local de CPU.

Por esto, el uso de una GPU es necesario si se quiere reducir notablemente los tiempos de ejecución.

Capítulo 3: Desarrollo del trabajo

En este apartado se explican los métodos empleados en la realización del proyecto. Primero, se explica el proceso seguido para la creación y ejecución de una solución para el proceso de preselección de una línea de manipulación de peras. Posteriormente, se expone el proceso que se ha realizado para obtener una solución al proceso de clasificación de las diferentes calidades de peras.

3.1 Proceso de preselección

Como se ha mencionado anteriormente, el proceso de preselección de la línea de manipulación de peras consiste en la clasificación de las peras en 2 clases, aptas o no aptas.La gran ventaja de este problema es la facilidad de reconocimiento de las zonas rotas o podridas de la fruta, tanto por su color como por su forma.

Debido a esto, se ha decidido utilizar clasificadores de aprendizaje tradicionales como solución a la preselección de peras, pues se adapta muy bien a problemas con clases claramente diferenciadas y presentan poco coste computacional.

La instalación de un clasificador de aprendizaje en la línea de manipulación constaría del siguiente esquema: se captura la imagen mediante una cámara, se segmenta la imagen para reconocer la pera como único objeto de interés, se extraen las características apropiadas y estas características se introducen en el clasificador entrenado para que devuelva como salida si la pera es apta o no. Esta salida la recibirá un actuador que se encargará de separar las peras defectuosas.

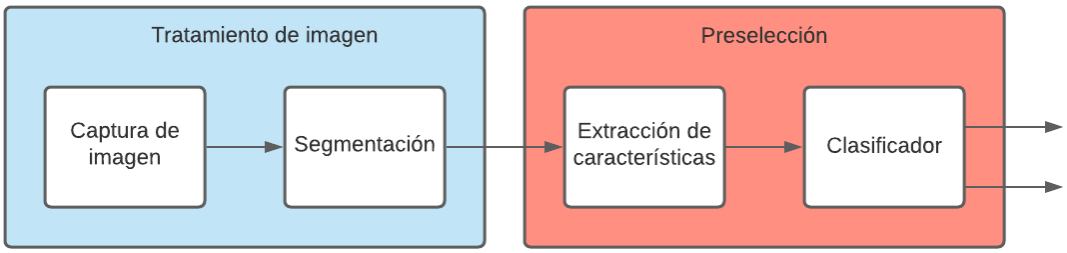


Figura 7. Esquema del proceso de preselección. Fuente: elaboración propia

El proceso de entrenamiento de los clasificadores de aprendizaje es equivalente al esquema mencionado anteriormente donde se parte de un conjunto de datos etiquetados. La principal diferencia es que se deberán seleccionar las características que discriminan estas dos clases.

Los clasificadores utilizados en este trabajo son los mencionados en el apartado 1.4: KNN, bayesiano y SVM. Para su implementación, se ha utilizado parte del dataset proporcionado por el Instituto de Automática e Informática Industrial (ai2) de la UPV, concretamente 796 imágenes en RGB y la misma cantidad en infrarrojos etiquetadas como aptas o no aptas. Las imágenes fueron revisadas con el fin de evitar posibles errores en el etiquetado.

3.1.1 Segmentación de las imágenes

Para poder realizar la clasificación con este tipo de clasificadores, es necesario segmentar las imágenes para poder realizar una correcta extracción de características de interés. En el caso de implementar una técnica de este estilo en la línea de manipulación, es necesario diferenciar los pixeles de interés de los pixeles del fondo de las imágenes capturadas.

El Dataset que se dispone para la realización del trabajo ha sido tratado previamente, eliminando el fondo de las imágenes, por lo que tiene todos los píxeles que no son de interés con valor 0, es decir, en negro.

Para realizar una buena extracción de características, es necesario procesar las imágenes previamente a su análisis. En el proyecto, es de interés detectar los píxeles que sean considerados como defectos. Para ello, se ha optado por dos estrategias diferentes que, con su uso simultaneo, permiten obtener un buen resultado.

La primera técnica de detección de las zonas con desperfectos consiste en un filtrado por color píxel a píxel. Los desperfectos en las peras son de un tono marrón el cual se representa con valores bajos para los tres colores primarios de la luz (rojo, verde y azul). Puesto que las zonas de interés de la pera tienen un valor alto de verde, se ha decidido utilizar el color verde como único filtro. En las fotografías de infrarrojos, los desperfectos se corresponden con zonas oscuras, por lo que se puede utilizar el mismo criterio para filtrar. Este filtro consiste en recorrer píxel a píxel la imagen y comprobar si supera el umbral fijado. En caso de que el color no sea superior al umbral, se fija ese píxel a un valor de 0.

La segunda técnica utilizada para la detección de zonas defectuosas es más compleja y por lo tanto más costosa computacionalmente. Esta técnica consiste en el filtrado de las imágenes utilizando sus propias características de media y varianza.

El primer paso de esta función consiste en recorrer todos los píxeles de la imagen y comprobar cuales de estos no tienen un valor nulo, es decir, cuales corresponden con una parte de la pera. En caso de que se trate de un píxel de interés, se sumará su valor a una variable y se aumentará el valor de un contador. Al terminar de recorrer la imagen entera, se calcula la división entre la suma de todos los valores y el número de píxeles correspondientes con peras para hallar la media de la imagen. En caso de las fotografías en RGB, se ha calculado la media del color verde únicamente.

A continuación, se vuelve a recorrer la imagen píxel a píxel, con la finalidad de calcular la varianza de esta. Para ello, en los píxeles de interés se calcula la raíz cuadrada del valor absoluto de la resta entre el valor del color del píxel y la media, y se suma este valor en una variable. Al terminar de recorrer, se calcula el cociente entre esta suma y el número de pixeles de interés calculado anteriormente. Esta división se corresponde con la varianza de la imagen.

Por último, se vuelve a recorrer por tercera vez la imagen, y en cada píxel de interés se calcula la resta entre el valor y la media. En caso de que esta resta sea menor a la varianza multiplicada por una constante negativa con la que se puede ajustar el filtrado, se fija su valor a 0. En caso de aplicar los dos filtros, es recomendado realizar la segunda técnica primero, ya que cambios previos en la imagen pueden afectar considerablemente a la media y la varianza de los colores.

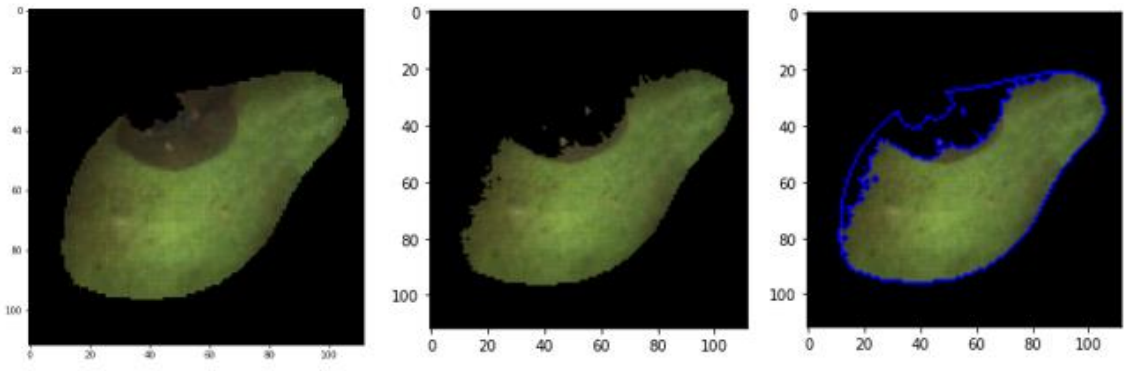


Figura 8. Esquema de un prepocesamiento de una imagen de una pera defectuosa. Fuente: elaboración propia

3.1.2 Extracción de características

Los clasificadores de aprendizaje tradicionales utilizan una serie de características para tener la capacidad de poder diferencias las distintas clases de imágenes aportadas. Dependiendo del caso a analizar, serán más interesantes unas u otras características.

Para la realización de la extracción de características, la librería de OpenCV mencionada en el apartado 2.3 es de gran utilidad. Esta librería dispone de la función *findContours* la cual recibe una imagen segmentada en blanco y negro y devuelve los contornos que encuentre en la imagen aportada. Estos contornos son de gran ayuda ya que se pueden calcular muchas características interesantes de estos.

En el desarrollo del presente trabajo, se han utilizado las siguientes características:

- *Área:*

El área de un contorno es la superficie acotada por este contorno. Esta característica resulta muy interesante de calcular ya que es de gran ayuda para el cálculo de características más complejas. Ya que existe la posibilidad de encontrar varios contornos al utilizar la función *findContours*, es interesante guardarse únicamente el tamaño del área mayor, el cual será el correspondiente al de la pera a analizar. Para la realización del cálculo del área, se utiliza la función *contourArea*, a la cual se le aporta el contorno de interés y devuelve un float con el valor del área.

- *Perímetro:*

De igual manera que pasa con el área, el perímetro es una característica muy interesante para el cálculo de otras propiedades. Para realizar su cálculo, se utiliza la función de OpenCV *arcLength*, a la cual se le introduce el contorno de interés y un booleano con valor True en caso de ser una superficie cerrada o False en caso de ser una curva. Esta función devuelve un valor float con el perímetro del contorno.

- *Solidez:*

La solidez de una imagen se calcula como el cociente entre su área y el área de su envolvente convexa. La envolvente convexa se define como el área mínima que envuelve a un contorno que no contiene partes cóncavas. Para calcular la envolvente convexa de un contorno se utiliza la función de OpenCV *convexHull*, la cual recibe el contorno de interés y nos devuelve el contorno de la envolvente convexa. Para calcular el área de este nuevo contorno, se utiliza la función mencionada anteriormente *contourArea*.

Figura 9. Ejemplo de diferencia de solidez. Se observa una pera de apta (izquierda) y una no apta (derecha). Fuente: elaboración propia

- *Color medio:*

En el apartado 3.1.1 se ha explicado como calcular la media de un color en la imagen. Esta característica es interesante de analizar debido a que las imágenes de la peor calidad contienen grandes zonas de desperfectos. Como ya se ha mencionado, las zonas oscuras corresponden con valores muy cercanos al 0, por lo que el color verde no es el único color de interés, una imagen con grandes zonas oscuras es probable que tenga una media de los tres valores muy baja. En las imágenes en blanco y negro correspondientes con los infrarrojos, también es un parámetro muy interesante de analizar.

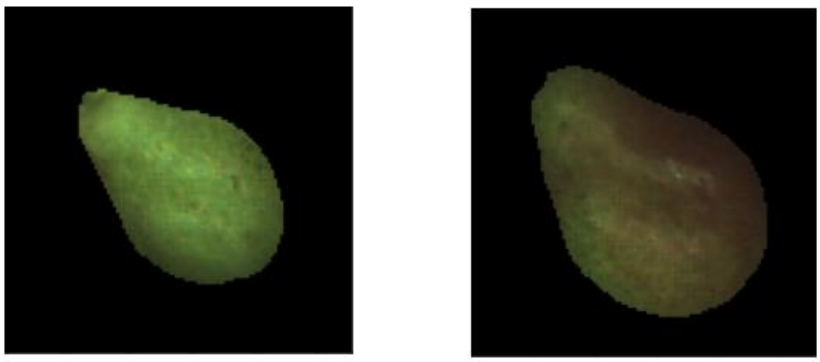


Figura 10. Ejemplo de diferencia de color. Se observa una pera apta (izquierda) y una no apta (derecha). Fuente: elaboración propia

- *Zonas defectuosas:*

Ya se ha hecho mención en el apartado 3.1.1 como detectar y eliminar las zonas defectuosas de la imagen, pero no se ha comentado como contabilizarlas. Se han utilizado 2 estrategias diferentes para contabilizarlas.

El primer método consiste en contabilizar el número de píxeles que se consideran como defectuosos. Para ello, en el proceso de procesamiento de la imagen, cada vez que se fuera a poner el valor de un píxel a 0 se aumentaría en 1 el valor de un contador. Al terminar de recorrer toda la imagen, la proporción de defectos se calcula como el cociente entre el número de píxeles cambiados y el número total de pixeles de la pera.

La segunda manera de calcular esta proporción se realiza utilizando los contornos de OpenCV. Este método consiste en calcular la suma de todos los contornos que se consideren defectos. Para ello, se recorren 1 a 1 todos los contornos que se encuentren y se suman las áreas de los contornos defectuosos. Para considerar que un contorno se trata de un defecto, se fija un umbral el cual consiste en que el área del contorno sea menor que un porcentaje del área de la pera. Este valor se puede ajustar para obtener diferentes resultados.

Este procedimiento tiene una gran ventaja, la cual es que es capaz de reconocer zonas defectuosas que ya estuvieran eliminadas. Alguna de las imágenes del Dataset contienen zonas internas negras que se corresponden con defectos eliminados en el proceso de eliminación de fondo.

Sin embargo, contiene una gran desventaja que consiste en la posibilidad de sumar áreas varias veces. Existe la posibilidad de que se encuentren contornos pequeños que contengan otros contornos aún más pequeños. Por lo tanto, es muy importante realizar un buen procesado de la imagen y un buen ajuste del umbral si se quiere utilizar este método.

3.1.3 Normalización

Antes de introducir los datos en los clasificadores, es importante realizar un proceso de normalización. Esto es debido a que algunos parámetros contienen valores muy elevados frente a otros los cuales oscilan entre 0 y 1. Si no se realizara un proceso de normalización, el clasificador daría más importancia a características como el color medio que a la solidez.

