En colaboración con

Modelo de aprendizaje profundo/red neuronal convolucional (CNN) para clasificación de calidad de ácidos grasos por imágenes de semillas de Helianthus annuus

Juan Manuel Vega Arias

Máster Universitario en Bioinformática y Bioestadística

Área: Estadística y Bioinformática

Directores: **Esteban Vegas Lozano** y **Ferran Reverter Comes**

Codirector: **Antonio J. Moreno Pérez**

B) GNU Free Documentation License (GNU FDL)

C) Copyright

FICHA DEL TRABAJO FINAL

Título:	Modelo de aprendizaje profundo/red neuronal convolucional (CNN) para clasificación de calidad de ácidos grasos por imágenes de semillas de Helianthus annuus
Autor:	Juan Manuel Vega Arias
Directores:	Esteban Vegas Lozano Ferran Reverter Comes
Co-director:	Antonio Moreno Pérez (CSIC)
Fecha:	Jun/2019
Título de Máster:	Máster Universitario en Bioinformática y Bioestadística
Area:	Estadística y Bioinformática
Idioma:	Castellano
Keywords:	CNN, Image classification, Deep learning

Abstract (in English, 250 words or less):

The fatty acids content classification in seeds for their later use in industry is a long and complicated process which aims to select the different seeds that would be used with the purposes each of these seeds are of best use. These purposes are, at the same time, determined by the quality content of the fatty acids in the seed.

Deep neural networks, especially convolutional neural networks, have shown a remarkable capacity for image classification and pattern abstraction in many different fields, obtaining better accuracy, and faster prediction results than those obtained by classic or human methods.

In this work we build a convolutional neural network model which can classify the sunflower seeds fatty acids quality through their images.

The model was developed separating the work in two main sections. First, an experimental portion in which we collected the necessary data to build our own data set from scratch to train the neural network, and second, the analytic component in which we developed the model using the data we previously collected.

This model shows a high accuracy classifying different types of sunflower (*Helianthus annuus* L.) seeds used for different purposes depending on their fatty acids quality content.

Contents

Lista de Tablas y figuras

1. Introducción

1.1 Contexto y justificación

El aceite de palma es uno de los aceites vegetales con mayor expansión de uso alrededor del mundo [1]. Podemos encontrar su presencia en un extenso rango de artículos en el mercado, entre estos tiene un especial puesto en los productos alimentarios debido a la composición de ácidos grasos que lo forman, que dotan a este aceite de altos puntos de fusión y de humo. Además, por su gran cantidad de producción por hectárea, es un aceite muy rentable económicamente [2].

Los principales efectos derivados de la agricultura de la palma son: la amenaza a la biodiversidad en las selvas tropicales por su monocultivo, además de efectos en la salud humana por la contaminación generada por el proceso de corte y quemado con el que se realiza la deforestación [3,4].

Es por esto que toma importancia el trabajo de investigación y desarrollo de especies vegetales que sirvan de alternativa viable, e incluso más saludables, al aceite de palma.

El cultivo de especies vegetales productoras de aceite ha ido en aumento en las últimas décadas, entre ellas la del girasol. Existe un especial interés en modificar la producción de ácidos grasos de estas mediante diferentes técnicas biotecnológicas como la mutagénesis inducida [5,6]. Este interés ha dado como resultado el desarrollo de líneas de girasol que muestran un contenido en ácidos grasos que dotan al aceite de las características adecuadas para su uso en diferentes sectores industriales y su subsecuente interés económico [7].

La caracterización del aceite contenido en semillas es un proceso optimizado, sin embargo, aún requiere de un tiempo de preparado de la semilla por metilación para la extracción de ácidos grasos metil ésteres (FAMES), para su posterior análisis [8]. A lo largo de la última década ha quedado demostrada la capacidad de las redes neuronales artificiales (ANN) para encontrar patrones en muy diversos ámbitos [9,10]. En los últimos años ha ido ganando terreno el uso de redes neuronales convolucionales (CNN) para clasificación de imágenes con unos resultados excelentes, equiparables a los obtenidos por especialistas [11]. En este trabajo realizamos un modelo de CNN clasificador de imágenes de semillas de girasol agrupadas por su contenido en ácidos grasos, así como la base de datos necesaria para su entrenamiento. De este modo podemos hacer una clasificación del contenido de ácidos grasos en la semilla por la imagen.

1.2 Objetivos

1.2.1 Objetivos generales

1) Clasificación de imágenes de semillas de girasol con distinta calidad de ácidos grasos.

1.2.2 Objetivos específicos:

1) Desarrollo de un modelo de CNN.

- 1.1. Creación desde cero de una base de datos para el entrenamiento del modelo.

- 1.2. Automatizar la extracción de características de la imagen.

- 1.3. Implementar y comprobar el algoritmo de clasificación.

1.3 Enfoque y método seguido

Seleccionaremos semillas de líneas puras parentales para el fotografiado de las mismas. Tras la toma de fotografías se sembrarán y cuando la semilla germine, se analizará el contenido graso de las germínulas. Se intentará crear una base de datos compuesta por las imágenes tomadas y las etiquetas creadas a través del análisis de FAMES de las germínulas.

Se hará una evaluación de las diferentes técnicas de clasificación de imágenes para encontrar la que mejor se ajuste a nuestro problema, así como un proceso de aprendizaje de construcción del modelo mediante la utilización de los frameworks TensorFlow y Keras implementados en Python 3.

Se probará el modelo en casos reales de semillas para clasificación de contenido de ácidos grasos de la misma. Se evaluará el modelo creando una tabla de confusión y a través de los diferentes estadísticos.

1.4 Planificación

1.4.1 Tareas

Objetivo 1.1

▪ Preparación en tandas de las semillas y toma de imágenes, posterior siembra.

▪ Análisis en tandas de contenido en ácidos grasos de germínulas.

▪ Creación de base de datos.

Objetivo 1.2

▪ Extracción automática de características:

o Preprocesado de las imágenes (normalizado, simplificado, segmentación)

o Técnicas de aumento de datos.

o Entrenar CNN (convolutional neural network) y extraer las últimas capas de descriptores

o Entrenar modelos supervisados (jerárquicos/no jerárquicos) sobre estos descriptores

▪ Entrenar, evaluar el modelo clasificador:

o Entrenamiento de la CNN para la generación del modelo y su posterior evaluación con matriz de confusión.

1.4.2 Calendario

▪ La obtención de imágenes comenzará en el mes de Marzo.

▪ La duración de la elaboración de la base de datos incluye:

o Preparación y fotografiado de las semillas

o Análisis de ácidos grasos de las germínulas

▪ Construcción del modelo a principios de Abril:

o Instalación de software necesario

o Prueba y realización de la tarea

o Entrega al tutor para revisión y corrección

▪ La redacción de la memoria se realizará en el periodo dispuesto para ello en el esquema general expuesto en la página de la asignatura, permitiendo la corrección por parte del tutor.

▪ Tan pronto como se acabe con la memoria del trabajo procederemos a la elaboración de la presentación, igualmente siguiendo las fechas sugeridas en la página de la asignatura.

1.4.3 ***Hitos***

• Hasta el **24/04/2019:**

▪ Preparación y confección de la base de datos:

o Preparación y fotografiado de las semillas

o Análisis de ácidos grasos de las germínulas

▪ Preparación y elección del entorno de trabajo:

o Python: TensorFlow, Keras, CUDA (para uso de GPU en entrenamiento de CNN)

▪ Extracción de descriptores a partir de CNN

• Hasta el 20/05/2019:

▪ Aprendizaje de CNN, evaluación del rendimiento del modelo.

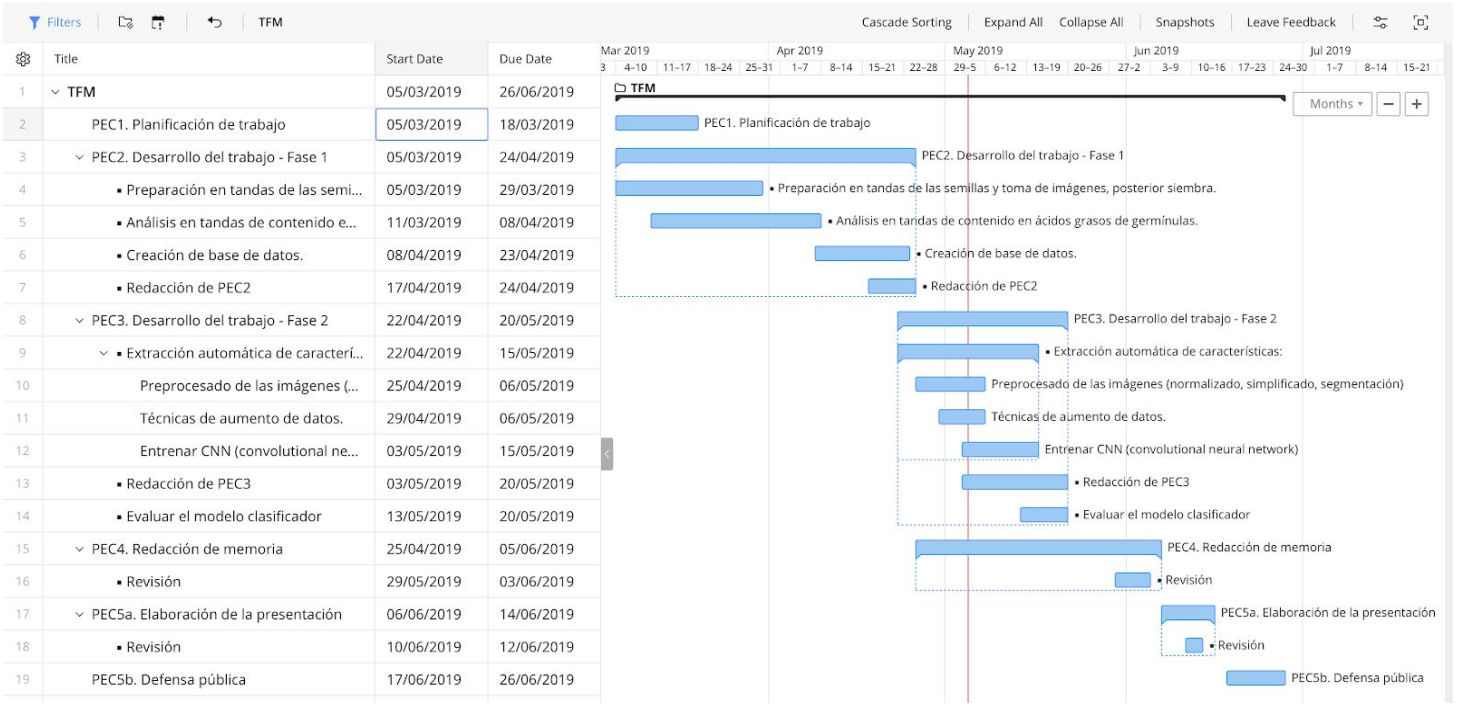
• Hasta el 05/06/2019:

▪ Cierre de memoria

• Hasta el 13/06/2019:

▪ Elaboración de la presentación.

