Escuela Técnica Superior de Ingeniería Informática

Ingeniería Del Software

CLASIFICACIÓN DE LESIONES EN LA PIEL USANDO APRENDIZAJE PROFUNDO

Realizado por

Elias Ibn Ziaten Cerezo

Tutorizado por

Rafaela Benítez Rochel y Enrique Domínguez Merino

Departamento

Lenguajes y Ciencias de la Computación

Fecha defensa:

Universidad de Málaga

Málaga, Junio de 2019

Resumen: Cada vez se realizan más búsquedas en internet sobre medicina y temas de salud. Las personas cada día depositan su confianza en internet para resolver todo tipo de problemas, y por desgracia, están dejando de acudir a los especialistas y en vez de ello, buscan en internet sus síntomas y dan fe de lo que el buscador les responda.
Con este Trabajo Fin de Grado se quiere dar solución a parte de esta tendencia creciente en la sociedad actual, proporcionando una herramienta capaz de asistir al usuario de manera efectiva cuando se trate de clasificar una lesión cutánea en una de las tres categorías que se han seleccionado para la clasificación.
El Trabajo Fin de Grado se sustenta en un sistema inteligente que ha sido entrenado con un conjunto de datos concreto y al que se la ha dotado de un enfoque conservador mediante la configuración de sus distintos parámetros.

Palabras claves: Deep Learning, Redes Neuronales, Clasificación, Lesiones Cutáneas

Abstract: There is a growing tendency on Internet searching about Medecine and health-related subjects. Everyday, people deposit their trust on Internet to solve all kinds of problems. Unfortunately they are stopping going to the specialists and instead of that, they search their symptoms on Internet and they trust whatever the searching engine retrieves them.
With this End-of-Degree Work we want to give a solution to a part of this growing tendency in society, providing a tool capable of assisting the user in an effective way when we are referring to classify any of the three categories of skin lesions that we have worked with.
This End-of-Degree Work uses an intelligent system that has benn trained with a concrete dataset. The intelligent system used in thios work has been given a conservative focus through it's parameter configuration
Keywords: Deep learning, Neural Networks, Classification, Skin Lesions

A mi madre y a José Manuel,
por estar siempre ahí.

Elias

Acrónimos

ETSIT	Escuela Técnica Superior de Ingeniería de Telecomunicación
UMA	Universidad de Málaga
TFG	Trabajo Fin de grado
SO	Sistema Operativo
CPU	Central Proccessing Unit o más comúnmente procesador
GPU	Graphics Proccessing Unit o más comúnmente tarjeta gráfica
ML	Machine Learning
DL	Deep Learning

Índice

Prólogo

El uso de páginas web y de las redes sociales para consultar cuestiones relacionadas con el diagnóstico o tratamiento de enfermedades está más en auge que nunca. El mundo digital alberga una cantidad ingente de información, muchas veces útil, pero otras tantas errónea o, sencillamente, no aplicable a la enfermedad o a los síntomas que consultamos.

Seis de cada 10 españoles utiliza Internet para informarse sobre salud, según la última encuesta del Observatorio Nacional de las Telecomunicaciones y de la Sociedad de la Información. Los buscadores web se han convertido en una especie de "consulta médica extraoficial" a la que el 85 % acude como primera y única opción cuando tiene algún tipo de síntoma o duda en relación a posibles enfermedades, diagnósticos o tratamientos.

La búsqueda de información relacionada con la salud a través de Internet crece cada año y el cáncer está entre las enfermedades más buscadas. Tener lunares puede ser un factor de riesgo de cáncer de piel pero es importante saber que la mayoría son benignos. Es por ello que uno de los síntomas que más aparece en las búsquedas es si una formación en la piel es un lunar, o es cáncer de algún tipo. Hay distintas páginas que nos dan información real, incluyendo fotos de en qué nos tenemos que fijar para saber si es un lunar o podría ser algo más. El problema de este tipo de webs que dan información es que tienen un espectro muy pequeño, pues cada cuerpo y ser humano somos distintos y lo que podrían ser signos de cáncer en algunos casos puede no serlo en otros.

La información está al alcance de cualquiera. Sin embargo, el paciente no siempre cuenta con las herramientas necesarias para interpretarla correctamente. Ésta es la idea en la que se centra el proyecto que aquí se describe.

A través de imágenes se podrían clasificar distintos tipos de cáncer de piel. Estos tipos son: carcinoma basocelular, carcinoma de células escamosas, nevus y melanomas malignos. El diagnóstico de cáncer de piel comienza con un examen visual. Un dermatólogo suele analizar la lesión sospechosa a simple vista y con la ayuda de un dermatoscopio, un microscopio de mano que proporciona una imagen de la situación al nivel justo debajo de la piel. Si estos métodos no son concluyentes o llevan al dermatólogo a creer que la lesión es cancerosa, se necesita hacer una biopsia.

Dado que el diagnóstico inicial del dermatólogo se lleva a cabo con un examen visual, se pensó que era posible desarrollar un TFG aplicando técnicas de reconocimiento y clasificación de imágenes mediante Inteligencia Artificial (IA), concretamente de Deep Learning (DL); y tras comunicarme con mi tutora, ambos llegamos a la conclusión de que era una buena idea.

Lo que se va a observar a partir de ahora son los frutos del trabajo descrito.

Parte I

Introducción

Introducción y visión general

Contenido

Sinopsis

En este capitulo vamos a realizar la introducción al TFG hablando de los objetivos , estado del arte, metodologías, estrucutra y ámbito de aplicación.

Objetivo

El objetivo de este TFG es la creación de varias aplicaciones que sean capaces de clasificar a través de imágenes si lo que se muestra en ellas se encuadra en uno de estos 3 grupos:

- Benigno: En este grupo se encuadraran tanto las lesiones que no son un indicativo maligno como imágenes de piel sana.
- Premaligno: Este grupo se refiere a aquellas lesiones que pueden indicar un origen maligno pero no tienen por qué serlo.
- Maligno: En este grupo encontraremos las lesiones que son un indicativo claro de que aquello que muestran es muy posible que sea un tumor maligno.

Esta aplicación se va a dividir en 3 partes claramente diferenciadas, la primera es el sistema inteligente, que será la encargada de discernir entre las imágenes que representan una lesión y las que no. La segunda parte está compuesta del servicio web que nos dará acceso a ese sistema y que recibirá y procesará las peticiones que nosotros realicemos. Por último, tenemos las aplicaciones cliente, una de ellas será una aplicación web y la otra será una aplicación para smartphones con Android como sistema operativo, pues éste es el SO más extendido en el sector móvil.

Las partes de las que este TFG se compone son:

- **El sistema inteligente**: Es la parte del TFG que ha llevado más tiempo desarrollar. Se encargará de clasificar las imágenes que le enviemos, ya sea mediante la cámara del teléfono o mediante el gestor de ficheros del sistema. Toda la información sobre el desarrollo paso a paso se encontrará en el capítulo 2 de esta memoria.
- **El servicio web**: Consiste en un servicio REST, que tendrá una serie de puntos a través de los cuales se podrá interactuar con el sistema inteligente.

- **Las aplicaciones cliente**: Se van a componer de dos aplicaciones, cuyo objetivo es el mismo, poner el sistema inteligente a disposición del usuario. La aplicación de escritorio consiste en una aplicación web, que corre sobre el servidor. Por otra parte, la aplicación móvil se pondrá en contacto con el servidor REST

Uno de los problemas que tiene el desarrollo de este TFG es el tiempo de ejecución. El equipo del que se va a disponer es medianamente potente para tareas de Deep Learning, pero aun así los tiempos de ejecución de entrenamiento son bastante extensos (pueden variar de menos de una hora hasta más de un día) por tanto, la cantidad de veces que se pueden entrenar los distintos modelos no es tan elevada como nos gustaría.

Estado del arte

Hay varios trabajos en esta linea de investigación, realizados por varios grupos de profesionales, incluso por una universidad norteamericana, pero en ninguno de los mencionados trabajos se ha desarrollado una aplicación móvil que ponga en contacto al usuario con el sistema inteligente.

En un primer momento se investigaron varios artículos en los que se había desarrollado un sistema similar al que se trata en este TFG, uno de ellos un articulo de Nature con identificador doi:10.1038/nature21056 en el que se usaba un conjunto de datos completamente novedoso y 2 ordenes de magnitud mas grande que el anterior trabajo con mayor numero de imágenes de ejemplo para el sistema inteligente.

En la plataforma Medium tambien he encontrado informacion sobre más personas trabajando sobre un problema parecido, pero todos usan el mismo conjunto de datos que yo, y no llegan a resultados concluyentes, con lo cual como al final no he usado nada de lo que proponían no aparecen citados en esta memoria.

Además de estos trabajos, también hay varios sitios en internet donde se pueden encontrar información varia, se han usado (`kaggle.com`) y (`medium.com`) para informarme acerca de las distintas estructuras de

redes neuronales, y de como usar las distintas librerías que se han usado en el TFG.

Metodología y directrices seguidas

Para la elaboración de este proyecto se han usado las conocidas como metodologías ágiles, concretando mas, SCRUM.

Se ha usado una herramienta para seguir los distintos requisitos, organizarlos y determinar que se iba a implementar en cada momento, añadiendo también las distintas dependencias entre requisitos y desglosando aquellos que eran demasiado generales en requisitos cada vez mas pequeños, hasta que resultaban implementables. El criterio de tiempo para determinar si eran implementables o no, era la implementabilidad en un día, con lo cual aquellos requisitos que se determinaba que eran demasiado extensos, se volvían a dividir.

Esta herramienta es Trello, se había usado previamente para hacer otros trabajos y organizar otros recursos, por tanto se tenía experiencia con ella.

Estructura del documento

Esta memoria está estructurada en 3 capítulos y una parte de conclusiones.

En el primer capítulo se va a realizar una introducción a los sistemas inteligentes, incluyendo respuestas a la pregunta ¿Qué son las redes neuronales?, algunos tipos de redes neuronales que existen a día de hoy, se hablan de las librerías que se han utilizado en el proyecto, y se hace mención al overfitting.

En el segundo capítulo vamos a encontrar toda la información del desarrollo; se hablará sobre el sistema inteligente, por qué se han escogido unos modelos neuronales para entrenar y no otros, requisitos, casos de uso, y todo el desarrollo tanto del sistema inteligente como de las

aplicaciones.

En el tercer y último capítulo se encuentran las pruebas realizadas tanto al sistema inteligente como a las aplicaciones, tomando más protagonismo las del sistema inteligente pues nos centramos en las distintas configuraciones de hiperparámetros y en qué resultados da cada una.

Por último en la parte de conclusiones se verá que se ha aprendido y las conclusiones a las que se ha llegado durante el aprendizaje, desarrollo y pruebas de este trabajo.

Ámbito de aplicación

El ámbito de aplicación de este trabajo es principalmente asistente, pues se estuvo investigando y hablando con un par de especialistas y ambos coincidían que si se dispusiera de una aplicación que fuera capaz de ayudar a preclasificar las lesiones antes de que el paciente acudiera a consulta, se ahorraría mucho tiempo y esperas en las consultas más concurridas. Sin embargo este TFG no pretende sustituir al veredicto de un dermatólogo especialista ni mucho menos, de hecho cuando se hable de las aplicaciones veremos las advertencias en las que se dice que cualquier veredicto que obtengamos estará en entredicho hasta que un especialista examine aquello que se ha fotografiado o enviado.

Parte II

Desarrollo del proyecto

Capítulo 1

Manual e introducción al desarrollo del proyecto.

Contenido

Sinopsis

Este capítulo, se va a usar como introducción a todas las tecnologías que se han usado en el proyecto, centradas principalmente en el sistema inteligente, aunque también hay un pequeño espacio reservado a introducir las tecnologías en las que se han desarrollado las aplicaciones, tanto de escritorio, como móvil.