Para realizar la normalización, se utiliza una fórmula matemática que consiste en calcular el cociente entre el valor a normalizar y el valor mínimo de esa característica entre el máximo y el mínimo. Existen las funciones *min* y *max* que nos permiten obtener estos valores de una lista de datos.

$$X_{Normalizada} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Ecuación 2. Fórmula de normalización

3.2 Proceso de clasificación por calidad

La clasificación realizada en la línea de manipulación de peras es un problema más complejo que la preselección, al haber 5 clases en lugar de 2 y al haber menos diferencias claras entre cada una de las clases. Por ello, fijar unas características que permitan hallar la clase a la que pertenece cada pera es una tarea muy compleja.

El modelo seleccionado para la resolución de este problema es la implementación de una red neuronal convolucional, la cual es capaz de aprender características automáticamente y permite realizar una clasificación más precisa a cambio de un coste computacional mayor.

El entrenamiento de una red neuronal consiste en un bucle que se retroalimenta. En primer lugar, se necesita una gran cantidad de datos etiquetados los cuales queramos clasificar, en el caso del proyecto las peras en función de su calidad. De este conjunto de datos, se escoge en torno a un 70% que se usará en el entrenamiento de la red. Estos datos de entrenamiento necesitan ser etiquetados manualmente. Como ocurre en la preselección, los datos han sido etiquetados por el Instituto de Automática e Informática Industrial (ai2) de la UPV y han sido revisados para realizar una corrección de los posibles errores. El entrenamiento de la red neuronal convolucional utilizada se detalla en el apartado 1.6.4.

Para la obtención de un mejor resultado y una facilidad en el entrenamiento, se puede utilizar una red preentrenada como base de la red a implementar. Esto es un proceso conocido como transferencia de aprendizaje.

Una vez entrenada la red, esta se utiliza para realizar predicciones con datos de validación que no hayan sido usados en el entrenamiento y se analizan los resultados obtenidos. En caso de no ser los resultados deseados, la red se puede volver a entrenar añadiendo nuevos datos y/o variando los hiperparámetros de entrenamiento.

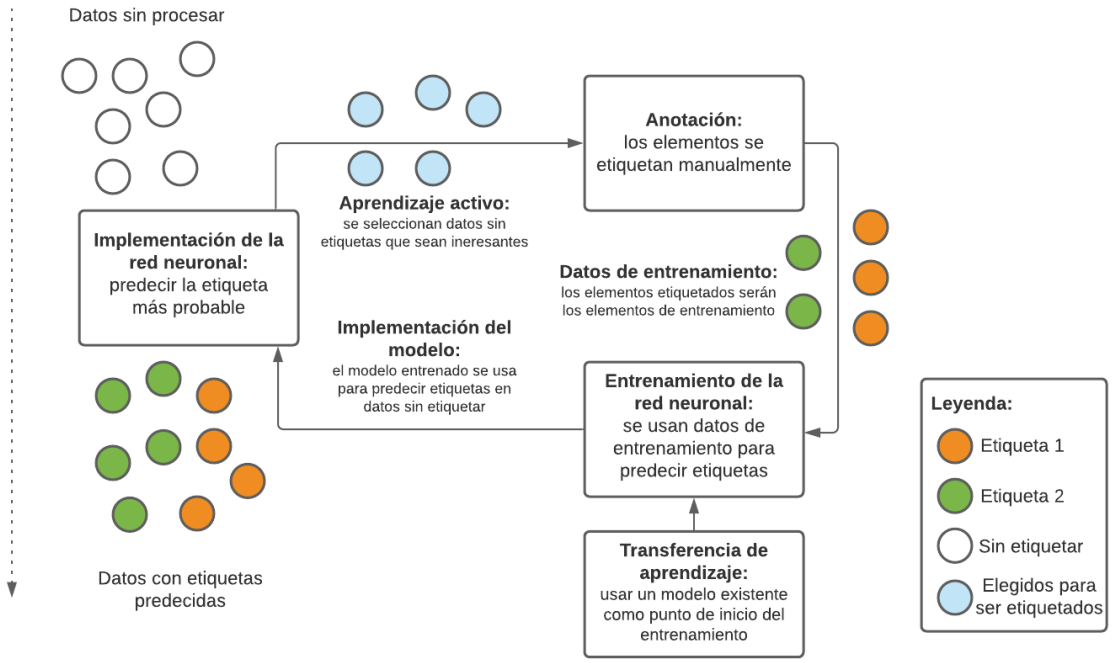


Figura 11. Esquema de implementación y entrenamiento de una red neuronal.
Fuente: elaboración propia

3.2.1 Transferencia de aprendizaje

Como se ha mencionado, una gran ventaja que nos proporciona la librería de PyTorch es la disponibilidad de redes preentrenadas que se pueden utilizar como ayuda en la resolución del problema. En eso se basa la transferencia de aprendizaje (en inglés *transfer learning*).

Dentro de la transferencia de aprendizaje, podemos diferenciar diferentes estrategias a seguir. En primer lugar, existe la posibilidad de utilizar como punto de partida los pesos de la red preentrenada, los cuales deberían ser un mejor punto de partida que la inicialización aleatoria. Una vez fijados los pesos iniciales, el siguiente paso a realizar es la eliminación de la última capa, la cual estaría enfocada a la resolución del problema de partida de la red preentrenada. En su lugar, se añadirán las capas necesarias para obtener un buen resultado para nuestro problema, indicando el número de clases que se desea obtener.

Otra estrategia a seguir mediante el uso de la transferencia de aprendizaje es la congelación de los pesos de todas las capas a excepción de la última, la cual se entrenará con la finalidad de obtener una buena extracción de características y eso se vea reflejado en una buena precisión. Este entrenamiento será poco costoso computacionalmente y de corta duración, puesto que no serán muchos los parámetros a ajustar. Si se quiere refinar aún más el funcionamiento de la red, existe la posibilidad de ir descongelando los parámetros del modelo progresivamente para conseguir un perfecto ajuste en su totalidad. A este método se le conoce como *fine tuning*.

Para la realización del proyecto se ha utilizado la red neuronal preentrenada *ResNet18*. Las redes con arquitectura *ResNet* son modelos ganadores de una competición realizada en 2015. Esta competición consistía en la utilización del Dataset del proyecto *ImageNet*, el cual consiste en un conjunto de más de 14 millones de imágenes y alrededor de 22000 etiquetas diferentes. Contar con una red preentrenada capaz de extraer características y generalizar tantas clases es de gran ayuda para la resolución de un problema de clasificación mediante imágenes.

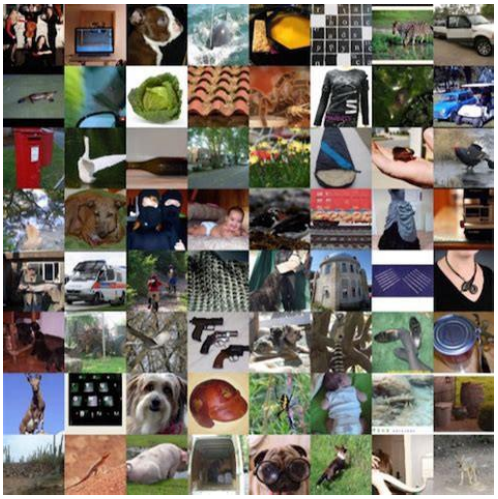


Figura 19. Dataset CIFAR-10, utilizado para el entrenamiento de la red ResNet18.
Fuente: https://www.researchgate.net/figure/Images-from-the-CIFAR-10-13-dataset-and-their-corresponding-classes-CIFAR-10_fig1_329969195

3.2.2 Arquitectura ResNet18

La red neuronal *ResNet18* cuenta con una estructura como se observa en la figura 21. Los procesos realizados en esta van a ser explicados a continuación. Una de las grandes propiedades de este tipo de red es la posibilidad de saltarse capas a la hora de calcular el gradiente en la retropropagación. De esta manera, se consigue disminuir eficazmente el problema del desvanecimiento de gradiente.

El desvanecimiento de gradiente es un problema del entrenamiento de redes neuronales artificiales que consiste en la disminución progresiva del gradiente. Esto produce que la red no pueda modificar de manera significativa los valores de los pesos sinápticos, al depender linealmente del valor del gradiente en su modificación.

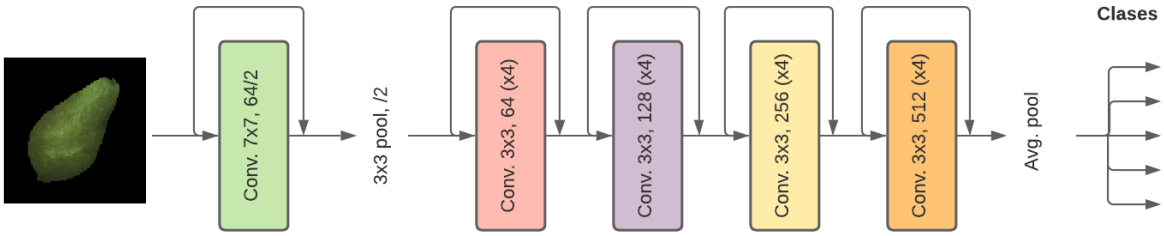


Figura 12. Esquema de la arquitectura ResNet18. Fuente: elaboración propia

Los procesos que realiza una red neuronal convolucional como la ResNet18 para poder realizar una clasificación mediante imágenes son los siguiente:

- *Convolución:*

La convolución es la operación que nos permite extraer la información de interés de las matrices que formen parte de la red neuronal convolucional. Para realizar esta operación se hace uso de un filtro matricial que sigue los mismos principios que un perceptrón.

Este filtro matricial consta del mismo número de matrices como tengan los datos de entrada, pero con un tamaño inferior (en el caso de la ResNet18, 7x7 en la primera y 3x3 en el resto) Estas matrices son conocidas como kernel, y están formadas por el equivalente a los pesos en un perceptrón.

La convolución consiste en recorrer con un patrón fijo las matrices de entrada e ir calculando el producto escalar de cada parte de la entrada con el kernel para así extraer la información y poder generar una salida. El cálculo del producto escalar se guarda en una nueva matriz llamada mapa de activación (*activation map*), el cual se hace pasar por la función de activación declarada.

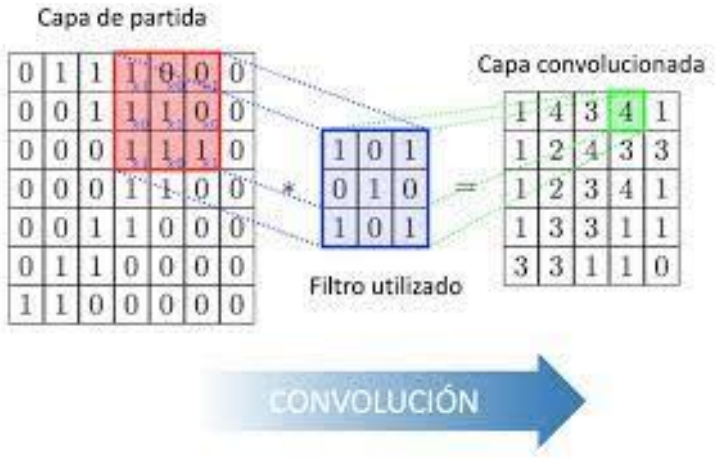


Figura 13. Esquema de una convolución. Fuente: https://www.diegocalvo.es/red-neuronal-convolucional/

El recorrido que realiza el kernel se puede modificar según la información que se pretenda extraer. El kernel se desplaza horizontal y verticalmente según un paso (*stride*) que se puede variar de valor para obtener mapas de activación de mayor o menor tamaño.

Para poder tener un mapa de activación del mismo tamaño que la entrada, existe la posibilidad de aplicar una operación conocida como *padding*, la cual consiste en agregar filas y columnas de valor nulo en el exterior de la matriz de entrada.

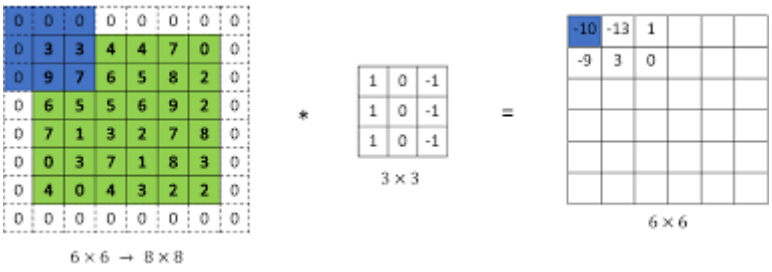


Figura 22. Esquema de un padding. Fuente: http://datahacker.rs/what-is-padding-cnn/

- *Muestreo:*

El muestreo (en inglés *pooling*) consiste en la reducción de las matrices con la intención de reducir el coste computacional de la red neuronal. Al realizar una reducción de parámetros, el modelo pierde precisión y por lo tanto existen diferentes métodos para preservar la información más importante.

El *Max-pooling* es una técnica de muestreo que consiste en la división de las matrices de entrada en submatrices de un tamaño 2x2 y generar una nueva matriz de menor tamaño con los valores máximos de cada submatriz.

El *Average pooling* es otra técnica de muestreo que consiste en un funcionamiento similar al del *Max-pooling* pero en lugar de obtener como salida los mayores valores de cada submatriz, la salida consiste en un valor medio de todos los valores de la misma submatriz.