Diagrama de Gantt



	Title	Start Date	Due Date
1	TFM	05/03/2019	26/06/2019
2	PEC1. Planificación de trabajo	05/03/2019	18/03/2019
3	PEC2. Desarrollo del trabajo - Fase 1	05/03/2019	24/04/2019
4	▪ Preparación en tandas de las semi...	05/03/2019	29/03/2019
5	▪ Análisis en tandas de contenido e...	11/03/2019	08/04/2019
6	▪ Creación de base de datos.	08/04/2019	23/04/2019
7	▪ Redacción de PEC2	17/04/2019	24/04/2019
8	PEC3. Desarrollo del trabajo - Fase 2	22/04/2019	20/05/2019
9	▪ Extracción automática de caracterí...	22/04/2019	15/05/2019
10	Preprocesado de las imágenes (...	25/04/2019	06/05/2019
11	Técnicas de aumento de datos.	29/04/2019	06/05/2019
12	Entrenar CNN (convolutional ne...	03/05/2019	15/05/2019
13	▪ Redacción de PEC3	03/05/2019	20/05/2019
14	▪ Evaluar el modelo clasificador	13/05/2019	20/05/2019
15	PEC4. Redacción de memoria	25/04/2019	05/06/2019
16	▪ Revisión	29/05/2019	03/06/2019
17	PEC5a. Elaboración de la presentación	06/06/2019	14/06/2019
18	▪ Revisión	10/06/2019	12/06/2019
19	PEC5b. Defensa pública	17/06/2019	26/06/2019

Tabla 1. Diagrama de Gantt de la planificación temporal.

1.5 Breve sumario de productos obtenidos

Base de datos de imágenes para el entrenamiento de la CNN.

Modelo de CNN de clasificación de calidad de contenido de ácidos grasos a través de imágenes de semillas de girasol. Posibilidad de publicación.

1.6 Breve descripción de los otros capítulos de la memoria

2. Redes neuronales convolucionales (CNN).
3. Creación de la Base de Datos
4. Material Vegetal
5. Análisis de FAMES
6. Material Software Utilizado
7. Hardware Utilizado
8. Modelo de clasificación de imágenes por CNN
9. Medidas de Rendimiento
10. Conclusiones
11. Glosario
12. Bibliografía
13. Anexos

2. Redes Neuronales Convolucionales (CNN)

2.1 Introducción

En la estructura jerarquizada que compone el cuerpo de la Inteligencia Artificial, tenemos varias subdivisiones entre las cuales está el Aprendizaje Automático. Este a su vez se divide en Aprendizaje Supervisado, No Supervisado y Aprendizaje Reforzado. El Aprendizaje Supervisado es aquel en el que cada uno de los ejemplos que forman la base de datos con los que se entrena el algoritmo de aprendizaje automático está correctamente etiquetado con la categoría o etiqueta que queremos obtener en la clasificación, el aprendizaje automático se subdivide en dos categorías, el Aprendizaje Superficial (Shallow Learning), como máquinas de vectores de soporte (SVM), árboles de decisión, modelos lineales e incluso redes neuronales artificiales (ANN) con una sola capa oculta, y Aprendizaje Profundo (Deep Learning), aquí tratamos con ANN de arquitectura más compleja, disponen de más de una capa oculta, como veremos más adelante. Las ANN fueron diseñadas como modelos computacionales de neuronas humanas por el neurofisiólogo **Warren McCulloch** y el neurocientífico computacional **Walter Pitts** en el año 1943, y al igual que el cerebro se compone de las neuronas, las redes neuronales se componen de unos nodos, que funcionan conectados unos a otros formando una red [12], esta red de nodos trabaja conjuntamente para estimar un resultado que se ajuste a un modelo no lineal con el mínimo error posible.

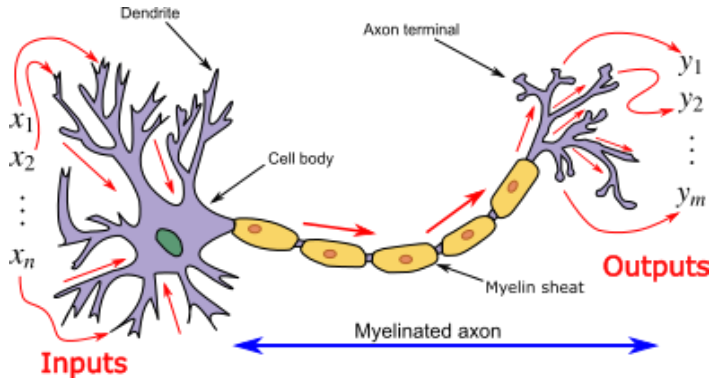


Figura 1. Esquema de neurona, Wikipedia.

El Aprendizaje Profundo a la vez se subdivide en otras categorías, cada una de estas categorías ha sido desarrollada y perfeccionada para resolver de forma más específica diferentes problemas, siendo así

las CNN el actual estado del arte para la clasificación de imágenes u objetos contenidos en esta.

Las CNN fueron creadas, al igual que las redes neuronales artificiales, basándose en la estructura de las neuronas de la corteza visual de mamíferos; gracias al trabajo de los científicos **David Hunter Hubel** y **Torsten Nils Wiesel** en 1959 [13]. Este trabajo inspiró a **Kuniko Fukushima** para su propuesta del "Neocognitron" en 1980, donde ya se reconoce la arquitectura característica de las CNN [14]. Más tarde el modelo de Fukushima es, a su vez, mejorado por investigadores actuales hasta las CNN de hoy.

2.2 Arquitectura

Este tipo de ANN están compuestas por una capa de input y una de output además de varias capas ocultas, algunas de las cuales son capas convolucionales con su función de activación, cuya labor describiremos adelante, capas de pooling y capas totalmente conectadas (FC) con su función de activación.

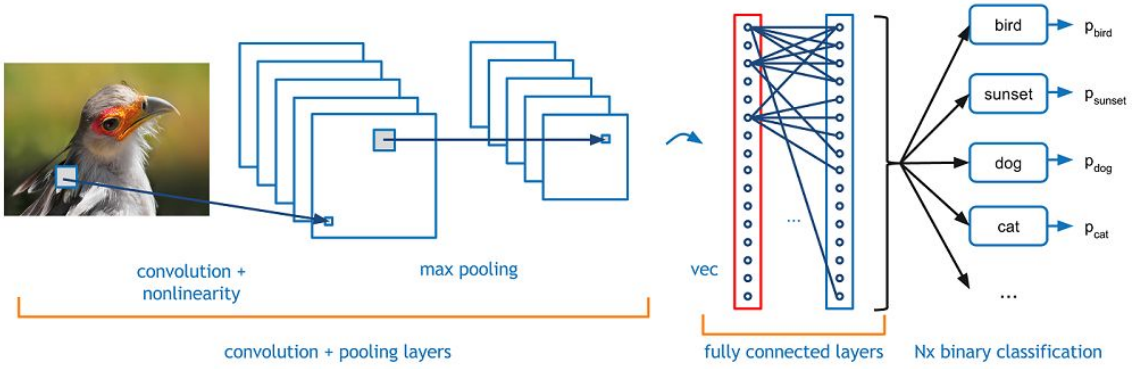


Figura 2. Esquema general de CNN

2.2.1 Capas convolucionales

Su aplicación es realizada en imágenes por las cuales pasamos matrices de 2D llamadas filtros o kernels que recorren la imagen a clasificar. En la operación de convolución, dichos filtros pasan por la matriz de píxeles de la que está compuesta la imagen solapándose entre un paso y el siguiente, a la amplitud del paso con el que el filtro se mueve se le llama stride.

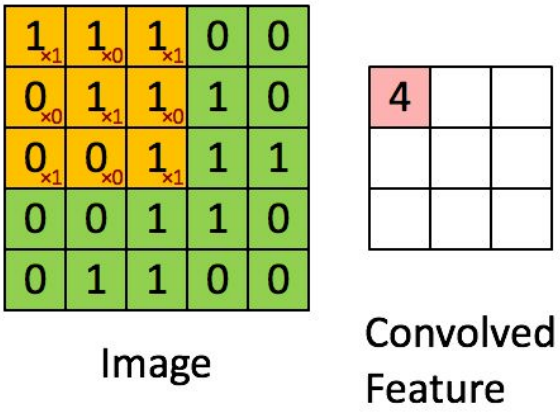


Figura 3. Visualización de convolución. Tenemos una imagen de 5x5, un filtro de 3x3 píxeles y un stride o paso de 1, ya que se mueve 1 píxel cada vez que el filtro se desplaza, siendo el output tras la convolución 3x3.