1.1. Introducción a sistemas inteligentes

Los sistemas inteligentes artificiales se definen como aquellos que presentan un comportamiento externo similar en algún aspecto a la inteligencia humana o animal. Se caracterizan por su capacidad para representar, procesar y modificar de forma explícita conocimiento sobre un problema, y para mejorar su desempeño con la experiencia. Esto les permite resolver problemas concretos determinando las acciones a tomar para alcanzar los objetivos propuestos, a través de la interacción con el entorno y adaptándose a las distintas situaciones.

Existen muchos tipos de sistemas inteligentes, de entre todos ellos se han elegido las redes de neuronas artificiales para desarrollar este TFG.

1.1.1. ¿Qué son las redes neuronales?

Una red neuronal es un modelo computacional dentro del aprendizaje automático inspirado en el comportamiento observado en el cerebro humano. Consiste en un conjunto de unidades de procesamiento llamadas neuronas, conectadas entre sí para así poder transmitirse señales. Estas unidades de proceso de pueden combinar de diferentes formas dando lugar diferentes arquitecturas de redes, sobre éstas se aplican diferentes dinámicas de computación.

En una dinámica de alimentación directa o tipo *forward*. La información de entrada atraviesa la red, sometiéndose a diversas operaciones según pasa por las distintas capas de neuronas, y finalmente se genera una salida.

Cada neurona está conectada con otras a través de unos enlaces. En estos enlaces el valor de salida de la neurona anterior es multiplicado por un valor de peso. Estos pesos en los enlaces pueden activar o inhibir las neuronas adyacentes. Del mismo modo, a la salida de la neurona, puede existir una función que modifica el valor antes de propagarse a otra neurona. Esta función se conoce como función de activación o de transferencia. Las redes neuronales actuales suelen contener desde unos miles a unos pocos millones de neuronas artificiales que se organizan en

estructuras llamadas capas

Estos sistemas aprenden y se forman a sí mismos, en lugar de ser programados de forma explícita, y sobresalen en áreas donde la detección de soluciones o características es difícil de expresar con la programación convencional. Para realizar este aprendizaje automático existen diferentes tipos de aprendizaje clasificables en *supervisado*, *no supervisado* o *reforzado*. Para nuestra aplicación usaremos un conjunto de datos de entrenamiento supervisado, es decir, disponemos de la salida correcta asociada a cada patrón con lo cual se usará un algoritmo de aprendizaje que intenta minimizar una función de pérdida o error que evalúa la red en su totalidad. Los valores de los pesos de las neuronas se van actualizando, tal y como el algoritmo de aprendizaje nos indique, buscando reducir el valor de la función de pérdida. Este proceso se realiza mediante la propagación hacia atrás. Los conceptos de función de pérdida o *loss*, pesos o *weights*, función de activación y propagación hacia atrás o *backpropagation* se definirán con detalle más adelante

El objetivo de la red neuronal es resolver los problemas de la misma manera que el cerebro humano, aunque las redes neuronales son más abstractas.

Las redes neuronales son expertas en detección y aprendizaje de patrones, y se componen de distintos tipos de capas. Cada capa realiza una serie de operaciones más o menos complejas, dependiendo del tipo de la capa. La variedad de capas se expondrá más tarde, cuando se trate implementación del proyecto, qué hacen cada una de ellas, y su importancia.

1.1.2. Conocimientos necesarios

En esta sección se van a incluir las definiciones mencionadas anteriormente y que a partir de ahora se usarán con bastante asiduidad.

- *weights* o pesos: Son los factores que ponderan las distintas entradas de las distintas capas de la red neuronal.
- *loss* o pérdida: Es una función estadística que nos dice la diferencia

entre el valor real que estamos intentando predecir y el dado por la red neuronal. Hay varias funciones a escoger, algunas de ellas son: *error cuadrático medio, error absoluto medio, error logarítmico absoluto medio, entropía cruzada, poisson* etc.

- *backpropagation* o propagación hacia atrás: Es una algoritmo de aprendizaje supervisado por corrección de error basado en la técnica de optimización del gradiente descendente aplicable a redes de neuronas artificiales de más de una capa oculta que tenga función de activación o transferencia distinta de la función lineal.

1.1.3. Tipos de redes y uso

Existen gran variedad de tipos de redes neuronales, en este apartado se dará una breve descripción de cada tipo, con el objetivo de enmarcar las que se han usado en este trabajo.

En la Figura 1.1 podemos observar los distintos tipos de redes neuronales que se van a tratar en este apartado

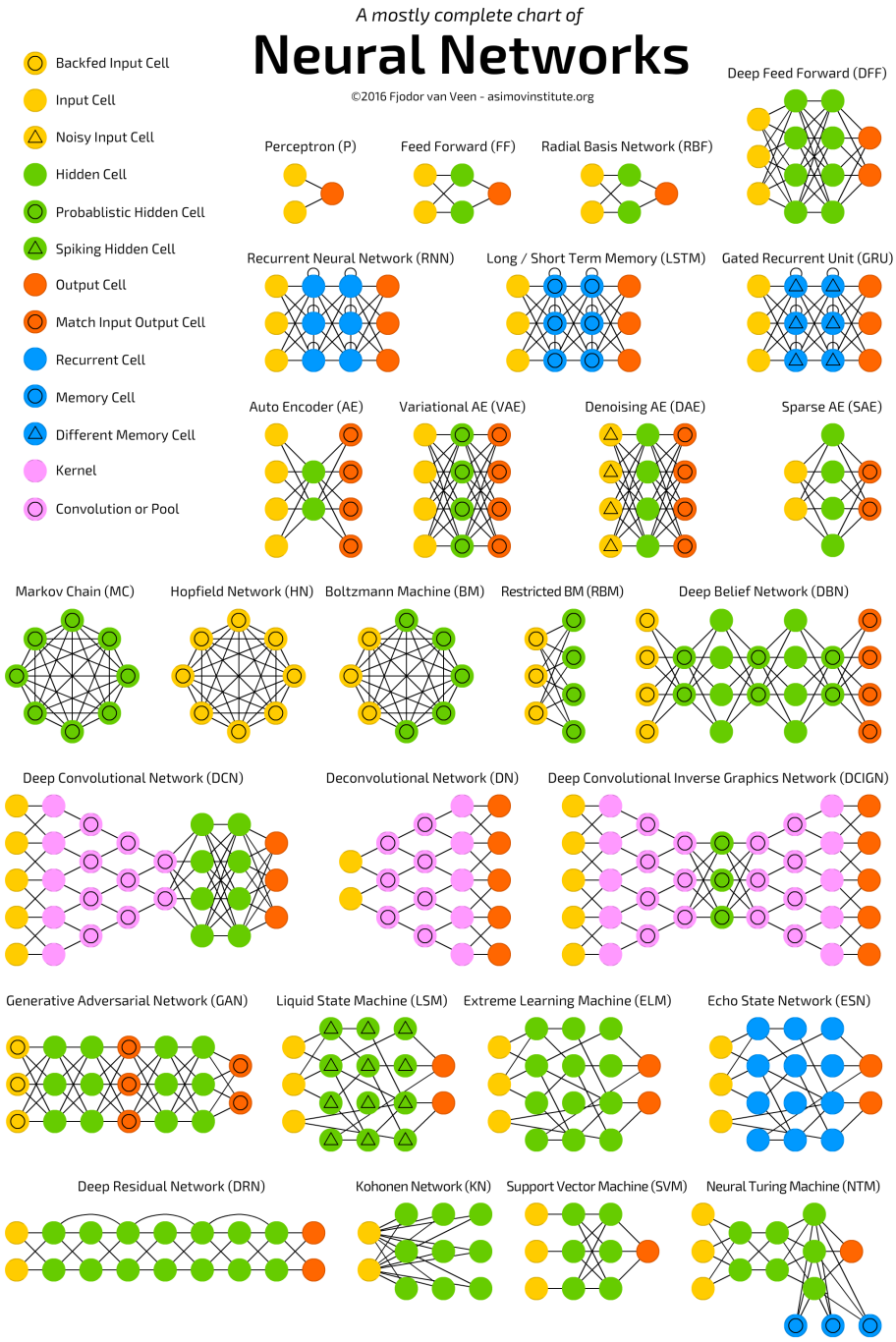


Figura 1.1: Diagrama de tipos de redes neuronales

Perceptron

El modelo más básico que se muestra en la figura es el del perceptrón simple. Su dinámica de la computación consiste en tomar las entradas, calcular la suma de dichas entradas ponderadas mediante los pesos sinápticos y aplicar una función de activación a esta señal combinada para calcular la salida.

Feed Forward (FF)

Todas sus neuronas están interconectadas, la activación se sucede desde la capa de entrada hacia la de salida, sin retroalimentación. Hay una capa entre la capa de entrada y la capa de salida (capa oculta).

Radial Basis Network (RBF)

Son redes del tipo FF que usan una función radial en vez de sigmoidal. La diferencia es que la función logística asigna a la salida un valor entre 0 y 1 (o -1 y 1), respondiendo así a preguntas de si o no, pero trabajan mal con valores continuos. La función radial en cambio responde a la pregunta ¿A qué distancia estamos del objetivo?, lo cual es perfecto para aproximación de funciones. Se podría decir que las funciones logísticas trabajan con valores discretos y las funciones radiales trabajan con valores continuos, aunque no sea verdad, pero ayuda a ver las diferencias claras entre ellas.

Deep Feed Forward (DFF)

Fue la estructura que abrió la caja de pandora del Deep Learning. Gracias a la capacidad de computación que se ganó en los primeros años del siglo XXI hicieron posible el entrenamiento de este tipo de redes neuronales con más de una capa oculta.

Recurrent Neural Networks (RNN)

Introducen un nuevo tipo de neuronas. Las neuronas recurrentes. Estas neuronas se diferencian de las vistas hasta ahora pues vuelven a

tomar la entrada pasado un tiempo determinado, esto hace que para ellas sea importante el contexto.

Long/Short Term Memory (LTSM)

Este tipo de red neuronal introduce un nuevo tipo de neurona, las neuronas con memoria. Estas pueden procesar datos que tienen diferencias en el tiempo. Si las RNN podían procesar datos en base a su contexto, estas pueden hacerlo en base a distintos instantes temporales. Por ello, son utilizadas para reconocimiento de vídeo, y también para procesamiento de lenguaje.

Grated Recurrent Unit (GRU)

Son LTSMs pero en vez de marcas temporales utilizan un periodo concreto, por ello son indicadas para reconocimiento musical y síntesis del habla.

Auto Encoder (AE)

Los Auto Encoders se utilizan para clasificación, agrupamiento y compresión de características. Cuando entrenamos una red del tipo FF, estamos realizando aprendizaje supervisado

Variational Auto Encoder (VAE)

La diferencia entre estos y los AE es que estos realizan probability compression en vez de feature compression, y utiliza un enfoque diferente a la hora de resolver la pregunta ¿Como generalizamos los datos?, pues la estructura se pregunta si 2 elementos son completamente independientes.

Denoising AE (DAE)

Utiliza la inclusión de ruido en las entradas para forzar a la red neuronal a quedarse con las características más generales, intentando tratar así el sobreajuste, concepto que se tratará más tarde.

Sparse AE (SAE)

Sae es otro autoencoder que puede ayudar a identificar patrones ocultos dentro de los datos. Lo que cambia de la estructura es el numero de neuronas de la capa oculta, pues es mayor que el numero de neuronas de la capa de entrada.

Markov Chain (MC)

Las cadenas de Markov son un concepto antiguo de grafos donde cada arista tiene una probabilidad asignada. Las cadenas de Markov no son redes neuronales clásicas. Se usan para clasificación basada en probabilidad, como los filtros de Bayes, para algunos tipos de clustering y como una maquina de estados finita.