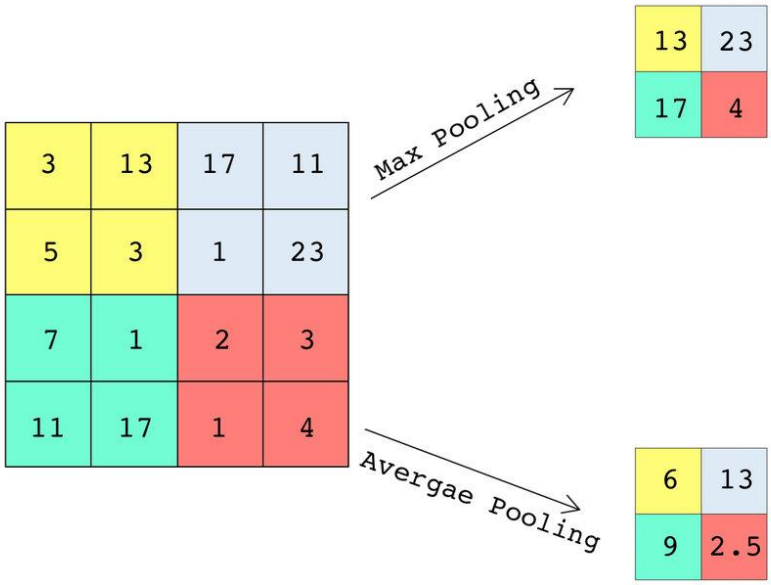


Figura 14. Esquema de Max pooling (arriba) y Average pooling (abajo). Fuente: https://www.researchgate.net/figure/Example-of-max-pooling-and-average-pooling-operations-In-this-example-a-4x4-image-is_fig4_332092821

- *Aplanamiento:*

Tras la realización de las operaciones mencionadas anteriormente, es necesario una última operación para poder calcular las salidas deseadas. Esto ocurre debido a que la salida de ambas sigue siendo una matriz. El aplanamiento (en inglés *flattening*) consiste en la transformación de las matrices producidas por las convoluciones en un único vector de una sola dimensión. De esta manera, es posible utilizar los datos de este vector como entrada de un modelo similar al perceptrón multicapa.

En la última capa de la red neuronal utilizada se aplica un proceso de aplanamiento y posteriormente se realiza una clasificación en 5 clases diferentes.

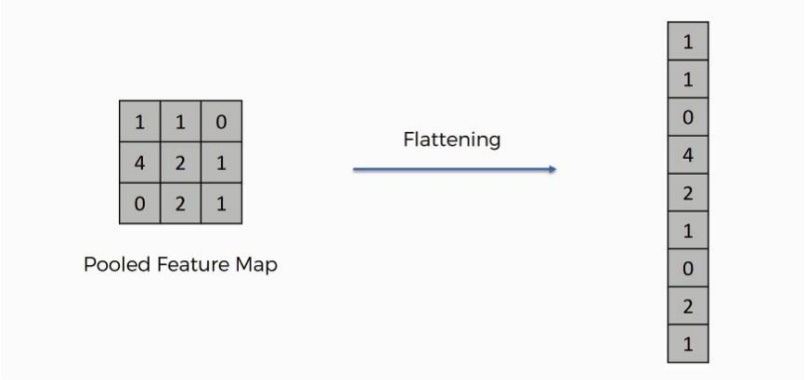


Figura 15. Esquema de un flattening. Fuente: https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-3-flattening

3.2.3 Creación de un DataLoader

El Dataset disponible para la creación de una red neuronal era muy desproporcionado, teniendo 398 peras de una calidad y 10768 de otra. Esto puede causar diversos problemas en el entrenamiento de la red, propiciando que esta dé más importancia a la clase con mayores inputs, por lo que el primer paso en la implementación fue la creción un Dataset equilibrado.

La técnica utilizada para la creación de nuevas imágenes fue la implementación de tres transformaciones diferentes a las imágenes originales. Estas son: un volteo horizontal, un volteo vertical y una rotación. Estas transformaciones han sido implementadas mediante funciones de OpenCV. El código encargado de la creación del nuevo Dataset escoge una imagen de una clase y de manera aleatoria elige si ejecutar cada una de las transformaciones y en caso de la rotación, que ángulo rotar. De esta manera, se ha creado un Dataset con 5000 imágenes de cada una de las clases, tanto en RGB como sus correspondientes en infrarrojos.

Para la utilización de este Dataset en la plataforma *Google Colab* es necesario subir la carpeta que contenga las imágenes a *Google Drive*. Es recomendado subir una carpeta comprimida y descomprimirla mediante el código "!unzip + dirección de la carpeta en Google Drive".

Como ya se ha mencionado en el apartado 2.4, PyTorch utiliza tensores en sus funciones para la creación y entrenamiento de redes neuronales. Para convertir un Dataset de imágenes en un conjunto de tensores, hay que realizar diferentes operaciones.

En primer lugar, hay que definir una serie de transforms. Los transforms son equivalentes a las transformaciones utilizadas en la creación del Dataset. Los transforms implementados en el trabajo son: *Resize,* que permite cambiar el tamaño de la imagen de entrada; *ToTensor*, que convierte las matrices que forman las imágenes en tensores y *Normalize*, que realiza una normalización de las entradas con el fin de facilitar el aprendizaje.

Una vez definidos los transforms a implementar, existen diferentes métodos para cargar un Dataset durante el proceso de aprendizaje. La técnica utilizada en el proyecto ha sido la ejecución de la función *ImageFolder*, que recibe la carpeta que contiene los datos y las transformaciones a aplicarles. Esta función devuelve las entradas con sus transformaciones y como etiqueta el nombre de la carpeta que les contenga, por lo que es importante que cada clase esté agrupada en una carpeta dentro de la carpeta que se le pase a la función.

Una vez definido el Dataset en PyTorch, hay que crear los DataLoaders que se vayan a utilizar en el proyecto. Un DataLoader es una clase de PyTorch que permite cargar los datos. Los DataLoaders permiten también fijar el tamaño de lote (en inglés *batch size*), un hiperparámetro muy importante en el entrenamiento de la red del que se hablará en el siguiente apartado. En la implementación de la red neuronal se han creado 2 DataLoaders, uno destinado al entrenamiento de la red, que contiene un 70% del Dataset y otro destinado a la validación con el 30% restante.

3.2.4 Hiperparámetros

El proceso de entrenamiento de una red neuronal introducido en el apartado 1.6.4 pretende optimizar la red neuronal. El entrenamiento de una red depende de diversos hiperparámetros que se deben ajustar empíricamente con la finalidad de conseguir un mejor rendimiento de la red. El proceso general de ajuste de los hiperparámetros consiste en realizar pruebas con diferentes valores, analizar los resultados sobre el dataset de validación y seleccionar la combinación que obtenga un mejor resultado.

A continuación, se explicarán los hiperparámetros utilizados en el proyecto y se explicará la importancia de cada uno y su reflejo en los resultados.

3.2.4.1 Función de Coste

La función de coste es una función que nos permite calcular el error entre la salida predicha por la red neuronal y el valor real. Los pesos sinápticos son modificados según varíe el error calculado con esta función, también llamado función de pérdidas.

Según el tipo de salida que genere la red y la tarea a la que esté destinada, se utilizan diversas funciones de coste. En el caso del proyecto, una clasificación de imágenes, se usará la función de la entropía cruzada (en inglés *Cross Entropy Loss*)

$$L(\hat{y}, y) = -\sum_{k}^{K} y^{(k)} \log \hat{y}^{(k)}$$

Ecuación 3. Función de Coste Cross Entropy Loss

3.2.4.2 Optimizador

El optimizador en una red neuronal es el encargado de variar los pesos de las capas con la finalidad de minimizar la función de coste y mejorar la precisión de la red. Para realizar esta modificación en el valor de los pesos, se calcula la derivada parcial de cada error y se varía de forma negativa en función de la tasa de aprendizaje. De esta manera, se intenta llegar al mínimo de la función de coste. Este método se llama descenso por gradiente (en inglés *Gradient Descend*)

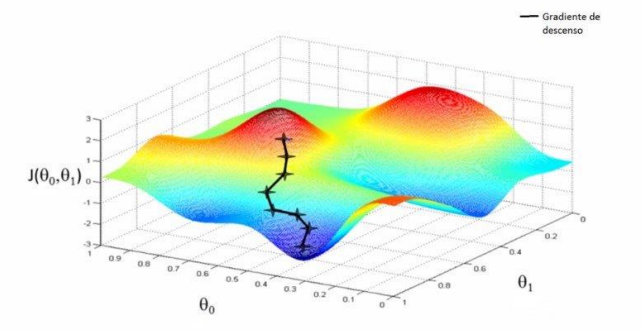


Figura 16. Esquema de un gradiente de descenso. Fuente: https://www.researchgate.net/figure/Optimizacion-gradiente-de-descenso4-Este-gradiente-es-hallado-despues-de-ejecutar-el_fig1_308783857

El optimizador más sencillo que se basa en el descenso del gradiente es el SGD (*Stochastic Gradient Descent*), en el cual el único factor que influye en la dimensión de la variación es la tasa de aprendizaje. Esto proporciona mucha irregularidad a la hora de hallar el mínimo de la función de coste, pudiéndote estancar en mínimos locales en casos de tasa de aprendizaje bajas o no pudiendo alcanzar el mínimo por ser demasiado grande.

$$W = W - \alpha * \Delta W$$

Ecuación 4. Optimizador SGD

Para intentar evitar esto, existen optimizadores que añaden más factores a la hora del cálculo del nuevo peso sináptico. Uno de los optimizadores más utilizados es el Adam (*Adaptative Moment Estimation*)

El optimizador Adam incorpora dos momentos, m y v, m modela la media de los gradientes y v hace lo mismo con la varianza. Los parámetros β1 y β2 se suelen fijar en 0.9 y 0.99 respectivamente. Estos valores se encargan de dar menos importancia a entradas anteriores respecto a las nuevas. Por esta razón, se ha decidido utilizar un optimizador Adam en la red neuronal creada para este proyecto.

$$m = \beta_1 * m + (1 - \beta_1) * \Delta W$$
$$v = \beta_2 * v + (1 - \beta_2) * \Delta W^2$$
$$W = W - \frac{\alpha * m}{\sqrt{v + \varepsilon}}$$

Ecuación 5. Optimizador Adam

3.2.4.3 Funciones de Activación

En apartados anteriores, se han introducido las funciones de activación y su función. En el apartado 1.5.4 se comentó la necesidad de funciones de activación derivables para aplicar el método del descenso de gradiente.

Existen diferentes tipos de funciones de activación derivables, todas ellas con una derivada simple para minimizar el coste computacional. Las funciones de activación más utilizadas son:

- *Sigmoide:*

La función sigmoide varía en un rango entre 0 y 1, tendiendo asintóticamente a 1 en los valores altos y a 0 en los bajos. Esta función fue la más popular durante muchos años, aunque recientemente ha perdido popularidad debido a que las redes neuronales con muchas capas tienen la dificultad de su entrenamiento a causa del problema de desvanecimiento del gradiente.

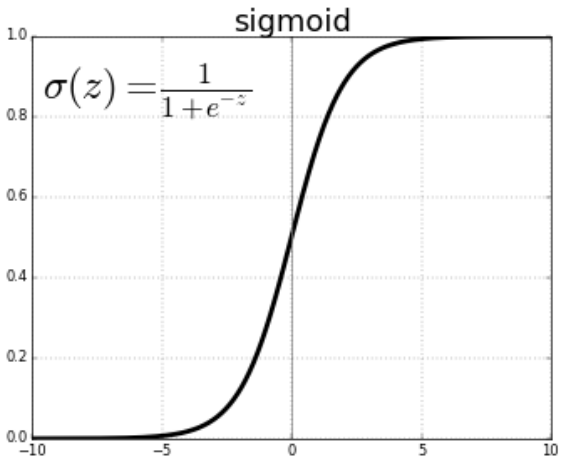


Figura 26. Gráfica Sigmoide. Fuente: elaboración propia

- *Tangente hiperbólica:*

La función tangente hiperbólica transforma los valores introducidos en un rango entre -1 y 1, de manera que los valores más grandes tienen a 1 y los bajos a -1. Esta función tiene el mismo problema que la sigmoide, y se utiliza principalmente en casos en los que hay que clasificar entre 2 opciones contrarias.

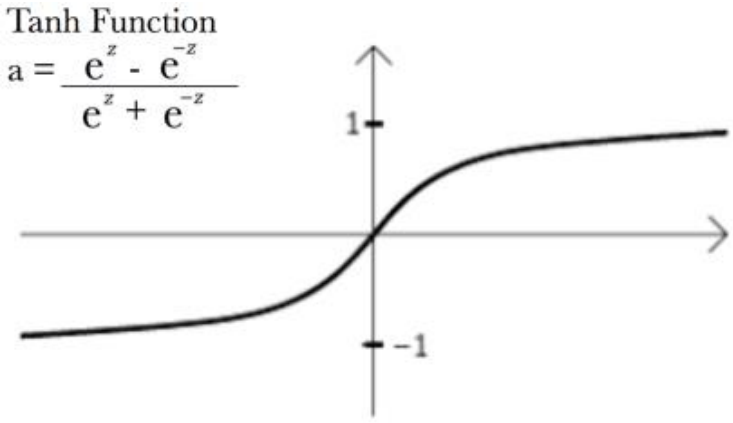


Figura 27. Gráfica tangente hiperbólica. Fuente: elaboración propia

- *ReLU:*

En la actualidad, la función de activación más utilizada es la ReLU. El funcionamiento de esta función de activación consiste en dejar pasar todos los valores positivos y anular los negativos. Esta función no está acotada, por lo que ayuda a evitar el problema del desvanecimiento del gradiente. La función de activación ReLU funciona muy bien en redes convolucionales, por lo que se ha decidido utilizar esta función en el proyecto.