2.2.2 *La operación de convolución*

También llamada de extracción de características, la matriz de la izquierda representa una imagen en blanco y negro. Cada entrada corresponde a un píxel, 0 para negro y 1 para blanco (normalmente es entre 0 y 255 para imágenes en escala de grises). La ventana que se desliza es llamada kerner, filtro o detector de características, la región de la matriz de la imagen de input del mismo tamaño del filtro es llamada campo receptivo. En cada paso se multiplican los valores del filtro por los del campo receptivo elemento a elemento y se suman, entonces el filtro se desliza un paso sobre el siguiente campo receptivo de la matriz y se repite la misma operación hasta que se haya pasado por la imagen al completo. El resultado de esta operación es un número entero del volumen del output. El output, será el input de la capa siguiente. Las capas convolucionales tienen como función de activación la función ReLU, que transforma los valores negativos a cero.

2.2.3 *La funcion de activación Relu*

Es una función que transforma los valores de entrada de un nodo en valores de salida. En una ANN sería la representación de lo que biológicamente ocurre en el soma de la neurona, donde después de recoger las diferentes señales de las sinapsis, si se supera cierto umbral se produce un potencial de acción.

En el caso de las CNN, todas las capas convolucionales tienen la función rectificadora, a cada nodo con esta función de activación se le llama ReLU por sus siglas en inglés, definida como:

$f(x) = \max(0, x)$

Donde x es el input de la neurona.

2.2.4 *Capa de agrupación o Max-pooling*

Es usada para reducir el volumen espacial de la imagen de input y normalmente va entre dos convoluciones ya que si aplicamos una capa totalmente conectada sería demasiado costoso.

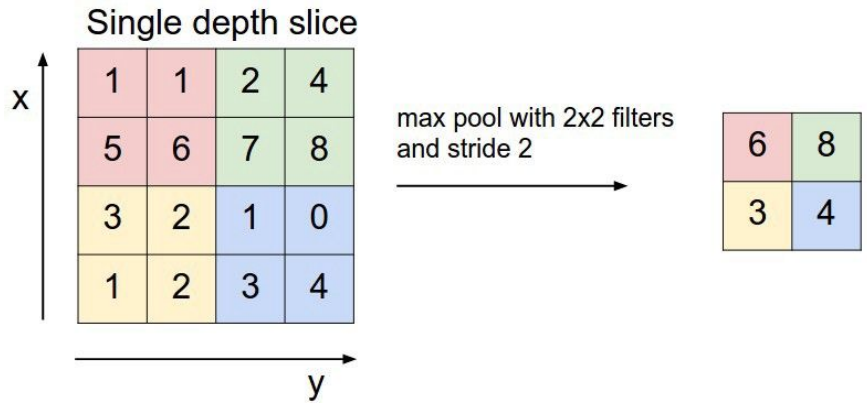


Figura 4. Resultado de operación de agrupación. En la imagen aplicamos un filtro de 2x2 y un paso de 2 píxeles para reducir la imagen de 4x4 a 2x2.

Realizamos Max-pooling para reducir el input sin perder información espacial.

2.2.5 *Capa totalmente conectada o Fully connected layer (FC).*

Conecta todas las neuronas de una capa con los elementos de la capa siguiente y se usa para clasificar las imágenes de diferentes categorías por entrenamiento.

2.2.6 *La funcion de activación Softmax*

Softmax es una función de activación para la capa FC que se usa para clasificación múltiple, como es nuestro caso. Para casos de clasificación binaria se usa la función logística.

2.2.7 *Capa de output*

Esta capa tendrá tantos nodos como clases queramos clasificar, perteneciendo cada uno a una clase, el resultado de la probabilidad de pertenencia a dicha clase será el que aparezca en este nodo.

2.2.8 *Función de coste o loss*

En clasificación de imágenes, como en cualquier ANN, los pesos o coeficientes asignados a cada uno de los nodos que forman nuestras capas de la red neuronal determinan nuestras predicciones son totalmente al azar al comenzar el entrenamiento. Para medir nuestro error necesitamos una función de coste. Esta normalmente se calcula como la diferencia entre el output real, la imagen etiquetada con el que entrenamos al algoritmo, y el output predicho por nuestro algoritmo. La meta es que nuestro modelo tenga un valor final de la función de coste lo más bajo posible, por lo tanto el error será mínimo. Dependiendo de la naturaleza de la predicción a realizar se utilizan diferentes funciones de coste.

2.2.9 *Propagación hacia atrás y función de optimización*

Una vez que se calcula el error necesitamos minimizarlo en una ANN esto se realiza por propagación hacia atrás, donde el error se propaga, como su propio nombre indica, hacia atrás a una capa previa donde es usado para modificar los pesos y la tendencia, de tal forma que el error sea minimizado. Esta modificación de los pesos es realizada por una función llamada función de optimización.

En nuestro modelo usamos la llamada root mean square (RMSprop), cuya función se expresa de la siguiente manera:

$$E[g^2]_t = \beta E[g^2]_{t-1} + (1-\beta)\left(\frac{\delta C}{\delta w}\right)^2$$

$$w_t = w_{t-1} - \frac{\eta}{\sqrt{E[g^2]_t}} \frac{\delta C}{\delta w}$$

La idea detrás de este optimizador es solucionar el problema de minibatches, en los que se divide entre un gradiente diferente cada vez. Con RMSprop obligamos a tener gradientes similares a minibatches adyacentes. Mantenemos las medias móviles de los gradientes al cuadrado para cada peso. Entonces dividimos el gradiente por la raíz del cuadrado de la media de ahí el nombre root mean square.

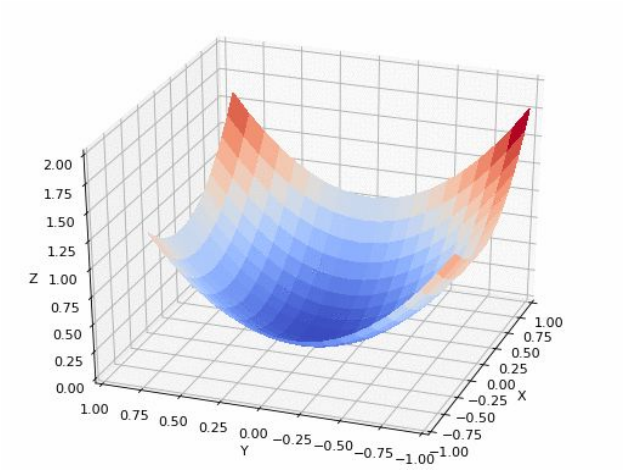


Figura 5. Descenso de gradiente

Las funciones de optimización calculan el gradiente, por ejemplo, la derivada parcial de la función de coste con respecto a los pesos, y los pesos se modifican en la dirección opuesta del gradiente calculado un paso cada vez. La magnitud de este paso es determinada por el ratio de aprendizaje o learning rate. Este ciclo es repetido hasta llegar a un mínimo de la función de coste.

Gracias a este proceso las redes neuronales pueden ajustar sus predicciones a patrones no lineales que son a la vez más complejos y más parecidos a los que se dan en condiciones naturales. Dicha capacidad para extraer patrones complejos está directamente relacionada con el número de capas que formen el modelo, puesto que las primeras capas extraerán patrones sencillos, estos irán subiendo en complejidad conforme vayamos a capas más profundas.

3. Base de Datos

3.1 Introducción

Las CNN pueden ser usadas para una variedad de fines. En este trabajo necesitamos clasificar imágenes a través de aprendizaje supervisado.

En este tipo de algoritmo de aprendizaje automático se necesita una base de datos de imágenes ya clasificadas y etiquetadas con las que realizaremos el proceso de entrenamiento.

3.2 Características de Imágenes

La base de datos usada para este trabajo se compone de 250 imágenes de semillas de girasol (3024x3024 RGB). Se divide en un set de entrenamiento y un set de validación, cada uno de estos sets tiene 5 clases diferentes (CAS-6, CAS-9, CATH, Daniel y White) con 33 imágenes por cada una de las clases en el set de entrenamiento, y 17 imágenes por cada una de las clases para el set de validación. Estas fueron clasificadas individualmente a través de un análisis de FAMES y posteriormente etiquetadas.

Figura 6-8. Muestra de imágenes de la base de datos. Imágenes de las clases CATH, CAS-9 y CAS-6 respectivamente.

3.3 Cámara

Para la toma de imágenes se utilizó una cámara de un iPhone 8 a la cual se le añadió un objetivo macro de x10 para móviles modelo Aukey.

Especificaciones técnicas:

12 MP, apertura $\mathit{f}/1.8$

Autofoco por detección de fase

Flash LED quad tono dual

HDR

Estabilización óptica de imagen

Geo-tagging

Foco táctil

Reconocimiento de rostro

Panorama

vídeo 2160p@30fps, 1080p@30fps, 1080p@240fps slo-mo, luz de vídeo.

Figura 9. Detalle de montaje de cámara y objetivo macro.

3.4 Cabina de Fotografía Portátil

Para una correcta iluminación se utilizó una mini cabina de fotografía portátil con iluminación de 40 bombillas LED delantera y trasera.

Especificaciones técnicas:

Tamaño de caja de luz de foto abierta: 9.4 x 9.1 x 8.7 in, Tamaño de tapa cerrada: 9.1 x 9.1 x 0.6 in

Bombillas LED: 40 （en 2 tiras LED)

Lumen: 2 * 550LM

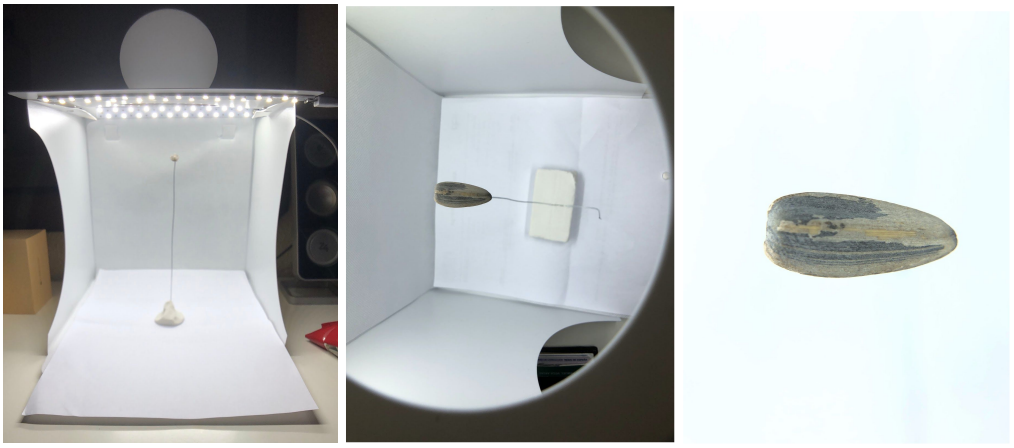
Voltaje de entrada:: 5V 1A

Material para el telón de fondo de Mini Folding The Light Box: tela de nylon

Mini Studio Photo Box Led Carpa Cuerpo Materiales: PP.