Hopfield Network (HN)

Las neuronas de tipo Hopfield son entrenadas con un numero de datos limitado para que respondan a un dato conocido con el mismo dato. Cada neurona es utilizada como entrada antes del entrenamiento, como neurona oculta durante el entrenamiento y como salida cuando es utilizada. Su uso principal es para eliminación de ruido y restauración de entradas.

Boltzmann Machine(BM)

Las maquinas de Boltzmann son muy parecidas a las HNs. Esta es la primera estructura neuronal que fue entrenada utilizando recocido simulado. Si unimos muchas maquinas de Boltzmann obtenemos una Deep Belief Network que veremos mas adelante.

Restricted BM (RBM)

Utilizan la estructura de las BM pero permiten el entrenamiento usando backpropagation como en las FF.

Deep Belief Network (DBN)

Las DBN son maquinas de Boltzmann conectadas y rodeadas por VAEs. Pueden ser conectadas unas con otras y generar datos a partir de patrones ya aprendidos,

Deep Convolucional Networks (DCN)

Estas son las que se van a usar en este TFG. Usan neuronas de convolución, capas de agrupamiento y núcleos, cada uno con una utilidad diferente. Los núcleos procesan los datos de entrada y las capas de agrupamiento lo simplifican usando funciones no lineales.

Normalmente se usan para reconocimiento de imágenes, usando un marco de la imagen (por ejemplo de 20*20). Este marco se desplaza por toda la imagen a modo de escáner para ir extrayendo las formas según van avanzando las iteraciones. Se combinan con DFFs al final para un procesamiento mas en profundidad.

Deep Convolucional Inverse Graphics Network (DCIGN)

Se usan mayormente para procesamiento de imágenes que la red no ha visto antes, estas redes gracias a su nivel de abstracción son capaces de eliminar ciertos objetos de la imagen, redibujarla, o insertar elementos que no estaban antes.

Generative Adversarial Network (GAN)

GAN representa una familia de redes neuronales dobles, compuestas por un generador y un discriminador que intentan mentirse mutuamente. Este tipo de estructuras evolucionan infinitamente siempre y cuando seamos capaces de mantener el balance en el entrenamiento.

Liquid State Machine (LSM)

LSM es una red neuronal no conexa donde las funciones de activación se convierten en umbrales de activación. Una neurona acumula valores

de muestras secuenciales y solo emite una salida cuando el umbral se alcanza. Se usan en visión por computador y reconocimiento del lenguaje.

Extreme Machine Learning (ELM)

ELM es un intento de reducir la complejidad creando capas ocultas no conexas entre ellas y con conexiones aleatorias. Son menos costosas computacionalmente pero depende mucho de los datos con los que vayamos a trabajar.

Support Vector Machines (SVM)

Se usan para clasificación binaria, no importan las entradas, la red procesara las entradas y la respuesta siempre será 'si' o 'no'.

Una vez vistas las redes neuronales mas importantes para dar contexto. Vamos a centrarnos en las que se usan en este TFG, y vamos a ver como vamos a operarlas

1.2. Keras

Figura 1.2: Logo de Keras

En esta sección se introduce la librería que vamos a usar en el desarrollo de la parte más importante del TFG pues las otras partes son interfaces para acceder al sistema inteligente.

1.2.1. TensorFlow

Keras es un framework de Machine Learning que puede ejecutarse sobre TensorFlow, Theano, o CNTK. Se ha escogido TensorFlow porque ya se tenia previo conocimiento de éste y además porque es lo más utilizado a fecha de la redacción de ésta memoria en materia de Machine Learning y de redes neuronales.

Por otra parte tensorflow tiene dos modos de ejecución. Uno basado en CPU y otro en GPU, aunque el código es común para ambos, lo que le aporta polivalencia a la hora de adaptarse a los distintos entornos de ejecución. De hecho es posible delegar ciertas tareas en la CPU en un sistema en el que estemos ejecutando modelos con GPU para aprovechar más el hardware y optimizar tiempos. Los tiempos de entrenamiento como veremos más adelante son muy variables, pues depende del hardware del que dispongamos, de si existe pérdida de rendimiento por temperatura mientras lo usamos; por supuesto del tamaño del modelo, de la cantidad de datos que tengamos que iterar sobre el mismo, además de las épocas, de si tenemos una serie de funciones activadas (callbacks) entre otras variables que pueden hacer que nuestro modelo sea viable computacionalmente o no.

Figura 1.3: Logo de TensorFlow

1.2.2. Diferencias en ejecución entre CPU y GPU

La GPU acelera muchísimo el proceso. En el caso de este TFG se cuenta con una Nvidia GTX 1070 para la computación de los modelos. La explicación de por qué una GPU es capaz de acelerar mas el proceso es sencilla. Las tarjetas gráficas tienen muchos núcleos (La GTX1070 tiene 1920), aunque son núcleos más limitados en variedad de operaciones respecto a los de la CPU. Pero las operaciones de las redes neuronales son lo suficientemente sencillas como para que estos núcleos sean capaces de computarlas.

El mejor CPU del mercado actualmente tiene 64 núcleos, y la GPU con mayor número tiene 3072 núcleos, con lo que podemos ver que hay un orden de magnitud de diferencia, además del hecho de que se pueden unir varias GPU dentro del mismo ordenador, cosa que no suele ocurrir con las CPU pues normalmente las placas base de los ordenadores para consumo personal solo traen un socket para CPU pero es mas común tener 2 o 3 puertos PCI-Express que son los que utilizan las GPU para comunicarse con el sistema.

Además de los núcleos también tenemos que tener en cuenta otras variables de la GPU como son el tamaño y la velocidad de la memoria que tiene (también conocida como VRAM). En el caso de la GPU que se esta utilizando en la elaboración de este TFG, ésta tiene 8GB de VRAM. Uno de los problemas de los modelos de redes neuronales grandes es que necesitan mucha memoria en la GPU, y cuando ésta se agota, si el modelo no cabe en ese espacio de memoria, la GPU se ve obligada a hacer *swap* a disco o a memoria RAM con lo cual se ralentiza el proceso de entrenamiento del modelo. Éste es un problema que afecta a la elaboración de este TFG pues el modelo no cabe en la VRAM disponible.

1.2.3. Servicios en la nube para procesamiento

La computación en la nube es una plataforma abierta a todo el mundo, que cada día evoluciona más. A fecha de la elaboración de este TFG ya hay al menos dos proveedores de servicios que publicitan plataformas para el aprendizaje automático en la nube. Éstas son Google con su servicio Cloud Machine Learning Engine, y Amazon, concretamente la división de Amazon Web Services, ofrece varias instancias en las que te proporcionan una GPU para aceleración del entrenamiento de modelos. Pero a los precios actuales que tienen ambas plataformas, si se va a realizar un desarrollo intensivo, no son recomendables pues el coste puede terminar siendo más elevado que el de una GPU con capacidad suficiente, aunque si pueden ser plataformas interesantes para empezar si no disponemos de una GPU.

1.3. Redes neuronales convolucionales

Como ya se describió antes, las redes neuronales convolucionales son las mas importantes a día de hoy pues nos permiten hacer cosas que hace no tanto tiempo nos parecerían increíbles, como que un coche sea capaz de conducir por si solo gracias al análisis de imágenes en tiempo real, o que con una simple imagen podamos saber si nuestra vida corre peligro o

no. Estas tareas y más son posibles gracias al análisis de las imágenes que nos permiten realizar las redes neuronales convolucionales. Tensorflow y Keras nos permiten crearlas de manera muy sencilla, sin necesidad de tener conocimientos previos de como funcionan, ni la curva de dificultad que tiene aprender a programarlas desde cero, proporcionando así una plataforma en la cual somos capaces de generar varios modelos diferentes e ir intercambiándolos rápidamente para ver las diferentes métricas entre unos y otros, comparar cual es mejor e incluso trabajar con todos los modelos al mismo tiempo y así ver cual es el que mejor se comporta frente a los cambios que se realizan tanto en la tasa de aprendizaje como en otros parámetros, o incluso ver la mejora o el deterioro de las métricas según se realizan cambios en el conjunto de entrenamiento.

1.3.1. ¿Cómo funcionan las redes neuronales convolucionales?

Las redes convolucionales, o *Convolucional Neural Networks* (a partir de ahora CNNs) como ya hemos explicado antes utilizan un filtro (anteriormente mencionado como marco), que no es más que una matriz de píxeles de diferente medida para recorrer y analizar la imagen con la que esta esté trabajando en cada momento. Gracias a esta matriz, con las iteraciones logra extraer patrones en las imágenes, como por ejemplo, los ojos de un gato, sus orejas, su boca etc, para finalmente tener todos estos datos en cuenta a la hora de tomar la decisión de en qué categoría se clasifica la imagen. En nuestro caso la red neuronal va a hacer lo mismo pero identificando formas en las imágenes, gradientes de color etcétera para terminar identificando de que tipo de lesión se trata, además de esto, se iniciará un modelo con redes convolucionales también para ver si se trata de un caso de piel sana o no.

El padding

Una de las variables que podemos controlar en el análisis de las imágenes es el padding, particularmente en este tipo de red neuronal, variable

que nos va a a ajustar hasta dónde vamos a desplazar el filtro dentro de la imagen, pues estas estructuras de redes neuronales actúan como un escáner, recorriendo la imagen de arriba a abajo y de izquierda a derecha. El problema es que los pixeles de las esquinas solo se analizan una vez, pues el filtro solo llega a ellos una vez, con su propia esquina, esto hace que se pierda información, pues si el píxel central del filtro pasara sobre él se analizaría mas de una vez por convolución. Por tanto lo que se hace es añadir información que no afecta al análisis en los bordes de la imagen. Esto último es lo que se conoce como padding.

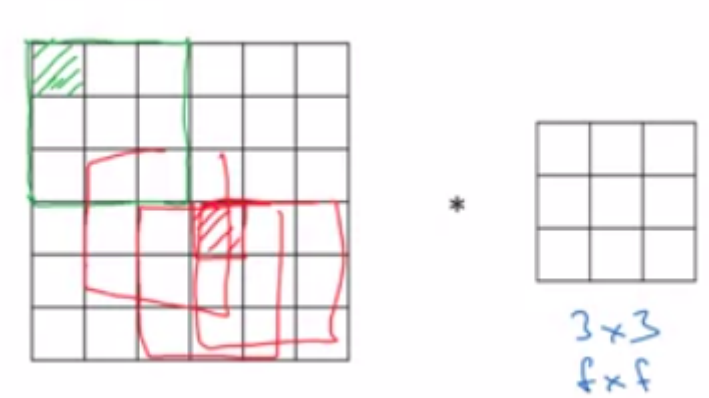


Figura 1.4: Ejemplo de red convolucional con filtro de 3x3

Por ejemplo, en la Figura 1.4 podemos observar lo que pasa cuando no hay padding perfectamente, como podemos ver, tenemos un filtro de 3x3 y, en la esquina superior derecha podemos ver que ese píxel solo se procesa una vez por convolución, sin embargo en el momento en que ponemos padding, el centro del filtro pasará por ese píxel, con lo cual no se analizará sólo una vez.