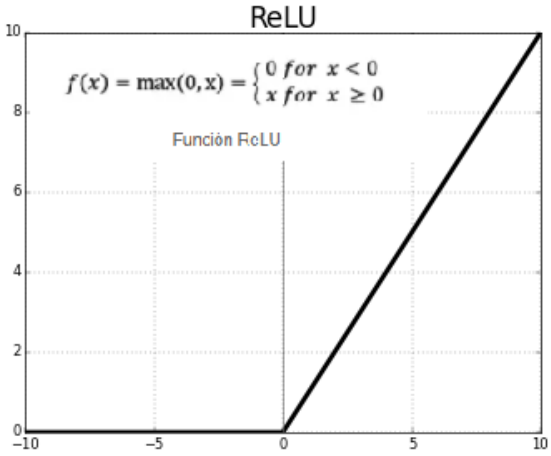


Figura 178. Gráfica ReLU. Fuente: elaboración propia

3.2.4.4 Tasa de Aprendizaje

La tasa de aprendizaje es el hiperparámetro más importante en las redes neuronales. Como ya se ha comentado, la tasa de aprendizaje es el valor por el que se multiplica el gradiente con la intención de minimizar el coste.

El principal problema de la tasa de aprendizaje es que se necesita un ajuste muy preciso. En caso de que la tasa de aprendizaje sea muy pequeña, la red neuronal sería capaz de llegar al mínimo, pero el aprendizaje será demasiado lento, produciendo tiempos de aprendizaje exageradamente elevados.

Por el contrario, si se escoge una tasa de aprendizaje demasiado grande para el problema, llegará un punto en el que no se podrá alcanzar el mínimo ya que el cambio en los pesos será excesivo y esto generará un "rebote", llegando al otro lado del mínimo sin alcanzarlo.

PyTorch ofrece la posibilidad de variar el valor de la tasa de aprendizaje a medida que se van realizando iteraciones en el entrenamiento. Esto se realiza con la función *StepLR*, a la cual se le pasa el optimizador previamente definido, el número de épocas en el que se quiere que cambie el valor y el valor por el que se multiplicará la tasa de aprendizaje.

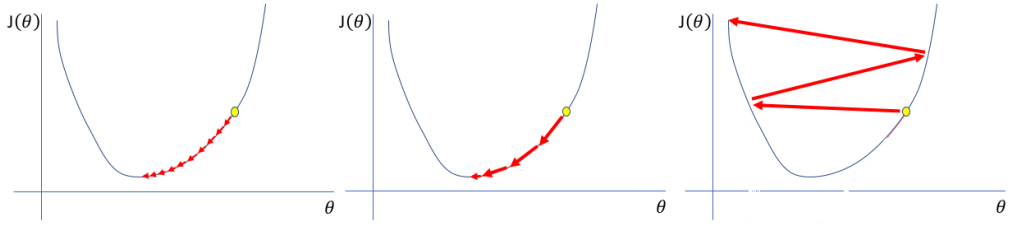


Figura 18. Ejemplo de tasa de aprendizaje pequeña (izquierda), correcta (centro) y grande (derecha). Fuente: https://www.jeremyjordan.me/nn-learning-rate/

3.2.4.5 Tamaño de Lote

El tamaño de lote es el número que indica los ejemplares de entrenamiento en una iteración. Cuanto mayor sea el tamaño de lote utilizado, mayor será el coste computacional y el proceso de entrenamiento necesitará más memoria. Se considera una iteración el cálculo del coste y el paso hacia atrás en el que se recalculan los nuevos pesos.

Se llama una época (en inglés *epoch*) cuando se realiza una iteración de todas las entradas de entrenamiento. Por lo tanto, considerando que tuviéramos 100 datos de entrada, si el tamaño de lote fueran 50 sólo se necesitarían 2 iteraciones, mientras que si el tamaño de lote fuera 20 se necesitarían 5 iteraciones.

Encontrar la combinación perfecta entre tamaño de lote e iteraciones es un proceso empírico. Por experiencia, se conoce que tamaños de lote muy elevados demandan mayor cantidad de memoria de GPU. Por el contrario, tamaños de lote excesivamente pequeños se traducen en peores resultados a causa del ruido inherente en el Dataset.

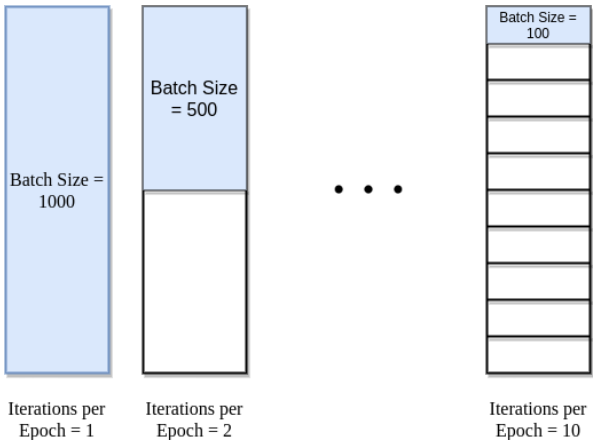


Figura 19. Esquema de relación entre tamaño de lote e iteraciones. Fuente: https://www.baeldung.com/cs/epoch-neural-networks

3.2.4.6 Weight Decay

El weight decay es un hiperparámetro de las redes neuronales cuya función es penalizar a la función de coste con pesos grandes y de esta manera consiga una generalización mejor, mejorando los problemas de sobreentrenamiento mencionados en el apartado 1.4.1. Esta penalización se realiza añadiendo un término a la función de coste.

El ajuste del weight decay, como el de otros hiperparámetros, es empírico. Valores muy altos de weight decay producirán que la red no sea capaz de entrenarse correctamente. Por el contrario, valores muy bajos no producirán ningún efecto notable en el rendimiento de la red

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left[-y^{(i)}\log(h_\theta(x^{(i)})) - (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2.$$

Ecuación 6. Fórmula del weigh decay

Capítulo 4: Resultados

En este apartado se muestran y comentan los resultados obtenidos con las diferentes técnicas de clasificación de objetos.

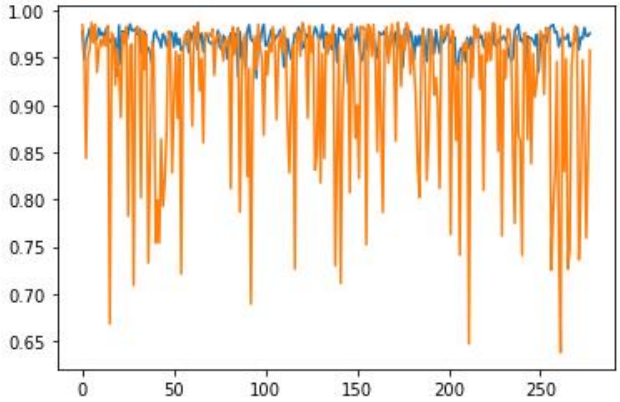
En primer lugar, se mostrarán los resultados obtenidos con cada una de las características extraídas. A continuación, se mostrarán los resultados de cada clasificador de aprendizaje tradicional, utilizado para realizar una preselección de las peras. Para cada clasificador hay un resultado usando las imágenes a color y otro con las infrarrojas. Los datos utilizados están separados en 2 categorías, apto y no apto.

En los resultados obtenidos con la red neuronal, utilizadas para realizar una clasificación en función de la calidad, se muestran 3 casos de estudio: 5000 imágenes de cada clase a color, 5000 infrarrojas de cada clase y 2500 de cada tipo. Los datos utilizados están clasificados en 5 categorías en función de su calidad.

Para cada caso, se especifican los hiperparámetros tasa de aprendizaje, tamaño de lote y *weight decay*, y se muestra una función de confusión que ayudará al estudio de los resultados.

4.1 Características

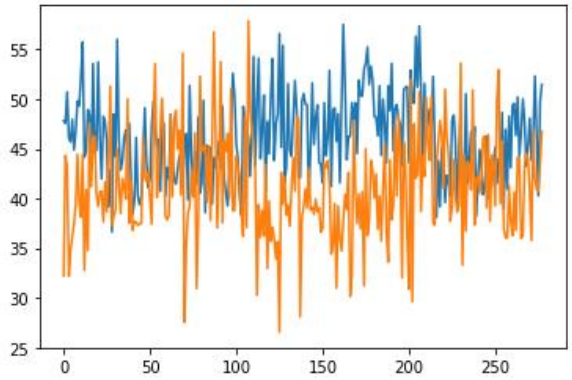
- *Solidez:*



Gráfica 1. Característica de solidez en peras de buena calidad (azul) y mala calidad (naranja). Fuente: elaboración propia

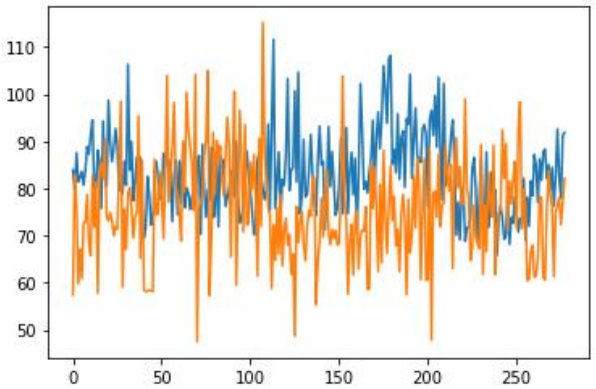
- *Media de colores:*

Azul:



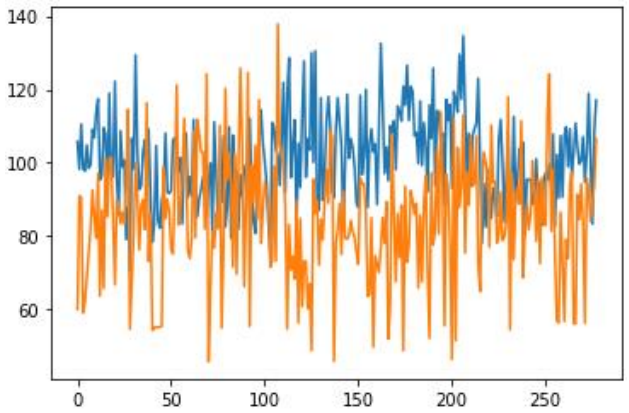
Gráfica 2. Característica de color azul medio en peras de buena calidad (azul) y mala calidad (naranja). Fuente: elaboración propia

Rojo



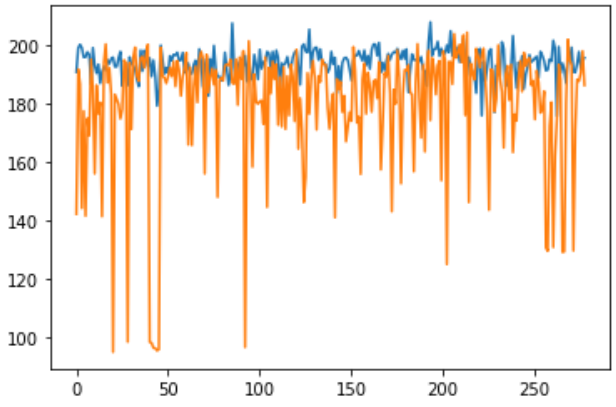
Gráfica 3. Característica de color rojo medio en peras de buena calidad (azul) y mala calidad (naranja). Fuente: elaboración propia

Verde:



Gráfica 3. Característica de color verde medio en peras de buena calidad (azul) y mala calidad (naranja). Fuente: elaboración propia

Infrarrojos:



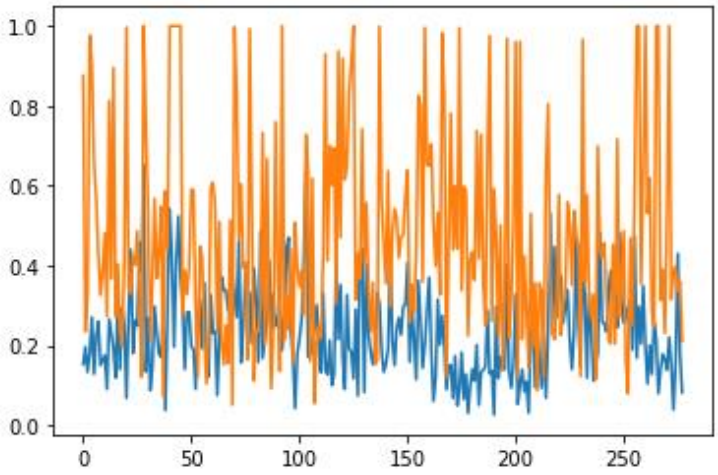
Gráfica 2. Característica de color medio en imágenes infrarrojas en peras de buena calidad (azul) y mala calidad (naranja). Fuente: elaboración propia

- Proporción de defectos

Imágenes RGB

Filtro utilizado: píxel defectuoso si su valor de verde es menor que 80 o su valor de rojo menor que 50

Estrategia utilizada: nº de píxeles defectuosos/nº de pixeles de pera

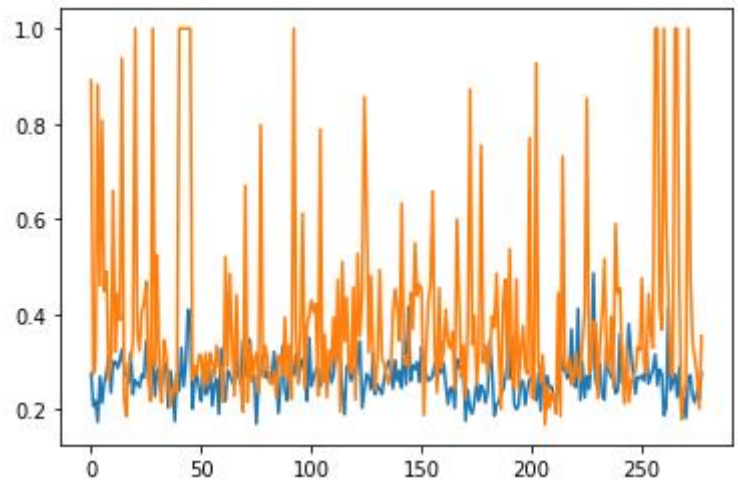


Gráfica 4. Característica de proporción de defectos en imágenes RGB en peras de buena calidad (azul) y mala calidad (naranja). Fuente: elaboración propia.