Figura 10. Cabina de fotografía portátil Nepter Lighting.

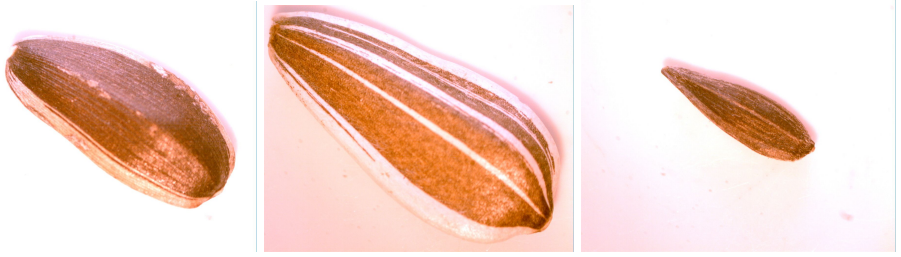
Para eliminar las sombras de fondo se construyó un pequeño soporte con cable de acero de 18 cm y base de plastilina blanca, las semillas fueron situadas en el extremo superior del soporte, solucionando el problema de la sombra a la vez que se consigue una imagen totalmente plana de la parte más ancha del aquenio.



Figuras 11-13. De izquierda a derecha: montaje del soporte, detalle de semilla sobre el soporte y resultado final.

3.5 Pruebas

Para la toma de imágenes se consideraron varias opciones antes de elegir la definitiva. En primer lugar se utilizó una lupa de laboratorio trinocular conectada a un ordenador. El resultado que se obtuvo con esta cámara fue inadecuado por la falta de correcta iluminación, además las semillas descansaban encima de un soporte blanco donde se proyectaba la sombra de estas. El aumento de la lupa permitía una buena definición, sin embargo, al variar los tamaños de la semilla había que reajustar el enfoque perdiendo el sentido de escala, y para algunas semillas de mayor tamaño el encuadre era insuficiente a la máxima distancia.



Figuras 14-17. Muestras de imágenes tomadas con lupa.

Como segunda opción se tomó una cámara digital Sony NEX 5N 16.1 MP AVCHD, con lente macro en el mismo soporte descrito anteriormente. Con esta opción persistían los problemas de iluminación y sombras además de la dificultad de manejo de la cámara.

Figuras 18-20. Muestras de imágenes tomadas con Sony NEX 5N 16.1 MP AVCHD.

Por estas razones se acabó utilizando el equipo final con el que se consiguió eliminar la sombra, así como la obtención de una iluminación y una calidad de imagen adecuadas para nuestra base de datos.

4. Material Vegetal

Las semillas usadas en este trabajo corresponden a plantas de la línea de control **CAS-6**, la mutante alto oleico (18:1) **CAS-9**, la cual se produjo a partir de líneas comerciales por mutagénesis química, y las líneas de semillas comestibles **White**, **CATH** y **Daniel**, 50 semillas de cada una de las líneas. Todos los caracteres han sido fijados por retrocruzamiento hasta que las plantas produjeron progenie con fenotipos uniformes. [15]

Las semillas de girasol fueron cosechadas de plantas cultivadas en cámaras de crecimiento en un ciclo a 25ºC/15ºC (día/noche), con un fotoperiodo de 16h, una densidad de flujo de fotones de 200 µmol · m^{-2} · s^{-1}, y líneas de fertirrigación.

5. Análisis de FAMES

Se realiza un procedimiento para la digestión de tejido fresco, transmetilación de lípidos, y la extracción de FAMES en un solo paso [8].

Se usaron semillas de girasol, aquenios pelados. Muestras de aproximadamente 50 mg junto con ácido heptadecanoico (7:0) como control interno fueron puestos en tubos con tapones revestidos de vitón o teflón. La eficiencia de las diferentes mezclas está expresada como % de transmetilación de glicerolípidos, la cual fue calculada con respecto al control interno. A excepción de cuando se indique, fueron usadas mezclas de metilación conteniendo metanol:benceno:DMP:H_2SO_4(37:20:5:2, por volumen); metanol:tolueno:DMP:H_2SO_4(39:20:5:2, por volumen) o metanol:tetrahidrofurano:DMP:H_2SO_4(31:20:5:2, por volumen). Todos los reactivos son de pureza analítica y las mezclas fueron almacenadas en recipientes oscuros en un refrigerador. Se añadieron a las muestras una cantidad de mezclas de 3.2, 3.3, o 2.9 ml, respectivamente y el resto de heptano 1.8, 1.7, o 2.1 ml, respectivamente hasta un volumen total de 5 ml. Los tubos fueron depositados en un baño de agua a 80ºC durante 1h para asegurar la extracción completa de lípidos y metilación. Después del calentamiento, los tubos son enfriados a temperatura ambiente. Se forman dos fases, la fase de la parte superior contiene los FAMES.

Las FAMES fueron analizadas por GLC usando un cromatógrafo de gases Hewlett- Packard 6890 (Palo Alto, CA, USA). La columna fue una Supelco SP-2380 fused-silica capillary column (30 m length, 0.25 mm i.d., 0.20 mm film thickness; Supelco, Bellefonte, PA, USA). Como gas de arrastre fue usado hidrógeno a 28 cm/s, y mientras la temperatura del detector e inyector fue 200ºC, la temperatura del horno fue 170ºC. Los diferentes metil ésteres fueron identificados por comparación con un control conocido [16].

El resultado de los análisis individuales junto con las clases se presenta en el Anexo correspondiente.

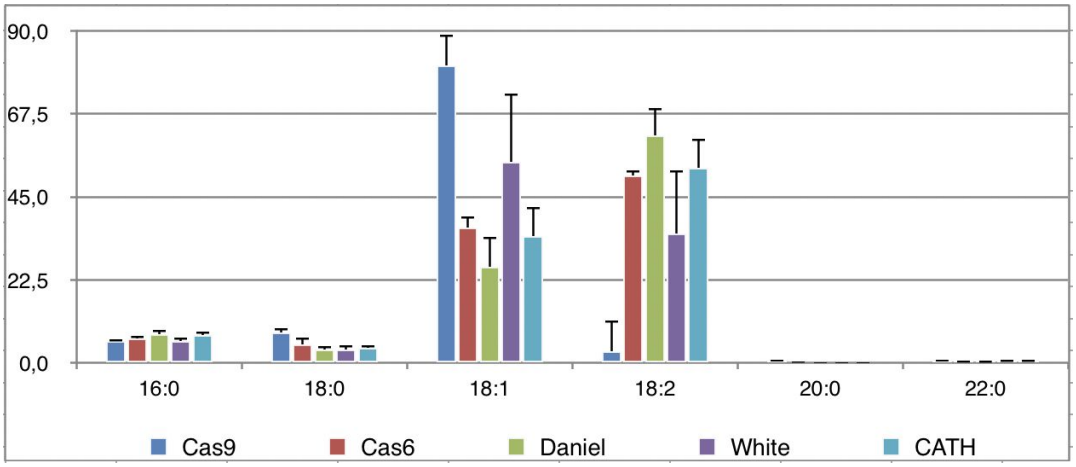


Figura 21. Análisis de FAMES. Diferentes clases y su contenido en los principales ácidos grasos.

6. Tecnologías Utilizadas

6.1 Python

Python es un lenguaje de programación creado por **Guido van Rossum** a finales de la década de los 80. Actualmente cuenta con una gran tasa de uso, esta popularidad se debe en gran medida a su alto nivel, gran número de librerías, como veremos más adelante, es un lenguaje que facilita la depuración del código, y además, es uno de los lenguajes de programación oficiales de Google. Por todas estas razones es un lenguaje con muchísimo soporte, manuales y relativamente fácil de aprender por lo que ha sido objeto de estudio en varias asignaturas del máster.

6.1.1 Instalar Python 3 para macOS

Bajamos el archivo desde la página de Python y lo instalamos regularmente siguiendo las indicaciones del instalador.