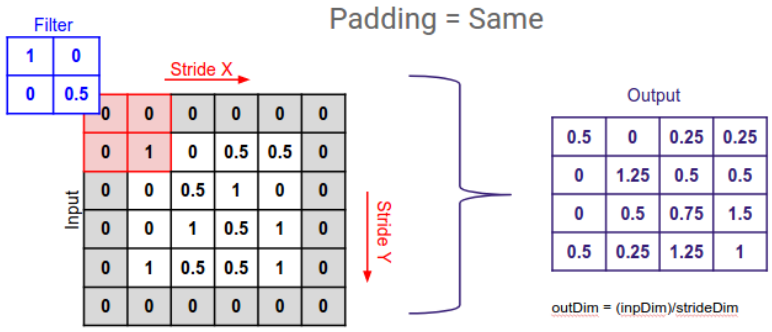


Figura 1.5: Ejemplo de la misma red pero esta vez con padding

Como se puede observar en la Figura 1.5 sucede justo lo que se ha explicado, el filtro analiza mas de una vez ese píxel, con lo cual no perdemos información.

El tamaño del filtro

Una vez que se sabe que existe una matriz de un tamaño determinado para analizar la imagen, la siguiente pregunta es: ¿Cómo afecta su tamaño al resultado final?. Bien, las diferencias entre un tamaño de filtro pequeño y un tamaño de filtro grande son las siguientes:

Tamaño de filtro pequeño	Tamaño de filtro grande
Cantidad de pesos menor, pero redes mas profundas.	Cantidad de pesos mayor, pero redes menos profundas.
Computacionalmente eficientes.	Computacionalmente ineficientes.
Con mas capas, aprendizaje de conceptos mas complejos.	Con menos capas, aprendizaje mas simple.
Necesita más memoria.	Necesita menos memoria.

Tabla 1.1: Tabla muy sencilla.

Como se ve en las diferencias, tiene más ventajas usar tamaños de filtros pequeños. Por ello siempre tenderemos hacia ellos. La siguiente pregunta que se nos plantea es: Si a menor tamaño mejor ¿Tiene sentido usar filtros de 1x1?. Pues la verdad es que no, pues esos filtros nos

pasan la imagen a la siguiente capa en el mismo estado en la que se la encuentra, sin extraer información de lo píxeles vecinos ni por tanto reduciendo la imagen a un tamaño menor.

Otra cuestión que es lógico plantearse es la posibilidad de usar filtros de dimensiones pares. El problema de estas dimensiones es que los filtros realizan una interpolación del resto de puntos de la matriz hacia el píxel central, por tanto si no hay píxel central no se puede hacer dicha interpolación, y estos tamaños de filtro no tienen mucho sentido por eso mismo. Por tanto, si no podemos usar ni 1x1 ni 2x2, la elección más común será usar un filtro de tamaño 3x3.

1.3.2. Hiperparámetros

Un hiperparámetro es un parámetro que puede ser ajustado antes del entrenamiento de un modelo. Es un término propio del Machine Learning (ML) y que no tiene definición en la Real Academia Española (RAE). vamos a comentar aquellos que se pueden modificar en las CNNs pues son aquellos que mas nos importan dado que este TFG está enfocado a ese tipo de redes.

Los hiperparámetros más comunes que podemos modificar a la hora de llevar a cabo el entrenamiento de una CNN son:

- Tasa de aprendizaje: Regula cuánto se pueden modificar los pesos en cada época
- Número de épocas: Número de veces que el conjunto de entrenamiento entero es procesado por la red neuronal.
- Batch size: Cantidad de imágenes que la red neuronal analiza a la vez.
- Función de activación: Es la función que decide cuándo se activa una neurona o no, introduce la no linearidad al modelo y permite que este sea mas complejo. Es la propiedad que confiere a las redes neuronales su potencia.

- Numero de capas ocultas y unidades: Son las capas que hay entre la capa de entrada y las de salida.
- Inicialización de pesos: Normalmente los pesos están iniciados con valores aleatorios, pero puede ser conveniente iniciarlos con los que ya tiene una red neuronal de clasificación de imágenes.
- Usar dropout: El dropout es una manera de evitar el sobreajuste u *overfitting*, y se basa en anular algunos valores de la red neuronal para que dejen de aprender a cierto nivel. Un valor de 0.5 es bastante recomendable.

Como podemos observar en esta lista, hay bastantes parámetros que podemos modificar a nuestro gusto, también hay muchas posibles permutaciones y pruebas que hacer para ver que es lo que mejor nos funciona. Pero este tema se tratara en el siguiente capitulo de la memoria, cuando hablemos de la implementación.

Sin embargo, sí que vamos a aclarar cuáles de ellos son los que tienen mayor importancia, o los que pueden cambiar el comportamiento del modelo drásticamente si cambiamos sus valores.

En primer lugar tenemos la tasa de aprendizaje o *Learning Rate*, que es el más crítico de todos. Este valor es muy sensible ya que una diferencia de 0.1 puede hacer que nuestro modelo funcione y trabaje correctamente o que sea totalmente impracticable. Por ello hay que ajustarlo muy bien, en el segundo capítulo de la memoria se describirán técnicas para que se escoja tal que el aprendizaje no sea muy lento pero estable.

En segundo lugar tenemos el número de épocas. Este valor también es clave, porque una época más puede hacer que nuestro modelo pase de algo bastante bueno, con buena capacidad de generalizar, a algo que no vale para nada, con una especificidad excesiva y un sobreajuste que hacen al modelo inservible.

El numero de capas es también bastante importante pero no tan crítico como los 2 anteriores. Un numero muy pequeño de capas puede hacer que nuestro modelo sea demasiado sencillo y que se centre demasiado

en los detalles, inutilizando su capacidad de generalizar. Sin embargo, un numero demasiado grande hará nuestro modelo 'computacionalmente imposible', lo escribo entre comillas porque si tuviéramos computador cuántico no tendríamos problemas, pero con una única tarjeta gráfica, y sin ser ésta de gama alta (El caso de este TFG), el entrenamiento del modelo puede resultar lento.

1.4. Overfitting

El *overfitting* o sobreajuste es un problema presente y a evitar en el entrenamiento de una red neuronal. Para no caer en el sobreajuste necesitamos una gran cantidad de datos y con mucha diversidad. Cuando la red neuronal empieza a ver los mismos datos una y otra vez, desarrolla asociaciones fijas y pierde capacidad de generalización.

En el caso de que se observe que ha aparecido este problema podría ayudar a evitarlo si se toman las siguientes medidas

- Aumentar la cantidad y variabilidad de los datos: Esta es la forma mas efectiva y más sencilla, a mayor cantidad y mayor variabilidad tengan los datos, mas difícil es que el modelo tenga overfitting pues cuantas más variaciones de los datos obtenga, mejor puede aprender las distintas características que hacen únicas a las clases que éste clasifica.

- Generar datos a partir de los que ya tenemos: Es una forma de aumentar la cantidad y la variabilidad, conocida como *data augmentation*. La idea es introducir modificaciones a las imágenes que sean grandes o pequeñas, para así enseñarle a la red, por ejemplo imágenes con menor calidad, más ruido, o que puedan tener pequeñas mutaciones. Así conseguimos generar más ejemplos tomando como base los que ya tenemos, esto tiene una ventaja y un inconveniente. La ventaja es obvia, más datos, pero el inconveniente es igual o más importante. Si las modificaciones son demasiado pe-

queñas corremos el riesgo de introducir overfitting en el sistema, y si las modificaciones son demasiado grandes estamos introduciendo falsos ejemplos de entrenamiento.

- Usar capas de Dropout: Si nuestro conjunto de entrenamiento está desbalanceado, cosa que es mas común de lo deseable tenemos la opción de usar el dropout .
- Usar arquitecturas adecuada: Puede que se hayan elegido pocas capas ocultas, con lo cual no se llega a aprender los rasgos mas generales o por el contrario, puede que el modelo elegido resulte demasiado complejo.

1.5. Tecnologías utilizadas para las aplicaciones de escritorio

En esta sección se van a presentar las tecnologías que se han utilizado para las apps tanto de escritorio como móvil asi como la arquitectura que se ha seguido para la comunicación entre ellas.

1.5.1. Arquitectura del sistema

Se ha decidido utilizar un paradigma cliente-servidor, donde la red neuronal va a estar alojada en un servidor, y las aplicaciones se comunicarán con este a través de una API RESTful, pero la implementación de la misma, así como los endpoints se discutirán en el segundo capítulo de esta memoria.

1.5.2. Aplicación de escritorio

Para la aplicación de escritorio se ha decidido realizar con Jinja, un lenguaje intermediario entre python y html. También se han añadido algunas animaciones con JavaScript, pero como se ha dicho antes, los detalles de esta implementación se describirán en el segundo capítulo de la memoria que va a tratar sobre la implementación del proyecto.

Se ha desarrollado una segunda versión de la aplicación de escritorio teniendo como base Angular y Typescript.

1.5.3. Aplicación móvil

Para la aplicación móvil se ha decidido optar por NativeScript, una tecnología que usa Typescript con Angular, aunque también es compatible con otros frameworks como pueden ser VUE, o con el lenguaje JavaScript.

Capítulo 2

Implementación del proyecto

Contenido

Sinopsis

En este capítulo se va a tratar todo el desarrollo, empezando por la librería escogida seguido de una explicación de los modelos utilizados, su estructura interna, requisitos, casos de uso, el conjunto de datos de entrenamiento y la elección final de las estructuras neuronales con las que se llevaría a cabo el proyecto, y se terminará con la descripción del sistema inteligente, la API REST y las aplicaciones cliente, dejando las pruebas de todo lo mencionado para el capítulo 3 de la memoria.

2.1. Selección del sistema inteligente

En esta sección vamos a tratar todo lo que concierne al desarrollo del sistema inteligente, desde las estructuras que se han planteado, hasta los modelos escogidos, entrenamiento de los mismos y un apartado breve de resultados, ya que la parte de pruebas del sistema en las que se incluirán los distintos resultados obtenidos es materia del capítulo 3 de la memoria.

2.1.1. Keras y Tensorflow

Estos 2 módulos de python son los que vamos a usar para construir nuestro sistema inteligente desde cero. Por suerte la documentación de Keras es bastante completa y será de bastante ayuda a lo largo del desarrollo. Por parte de TensorFlow (a partir de ahora TF), esta es la librería que Keras ejecuta, simplificando muchas de las dificultades de TF, y también perdiendo rendimiento respecto a programar usando TensorFlow, pero es mas importante esa simplificación que hace a la hora de crear, entrenar y probar modelos que la perdida de rendimiento que sufre TF a la hora de ejecutarse. Además de estos paquetes usaremos unos cuantos más, que se irán comentando según se vaya avanzando en esta sección.

2.1.2. Estudio de modelos

A la hora de construir un sistema inteligente es vital saber escoger el modelo correcto para el problema que se intenta resolver, en el caso de este TFG se ha hecho una investigación previa de cara a tener varias opciones entre las que elegir por lo que ya se comentó en el capitulo 1. Uno de los errores que podíamos cometer a la hora de tener sobreajuste era escoger un modelo que generalizara mal, por ello se ha hecho este estudio, para descartar los modelos que generalizan mal del espectro de los posibles a escoger, y una vez que se hayan seleccionado aquellos que tienen buenas posibilidades probarlos con todas las variaciones de

hiperparámetros posibles para ver qué modelo con qué configuración particular generaliza mejor. Antes de ver las distintas estructuras de redes neuronales, vamos a explicar un par de elementos que aparecen en algunas de ellas.

¿Qué es ResNet?

ResNet es una abreviatura de Residual Network. Como el nombre de la red indica, este tipo de redes introduce aprendizaje residual, pero ¿Que es el aprendizaje residual y cual es su utilidad?.

En general en las redes de aprendizaje profundo (o Deep Learning) varias capas de neuronas se acoplan y se entrenan en conjunto. La red aprende las distintas características de los datos. En el aprendizaje residual, en vez de intentar aprender nuevas características, lo que se intenta es aprender las características remanentes. Por remanente entendemos extracción de la característica aprendida a la entrada de esa capa. ResNet hace esto haciendo atajos (conectando directamente la salida de la capa n con la capa n+x). Se ha probado que esta forma de entrenamiento es más fácil que entrenar las redes neuronales profundas per se, y además se ha resuelto el problema de la degradación de la precisión de la red.