Imágenes Infrarrojas:

Filtro utilizado: píxel defectuoso si su valor es menor que 170.

Estrategia utilizada: nº de píxeles defectuosos/nº de pixeles de pera



Gráfica 5. Característica de proporción de defectos en imágenes infrarrojas en peras de buena calidad (azul) y mala calidad (naranja). Fuente: elaboración propia

4.2 Clasificadores

- KNN

RGB:

Características utilizadas: Solidez, color verde medio, color rojo medio, proporción de defectos

k = 7

Precisión: 77.08%

```
      testLabels:      [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

```
      predictedLabels: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 1 1 1 1 1 0 0 0 1 1 1 0 1 0 0 1 1 0 0 0 0 1 0 0 1 0 0 1 1 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0 0 1 1 1 1 1 0 0 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 0 1
0 1 0 0 1 0 1 1 1 0 0 1 1 1 0 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 0 1 0 0 1
1 1 0 1 1 0 1 0 1 1 1 0 1 1 0 1 1 1 0 0 1 1 0 0 1 0 1 1 1 1 1 1 0 1 1 1 0
1 1 1 1 1 1 1 1 0 1 1 0 1 0 1 0 1 0]
```

Infrarrojos:

Características utilizadas: Solidez, color medio, proporción de defectos.

k = 5

Precisión: 76.67%

```
      testLabels:      [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

```
      predictedLabels: [0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 1 0 1 1 0 1 0 1 0 0 0 0 1 0 1 0 0 0 0 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0 1 0 1
0 1 0 1 0 0 0 0 1 1 0 1 1 0 1 1 1 1 1 0 1 0 1 1 1 0 1 1 0 1 1 1 0 1 0 1 1
1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 0 1 1 0 0 1 1 0 1 1 1 1 0 1 0 0 1 1 1 1 1 0
1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 0 1 0]
```

- Bayesiano:

RGB:

Características utilizadas: Solidez, color verde medio, color rojo medio, color azul medio y proporción de defectos

Precisión: 75.83%

```
testLabels:      [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

```
predictedLabels: [0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1
 0 0 1 0 1 1 0 0 0 1 1 1 0 1 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0
 0 1 0 0 1 0 0 0 0 1 1 0 0 1 1 0 0 0 1 1 1 1 1 1 1 0 1 1 1 0 0 1 1 0 1 0 1
 0 1 0 1 1 0 1 1 1 0 0 0 1 0 0 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 0 1 0 0 1
 1 1 1 1 0 0 1 0 1 1 1 0 1 1 0 1 1 1 0 0 1 1 0 0 1 0 1 1 1 1 1 1 1 1 1 1 0
 1 1 1 1 1 1 1 1 0 1 1 0 1 0 1 0 1 0]
```

Infrarrojos:

Características utilizadas: Solidez, color medio, proporción de defectos.

Precisión: 75.42%

```
testLabels:      [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

```
predictedLabels: [0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 1 0 0 1 0 0 0 0 1 1 0 1 0 1 0 1 1 1 0 1 1 1 1 1 0 1 0 1 0 0 1 0 0 0 0 1
 0 1 0 1 0 0 0 0 0 0 0 0 1 0 1 1 1 0 1 0 1 0 1 1 1 0 1 1 0 1 1 1 0 1 0 1 0
 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 0 0 1 0 0 0 1 0 0 1 1 0 0 1 0 0 1 1 1 1 1 0
 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 0 0 0]
```

- SVM:

RGB:

Características utilizadas: Solidez, color verde medio, color rojo medio, color azul medio y proporción de defectos

Precisión: 78.333%

C = 20

```
      testLabels:     [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

```
      predictedLabels: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.
0. 1. 0. 0. 1. 1. 1. 1. 0. 0. 0. 1. 1. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
1. 0. 0. 1. 0. 0. 1. 1. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0.
1. 1. 1. 0. 1. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 0. 1. 1. 1.
0. 1. 0. 1. 0. 1. 1. 0. 1. 0. 1. 1. 1. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 0.
1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 0. 1. 1. 1. 1. 0. 1. 1. 0. 1.
0. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 0. 1. 1. 0. 0. 1. 0. 1. 1. 1. 1. 1.
1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 0. 1. 0. 1. 0.]
```

Infrarrojos:

Características utilizadas: Solidez, color medio, proporción de defectos.

Precisión: 79.583%

C = 60

```
      testLabels:     [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

```
      predictedLabels: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.
0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0.
1. 1. 0. 1. 0. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 0. 1. 1. 0.
1. 0. 0. 1. 0. 1. 0. 1. 1. 1. 0. 0. 1. 0. 0. 1. 1. 0. 1. 1. 1. 0. 1. 0.
1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 0. 1. 0. 1. 0. 1. 1. 1. 1. 1. 0. 1.
0. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 0. 0. 0. 1. 0. 0. 1. 1. 0. 1. 1. 0. 0.
0. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 0. 1. 0. 0. 0.]
```

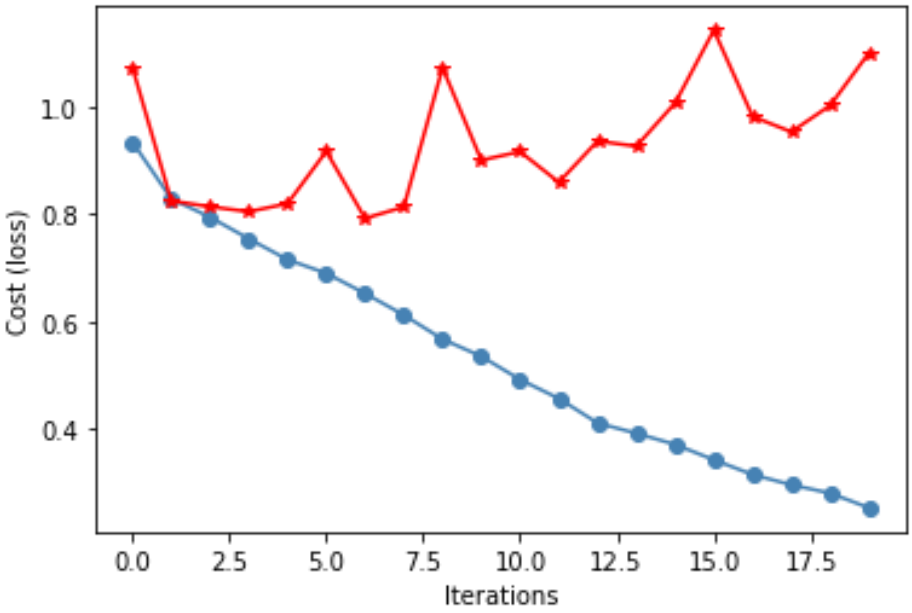
4.3 ResNet18

- Imágenes a color:

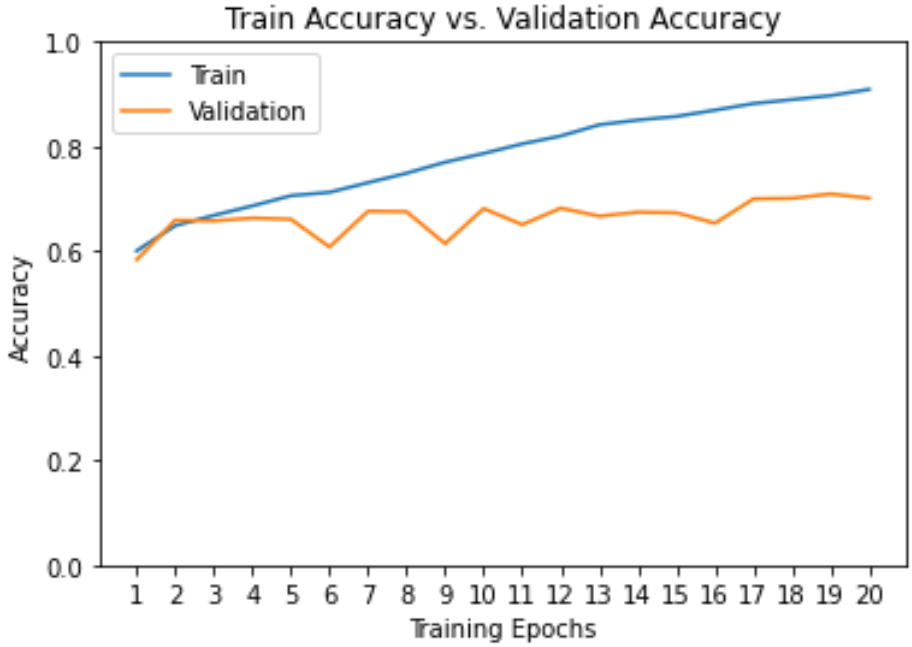
Tasa de aprendizaje: 0.001

Tamaño de lote: 64

Weight decay: 0.001



Gráfica 6. Coste de entrenamiento en imágenes a color. En rojo el coste de validación y en azul el coste de entrenamiento. Fuente: elaboración propia



Gráfica 7. Precisión en imágenes a color. En naranja la precisión de validación y en azul la precisión de entrenamiento. Fuente: elaboración propia

Tiempo de entrenamiento: 39.45 minutos, 1.9725 min/epoch, 633 imágenes/minuto

Mejor precisión: 70.9239%

Precisión por clase:

```
1 | Correct: 1307.0 | Total: 1480.0 | Accuracy: 0.8831081081081081
2 | Correct: 857.0 | Total: 1512.0 | Accuracy: 0.5667989417989417
3 | Correct: 800.0 | Total: 1500.0 | Accuracy: 0.5333333333333333
4 | Correct: 1018.0 | Total: 1517.0 | Accuracy: 0.6710613052076466
5 | Correct: 1338.0 | Total: 1492.0 | Accuracy: 0.8967828418230563
```

Matriz de confusión:

		Clase predicha				
		Clase 1	Clase 2	Clase 3	Clase 4	Clase 5
Clase real	Clase 1	1307	155	16	2	0
	Clase 2	297	857	302	51	5
	Clase 3	82	338	800	260	20
	Clase 4	6	43	242	1018	208
	Clase 5	1	3	18	132	1338

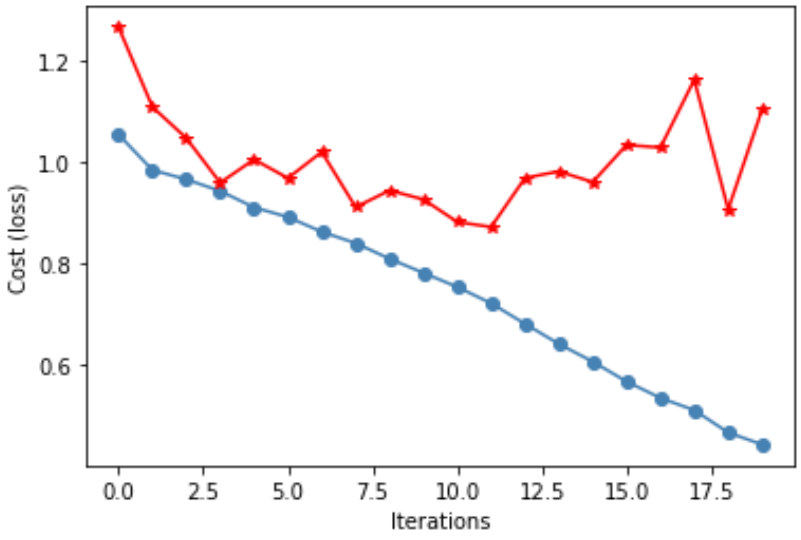
Gráfica 8. Matriz de confusión en imágenes a color. Fuente: elaboración propia

- Imágenes infrarrojas:

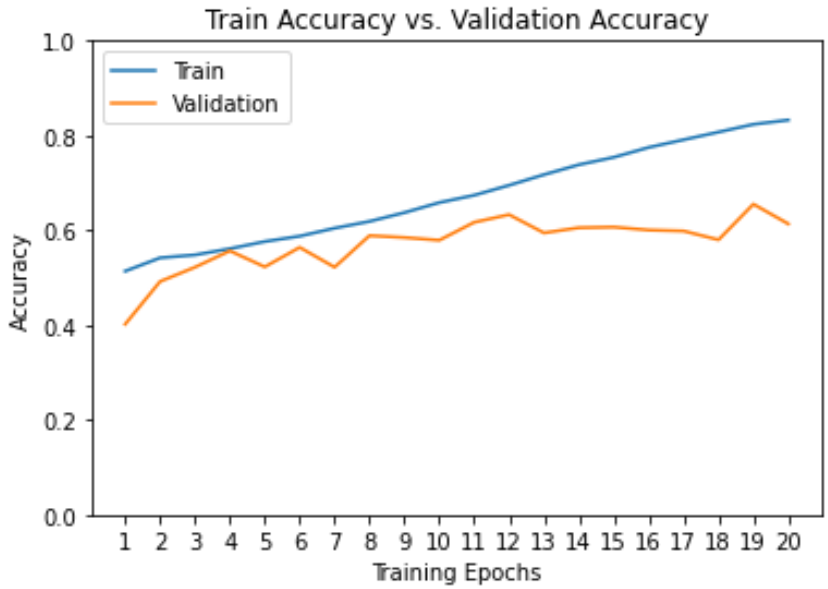
Tasa de aprendizaje: 0.001

Tamaño de lote: 64

Weight decay: 0.002



Gráfica 11. Coste de entrenamiento en imágenes infrarrojas. En rojo el coste de validación y en azul el coste de entrenamiento. Fuente: elaboración propia



Gráfica 12. Precisión en imágenes infrarrojas. En naranja la precisión de validación y en azul la precisión de entrenamiento.
Fuente: elaboración propia