Comprobamos la instalación

```
$ python3
Python 3.6.3 (v3.6.3:2c5fed86e0, Oct  3 2017, 00:32:08)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

6.1.2 Paquetes, módulos y librerías de Python utilizados

Para la elaboración del modelo CNN, y el preproceso de las imágenes fueron utilizados los siguientes paquetes:

Matplotlib: paquete de Python para gráficos en 2D, utilizado para visualización rápida de datos y figuras.

Numpy: es un paquete que provee a Python con arreglos multidimensionales altamente eficientes con diseño especial para el cálculo numérico, operaciones de álgebra lineal, operaciones con matrices, etc.

Pickle: convierte objetos de Python en bytes (serialización de objeto), estos bytes pueden ser almacenados para luego reconstruir un objeto con las mismas características.

Display: es un módulo que contiene herramientas de presentación.

CV2 (OpenCV): es una biblioteca utilizada para el desarrollo de aplicaciones de visión artificial.

PIL (pillow): es una librería que permite la edición de imágenes.

Sklearn: es una librería con utilidades para machine learning.

6.2 Keras y TensorFlow

Keras es una librería de alto nivel y de código abierto para redes neuronales escrita en Python. Puede funcionar con TensorFlow, CNTK o Theano como motores de backend. Fue diseñada para permitir una experimentación rápida. Funciona perfectamente con GPU o CPU. También soporta CNNs, redes neuronales recurrentes (RNN) y una mezcla de ambas, además de ser compatible con Python 2.7-3.6. Keras funciona con TensorFlow como backend por defecto.

TensorFlow es una librería de código abierto para machine learning desarrollada originalmente por científicos e ingenieros del equipo Google Brain del departamento de machine learning de Google. Es una plataforma con soportes para construir gran variedad de redes neuronales aunque gracias a su flexibilidad y gran rendimiento para computación numérica también es usada en otros campos científicos.

6.2.1 Instalación

Antes de instalar Keras se recomienda instalar el motor de backend elegido, en nuestro caso utilizaremos TensorFlow.

Instalar TensorFlow para macOS

```
$ pip3 install tensorflow
```

Comprobamos la instalación

```
$ pip3 show tensorflow
```

```
Name: tensorflow
Version: 1.13.1
Summary: TensorFlow is an open source machine learning framework for
everyone.
Home-page: https://www.tensorflow.org/
Author: Google Inc.
Author-email: opensource@google.com
License: Apache 2.0
Location:
/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packa
ges
Requires: keras-applications, grpcio, tensorflow-estimator, astor, six,
protobuf, termcolor, wheel, gast, keras-preprocessing, absl-py, numpy,
tensorboard
Required-by:
```

Instalamos y comprobamos Keras

```
$ pip3 install keras
```

```
$ pip3 show keras

Name: Keras
Version: 2.2.4
Summary: Deep Learning for humans
Home-page: https://github.com/keras-team/keras
Author: Francois Chollet
Author-email: francois.chollet@gmail.com
License: MIT
Location:
/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packa
ges
Requires: numpy, keras-preprocessing, keras-applications, six, h5py,
pyyaml, scipy
Required-by:
```

7. Hardware Utilizado

En la realización de este trabajo se ha utilizado una computadora modelo MacBook Pro (13-inch, 2017, Four Thunderbolt 3 Ports).

Especificaciones técnicas:

Nombre del modelo: MacBook Pro

Identificador del modelo: MacBook Pro 14,2

Nombre del procesador: Intel Core i5

Velocidad del procesador: 3,1 GHz

Cantidad de procesadores: 1

Cantidad total de núcleos: 2

Caché de nivel 2 (por núcleo): 256 KB

Caché de nivel 3:4 MB

Memoria: 8 GB

Versión de la ROM de arranque: 190.0.0.0.0

Versión SMC (sistema): 2.44f1

Gráficos: Intel Iris Plus Graphics 650 1536 MB

Para el entrenamiento del modelo se planteó en primer lugar el uso de un cluster de supercomputación al que se podía tener acceso en caso de necesidad. Sin embargo, se llegó a la obtención de buenos resultados con tiempos de entrenamiento considerablemente cortos, por lo que no se usaron otros recursos.

8. Modelo de Clasificación de Imágenes por CNN

Para la realización de todos los apartados de este capítulo es precisa la carga previa de los paquetes y módulos que se van a utilizar en el mismo.

```
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
from tensorflow.keras.preprocessing.image import ImageDataGenerator,
array_to_img, img_to_array, load_img
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.layers import Activation, Dropout, Flatten, Dense
from tensorflow.keras import backend as K
from tensorflow.keras.callbacks import TensorBoard, ReduceLROnPlateau,
EarlyStopping, ModelCheckpoint
from tensorflow.keras.utils import plot_model
from IPython.display import display
from PIL import Image
from sklearn.metrics import confusion_matrix,classification_report
```

Tras dicha carga de paquetes ya estaríamos listos para empezar a actuar en cada apartado.

8.1 Procesamiento de datos

8.1.1 Escalado

Para poder entrenar nuestra red neuronal con nuestras imágenes primero se tuvo que plantear un cambio del tamaño de estas. Nuestras imágenes originales tienen un tamaño de 3024 x 3024 píxeles siendo este demasiado grande al dar una cantidad enorme de parámetros que entrenar. Se pensó en ajustar este tamaño a un número de píxeles mucho más bajo e ir probando con algunos tamaños mayores si no se conseguían buenos resultados. Dejándose como opción final 150 x 150 píxeles.

```
# dimensions of our images.
img_size = 150

train_data_dir = './data/train'
validation_data_dir = './data/validation'

# we also fix some useful variables
nb_train_samples = 33 * 5
nb_validation_samples = 17 * 5
epochs = 100
batch_size = 3

if K.image_data_format() == 'channels_first':
   input_shape = (3, img_size, img_size)
else:
   input_shape = (img_size, img_size, 3)
```

Se disponen dos carpetas con las imágenes de validación y entrenamiento cada una contiene cinco carpetas con el nombre de cada una de las clases previamente establecidas, en cada una de estas carpetas están las imágenes de cada clase numeradas. Esta disposición es importante para las funciones que vamos a emplear más adelante.

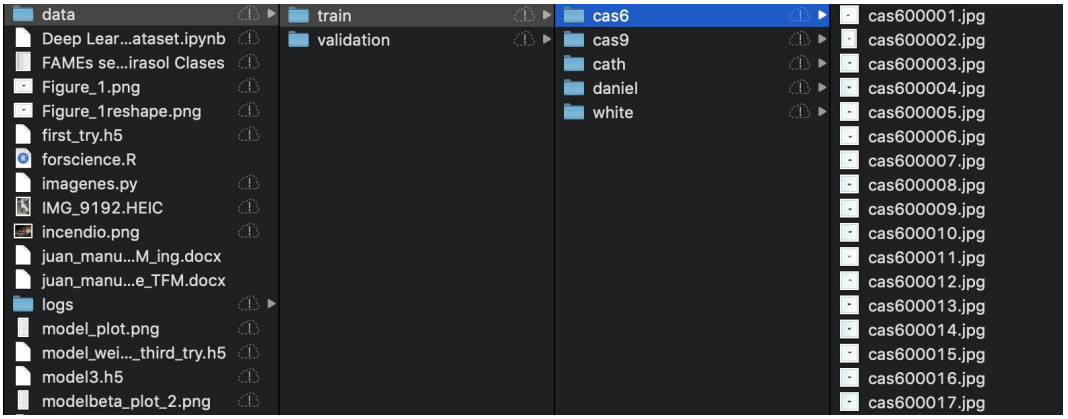


Figura 22. Disposición de archivos que forman la base de datos.

8.1.2 ***Aumento de datos***

Desde el principio se contempló esta opción ya que se disponían de muy pocas muestras de semillas. Keras viene con una módulo de procesado de imágenes que tiene una función que crea imágenes modificadas en base a las de nuestra base de datos y la red neuronal las interpreta como nuevos datos.

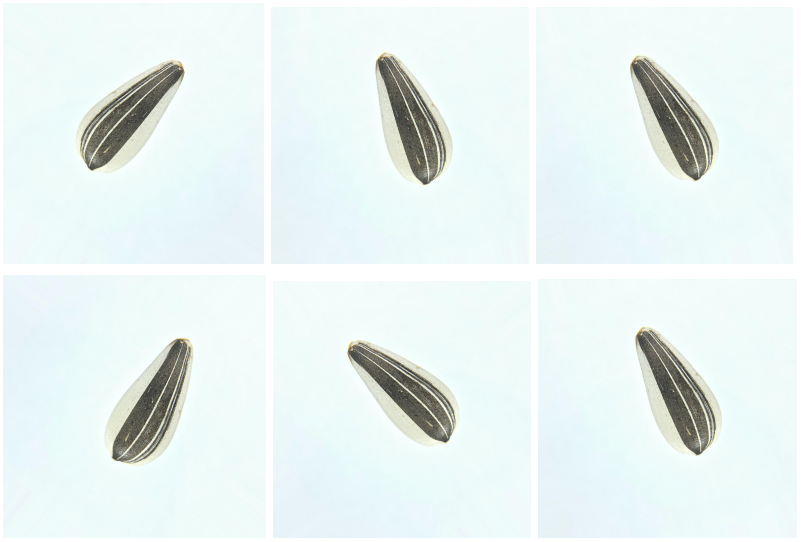
Se realiza un bucle con la función `train_datagen.flow()`, de esta forma vemos cómo quedarán nuestras imágenes generadas en base a una imagen de la base de datos.

```
# With this for loop we can see a sample of generated images based in a single
picture
img                                                                          =
load_img('/Users/jschrodinger/Desktop/TFM/data/train/daniel/daniel00002.jpg')     #
this is a PIL image

x = img_to_array(img)  # this is a Numpy array with shape (3, 150, 150)
x = x.reshape((1,) + x.shape)  # this is a Numpy array with shape (1, 3, 150, 150)

# the .flow() command below generates batches of randomly transformed images and
saves the results to the `preview/` directory
i = 0
for   batch   in   train_datagen.flow(x,   batch_size=1,   save_to_dir='preview',
save_prefix='daniel', save_format='jpeg'):
   i += 1
   if i > 5:
       break  # otherwise the generator would loop indefinitely
```

El resultado es el siguiente.



Figuras 23-28. Prueba de transformaciones realizadas por data augmentation.

Cuando usemos la función para generar imágenes de entrenamiento en nuestro script estas imágenes se irán tomando al azar de un directorio por el comando `.flow_from_directory()`.

En el siguiente script se establecen los cambios que la función `ImageDataGenerator()` realizará en las imágenes de test y las de validación. Las transformaciones aplicadas se indican con los argumentos. La documentación esta en https://keras.io/preprocessing/image/#imagedatagenerator-class

```
# this is the augmentation configuration we will use for training
train_datagen = ImageDataGenerator(
    rescale=1. / 255,
    rotation_range=40,
    shear_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')

# this is the augmentation configuration we will use for testing:
test_datagen = ImageDataGenerator(
    rescale=1. / 255,
    rotation_range=40,
    shear_range=0.2,
```

```
    horizontal_flip=True,
    fill_mode='nearest')

train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size = (img_size, img_size),
    batch_size = batch_size,
    class_mode = 'categorical')

validation_generator = test_datagen.flow_from_directory(
    validation_data_dir,
    target_size = (img_size, img_size),
    batch_size = batch_size,
    class_mode = 'categorical',
    shuffle=False)
```

rescale: es el factor de escalado, se multiplican los datos por el valor provisto.

rotation_range: rango en el que rotará la imagen aleatoriamente.

Shear_range: es el ángulo de corte en sentido contrario a las agujas del reloj.

horizontal_flip: si 'True' voltea aleatoriamente el input horizontalmente.

fill_mode: forma de rellenar los bordes con la opción 'nearest': aaaaaaaa|abcd|dddddddd

El resultado de dichas transformaciones es de aproximadamente 165 imágenes aleatoriamente creadas por cada epoch en el set de entrenamiento o un total de más de 16500 imágenes después de entrenar por 100 epochs.

Figuras 29-34. Ejemplo de transformaciones realizadas por data augmentation.

8.2 Creación de la estructura de la CNN

Como vimos en la introducción las CNN tienen como esquema general de estructura una capa input, seguida de una capa de convolución más una de agrupación un número n de veces, por último, una capa FC y una capa de output. El esquema general que sigue nuestra CNN es el siguiente.