Las redes convolucionales han conducido a una serie de logros en cuanto a clasificación de imágenes, usando el aprendizaje residual. Otros problemas de visión por computador han sido resueltos por redes neuronales profundas. Entonces, con el paso de los años, ha habido una tendencia de crear redes más profundas para resolver problemas más complejos y para mejorar la precisión de la clasificación y reconocimiento. Pero a la vez que creamos estos modelos más profundos, el entrenamiento de los mismos se vuelve cada vez más complejo y la precisión se empieza a saturar y a degradar. El aprendizaje residual intenta solventar estos dos problemas.

¿Qué es Imagenet?

Imagenet es una base de datos organizada de acuerdo a la jerarquía de Wordnet, lo que se conoce en terminología de machine learning como un *dataset* o conjunto de datos en la que en cada nodo está repleto de cientos y miles de imágenes. Actualmente se tienen unas 500 imágenes por nodo (un nodo es una clase, tal como coche, bicicleta o camión).

2.1.3. Inception ResNetV2

Es una aplicación de Keras basada en la ResNet V2 de Google, y ya viene con pesos predefinidos basados en el entrenamiento con el dataset Imagenet.

Inception ResNet V2 es una red neuronal convolucional (CNN) que es lo último en términos de aciertos en el Benchmark de ILSVRC para clasificación de imágenes. La red Inception ResNet V2 es una variación de la arquitectura Inception V3 de google, que toma algunas ideas de las publicaciones científicas de Microsoft ResNet. Las conexiones residuales permiten hacer atajos en el modelo y esto ha permitido a los investigadores entrenar redes neuronales incluso más profundas, que han dado mejor rendimiento. Esto ha permitido una simplificación importante de los bloques del principio de la red.

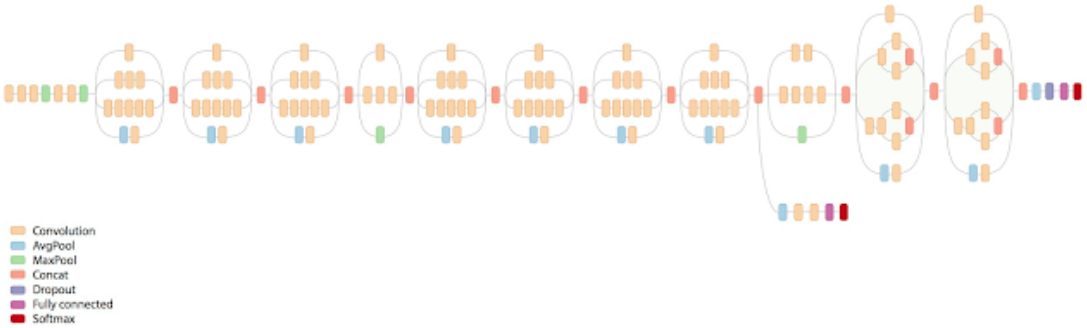


Figura 2.1: Estructura de la red Inception V3

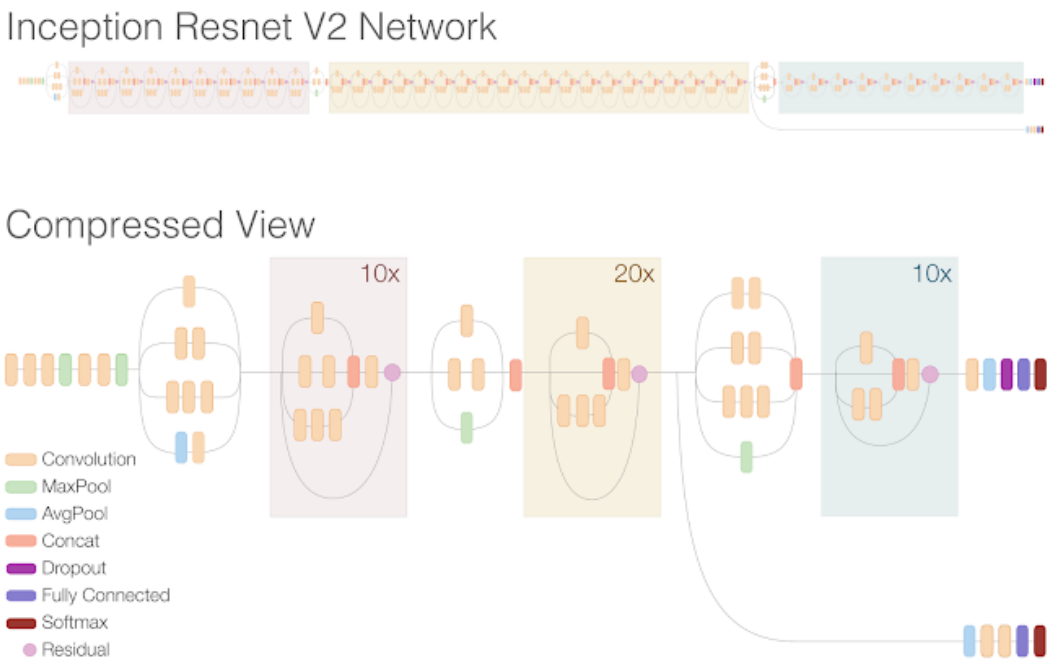


Figura 2.2: Estructura simplificada de la Inception ResNetV2

En las Figuras 2.1 y Figura 2.2 podemos ver las diferencias entre Inception ResNet V2, e Inception V3, estructura que veremos más tarde

2.1.4. AlexNet

Con la publicación *ImageNet Classification with Deep Convolutional Neural Networks* de Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton en 2017. Se demostró que con Alexnet se podían entrenar modelos profundos en GPUs. AlexNet usa el paralelismo para dividir la red entre varias GPUs, la arquitectura usa 4 grupos de convolutional activation, pooling layers, y 3 fully-connected layers de 4k, 4k y 1000 neuronas con un vector de salida con dimensión 1000.

Sin embargo, se usa una capa adicional llamada Batch Normalization Layer. La motivación detrás de ella es puramente estadística. Se ha demostrado que los datos normalizados (por ejemplo con media 0 y varianza 1) permite a las redes converger mucho más rápido. Por ello, queremos normalizar nuestro mini-batch pero tras aplicar una convolu-

ción los datos no tendrán media 0 y varianza 1. Para ello se aplica la normalización tras cada una de las convoluciones. La implementación de la normalización es bastante simple.

Con todo esto lo que se pretende decir es que se consiguió paralelizar toda la ejecución del entrenamiento pudiendo así aprovechar el mayor número de núcleos que ofrece una tarjeta grafica (o GPU) para agilizar los cálculos.

2.1.5. GoogleNet

Esta red ganó el ILSVRC (Imagenet Large Scale Visual Recognition Challenge) en 2014, este desafío evalúa los algoritmos para la detección de objetos y la clasificación de imágenes a gran escala. La novedad que introduce esta red son los *Inception Modules* o módulos de inicio. La red simplemente acopla estos inception modules juntos.

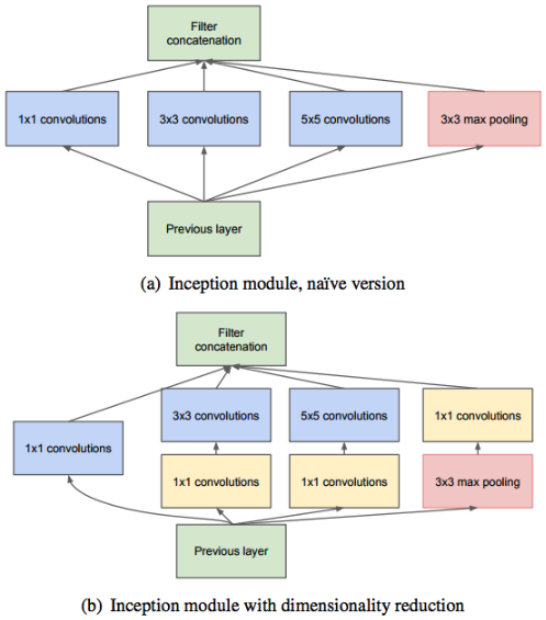


Figura 2.3: Modulo Inception de GoogleNet

La motivación detrás de estos inception modules era la problemática existente de seleccionar el tamaño del tamaño del escáner. Siempre

se suelen usar filtros pequeños; de 3x3, de 5x5 o de 7x7 pero ¿Cual de ellos funciona mejor? Esta red, en vez de escoger uno de ellos con las limitaciones que eso conlleva, usa todos ellos y concatena los resultados. Tomando los mapas de activación de las características de la convolución anterior, se aplican 3 convoluciones (una con cada tamaño de filtro) y después se concatenan los 3 mapas de activación.

El único problema con este planteamiento es la complejidad computacional. Un inception module es lento para entradas grandes.

Una vez explicado el concepto de inception module, vamos a ver cómo los usa la GoogleNet. Como se dijo antes, simplemente acoplamos muchos inception modules uno encima del otro para crear la GoogleNet (También hay unas cuantas capas antes de llegar a los bloques de inception modules y la siguiente imagen también muestra unos clasificadores auxiliares.

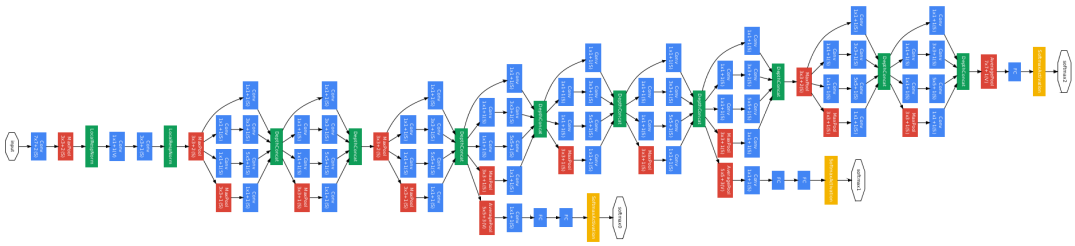


Figura 2.4: GoogleNet

Una vez que hemos explicado que estructuras neuronales hemos tenido en cuenta vamos a ver cuales de ellas han sido las seleccionadas para ser entrenadas.

2.2. Selección de estructuras para entrenar

Para la resolución de este problema en principio se escogieron 2 estructuras a fin de probarlas y ver cuál de ellas daba mejor resultado con las distintas variaciones del conjunto de entrenamiento. Las 2 escogidas

fueron Inception ResNet V2 pues tras estar investigando un tiempo y tras una serie de pruebas preliminares, se determinó que daba muy buenos resultados cuando se enfrentaba a problemas de análisis de imágenes y la segunda estructura seleccionada fue la Inception V3, pues es una estructura que desarrolla Google, que es la predecesora de la GoogleNet y tras ver que arrojaba resultados prometedores en el análisis preliminar. Se han descartado la GoogleNet por falta de potencia computacional y la AlexNet al ser una estructura más antigua y ver que no daba buenos resultados en un análisis preliminar.

2.3. Requisitos Funcionales

En esta sección y en las subsiguientes se van a tratar los aspectos referentes a la ingeniería de requisitos del proyecto; en este apartado, los requisitos funcionales.

- RF1: El usuario podrá leer las advertencias de la aplicación.
- RF2: El usuario podrá clasificar lesiones de la piel presentes en una imagen.

Es una lista de requisitos pequeña pero como ya se comentó en el anteproyecto, el grueso de este TFG era hacer funcionar el sistema inteligente de una manera razonable, pues el resto es poner a disposición del usuario el funcionamiento de ese sistema inteligente a través de las aplicaciones cliente. Todas las ejecuciones, con la matriz de confusión y distintos enfoques y configuraciones se podrán ver en el capítulo 3 de esta memoria.