Tiempo de entrenamiento: 39.4 minutos, 1.97 min/epoch, 634 imágenes/minuto

Mejor precisión: 65.5246 %

Precisión por clase:

```
1 | Correct: 807.0 | Total: 1488.0 | Accuracy: 0.5423387096774194
2 | Correct: 1111.0 | Total: 1502.0 | Accuracy: 0.7396804260985352
3 | Correct: 755.0 | Total: 1536.0 | Accuracy: 0.4915364583333333
4 | Correct: 1074.0 | Total: 1471.0 | Accuracy: 0.7301155676410604
5 | Correct: 1168.0 | Total: 1504.0 | Accuracy: 0.776595744680851
```

Matriz de confusión:

		Clase predicha				
		Clase 1	Clase 2	Clase 3	Clase 4	Clase 5
Clase real	Clase 1	807	613	37	28	3
	Clase 2	164	1111	179	45	3
	Clase 3	39	352	755	364	26
	Clase 4	5	63	178	1074	151
	Clase 5	0	16	14	306	1168

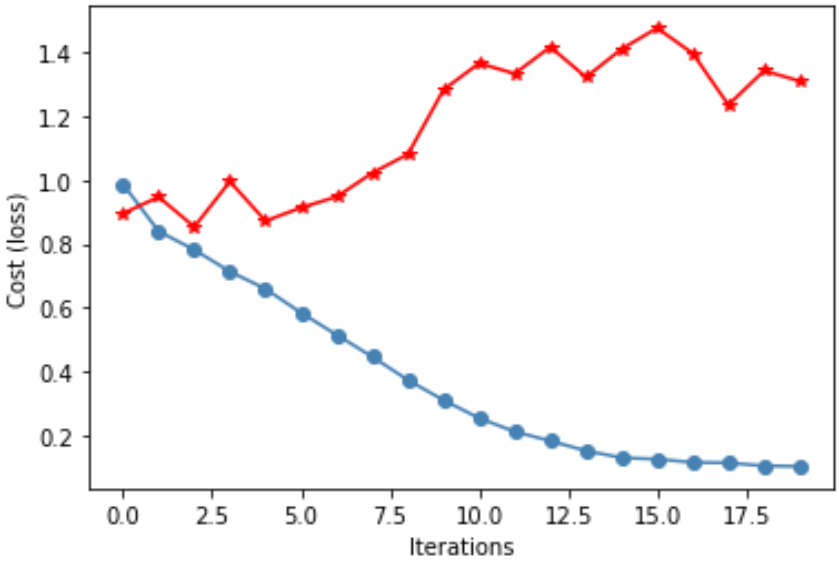
Gráfica 13. Matriz de confusión en imágenes infrarrojas. Fuente: elaboración propia

- Mezcla de imágenes:

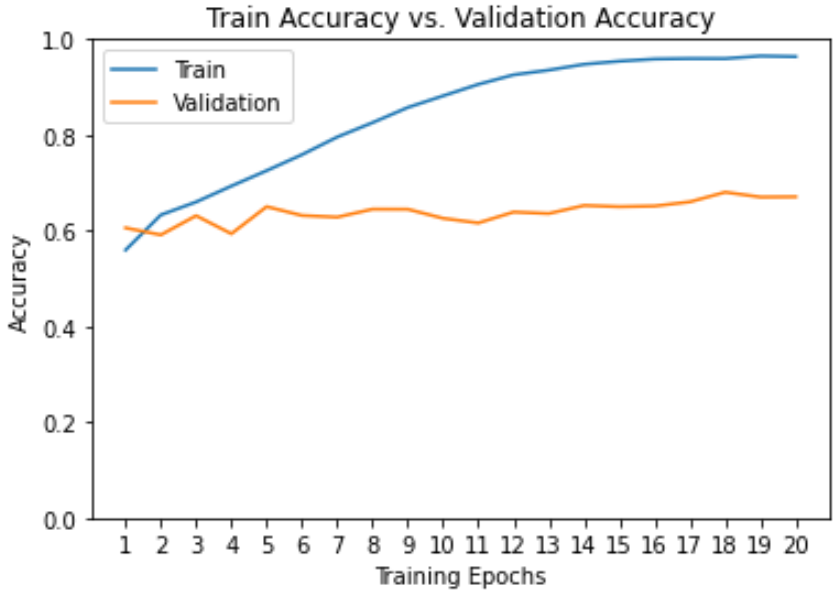
Tasa de aprendizaje: 0.0005

Tamaño de lote: 64

Weight decay: 0.0005



Gráfica 14. Coste de entrenamiento en imágenes mezcladas. En rojo el coste de validación y en azul el coste de entrenamiento.
Fuente: elaboración propia



Gráfica 10. Precisión en imágenes mezcladas. En naranja la precisión de validación y en azul la precisión de entrenamiento. Fuente: elaboración propia

Tiempo de entrenamiento: 39.25 minutos, 1.9625 min/epoch, 637 imágenes/minuto

Mejor precisión: 68.0395 %

Precisión por clase:

```
1 | Correct: 1257.0 | Total: 1521.0 | Accuracy: 0.8264299802761341
2 | Correct: 683.0 | Total: 1455.0 | Accuracy: 0.46941580756013745
3 | Correct: 904.0 | Total: 1497.0 | Accuracy: 0.603874415497662
4 | Correct: 951.0 | Total: 1521.0 | Accuracy: 0.6252465483234714
5 | Correct: 1310.0 | Total: 1509.0 | Accuracy: 0.8681245858184228
```

Matriz de confusión:

		Clase predicha				
		Clase 1	Clase 2	Clase 3	Clase 4	Clase 5
Clase real	Clase 1	1257	208	51	2	3
	Clase 2	398	683	324	45	5
	Clase 3	105	253	904	218	17
	Clase 4	18	54	290	951	208
	Clase 5	2	4	37	156	1310

Gráfica 16. Matriz de confusión en imágenes mezcladas. Fuente: elaboración propia

4.4 Análisis de los resultados

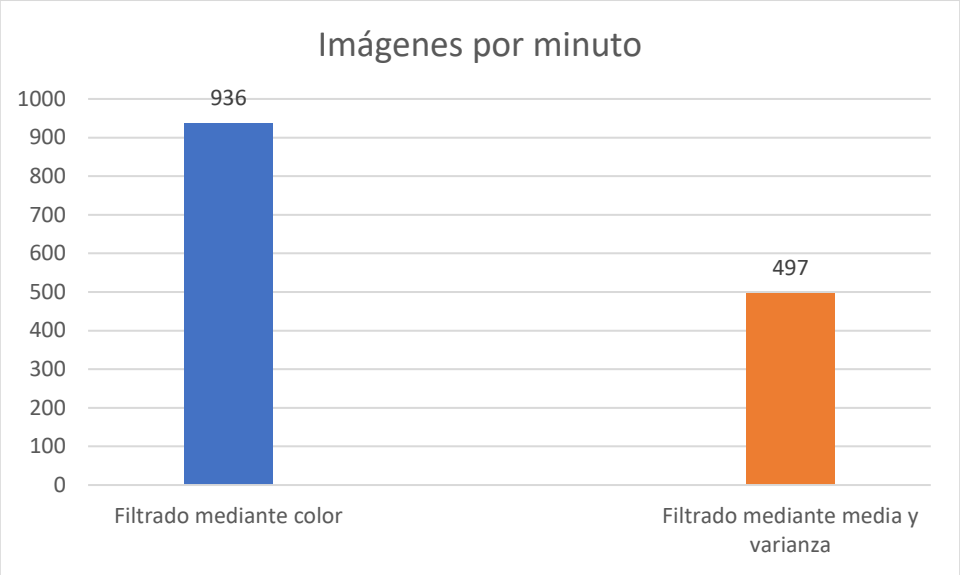
El objetivo del proyecto es analizar diferentes técnicas de clasificación y su rendimiento con diferentes tipos de entrada. Se pretende optimizar la precisión de aciertos, teniendo en cuenta el coste temporal de su entrenamiento entre otros factores.

4.4.1 Tiempo de entrenamiento

Una de las principales diferencias entre las técnicas de clasificación por visión artificial es el coste computacional de cada una de ellas, afectando en su coste temporal.

En cuanto al tiempo de ejecución de los clasificadores de aprendizaje tradicional, el mayor tiempo lo encontramos en la extracción de características. En este proceso, podemos notar una diferencia notable en función de si se realiza un preprocesamiento de la imagen. En concreto, el método de cálculo de la media y la varianza de las imágenes para reconocer defectos es el más largo, al tener que recorrer las imágenes píxel a píxel varias veces.

Sin embargo, se considera que los tiempos obtenidos con ambos métodos cumplen satisfactoriamente, al no ser un tiempo excesivamente elevado. Además, cabe comentar que el tiempo de ejecución dependerá en función del dispositivo utilizado. En el uso de estas técnicas no se ha utilizado una GPU, por lo que su uso aceleraría considerablemente los cálculos.

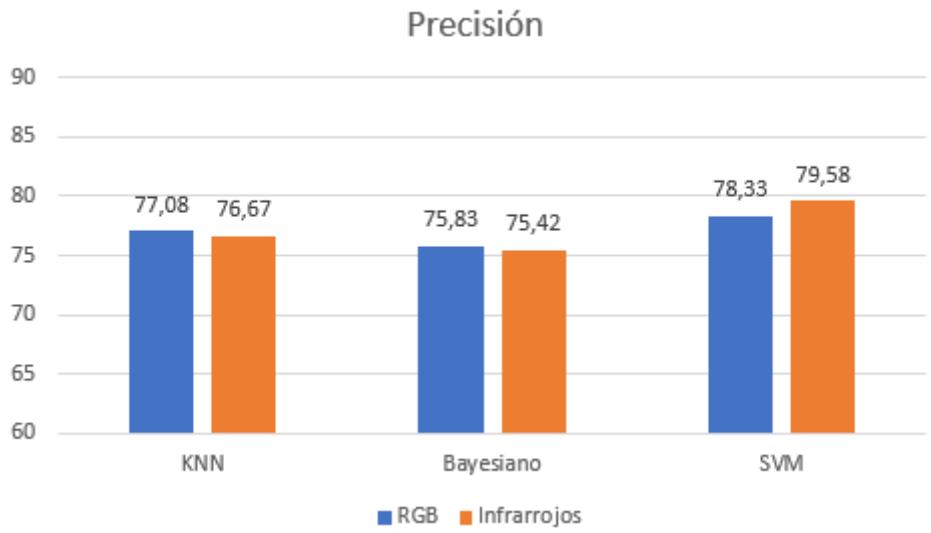


Gráfica 17. Tiempo de ejecución de dos métodos de preprocesamiento de imagen

Respecto al tiempo de entrenamiento de las redes neuronales, el factor más importante es también el entorno en el que se ejecute. Como ya se comentó en el apartado 2.5, *Google Colab* nos proporciona una GPU con la que se han realizado los entrenamientos de las redes. Respecto a los 3 casos analizados, imágenes a color, infrarrojos y una mezcla de ambas, no se encuentran diferencias notables en el tiempo de ejecución.

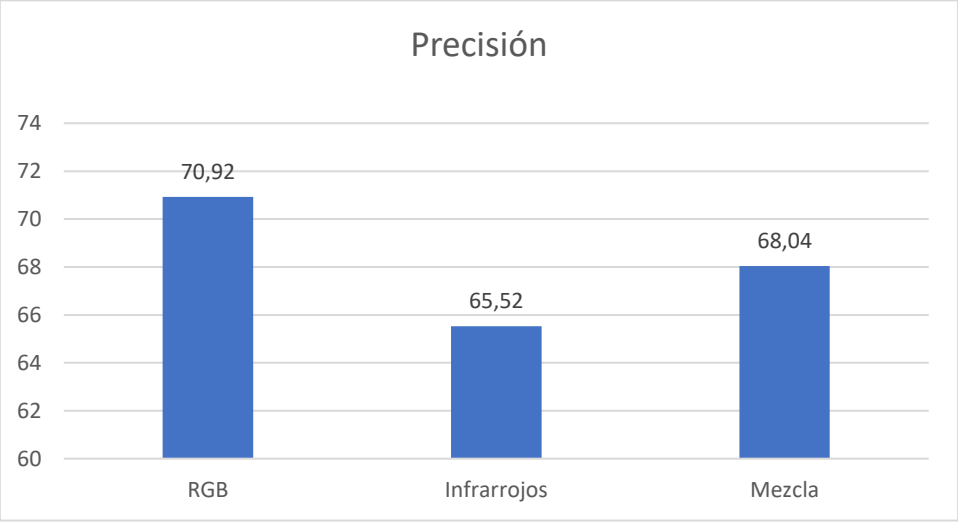
4.4.2 Precisión

La precisión ha sido el principal parámetro a optimizar en el trabajo. En cuanto a los resultados de los clasificadores de aprendizaje tradicional enfocados en la preselección en la línea de manipulación, se obtienen resultados similares en todos los casos analizados, siendo el SVM el que mejores resultados obtiene. En general, no existe una diferencia notable entre el uso de imágenes RGB o infrarrojos, pero en el SVM se obtiene un poco más de precisión con el uso de infrarrojos. Esto es debido a que con las imágenes infrarrojas existe la posibilidad de encontrar defectos internos además de los externos. Otra razón de esta diferencia es la mayor facilidad de reconocimiento de las zonas defectuosas, al tratarse de imágenes en blanco y negro y por lo tanto ser una única matriz en lugar de 3.



Gráfica 18. Precisión de los clasificadores de aprendizaje reforzado. Fuente: elaboración propia

En cuanto a las redes neuronales convolucionales utilizadas para la clasificación en función de la calidad se advierten ligeras mejoras en función del uso de imágenes a color. Esto es debido a que la diferenciación entre calidades de pera se realiza principalmente en función del color y la forma.