```
model = Sequential()
model.add(Conv2D(32, kernel_size = 3, input_shape=input_shape, activation
= 'relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, kernel_size = 3, activation = 'relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, kernel_size = 3, activation = 'relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(64, activation = 'relu'))
model.add(Dropout(0.5))
model.add(Dense(5, activation = 'softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
```

Con este número de nodos y de tamaño de agrupación conseguimos reducir a la mitad el input cada vez que se realizan un par de operaciones convolución-agrupación. Añadimos una capa de dropout al 0.5 para evitar sobreajuste de los datos.

Podemos ver un esquema de la arquitectura del modelo con el comando `print(model.summary())`.

```
print(model.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 72, 72, 32)        9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 36, 36, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 34, 34, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 17, 17, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 18496)             0
_________________________________________________________________
dense (Dense)                (None, 64)                1183808
_________________________________________________________________
dropout (Dropout)            (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 5)                 325
=================================================================
Total params: 1,212,773
Trainable params: 1,212,773
Non-trainable params: 0
_________________________________________________________________
None
```

Al compilar el modelo se establece la función de coste '**categorical_crossentropy**', que es específica para clasificación de múltiples clases, y el optimizador '**rmsprop**' y se mide la precisión o '**accuracy**'.

La decisión de utilización del optimizador '**rmsprop**' y no '**adam**', como se suele hacer, se tomó al obtenerse mejores resultados con la primera. Aunque las diferencias son pequeñas, los resultados obtenidos en la matriz de confusión fueron un tanto mejores. Esto lo veremos en el capítulo destinado a ello.

8.2.1 ***Establecimiento de callbacks***

Los callbacks son de las funciones más útiles en Keras. Estas funciones se irán aplicando en distintas etapas del entrenamiento según proceda. Con ellas podemos obtener información sobre el estado interno y estadísticas del modelo e incluso cambiar algunos parámetros durante el proceso de entrenamiento. La documentación está en http://keras.io/callbacks/

Para nuestro modelo establecemos los siguientes callbacks.

TensorBoard

Esta es una herramienta de visualización de TensorFlow. Con ella podemos escribir un registro en un archivo que permitirá la confección de un archivo HTML en que se permite la visualización de gráficos dinámicos de los parámetros de medida del entrenamiento y la validación.

```
NAME = "CNN-seed-classifier_rmsprop"
# with tensorboard we can visualize the loss and accuracy values in graph
#  typing on console 'tensorboard --logdir=logs/'
tensorboard = TensorBoard(log_dir='logs/{}'.format(NAME))
```

No añadimos argumentos a esta función tan solo el directorio donde queremos guardar los registros con **`log_dir`**.

ReduceLROnPlateau

Reduce el learning rate cuando un parámetro prefijado ha dejado de mejorar.

`monitor`: el parámetro a monitorizar. En nuestro caso '`val_loss`'

`factor`: es el factor de reducción por el cual se multiplicará el learning rate. `new_lr = lr * factor`

`patience`: es el número de epochs sin mejoría del parámetro antes de que se reduzca el learning rate.

`min_lr`: es el mínimo al que puede llegar el valor del learning rate.

La reducción del learning rate cuando no hay mejoría en los parámetros es importante ya que permite al modelo dar pasos más cortos hacia los mínimos locales de la función de coste sin pasarse.

```
# this reduce learning rate when a metric has stopped improving
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5,
min_lr=0.001)
```

ModelCheckpoint

Este callback guarda el modelo tras cada epoch cuando un parámetro preestablecido mejora, gracias a esto podemos recuperar el modelo con mejores valores tras un proceso de entrenamiento y usarlo para predicción.

`filepath:` en este path se guardará el modelo, con el nombre del archivo '`model_rmsprop.hdf5`'.

`verbose:` modo verbose activado, nos dará información de la acción en cada epoch.

`save_the_best_only:` sólo se guardará el modelo con los mejores parámetros.

```
# ModelCheckpoint saves the model after each epoch if the validation loss
improves
checkpointer = ModelCheckpoint(filepath='./tmp/model_rmsprop.hdf5',
verbose=1, save_best_only=True)
```

8.3 Entrenamiento del modelo

En este apartado usamos la función `model.fit_generator()` que recibe como argumentos las variables `train_generator()` y `validation_generator()` asignadas a las funciones `train_datagen.flow_from_directory()` y `test_datagen.flow_from_directory()` respectivamente.

`Callbacks` recibe una lista con las variables asignadas a los citados callbacks del paso anterior.

`Verbose` está activado con lo que iremos recibiendo información en consola en cada epoch.

```
model.fit_generator(
   train_generator,
   steps_per_epoch = nb_train_samples // batch_size,
```

```
    epochs = epochs,
    validation_data = validation_generator,
    validation_steps = nb_validation_samples // batch_size,
    callbacks = [tensorboard, reduce_lr, checkpointer],
    verbose = 2)
# steps_per_epoch = 33*5 (165)  // 3 = 55
# validation_steps = 17*5 (85) // 3 = 28.3 = 29
```

Entrenamos varios modelos durante por diferente número de epochs, finalmente se decidió comparar dos modelos con diferente función de activación por 100 epochs.

9. Medidas de Rendimiento.

En este apartado medimos el rendimiento del modelo con mejores parámetros, este fue previamente guardado con el callback `ModelCheckpoints()`. También veremos el historial de registros de los parámetros más importantes de varios modelos con el callback `TensorBoard()`.

9.1 Matriz de Confusión y Reporte de Clasificación

Gracias a las funciones `confusion_matrix()` y `classification_report()` podremos obtener la Matriz de confusión y el reporte de clasificación respectivamente, gracias a estos podemos observar de una manera más visual el resultado del rendimiento de nuestro modelo clasificador una vez entrenado.

```
# Performance Measurements
# Confusion matrix is created with the best model saved by Checkpoints
model                                                                                    =
tf.keras.models.load_model('/Users/jschrodinger/Desktop/TFM/tmp/model_rmsprop.hdf5')

predictions  =  model.predict_generator(validation_generator,   nb_validation_samples  //
batch_size + 1)

y_pred = np.argmax(predictions, axis=1)

class_labels = list(validation_generator.class_indices.keys())
print('\nConfusion Matrix\n')
print(class_labels)
```

```
print(confusion_matrix(validation_generator.classes, y_pred))

# Confusion Metrics: Accuracy, Precision, Recall & F1 Score
report = classification_report(validation_generator.classes, y_pred, target_names=
class_labels)
print('\nClassification Report\n')
print(report)
```

Primero debemos cargar el modelo con el que hacer las predicciones, este será el que previamente se guardó con el callback ModelCheckpoints(), y seguidamente generar unas predicciones con la función model.predict_generator(). Una vez hechas las predicciones las comparamos con las etiquetas originales en la función confusion_matrix() para obtener la matriz de confusión. Luego con las mismas predicciones generamos el reporte de clasificación. El resultado que se muestra en consola es el obtenido para el modelo '`model_rmsprop.hdf5`'.

```
Confusion Matrix

[[15  2  0  0  0]
 [ 0 17  0  0  0]
 [ 0  0 17  0  0]
 [ 0  0  0 16  1]
 [ 0  0  0  0 17]]

Classification Report

              precision    recall  f1-score   support

        cas6       1.00      0.88      0.94        17
        cas9       0.89      1.00      0.94        17
        cath       1.00      1.00      1.00        17
      daniel       1.00      0.94      0.97        17
       white       0.94      1.00      0.97        17

    accuracy                           0.96        85
   macro avg       0.97      0.96      0.96        85
weighted avg       0.97      0.96      0.96        85
```

Probamos con el modelo que usó la función adam como optimizador y obtenemos los siguientes resultados. Para ello cambiamos el modelo a '`model_adam.hdf5`' con el siguiente comando y corremos el script en consola para ver los resultados.