2.4. Requisitos No Funcionales

- RNF1: El sistema operativo de la aplicación móvil será Android.
- RNF2: Los formatos de imagen soportados serán .jpg, .png, y .bmp.

- RNF3: Las imágenes enviadas para su análisis pueden ser de cualquier tamaño.
- RNF4: Se exigirá un nivel de precisión mínima, la lesión a analizar debe estar encuadrada, a ser posible en el centro de la imagen.
- RNF5: Las aplicaciones, tanto móvil como de escritorio requieren conexión a internet.
- RNF6: El procesamiento se hace fuera de la aplicación debido a un tema de eficiencia, pues la energía de la que disponemos en un dispositivo móvil es limitada y para ello es necesario el RNF5.
- RNF7: Para que las imágenes sean adecuadas para su análisis deberán tener una buena definición y un brillo adecuado, pues si estas características no están bien el análisis puede ser erróneo.
- RNF8: La resolución de la imagen enviada debe ser de al menos 224x224.
- RNF9: La versión de Android mínima para usar la aplicación será Android 5.1.

2.5. Casos de uso

2.5.1. El usuario podrá leer las advertencias

Descripción	El usuario podrá leer las advertencias.
Prioridad	Alta
Actores	1. Usuario 2. Sistema
Escenario principal	
1. El usuario pulsa sobre acceder. 2. El sistema carga la vista de las advertencias. 3. El usuario lee las advertencias.	
Escenario alternativo	
Postcondición	1. El usuario ha leído las advertencias

2.5.2. El usuario podrá clasificar lesiones de la piel presentes en una imagen.

Descripción	El usuario podrá clasificar lesiones de la piel presentes en una imagen.
Prioridad	Alta
Actores	1. Usuario 2. Sistema
Escenario principal	
1. El usuario pulsa sobre acceder. 2. El sistema carga la vista de las advertencias. 3. El usuario lee las advertencias. 4. El usuario pulsa sobre aceptar y analizar. 5. El sistema carga la vista de análisis. 6. El usuario selecciona una imagen para analizar. 7. El sistema analiza la imagen. 8. El sistema carga la vista de resultados y muestra el veredicto. 9. El usuario visualiza el resultado al análisis de la imagen enviada.	
Escenario alternativo	
1. El usuario no acepta las advertencias. 2. El usuario toma una foto en vez de seleccionarla de la galería, o del sistema de ficheros.	
Postcondición	1. El usuario ha analizado una imagen y ha obtenido un resultado

2.6. Diagramas de secuencia

Aquí se van a incluir los diagramas de secuencia para los dos sistemas.

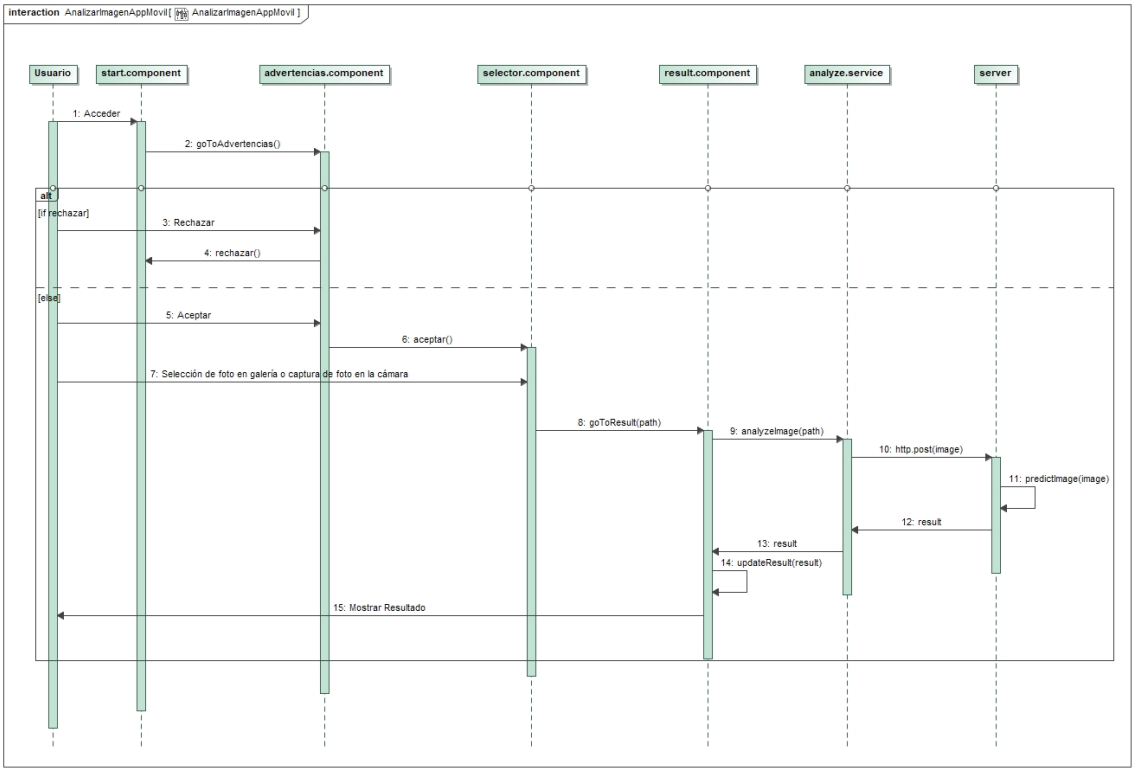


Figura 2.5: Diagrama de secuencia de la interacción entre el usuario y la aplicación móvil

En este diagrama lo que se observa es la interacción del usuario con el sistema en la aplicación móvil. Incluye una alternativa que es la opción de que el usuario no acepte las advertencias y un bucle pues el usuario puede analizar todas las imágenes que quiera.

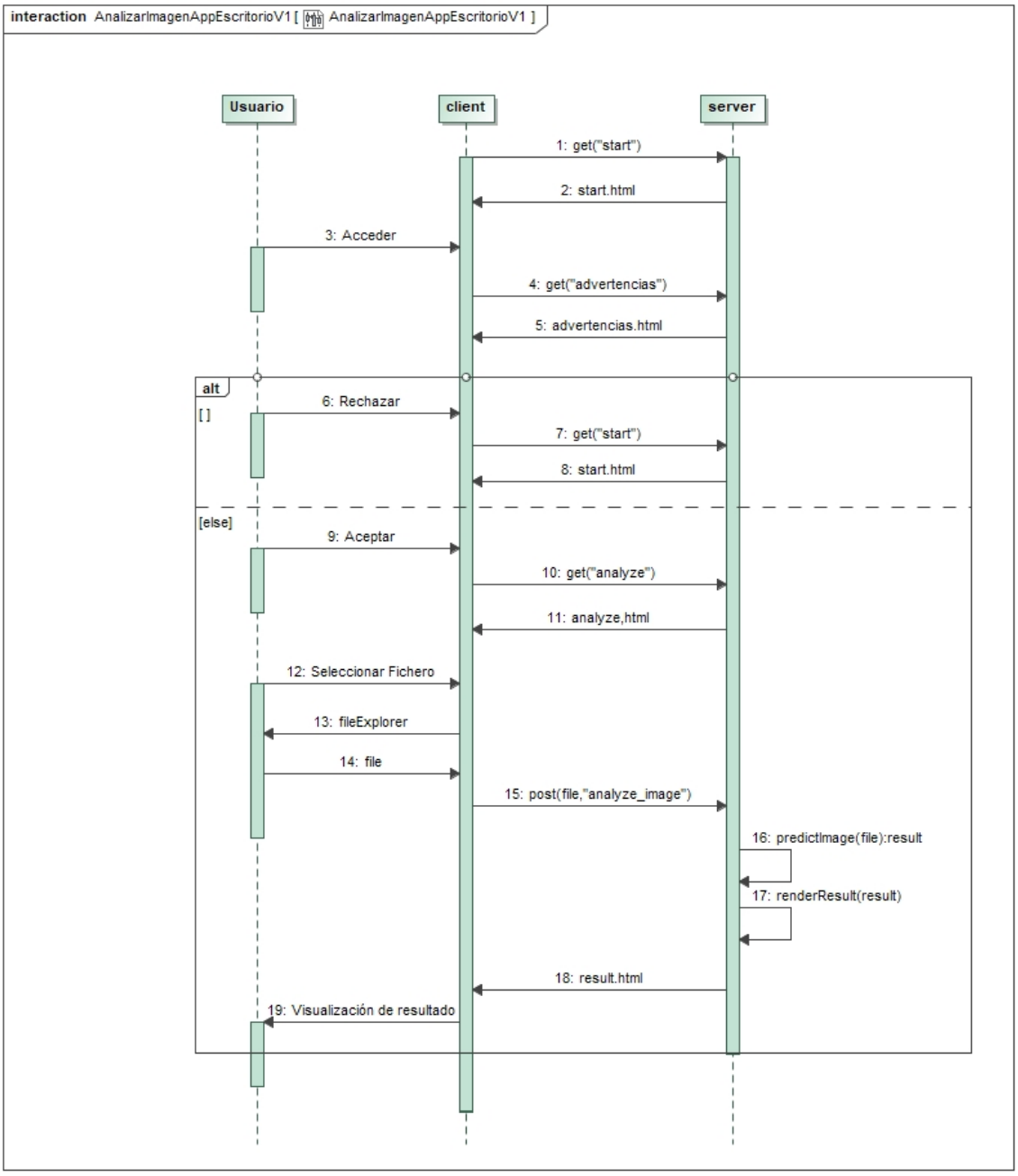


Figura 2.6: Diagrama de secuencia de la interacción entre el usuario y la primera versión de escritorio

En este diagrama lo que se observa es la interacción del usuario con el sistema en la aplicación de escritorio. Incluye una alternativa que es

la opción de que el usuario no acepte las advertencias.