Gráfica 19. Precisión de los casos de estudio de la red ResNet18. Fuente: elaboración propia

El primer objetivo que se pretendía alcanzar con el desarrollo de este trabajo es el desarrollo de una técnica de visión artificial capaz de realizar una clasificación de las peras en aptas o no aptas para su futura implementación en el proceso de preselección de una línea de manipulación.

Este objetivo se considera que se ha cumplido ya que, como se observa en la gráfica 14, las peras de peor calidad se clasifican con un 80% de aciertos, lo que se considera una buena precisión para el alcance del trabajo.

El segundo objetivo consistía en realizar una clasificación de las peras en función de su calidad, para implementarlo en el proceso de encajado del almacén. Se han diferenciado 5 calidades de peras, correspondiendo con las clases proporcionadas por el Instituto de Automática e Informática Industrial (ai2).

Con una precisión de hasta un 70% con las redes neuronales convolucionales, se considera que el objetivo se ha cumplido. Es cierto que existe un margen de mejora posible, pero se considera una buena aproximación ya que en caso de fallo se aproxima mucho a las calidades cercanas. Esto se puede observar analizando las matrices de confusión.

Las matrices de confusión consisten en una representación del desempeño de un algoritmo de clasificación. En las matrices de confusión del proyecto, se representan en cada una de las filas el valor real de la clase a la que pertenecen las peras y en cada una de las columnas el valor predicho. Por lo tanto, un clasificador perfecto tendría una matriz de confusión puramente diagonal.

Si se analiza el desempeño de la red con la ayuda de la matriz de confusión, se puede observar un mejor desempeño del percibido con las precisiones totales. Al realizar el cálculo de las peras que han sido clasificadas en una sola clase superior o inferior, se concluye que la red ofrece una buena aproximación a la calidad de la pera.

	Acierto (%)	Errores por 1 sola clase (%)	Errores restantes (%)
RGB	70,92	25,78	3,29
Infrarrojos	65,52	30,76	3,72
Mezcla	68,04	27,39	4,57

Gráfica 20. Precisión de las redes neuronales. Fuente: elaboración propia

Capítulo 5: Conclusiones

En el presente trabajo se han implementado y evaluado diferentes técnicas de visión por computador con la finalidad de realizar una clasificación de las peras de un almacén en función de su calidad.

Los objetivos de clasificación tanto en aptos y no aptas como en diferentes clases según su calidad han sido cumplidos. Para lograr esto, se han optimizado empíricamente todas las técnicas utilizadas en la realización del proyecto.

Las técnicas de clasificación mediante un aprendizaje supervisado tradicional han resultado ser unos métodos con un gran potencial para el filtrado inicial en el proceso de clasificación de peras. Estos métodos han demostrado tener un gran rendimiento a pesar de su bajo coste computacional. La aplicación de estas técnicas para resolver el problema de clasificación en función de la calidad resultaría más complejo de realizar, debido a las menores diferencias entre clases y la dificultad que eso supondría en la selección de las características a utilizar.

Las redes neuronales convolucionales aplicadas a la clasificación han demostrado ser una técnica con un gran potencial en la automatización del proceso de clasificación en un almacén de peras. Pese a esto, todavía existe margen de mejora en cuanto a la precisión final.

Como se puede observar en las gráficas de la precisión y coste en el entrenamiento de las redes, existe un gran sobreentrenamiento, ya que se logran obtener precisiones de más del 90% para los datos de entrenamiento, pero sin mejorar la precisión de los datos de validación.

Una de las formas de mejora consistiría en un aumento del Dataset utilizado. Como se comentó en el apartado 3.2.3, existe la posibilidad de crear más imágenes a partir de las ya disponibles. Sin embargo, un aumento con nuevos datos etiquetados sería de gran utilidad debido a la disparidad de datos disponibles en el Dataset, teniendo 10 veces más de una clase que de otra.

Otra de las formas de mejora sería un mejor ajuste en los hiperparámetros de la red. Este ajuste se ha realizado de manera empírica y se ha obtenido un buen resultado con el tiempo como uno de los factores limitantes.

Ambos métodos de mejora suponen un coste computacional que, debido a los límites de *Google Colab* han supuesto una barrera en el desarrollo del proyecto. Con la adquisición de un entorno local de ejecución se podrían obtener mejores resultados, a costa de encarecer el proyecto.

Un trabajo futuro sería realizar el estudio e implantación de este sistema de clasificación en un almacén de peras con el fin de automatizar el proceso de clasificación, además de la evaluación de los resultados obtenidos en su funcionamiento.

Bibliografía

[1] Benlloch, J. V., Agusti, M., Sanchez, A., & Rodas, A. (1995, October). Colour segmentation techniques for detecting weed patches in cereal crops. In *Proc. of Fourth Workshop on Robotics in Agriculture and the Food-Industry* (pp. 30-31).

[2] Sánchez, A. J., Albarracin, W., Grau, R., Ricolfe, C., & Barat, J. M. (2008). Control of ham salting by using image segmentation. *Food Control*, *19*(2), 135-142.

[3] Sánchez, A., & Ramos, C. (2000). SEGUIMIENTO VISUAL DE OBJETOS UTILIZANDO TÉCNICAS DE PREDICCIÓN, XXI Jornadas de Automatica.

[4] Sanchez, A. J., & Marchant, J. A. (2000a). Fusing 3D information for crop/weeds classification. In Proceedings 15th International Conference on Pattern Recognition. ICPR-2000 (Vol. 4, pp. 295-298). IEEE.

[5] Ricolfe-Viala, C., & Sanchez-Salmeron, A. J. (2011, September). Optimal conditions for camera calibration using a planar template. In 2011 18th IEEE International Conference on Image Processing (pp. 853-856). IEEE.

[6] Ivorra, E., Amat, S. V., Sánchez, A. J., Barat, J. M., & Grau, R. (2014). Continuous monitoring of bread dough fermentation using a 3D vision Structured Light technique. Journal of Food Engineering, 130, 8–13. https://doi.org/10.1016/j.jfoodeng.2013.12.031.

[7] Ivorra, E., Sánchez, A. J., Camarasa, J. G., Diago, M. P., & Tardaguila, J. (2015). Assessment of grape cluster yield components based on 3D descriptors using stereo vision. Food Control, 50, 273-282.

[8] Verdú, S., Ivorra, E., Sánchez, A. J., Barat, J. M., & Grau, R. (2015a). Relationship between fermentation behavior, measured with a 3D vision Structured Light technique, and the internal structure of bread. Journal of Food Engineering, 146, 227–233. https://doi.org/10.1016/j.jfoodeng.2014.08.014.

[9] Grau, R., Sánchez, A. J., Girón, J., Iborra, E., Fuentes, A., & Barat, J. M. (2011). Nondestructive assessment of freshness in packaged sliced chicken breasts using SW-NIR spectroscopy. Food Research International, 44(1), 331-337.

[10] Ivorra, E., Sánchez, A. J., Verdú, S., Barat, J. M., & Grau, R. (2016). Shelf life prediction of expired vacuum-packed chilled smoked salmon based on a KNN tissue segmentation method using hyperspectral images. Journal of Food Engineering, 178, 110-116.

[11] Verdú, S., Ivorra, E., Sánchez, A. J., Barat, J. M., & Grau, R. (2015b). Study of high strength wheat flours considering their physicochemical and rheological characterisation as well as fermentation capacity using SW-NIR imaging. Journal of Cereal Science, 62, 31-37.

[12] E.M. Berti, A.J.S. Salmeron, F. Benimeli (2012a). Kalman filter for tracking robotic arms using low cost 3D vision systems. The Fifth International Conference on Advances in Computer–Human Interactions, Valencia, Spain, Jan. 30–Feb. 4, pp. 236-240.

[13] Berti, E.M.; Salmerón, A.J.S.; Benimeli, F. (2012b). Human-Robot Interaction and Tracking Using low cost 3D Vision Systems. Romanian J. Tech. Sci. Appl. Mech. 2012, 7, 1–15.

[14] Martinez, E.; Sanchez, A.; Ricolfe, C.; Nina, O. (2016). Human Pose Estimation for RGBD Imagery with Multi-Channel Mixture of Parts and Kinematic Constraints. WSEAS Trans. Comput. 2016, 15, 279–286.

[15] Martinez, E.; Sanchez-Salmeron, A.J.; Ricolfe-Viala, C. (2017a). 4D-DPM model for pose estimation using Kalman filter constraints. Int. J. Adv. Robot. Syst., 14, 1–13.

[16] Martinez, E.; Nina, O.; Sanchez, A.; Ricolfe, C. (2017b). Optimized 4D-DPM for Pose Estimation on RGBD Channels using polisphere models. In Proceedings of the 12th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications, Porto, Portugal, 27 February–1 March 2017; Volume 5, pp. 281–288.

[17] Martinez-Berti, E., Sánchez-Salmerón, A. J., & Ricolfe-Viala, C. (2017). Dual quaternions as constraints in 4d-dpm models for pose estimation. Sensors (Switzerland).

PRESUPUESTO

3. Presupuesto

El presupuesto consiste en una valoración económica del proyecto "Diseño, implementación y evaluación de técnicas de clasificación de peras mediante técnicas de visión por computador".

La elaboración de este capítulo se ha realizado consultando el documento recomendado por la UPV: "*Recomendaciones en la elaboración de presupuestos en actividades de I+D+I".*

El presupuesto se compone de dos conceptos diferentes: coste del personal y material inventariable.

6.1 Coste del personal

El coste de cada participante del proyecto se calcula según la fórmula:

Coste (€) = Coste horario (€/h) * Dedicación en horas (h)

6.2 Material inventariable

Las amortizaciones de los equipos y licencias informáticas se han calculado según la siguiente fórmula:

$$Coste\ (€) = \frac{\text{Tiempo de uso (meses)} * Coste\ del\ equipo\ (€)}{\text{Periodo de amortización (años)} * 12}$$

Clasificación económica del gasto	Amortización (años)
Adquisición de equipo para procesos de información	6
Adquisición de aplicaciones informáticas	6

6.3 Costes totales

Coste de personal

Denominación del personal	Precio (€)	Cantidad (horas)	Total (€)
Tutor	51,4	30	1542
Graduado de Ingeniería de Tecnologías Industriales	20	300	6000
		Total Personal:	7542

Coste de material inventariable

Concepto	Precio (€)	Amortización (meses)	T. uso	Cantidad	Total
Ordenador	660	6	4	1	36,67
Internet/WIFI	30	6	4	4	1,67
Windows 10	135	6	4	1	7,5
				10%	4,58
				Total Material Inventariable:	50,42

Concepto	Coste Total (€)
Coste de personal	7542
Coste de material inventariable	50,42
Presupuesto de Ejecución Material	7592,42

		Importe
Presupuesto de Ejecución Material		7592,42€
Gastos Generales (13%)		987,01€
Beneficio industrial (6%)		455,55€
Presupuesto de Ejecución por Contrata		9034,98€
IVA (21%)		1897,34€
	Presupuesto de Base de Licitación	10932,00€

El coste total del proyecto asciende a **DIEZ MIL NOVECIENTOS TREINTA Y DOS EUROS**