```
model = tf.keras.models.load_model('/Users/jschrodinger/Desktop/TFM/tmp/model_adam.hdf5')
```

```
Confusion Matrix

[[13  4  0  0  0]
 [ 3 14  0  0  0]
 [ 0  0 17  0  0]
 [ 0  0  0 17  0]
 [ 0  0  0  0 17]]

Classification Report

              precision    recall  f1-score   support

        cas6       0.81      0.76      0.79        17
        cas9       0.78      0.82      0.80        17
        cath       1.00      1.00      1.00        17
      daniel       1.00      1.00      1.00        17
       white       1.00      1.00      1.00        17

    accuracy                           0.92        85
   macro avg       0.92      0.92      0.92        85
weighted avg       0.92      0.92      0.92        85
```

En los resultados anteriores tenemos diferentes parámetros que son Accuracy, Precision, Recall y F1 score. Para la interpretación de estos parámetros se usan las medidas de los True Positives (TP) True Negatives (TN) False Positives (FP) False Negatives (FN).

True Positives (TP): son aquellos valores de clases correctamente clasificadas como valores positivos. Pertenece a la clase alfa y ha sido clasificado como tal.

True Negatives (TN): son aquellos valores de clases correctamente clasificadas como valores negativos. Pertenece a la clase beta y ha sido descartado como alfa.

False Positives (FP): son aquellos valores de clases erróneamente clasificadas como positivas. Pertenece a la clase beta y es clasificada como alfa.

False Negatives (FN): son aquellos valores de clases erróneamente clasificadas como negativas. Pertenece a la clase alfa y ha sido descartado del grupo de los alfa.

	Predicted class		
Actual Class		Class = Yes	Class = No
	Class = Yes	True Positive	False Negative
	Class = No	False Positive	True Negative

Figura 35. Esquema general de matriz de confusión.

Una vez entendidos estos parámetros podemos definir los estadísticos Accuracy, Precision, Recall, y F1 score.

Accuracy
Es una proporción de las observaciones correctamente predichas entre el total de observaciones. Para nuestro modelo 'rmsprop' obtenemos un 0.96, lo que significa que es correcto en un 96% de las veces. En nuestro modelo 'adam' tenemos un 0.92 o predicciones correctas un 92% de las veces.
Accuracy = TP+TN/TP+FP+FN+TN
RMSprop
17+17+17+16+15 / 85 = 0.96
Adam
17+17+17+14+13 / 85 = 0.92

Precision
Es la proporción de observaciones positivas predichas correctamente entre el total de observaciones positivas predichas ya sea correcta o incorrectamente. Se obtiene para cada una de las clases
Precision = TP/TP+FP
RMSprop Precision CAS-6
15 / 15 = 1.00
Adam Precision CAS-6
13 / 13+3 = 0.8125

Recall (sensitibity)
Es la proporción de observaciones positivas predichas correctamente entre todas las observaciones de la clase. La pregunta que se responde con este estadístico es ¿de todas las semillas que son realmente de la clase alfa, cuántas etiquetamos? En nuestro caso tenemos unos valores por encima de 0.5 lo cual es un buen indicador.
Recall = TP/TP+FN
RMSprop Recall CAS-6
15 / 15+2 = 0.88
Adam Recall CAS-6
13 / 13+4 = 0.7647

F1 Score: es la media ponderada de Precision y Recall. Por tanto, esta puntuación toma en consideración los FN y FP. Accuracy es mejor estadístico si FN y FP tienen el mismo costo, como en nuestro caso.
F1 Score = 2*(Recall * Precision) / (Recall + Precision)
RMSprop F1 Score CAS-6
(0.88 *1 / 1.00+0.88) * 2 = 0.94
Adam F1 Score CAS-6
(0.76 * 0.81 / 0.76 + 0.81) * 2 = 0.79

9.2 Gráficos de TensorBoard

Aquí veremos el resultado del callback `TensorBoard()` que, como ya explicamos anteriormente, nos produce unos gráficos dinámicos en HTML con los que visualizamos los parámetros más importantes en cada epoch.

Para ello debemos usar el comando '`tensorboard --logdir=logs/`'

```
$ tensorboard --logdir=logs/

TensorBoard 1.13.1 at http://MacBook-Pro-de-Juanma.local:6006 (Press CTRL+C to quit)

I0602 19:49:33.149888 123145432072192 _internal.py:122] ::ffff:192.168.1.133 - - [02/Jun/2019 19:49:33] "GET /data/environment HTTP/1.1" 200 -
```

Copiamos la dirección 'http://MacBook-Pro-de-Juanma.local:6006' y la pegamos en la barra de direcciones, así se nos abrirá la aplicación con los gráficos.

Figura 36. Gráfica de TensorBoard. Validation loss en eje de ordenadas frente a epochs en eje de abscisas.

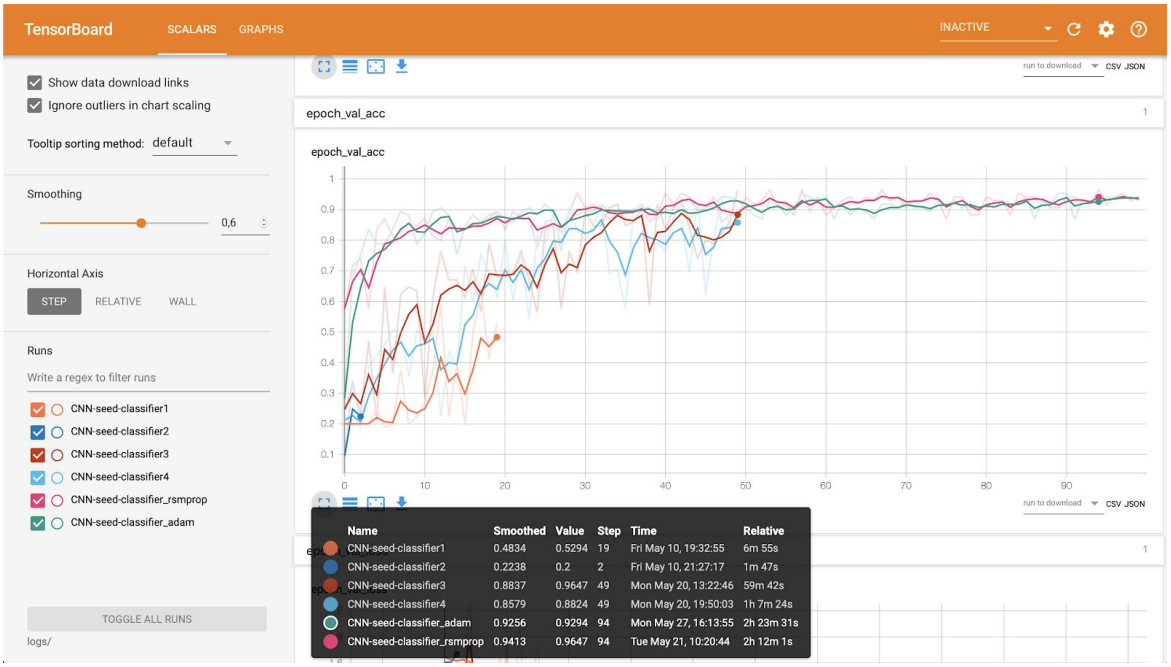


Figura 37. Gráfica de TensorBoard. Validation accuracy en el eje de ordenadas frente a epochs en el eje de abscisas.

El modelo con el que se han obtenido la matriz de confusión y el reporte de clasificación es 'CNN-seed-classifier_rmsprop' en rojo y con el que comparamos en los gráficos es 'CNN-seed-classifier_adam' nombrados según sus optimizadores y entrenados sobre los datos por 100 epochs. En ambos gráficos vemos marcada la epoch (Step = 94) donde el clasificador obtuvo mejores resultados. Los restantes registros de las CNN nombradas de la 1 al 4 fueron diferentes pruebas para descartar arquitecturas opcionales.

El optimizador adam debería de haber arrojado mejores parámetros que rmsprop. Este resultado poco convencional puede ser debido a que el reparto de los pesos en el principio del entrenamiento es al azar, habiendo obtenido pesos con mejores condiciones un modelo que otro. Se podría tomar como posible fuente de más exhaustiva investigación un mayor número de pruebas con las funciones de optimización, para comprobar esta hipótesis.

10. Conclusiones

En este trabajo se ha conseguido confeccionar una base de datos desde cero para la clasificación de imágenes de semillas de girasol, la cual se ha usado para el entrenamiento de una CNN resultando un modelo cuyos parámetros de medida del rendimiento de clasificación obtenidos superan los valores mínimos para concluir que el modelo obtenido es de calidad.

Para elaborar la base de datos se han utilizado herramientas y técnicas fotográficas especiales por la particularidad de los objetos a fotografiar. Una CNN necesita gran cantidad de datos para su entrenamiento, así que por la escasa disponibilidad de semillas para la elaboración de nuestra base de datos se han tenido que usar técnicas de aumento de datos, generando más de 16500 imágenes a partir de las 165 disponibles en el set de entrenamiento. También se han aplicado soluciones para evitar el sobreajuste como el dropout, al igual que se han probado varios métodos que facilitan la optimización de los coeficientes para llegar a la convergencia de la función de coste como la prueba de diferentes funciones optimizadoras y un learning rate dinámico si se encontraba un punto de silla.

Aunque en este trabajo se ha conseguido la obtención de un clasificador de calidad de semillas de líneas puras, no se ha trabajado con líneas cruzadas, donde la clasificación de ácidos grasos podría ser mucho más difícil debido a una diferencia más sutil en la composición de estos. Tampoco se hizo un seguimiento de la viabilidad estudiando las germínulas, lo cual era otro de los objetivos de este trabajo. Los motivos por los que no se incluyeron en la experimentación fueron la falta de tiempo y la falta de ejemplares, una parte considerable de las semillas eran demasiado antiguas e inviables, por lo que se decidió simplificar el modelo para así tener una base desde la que construir estudios más complejos en el futuro. Dichos estudios deberían integrar también un mayor número de pruebas con los diferentes optimizadores y tratar de explicar de manera formal los resultados obtenidos.

Por tanto, como conclusiones finales de este trabajo se pueden subrayar:

- Se han obtenido las competencias para la creación de una base de datos de imágenes y las técnicas de aumento de datos si se requieren.

- Se han conseguido competencias necesarias para la utilización de los paquetes Keras y TensorFlow para la creación de una CNN de alto rendimiento para la clasificación de semillas de girasol.

- Debido a la falta de tiempo y de ejemplares el modelo se tuvo que simplificar, no incluyendo en la clasificación líneas cruzadas. También se obtuvo un resultado inesperado con las funciones de optimización.

- Como posibles líneas de investigación futuras se podría proponer la construcción de un modelo más complejo que incluya las

estirpes cruzadas y mayor número de experimentos con los diferentes optimizadores.

11. Glosario

ANN	Artificial Neural Network (red neuronal artificial)
CNN	Convolutional Neural Network (red neuronal convolucional)
FAMES	Fatty Acids Methyl Esters (ácidos grasos metil ésteres)
GPU	Graphics Processing Unit (unidad de procesamiento de gráficos)
SVM	Support Vector Machine (máquina de vectores de soporte)
FC	Fully Connected
ReLU	Rectified Linear Unit
RMSprop	Root Mean Square Propagation
RGB	Red Green Blue
MP	Megapíxeles
LED	Light Emitting Diode
HDR	High Dynamic Range
FPS	Frames Per Second
LM	Lumen
PP	Polipropileno
AVCHD	Advanced Video Coding High Definition
DMP	Dimethoxypropane
GLC	Gas Liquid Chromatography
RNN	Recurrent Neural Networks
lr	Learning rate

12. Bibliografía

1. Global production volume palm oil, 2012–2018 [internet]. Hamburg: Statista; 2018. Available from: https://www.statista.com/statistics/613471/palm-oil-production-volume-worldwide/http://

2. Palm oil consumption [internet]. Zoetermeer: European Palm Oil Alliance. Available from: https://www.palmoilandfood.eu/en/palm-oil-consumptionhttp://

3. Wilcove, D. S., & Koh, L. P. (2010). *Addressing the threats to biodiversity from oil-palm agriculture. Biodiversity and Conservation, 19(4), 999–1007.*doi:10.1007/s10531-009-9760-x.

4. Kadandale, S., Marten, R., & Smith, R. (2018). *The palm oil industry and noncommunicable diseases. Bulletin of the World Health Organization, 97(2), 118–128.*doi:10.2471/blt.18.220434

5. Garcés, R., García, J. M., & Mancha, M. (1989). Lipid characterization in seeds of a high oleic acid sunflower mutant. Phytochemistry, 28(10), 2597–2600. doi:10.1016/s0031-9422(00)98047-6

6. Miller, J. F., Zimmerman, D. C., & Vick, B. A. (1987). *Genetic Control of High Oleic Acid Content in Sunflower Oil1. Crop Science, 27(5), 923.*doi:10.2135/cropsci1987.0011183x002700

7. Martínez-Force, E., Álvarez-Ortega, R., Cantisán, S., & Garcés, R. (1998). Fatty Acid Composition in Developing High Saturated Sunflower (Helianthus annuus) Seeds: Maturation Changes and Temperature Effect. Journal of Agricultural and Food Chemistry, 46(9), 3577–3582.doi:10.1021/jf980276e

8. Garces, R., & Mancha, M. (1993). One-Step Lipid Extraction and Fatty Acid Methyl Esters Preparation from Fresh Plant Tissues. Analytical Biochemistry, 211(1), 139–143.doi:10.1006/abio.1993.1244

9. Almeida, L. G., Backović, M., Cliche, M., Lee, S. J., & Perelstein, M. (2015). Playing tag with ANN: boosted top identification with pattern recognition. Journal of High Energy Physics, 2015(7).doi:10.1007/jhep07(2015)086

10. Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., … Hassabis, D. (2016). Mastering the game of Go

with deep neural networks and tree search. Nature, 529(7587), 484–489.doi:10.1038/nature16961

11. Esteva, A., Kuprel, B., Novoa, R. A., Ko, J., Swetter, S. M., Blau, H. M., & Thrun, S. (2017). Dermatologist-level classification of skin cancer with deep neural networks. Nature, 542(7639), 115–118. doi:10.1038/nature21056
12. McCulloch, W. S., & Pitts, W. (1943). *A logical calculus of the ideas immanent in nervous activity. The Bulletin of Mathematical Biophysics, 5(4), 115–133.* doi:10.1007/bf02478259
13. Hubel, D. H., & Wiesel, T. N. (1959). *Receptive fields of single neurones in the cat's striate cortex. The Journal of Physiology, 148(3), 574–591.*doi:10.1113/jphysiol.1959.sp006308
14. Fukushima, K. (1980). *Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biological Cybernetics, 36(4), 193–202.*doi:10.1007/bf00344251
15. Garcés, R., Martínez-Force, E., Salas, J. J., & Venegas-Calerón, M. (2009). Current advances in sunflower oil and its applications. Lipid Technology, 21(4), 79–82.doi:10.1002/lite.200900016
16. Salas, J. J., Moreno-Pérez, A. J., Martínez-Force, E., & Garcés, R. (2007). Characterization of the glycerolipid composition of a high-palmitoleic acid sunflower mutant. European Journal of Lipid Science and Technology, 109(6), 591–599.doi:10.1002/ejlt.200600285

13. Anexos

Código completo del modelo