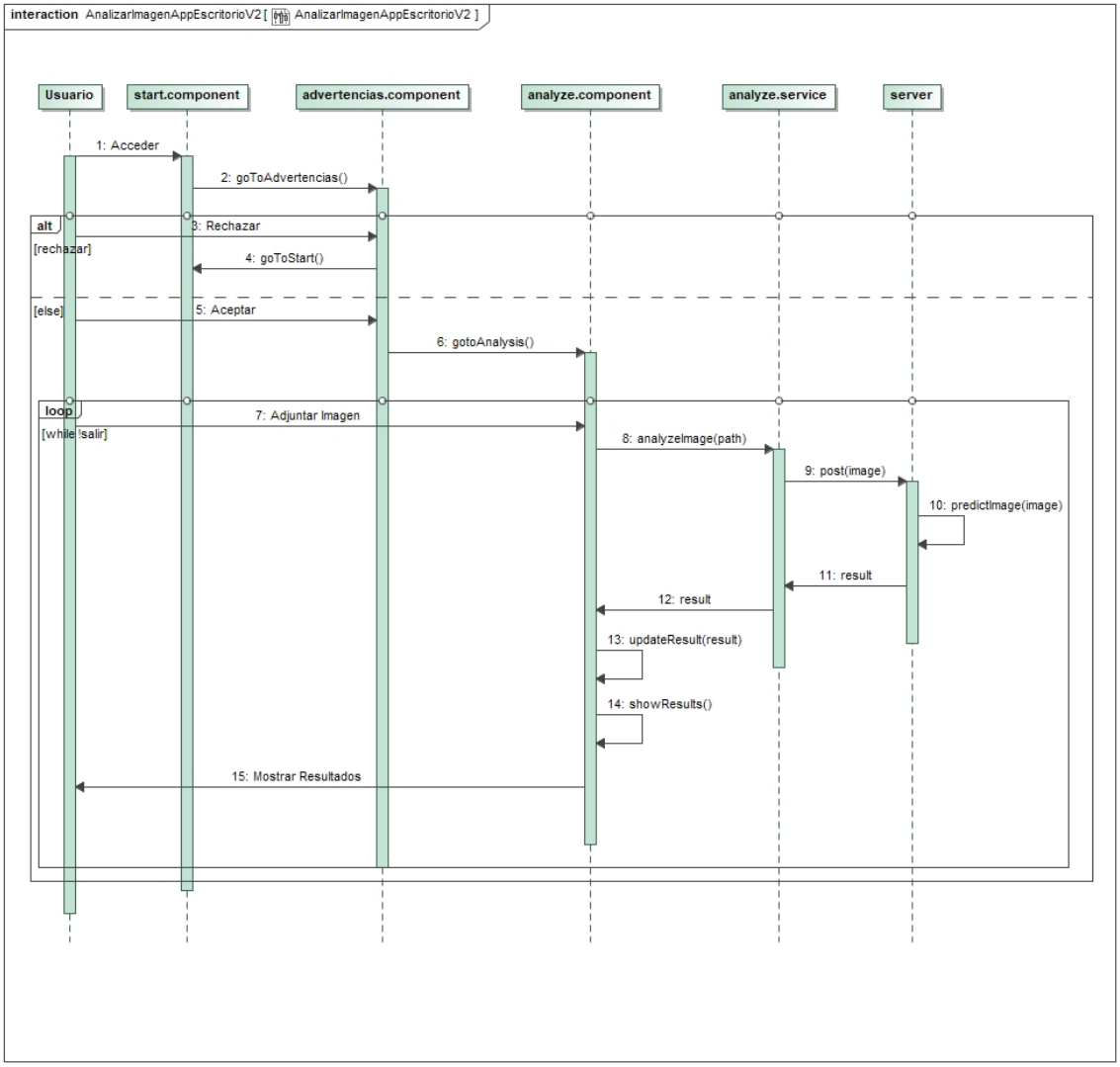


Figura 2.7: Diagrama de secuencia de la interacción entre el usuario y la primera versión de escritorio

En este diagrama lo que se observa es la interacción del usuario con el sistema en la aplicación de escritorio. Incluye una alternativa que es la opción de que el usuario no acepte las advertencias.

2.7. Introducción al desarrollo

En esta sección se describe cómo se ha llevado a cabo la implementación del proyecto: las tecnologías usadas, qué se ha hecho en las distintas iteraciones, directrices que se han llevado a cabo, problemas que han surgido y soluciones aportadas a dichos problemas.

2.7.1. La elección de tecnología y preparación previa de la misma.

Una vez realizado un amplio estudio de posibles técnicas de aplicación para el problema concreto de reconocimiento de imágenes que ocupa este TFG se decide usar Keras como API de Tensorflow para el desarrollo del proyecto. Esta decisión creó la necesidad de ampliar mis conocimientos sobre aprendizaje supervisado, redes convolucionales, y programación usando Keras.

2.7.2. Elección del conjunto de datos

Se ha elegido un conjunto de datos de libre disposición que publica la Universidad de Harvard, el Human Against Machine 10000, al cual se le ha aplicado algunas técnicas de preprocesamiento con el objetivo de mejorar su calidad.

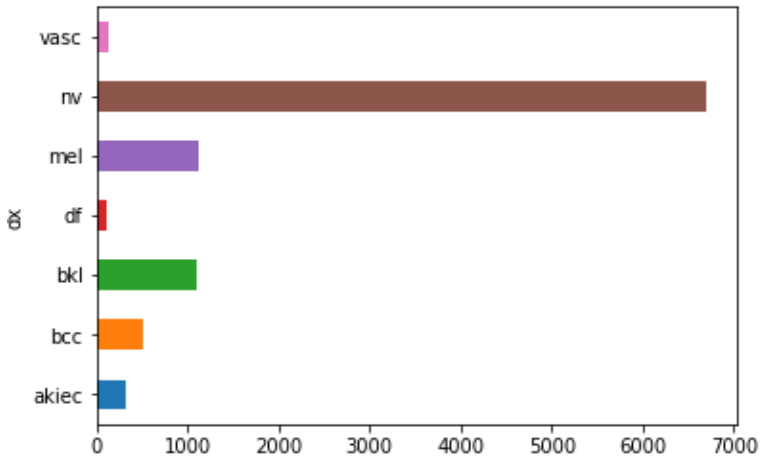


Figura 2.8: Distribución original de las imágenes entre las distintas clases de las que disponía el dataset

En primer lugar vamos a describir las clases que se ven en esa imagen, pues con las abreviaciones es complejo.

- vasc : Lesiones Vasculares, de cualquier tipo
- nv : Nevus o más conocidos como lunares
- mel : Melanomas
- df : Dermatofibromas
- bkl : Lesiones benignas parecidas a las keratosis
- bcc : Carcinoma basocelular
- akiec : Keratosis acníticas

Como se puede ver en la imagen el dataset está claramente desbalanceado a favor de los nevus, dejando al resto sin apenas representación. Está claro que esta característica conducirá a un sobreajuste desmedido (se va a incluir una ejecución con este paradigma en el capítulo 3) con lo cual se tuvo que pre procesar los datos.

Además de este dataset, que era el único que estaba disponible públicamente y sin restricciones, se realizaron peticiones a varios sitios que

tenían imágenes que podían ser utilizadas para la elaboración de este TFG. Esos sitios fueron Derm101, The Cancer Image Archive, SkinVision, Cancer Research UK y ISIC. Ninguno de ellos salvo ISIC permitió el acceso o el uso de sus imágenes de forma que no se ha podido balancear el dataset añadiendo imágenes nuevas y por tanto nos hemos limitados a los resultados que se puedan extraer de este dataset.

2.7.3. Idea inicial

La idea inicial del proyecto era la de crear un clasificador de lesiones cutáneas de carácter canceroso distinguiendo entre los tipos que se han visto en la figura 2.8. Para ello se desarrollaría un sistema inteligente, que trabajara con las imágenes de las que se disponían y las clasificara en sus respectivas clases. El primer planteamiento respecto a los datos, una vez que ya se tuvo toda la estructura de preprocesamiento, generación, entrenamiento y test, fue el de probar una red neuronal creada en base a los conocimientos de los que se disponían en aquel momento. Las pruebas preliminares confirmaron lo que a nivel teórico se anunciaba; hacer esto con este conjunto de datos es virtualmente imposible. La efectividad de la misma no subía del 10 % de aciertos. Dada esta situación, se decidió investigar en Kaggle cuáles eran unas buenas estructuras neuronales para clasificar imágenes, y se dió con parte de la solución que se ha aplicado finalmente. Keras tiene una serie de estructuras neuronales que puedes utilizar simplemente reverenciándolas, y cuentan con una serie de opciones bastante interesantes que comentaremos en la siguiente subsección.

2.7.4. Aplicaciones de Keras

Las aplicaciones son unas estructuras neuronales que nos proporciona Keras que ya están formadas y probadas, y de hecho una de las opciones a las que podemos optar es a utilizar los pesos del conjunto de datos Imagenet en el que las estructuras vienen preentrenadas. Vamos a ver las distintas aplicaciones que se han probado en este TFG, aunque en la

pagina de la documentación de Keras se pueden consultar todas las que están disponibles.

- Inception V3
- Inception ResNet V2
- VGG16

Las 2 primeras han tenido su explicación arriba, pero VGG16 se introdujo avanzado el proyecto como un mecanismo de encontrar errores, cosa que veremos cuando se hable de las etapas de desarrollo del proyecto y de los distintos problemas que han ido surgiendo a lo largo del desarrollo del proyecto, pues las 2 primeras redes estaban dando resultados sin mucho sentido y se introdujo esta última en búsqueda de errores.

2.7.5. Callbacks

Se describe aquí el concepto de callback ya introducido en el capitulo 1. Los callbacks son funciones que Keras nos proporciona para que se apliquen en cierto momento del proceso de aprendizaje, esto es en tiempo de ejecución. Éstos se pasan como un argumento al método 'fit' del modelo que creamos y se ejecutan cuando sea pertinente. Lo interesante de los callbacks es la variedad de cosas que se pueden hacer con ellos, así que vamos a ver los más importantes:

ModelCheckpoint

Esta es una función que nos permite guardar el modelo en un fichero .h5 una vez que se den las condiciones que se le pasan como argumento a este callback. Éste se ejecuta cada época pero no en todas las épocas se guarda el modelo. Como argumentos recibe una ruta al fichero donde queremos guardarlo, en qué parámetro se ha de fijar para saber si ha de guardarlo o no, un argumento para guardar sólo el modelo que de mejor resultado en la métrica pasada anteriormente, otro argumento para indicar si queremos guardar el modelo entero o solo los pesos, un modo de configuración y un periodo.

EarlyStopping

Early Stopping es un callback que permite interrumpir el entrenamiento cuando una determinada métrica deja de mejorar durante un tiempo. Como argumentos recibe la métrica, el período durante el que debe esperar hasta cortar el entrenamiento y un modo de configuración. El parámetro mas interesante se llama Restore Best Weights que nos permite que una vez que se corte el entrenamiento se recuperen los mejores pesos del modelo, antes de realizar los tests.

LearningRateScheduler

Permite programar el learning rate que se usará en cada época. Recibe una función que debe recibir el numero de época y el learning rate actual y devolver el nuevo.

ReduceLROnPlateau

Reduce el learning rate cuando una métrica deja de mejorar, esto se hace porque puede que el learning rate sea demasiado elevado y la red no pueda mejorar más con ese tamaño de paso. Al reducirlo, este ayuda a terminar de refinar el entrenamiento y a buscar el mínimo hacia el que va enfocado todo el entrenamiento de la red. Recibe como argumentos:

- La métrica a monitorizar
- Factor por el que será reducido el learning rate (éste no es necesario y si no se incluye la función lo ajusta automáticamente)
- Período que la métrica tiene que dejar de mejorar para que el learning rate se reduzca.
- Modo de configuración
- Learning rate mínimo que la función nunca excederá en su reducción.

Crear nuestros propios callbacks

Keras ofrece la capacidad de crear nuestros propios callbacks, lo cual es una opción a tener en cuenta aunque en este TFG no se han creado ninguno nuevo.

En este proyecto se ha estado trabajando con los callbacks ModelCheckpoint, ReduceLROnPlateau y EarlyStopping porque son los que se han considerado más útiles.

2.8. Desarrollo del proyecto

En esta sección se van a tratar las distintas fases del proyecto, los problemas que fueron surgiendo en cada una de ellas y las soluciones que se fueron adoptando hacia los mismos.

2.8.1. Fase 1: Investigación y puesta a punto

La primera fase se dedicó a investigar todo lo que se ha mencionado arriba y hacerlo funcionar. La fase 1 comienza sin tener nada instalado, y termina cuando se consigue hacer funcionar una red neuronal prototipo creada para ello. En esta primera fase es donde se dedica la mayor parte del tiempo a solucionar pequeños problemas mientras se construye el proyecto, como por ejemplo, tener la versión de python incorrecta, todos los problemas que da instalar TensorFlow para GPU, con CUDA y cuDNN, cuyas versiones no son las ultimas, entonces hay que desinstalar y volver a instalar otra vez... Al final son pequeñas pérdidas de tiempo pero hace que sea tedioso configurarlo todo perfectamente, pero una vez superados esos primeros problemas de no mucha relevancia y ya que tenemos configurado el proyecto, llega la hora de empezar a probar todo esto que se ha comentado anteriormente.

Tras descargar y descomprimir el conjunto de datos, nos encontramos con un archivo con extensión .csv, con la cual no se había trabajado antes, con lo cual más tiempo invertido en manejar estos ficheros que al final han resultado ser muy útiles, pues son los que nos permiten ma-

pear los ficheros de imágenes y el sistema inteligente. Éstos pueden ser modificados para reducir, aumentar, o cambiar las etiquetas que tienen los elementos allí presentes.