ANEXOS

1. Código de clasificación para la preselección

```
import numpy as np                # el módulo numpy se asocia con alias np
import cv2                        # módulo de OpenCV
import matplotlib                 # módulo de visualización
import matplotlib.pyplot as plt   # el módulo matplotlib.pyplot con alias plt
from IPython.display import clear_output, Image, display, HTML
import tools_creaVideo as crea    # módulo para generar imágenes virtuales
import ipywidgets as widgets      # funcionalidad de interacción con las gráficas
import os
```

```
features = [[],[]]
clase=0
i=0
carpetas = os.listdir("datasets\\dataset")
for carpeta in carpetas:
    imgs_dir="datasets\\dataset\\"+carpeta+"\\"
    imagenes=os.listdir(imgs_dir)
    for imagen in imagenes:
        i=i+1
        if imagen.endswith(".pgm"):
            img=cv2.imread("datasets\\dataset\\" + carpeta + "\\"+ imagen)
            img_procesada = img.copy()
            imgray = cv2.cvtColor(img_procesada, cv2.COLOR_BGR2GRAY)
            ret, thresh = cv2.threshold(imgray, 0, 255, 0)
            contours, xd = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
            areamax=0
            for cnt in contours:
                area = cv2.contourArea(cnt)
                if (area>areamax):
                    areamax=area
                    perimetro = cv2.arcLength(cnt,True)
                    (x,y),radius = cv2.minEnclosingCircle(cnt)
                    radius = int(radius)
                    hull = cv2.convexHull(cnt)
                    hull_area = cv2.contourArea(hull)
                    solidity = float(area)/hull_area
                    circularidad = (4*np.pi*area)/(perimetro*perimetro)
            gris = 0
            n = 0
            defecto=0
            for x in range(img.shape[0]):
                for y in range(img.shape[1]):
                    if (imgray[x,y]!=0):
                        gris += float(img[x][y][0])
                        n+=1
            for x in range(img.shape[0]):
                for y in range(img.shape[1]):
                    if (imgray[x,y]!=0):
                        if (img[x][y][1]<170):
                            defecto+=1
            proporcion=defecto/n
            gris_medio = gris/n
            features[int(clase)] = features[int(clase)] + [[solidity, gris_medio, proporcion]]
    clase=clase+1
```

```
# División del dataset en train y test
num_datos_1 = len(features[0])
num_datos_2 = len(features[1])
print('Número de datos iniciales:', num_datos_1, num_datos_2)

last_data = min(num_datos_1, num_datos_2)
num_train = int(0.7*last_data)

train_dataset = []
test_dataset  = []
train_dataset = train_dataset + [features[0][0:num_train], features[1][0:num_train]]
test_dataset = test_dataset + [features[0][num_train:last_data], features[1][num_train:last_data]]
print('Número de datos de entrenamiento:', num_train,'y de test:', last_data - num_train    )

# Características de los objetos de interés en formato filas
trainData = np.asarray(train_dataset[0]+train_dataset[1]).astype(np.float32)

# Etiquetas
labels_0 = list(np.zeros(num_train))
labels_1 = list(np.ones(num_train))
trainLabels = np.asarray(labels_0+labels_1).astype(np.int32)

# Para visualizar el dataset debemos formatear los datos por columnas
# Se separan los datos de cada clase
datos_1 = trainData[trainLabels.ravel()==0]
datos_2 = trainData[trainLabels.ravel()==1]

# Se cambia el formato de filas a columnas o viceversa con la tranpuesta
datos_1_column = np.transpose(datos_1, (1, 0))
datos_2_column = np.transpose(datos_2, (1, 0))

# Se visualizan las características individualemnte
num_features = len(features[0][1])
minimos = np.zeros((num_features))
maximos = np.zeros((num_features))
for i in range(num_features):
    fig = plt.figure()
    ax = fig.add_subplot(1, 1, 1)
    minimos[i]=min(min(datos_1_column[i]), min(datos_2_column[i]))
    maximos[i]=max(max(datos_1_column[i]), max(datos_2_column[i]))
    ax.plot(datos_1_column[i], color='tab:blue')
    ax.plot(datos_2_column[i], color='tab:orange')
    plt.title('Característica '+str(i) )
    plt.show()

print('Valores mínimos:', minimos)
print('Valores maximos:', maximos)
```

```
# Características de los objetos de interés por filas
testData = np.asarray(test_dataset[0]+test_dataset[1]).astype(np.float32)

# Etiquetas
num_test = len(test_dataset[0])
labels_0 = list(np.zeros(num_test))
labels_1 = list(np.ones(num_test))
testLabels = np.asarray(labels_0+labels_1).astype(np.int32)


def get_accuracy(predictions, labels):
    """Devuelve la precisión según las coincidencias entre las predicciones y las etiquetas"""
    accuracy = (np.squeeze(predictions) == labels).mean()
    return accuracy * 100

trainData_col = np.transpose(trainData.copy(), (1, 0))
for i, (min, max) in enumerate(zip(minimos,maximos)):
    trainData_col[i]=np.true_divide((trainData_col[i]-min), max-min).astype(np.float32)


# testData normalizado en formato columna
testData_col = np.transpose(testData.copy(), (1, 0))
for i, (min, max) in enumerate(zip(minimos,maximos)):
    testData_col[i]=np.true_divide((testData_col[i]-min), max-min).astype(np.float32)


# Datos normalizados en formato fila
trainData_norm = np.transpose(trainData_col.copy(), (1, 0))
testData_norm = np.transpose(testData_col.copy(), (1, 0))


model_knn = cv2.ml.KNearest_create()
model_knn.train(trainData_norm, cv2.ml.ROW_SAMPLE, trainLabels)

#Normalización de los datos
trainData_col = np.transpose(trainData.copy(), (1, 0))
for i, (min, max) in enumerate(zip(minimos,maximos)):
    trainData_col[i]=np.true_divide((trainData_col[i]-min), max-min).astype(np.float32)


# testData normalizado en formato columna
testData_col = np.transpose(testData.copy(), (1, 0))
for i, (min, max) in enumerate(zip(minimos,maximos)):
    testData_col[i]=np.true_divide((testData_col[i]-min), max-min).astype(np.float32)


# Datos normalizados en formato fila
trainData_norm = np.transpose(trainData_col.copy(), (1, 0))
testData_norm = np.transpose(testData_col.copy(), (1, 0))

#Se crea un clasificador KNN
model_knn = cv2.ml.KNearest_create()
model_knn.train(trainData, cv2.ml.ROW_SAMPLE, trainLabels)
```

```
# Prueba valores de k=1, k=3, k=5, k=7, k=9
num_test = len(testData)
results=[]
for k in np.arange(1, 10, 2):
    ret, predictedLabels, neighbours, dist = model_knn.findNearest(testData, k)
    predictedLabels = np.reshape(predictedLabels, (num_test)).astype(np.int32)
    acc = get_accuracy(predictedLabels, testLabels)
    print("k=", k, "-> Porcentaje de aciertos: {} %".format("%.2f" % acc))

    print('        testLabels:     ', testLabels)
    print('        predictedLabels:', predictedLabels)

    results.append(acc)

# Se crea y se entrena un modelo con los datos de train
model_bayes = cv2.ml.NormalBayesClassifier_create()
model_bayes.train(trainData, cv2.ml.ROW_SAMPLE, trainLabels)

# Se predicen las etiquetas de los datos de test
predictedLabels = model_bayes.predict(testData)[1].ravel()

# Se calcula la precisión (porcentaje de aciertos)
acc = get_accuracy(predictedLabels, testLabels)
print("Porcentaje de aciertos: {} %".format("%.2f" % acc))

print('testLabels:     ', testLabels)
print('predictedLabels:', predictedLabels)

# Asignar los valores de las etiquetas a -1, 1
svm_trainLabels = trainLabels.copy()
indices = np.where(svm_trainLabels == 0)[0]
svm_trainLabels[indices] = -1

svm_testLabels = testLabels.copy()
indices = np.where(svm_testLabels == 0)[0]
svm_testLabels[indices] = -1

# Inicializar el modelo SVM:
model_svm = cv2.ml.SVM_create()
model_svm.setKernel(cv2.ml.SVM_LINEAR)
#model_svm.setKernel(cv2.ml.SVM_RBF)
model_svm.setType(cv2.ml.SVM_C_SVC)
model_svm.setTermCriteria((cv2.TERM_CRITERIA_MAX_ITER, 1000, 1e-6))

c_values = [0.001, 0.01, 0.1, 1.0, 10.0, 60.0]
#gamma_values = [0.001, 0.01, 0.1, 1.0, 10.0, 100.0]
for c in c_values:
    #for g in gamma_values:

        #model_svm.setGamma(g)
        model_svm.setC(c)
        model_svm.train(trainData_norm, cv2.ml.ROW_SAMPLE, svm_trainLabels)

        predictedLabels = model_svm.predict(testData_norm)[1].ravel()

        acc = get_accuracy(predictedLabels, svm_testLabels)
        print ("C=","{:7.3f}".format(c),"-> Porcentaje de aciertos:", "{:7.3f} %".format(acc))
```

1.2 Código de clasificación para la selección según calidad

```
from __future__ import print_function
from __future__ import division
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
import torchvision
from torchvision import datasets, models, transforms
import matplotlib.pyplot as plt
import time
import os
import copy
from torchvision.datasets import ImageFolder
from torch.utils.data import DataLoader
from torch.optim import lr_scheduler
print("PyTorch Version: ",torch.__version__)
print("Torchvision Version: ",torchvision.__version__)
```

```
#Solo para feature extracting
def set_parameter_requires_grad(model, feature_extracting):
    if feature_extracting:
        for param in model.parameters():
            param.requires_grad = False
```

```
def initialize_model(num_classes, feature_extract, use_pretrained=True):
    model_ft = None
    input_size = 0
    model_ft = models.resnet18(pretrained=use_pretrained)
    set_parameter_requires_grad(model_ft, feature_extract)
    num_ftrs = model_ft.fc.in_features
    model_ft.fc = nn.Linear(num_ftrs, num_classes)
    input_size = 224
    return model_ft, input_size
```

```
num_classes=5
feature_extract=False
```

```
model_ft, input_size = initialize_model(num_classes, feature_extract, use_pretrained=True)
```

```
transform = transforms.Compose([transforms.Resize((224, 224)),
                                transforms.ToTensor(),
                                transforms.Normalize((0.5, 0.5, 0.5),
                                                     (0.5, 0.5, 0.5))])
dataset = ImageFolder('perasDevoeRGB', transform=transform)
classes = dataset.classes
```

```
from torch.utils.data import random_split
train_set, valid_set = random_split(dataset, (int(len(dataset) * 0.7) + 1, int(len(dataset) * 0.3)))
```

```
train_loader = DataLoader(train_set, batch_size=64)
valid_loader = DataLoader(valid_set, batch_size=1)
```

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

```
model_ft = model_ft.to(device)

params_to_update = model_ft.parameters()
print("Params to learn:")
if feature_extract:
    params_to_update = []
    for name,param in model_ft.named_parameters():
        if param.requires_grad == True:
            params_to_update.append(param)
            print("\t",name)
else:
    for name,param in model_ft.named_parameters():
        if param.requires_grad == True:
            print("\t",name)

optimizer_ft = optim.Adam(params_to_update, lr=0.0005, weight_decay=0.0015)
```

```
def train_model(model, dataloaders, criterion, optimizer, num_epochs):
    since = time.time()

    val_acc_history = []
    train_acc_history = []
    loss_train = np.zeros((num_epochs, 1))
    loss_val = np.zeros((num_epochs, 1))
    total_step = len(train_loader)
    max_iter = total_step*num_epochs
    cost = np.zeros((max_iter, 1))
    cont = 0
    cont2 = 0

    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0
```

```
for epoch in range(num_epochs):
    print('Epoch {}/{}'.format(epoch, num_epochs - 1))
    print('-' * 10)

    # Cada época tiene una fase de entrenamiento y una de validación
    for phase in ['train', 'val']:
        if phase == 'train':
            model.train()  # Se fija el modelo en modo entrenamiento
        else:
            model.eval()   # Se fija el modelo en modo validación

        running_loss = 0.0
        running_corrects = 0
```

```
        for inputs, labels in dataloaders[phase]:
            inputs = inputs.to(device)
            labels = labels.to(device)

            optimizer.zero_grad()

            with torch.set_grad_enabled(phase == 'train'):

                outputs = model(inputs)
                loss = criterion(outputs, labels)

                _, preds = torch.max(outputs, 1)


            if phase == 'train':
                loss.backward()
                cost[cont]=loss.item()
                cont+=1
                optimizer.step()


            running_loss += loss.item() * inputs.size(0)
            running_corrects += torch.sum(preds == labels.data)
```

```
            epoch_loss = running_loss / len(dataloaders[phase].dataset)
            epoch_acc = running_corrects.double() / len(dataloaders[phase].dataset)

            print('{} Loss: {:.4f} Acc: {:.4f}'.format(phase, epoch_loss, epoch_acc))


            if phase == 'val' and epoch_acc > best_acc:
                best_acc = epoch_acc
                best_model_wts = copy.deepcopy(model.state_dict())
            if phase == 'val':
                val_acc_history.append(epoch_acc)
                loss_val[cont2]=epoch_loss
                cont2 += 1
            if phase == 'train':
                train_acc_history.append(epoch_acc)
                loss_train[cont2]=epoch_loss
```

```
    time_elapsed = time.time() - since
    print('Training complete in {:.0f}m {:.0f}s'.format(time_elapsed // 60, time_elapsed % 60))
    print('Best val Acc: {:4f}'.format(best_acc))

    # load best model weights
    model.load_state_dict(best_model_wts)
    return model, train_acc_history, val_acc_history, cost, loss_train, loss_val
```

```
dataloaders_dict = {"train": train_loader,"val": valid_loader}
criterion = nn.CrossEntropyLoss()

num_epochs=20
# Se entrena y evalua el modelo
model_ft, hist_train, hist_val, cost, loss_train, loss_val = train_model(model_ft, dataloaders_dict, criterion, optimizer_ft, num_epochs=num_epochs)
```

```
nb_classes = 5

confusion_matrix = torch.zeros(nb_classes, nb_classes)
with torch.no_grad():
    for i, (inputs, classes) in enumerate(dataloaders_dict['val']):
        inputs = inputs.to(device)
        classes = classes.to(device)
        outputs = model_ft(inputs)
        _, preds = torch.max(outputs, 1)
        for t, p in zip(classes.view(-1), preds.view(-1)):
                confusion_matrix[t.long(), p.long()] += 1

print(confusion_matrix)
```

1.3 Ampliación del Dataset

```
import cv2
import random

def horizontal_flip(img):
        return cv2.flip(img, 1)

def vertical_flip(img):
        return cv2.flip(img, 0)

def rotation(img):
    angle = int(random.uniform(-30, 30))
    h, w = img.shape[:2]
    M = cv2.getRotationMatrix2D((int(w/2), int(h/2)), angle, 1)
    img = cv2.warpAffine(img, M, (w, h))
    return img
```

```
import random
import os

# La ruta del fichero que contenga las imágenes
folder_path = 'datasets\\perasDevoeColor\\1'
# El número de ficheros a generar
num_files_desired = 2500


images = [os.path.join(folder_path, f) for f in os.listdir(folder_path) if os.path.isfile(os.path.join(folder_path, f))]

num_generated_files = 0
while num_generated_files <= num_files_desired:
    # Se escoge una imagen aleatoria
    image_path = random.choice(images)
    image_to_transform = img = cv2.imread(image_path)


    available_transformations = {
    'rotate': rotation,
    'horizontal_flip': horizontal_flip,
    'vertical_flip': vertical_flip
    }

    # Se escoge un número aleatorio de transformaciones que realizar
    num_transformations_to_apply = random.randint(1, len(available_transformations))
    new_folder_path = folder_path = 'datasets\\perasDevoeMezclaBMP\\1'
    num_transformations = 0
    transformed_image = None
    while num_transformations <= num_transformations_to_apply:
        key = random.choice(list(available_transformations))
        transformed_image = available_transformations[key](image_to_transform)
        new_file_path = '%s/augmented_image_%s.bmp' % (new_folder_path, num_generated_files)
        cv2.imwrite(new_file_path, transformed_image)
        num_transformations += 1
    num_generated_files += 1
```