```
import tensorflow as tf

import matplotlib.pyplot as plt

import numpy as np

from tensorflow.keras.preprocessing.image import ImageDataGenerator,
   array_to_img, img_to_array, load_img

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D

from tensorflow.keras.layers import Activation, Dropout, Flatten, Dense

from tensorflow.keras import backend as K

from tensorflow.keras.callbacks import TensorBoard, ReduceLROnPlateau,
   EarlyStopping, ModelCheckpoint

from tensorflow.keras.utils import plot_model

from IPython.display import display

from PIL import Image

from sklearn.metrics import confusion_matrix,classification_report




# dimensions of our images.

img_size = 150


train_data_dir = './data/train'

validation_data_dir = './data/validation'

nb_train_samples = 33 * 5

nb_validation_samples = 17 * 5
```

```
epochs = 100

batch_size = 3


if K.image_data_format() == 'channels_first':

   input_shape = (3, img_size, img_size)

else:

   input_shape = (img_size, img_size, 3)


model = Sequential()

model.add(Conv2D(32, kernel_size = 3, input_shape=input_shape, activation =
   'relu'))

model.add(MaxPooling2D(pool_size=(2, 2)))


model.add(Conv2D(32, kernel_size = 3, activation = 'relu'))

model.add(MaxPooling2D(pool_size=(2, 2)))


model.add(Conv2D(64, kernel_size = 3, activation = 'relu'))

model.add(MaxPooling2D(pool_size=(2, 2)))


model.add(Flatten())

model.add(Dense(64, activation = 'relu'))

model.add(Dropout(0.5))

model.add(Dense(5, activation = 'softmax'))




model.compile(loss='categorical_crossentropy',

             optimizer='rmsprop',

             metrics=['accuracy'])
```

```
# Earlystopping stops the learning process when some values are reached
#early_stop = EarlyStopping(monitor='val_loss', patience = 10, verbose=2,
   mode='min', baseline= 0.3, restore_best_weights=True)

NAME = "CNN-seed-classifier_rmsprop"
# with tensorboard we can visualize the loss and accuracy values in graph
#  typing on console 'tensorboard --logdir=logs/'
tensorboard = TensorBoard(log_dir='logs/{}'.format(NAME))

# this reduce learning rate when a metric has stopped improving
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5,
   min_lr=0.001)

# ModelCheckpoint saves the model after each epoch if the validation loss
   improves
checkpointer = ModelCheckpoint(filepath='./tmp/modeltweekedadam.hdf5',
   verbose=1, save_best_only=True)

# this is the augmentation configuration we will use for training
train_datagen = ImageDataGenerator(
   rescale=1. / 255,
   rotation_range=40,
   shear_range=0.2,
   horizontal_flip=True,
   fill_mode='nearest')


# With this for loop we can see a sample of generated images based in a single
   picture
```

```
#img =
   load_img('/Users/jschrodinger/Desktop/TFM/data/train/daniel/daniel00002.jpg'
   )  # this is a PIL image

#img.size

#x = img_to_array(img)  # this is a Numpy array with shape (3, 500, 500)

#x = x.reshape((1,) + x.shape)  # this is a Numpy array with shape (1, 3, 500,
   500)


# the .flow() command below generates batches of randomly transformed images
   and saves the results to the `preview/` directory

#i = 0

#for batch in train_datagen.flow(x, batch_size=1, save_to_dir='preview',
   save_prefix='daniel', save_format='jpeg'):

#    i += 1

#    if i > 5:

#        break  # otherwise the generator would loop indefinitely




# this is the augmentation configuration we will use for testing:


test_datagen = ImageDataGenerator(

   rescale=1. / 255,

   rotation_range=40,

   shear_range=0.2,

   horizontal_flip=True,

   fill_mode='nearest')


train_generator = train_datagen.flow_from_directory(

   train_data_dir,

   target_size = (img_size, img_size),
```

```
    batch_size = batch_size,
    class_mode = 'categorical')

validation_generator = test_datagen.flow_from_directory(
    validation_data_dir,
    target_size = (img_size, img_size),
    batch_size = batch_size,
    class_mode = 'categorical',
    shuffle=False)

model.fit_generator(
    train_generator,
    steps_per_epoch = nb_train_samples // batch_size,
    epochs = epochs,
    validation_data = validation_generator,
    validation_steps = nb_validation_samples // batch_size,
    callbacks = [tensorboard, reduce_lr, checkpointer],
    verbose = 2)
# steps_per_epoch = 33*5 (165)  // 3 = 55
# validation_steps = 17*5 (85) // 3 = 28.3 = 29

# Performance Measurements
# Confusion matrix is created with the best model saved by Checkpoints
model =
    tf.keras.models.load_model('/Users/jschrodinger/Desktop/TFM/tmp/modeltweeked
    .hdf5')

predictions = model.predict_generator(validation_generator,
    nb_validation_samples // batch_size + 1)
```

```
y_pred = np.argmax(predictions, axis=1)


true_classes = validation_generator.class_indices


class_labels = list(validation_generator.class_indices.keys())
print('\nConfusion Matrix\n')
print(class_labels)


print(confusion_matrix(validation_generator.classes, y_pred))


# Confusion Metrics: Accuracy, Precision, Recall & F1 Score
report = classification_report(validation_generator.classes, y_pred,
   target_names= class_labels)
print('\nClassification Report\n')
print(report)


# Plotting the model architecture and saving it in a .png file
plot_model(model, to_file='modelbetatweekedadam_plot.png', show_shapes=True,
   show_layer_names=True)


# Printing the model Summary
print(model.summary())
```

Muestras de imágenes de cada clase

White

Daniel

Cath

CAS-9

CAS-6