Los ficheros csv, responden a las siglas de Comma-Separated Values tienen la siguiente estructura. En la primera línea se incluyen, separados por comas, los nombres de las columnas que tiene el fichero. En las lineas sucesivas, cada línea es un registro compuesto por valores separados por comas, donde cada coma separa dos columnas. Algo que los hace diferentes y que me llamó la atención fue la capacidad de poder mezclar diferentes columnas en un mismo csv. Esto permite hacer cosas tan interesantes como mezclar distintos conjuntos de datos que tengas almacenados en distintas carpetas, y que sólo tengas que preocuparte de que las columnas que vas a usar tengan valores.

```
40.0,,bkl,consensus,ISIC_0032655,HAM_0003807,unknown,D:\Data_TFG\Images\ISIC_0032655.jpg,male,benign
40.0,,bkl,consensus,ISIC_0033620,HAM_0003807,unknown,D:\Data_TFG\Images\ISIC_0033620.jpg,male,benign
40.0,,bkl,consensus,ISIC_0034040,HAM_0003808,unknown,D:\Data_TFG\Images\ISIC_0034040.jpg,male,benign
70.0,,mel,histo,ISIC_0032258,HAM_0003521,back,D:\Data_TFG\Images\ISIC_0032258.jpg,female,malignant
,melanoma,mel,,,,,D:\Data_TFG\Images\images_new\Images\ISIC_0000002.jpeg,,malignant
,melanoma,mel,,,,,D:\Data_TFG\Images\images_new\Images\ISIC_0000004.jpeg,,malignant
,melanoma,mel,,,,,D:\Data_TFG\Images\images_new\Images\ISIC_0000013.jpeg,,malignant
,melanoma,mel,,,,,D:\Data_TFG\Images\images_new\Images\ISIC_0000022.jpeg,,malignant
,melanoma,mel,,,,,D:\Data_TFG\Images\images_new\Images\ISIC_0000026.jpeg,,malignant
```

Figura 2.9: Extracto del csv preparado para el entrenamiento

Como se puede observar en la figura 2.9 se pueden ver 2 tipos de lineas, unas que tienen más datos que otras, pero esto no impide la compatibilidad de las mismas pues solo interesan la tercera, a la antepenúltima y la última; correspondientes al diagnóstico codificado, a la ruta de la imagen dentro del sistema y a la clasificación trivaluada asignada (benigno, premaligno y maligno).

Sin embargo, la figura 2.9 corresponde a una versión del fichero csv original modificada, añadiendo valores nuevos y eliminando algunos otros.

```
lesion_id,image_id,dx,dx_type,age,sex,localization
HAM_0000118,ISIC_0027419,bkl,histo,80.0,male,scalp
HAM_0000118,ISIC_0025030,bkl,histo,80.0,male,scalp
HAM_0002730,ISIC_0026769,bkl,histo,80.0,male,scalp
HAM_0002730,ISIC_0025661,bkl,histo,80.0,male,scalp
HAM_0001466,ISIC_0031633,bkl,histo,75.0,male,ear
HAM_0001466,ISIC_0027850,bkl,histo,75.0,male,ear
HAM_0002761,ISIC_0029176,bkl,histo,60.0,male,face
HAM_0002761,ISIC_0029068,bkl,histo,60.0,male,face
HAM_0005132,ISIC_0025837,bkl,histo,70.0,female,back
HAM_0005132,ISIC_0025209,bkl,histo,70.0,female,back
HAM_0001396,ISIC_0025276,bkl,histo,55.0,female,trunk
HAM_0004234,ISIC_0029396,bkl,histo,85.0,female,chest
HAM_0004234,ISIC_0025984,bkl,histo,85.0,female,chest
HAM_0001949,ISIC_0025767,bkl,histo,70.0,male,trunk
HAM_0001949,ISIC_0032417,bkl,histo,70.0,male,trunk
HAM_0007207,ISIC_0031326,bkl,histo,65.0,male,back
HAM_0001601,ISIC_0025915,bkl,histo,75.0,male,upper extremity
HAM_0001601,ISIC_0031029,bkl,histo,75.0,male,upper extremity
HAM_0007571,ISIC_0029836,bkl,histo,70.0,male,chest
HAM_0007571,ISIC_0032129,bkl,histo,70.0,male,chest
HAM_0006071,ISIC_0032343,bkl,histo,70.0,female,face
HAM_0003301,ISIC_0025033,bkl,histo,60.0,male,back
HAM_0003301,ISIC_0027310,bkl,histo,60.0,male,back
HAM_0004884,ISIC_0032128,bkl,histo,75.0,male,upper extremity
```

Figura 2.10: Fragmento del csv original que venía con el dataset

Como podemos ver en la figura 2.10 se puede observar que es mucho mas simple, y de hecho se ha tenido que modificar para adaptarlo al problema, cosa que como se ha comentado antes, también ha sido muy útil para su uso en las practicas y a la hora de realizar modificaciones en el conjunto de datos sin que esto implique eliminar las imágenes ni hacer modificaciones sobre la estructura de los ficheros.

La librería usada para este propósito de manejar los ficheros con extensión .csv ha sido pandas, que nos permite de una manera sencilla leerlos, mostrarlos y modificarlos. Gracias a ella se terminaron de solventar los problemas, como que a los ficheros les faltaba la extensión y la tarea de importar los datos y prepararlos estaba terminada.

Tras esto lo único que se tuvo que hacer fue crear una red neuronal con unas cuantas capas y ejecutarlo. Ya se disponía de la primera red con capacidad de entrenamiento.

2.8.2. Fase 2: Presentación de datos al sistema

El esquema de trabajo usado en el desarrollo e implementación del modelo es el siguiente: se comenzó con una red neuronal sencilla y se fui incrementando y complicando conforme se fue necesitando. La idea bajo esta forma de trabajo es quedarnos con el modelo más simple posible lo cual supone un ahorro considerable de parámetros con necesidad de ajuste y por tanto, un aprovechamiento máximo de la información proporcionada por nuestra base de datos

Tras la creación de la primera red neuronal entrenable en la fase uno, se fue refinando la arquitectura y parámetros pero, dado el tiempo de investigación que esto suponía decidí usar las aplicaciones de Keras. Esta API incluye estructuras ya creadas e incluso entrenadas en el conjunto de datos imagenet. Una vez cargada esa estructura mucho mas compleja que la red inicial que yo tenía (véase figura 2.2) apareció un problema de memoria al se cargase en memoria todas las imágenes en resolución original, sin redimensionarlas y todas en una estructura única (en un único tensor). Para resolver tal inconveniente se usaron los generadores, unas estructuras muy interesantes y eficientes en memoria, en vez de cargar todo el conjunto de datos en memoria se crea un tensor único y se van sustituyendo sus componentes, ahorrando así muchísima memoria y haciendo el software mas utilizable por usuarios que no dispongan de un hardware tan avanzado.

En cuanto a métricas de la estructura, Keras nos da 4 por cada época: precisión en el entrenamiento, pérdida en el entrenamiento, precisión y pérdidas en validación. Las métricas que a nosotros realmente nos interesan son las de los datos de validación pues son los datos que la red neuronal ve al final de cada época a modo de test parcial. También se tienen unas imágenes que la red neuronal no usa nunca, hasta que se completa el entrenamiento para ver cómo generaliza ésta al final de la etapa de entrenamiento. Los datos de test son realmente importantes pues ahí es donde se puede empezar a observar la calidad de la generalización que está realizando la red.

2.8.3. Fase 3: Desarrollo de la primera versión de la aplicación de escritorio

Tras realizar unas cuantas ejecuciones y pruebas ajustando las épocas y mejorando significativamente las métricas que los distintos modelos proporcionaban se decidió empezar a realizar una primera versión sencilla de la aplicación de escritorio de cara a realizar pruebas de manera más fácil desde el punto de vista del desarrollo. Esta aplicación tiene como objetivo proporcionar un punto de acceso al sistema inteligente para el usuario.

Para la primera versión de la aplicación de escritorio se hizo una estructura sencilla basada en Flask y Jinja2, un microframework para desarrollo de APIs REST y un motor de plantillas para python, respectivamente. Se desarrollan los end-points usados para comunicar la aplicación de escritorio con el sistema inteligente. Tras esto se escoge un framework de CSS para darle estilo a la aplicación.

En cuanto a organización se decidió que la aplicación partiera de una pagina inicial, con un logo y un botón, la página de advertencias que el usuario debe leer y aceptar si quiere analizar una imagen. En la siguiente vista se tiene un botón para selección de fichero y un botón de enviar. Por ultimo una pagina de resultados donde se nos da la imagen que hemos enviado, la que realmente se ha analizado (ya que para su evaluación tendrá que pasar por un proceso de redimensionado) y por ultimo el veredicto. Se hizo bastante simple, en una primera versión, para poder probar el modelo con efectividad y simpleza.

Es en esta fase del proyecto donde se decide cambiar el enfoque a la hora de asesorar al cliente, debido a problemas de ajuste como se describe en la siguiente fase

En el momento en el que se empezó a probar el modelo, quedó patente que había sobreajuste por todos los lados, cosa, que mirando con

perspectiva era lógico que pasara, con lo cual tenia una tarea pendiente que era ver como solventar este problema.

Se intentó a través de 3 enfoques: El primero fue hacer un submuestreo de las ocurrencias de los nevus, pero no dio resultado, tenia muy pocas imágenes de algunos de los tipos que se querían clasificar.

El segundo enfoque fue hacer lo contrario, con técnicas de data augmentation, hacer un sobremuestreo de las imágenes que se tenían pero no solo no dio resultado sino que además empeoró el sobreajuste ya latente en el modelo.

Por último se intentó modificar la estructura de la red neuronal añadiendo capas de dropout porque es una de las formas que recomendaban en internet para reducir este problema, pero no dio resultado.

Tras ver que con esa cantidad tan pequeña de datos era imposible hacer nada se optó por hacer un cambio de paradigma

2.8.4. Fase 4: Cambio de enfoque y aplicación móvil

En los análisis preliminares realizados en la Fase 3 se puso de manifiesto que la base da datos no nos proporcionaba la información necesaria para resolver el problema inicialmente propuesto. Antes de decidirnos por el cambio se trabajó en 3 enfoques: El primero fue hacer un submuestreo de las ocurrencias de los nevus, pero no dio resultado, tenia muy pocas imágenes de algunos de los tipos que se querían clasificar. El segundo enfoque fue hacer lo contrario, con técnicas de data augmentation, hacer un sobremuestreo de las imágenes que se tenían pero no solo no dio resultado sino que además empeoró el sobreajuste ya latente en el modelo.Por último se intentó modificar la estructura de la red neuronal añadiendo capas de dropout pero no dio resultado. Es por esto que se decidió enfocar el proyecto hacia otra vía. Se decidió realizar una clasificación de las lesiones cutáneas en tres posibles clases: benigna, pre-maligna y maligna y para ello se utilizó una base de datos que contenía cerca de 25000 imágenes (ISIC).

Los datos finales que recibiría el sistema inteligente quedaron final-

mente de esta forma:

```
tipo
benign          2510
malignant       2510
premalignant    2510
```

Figura 2.11: Número de muestras de cada tipo utilizadas para entrenamiento, validación y pruebas

2.8.5. Fase 5: Pruebas y reajuste

Una vez que se consiguió que la red neuronal funcionara de una manera estable, se empezaron a cambiar los hiperparámetros para optimizarla, sin llegar a tener sobreajuste. Todas las trazas de ejecución y las configuraciones con estos hiperparámetros se discutirán en el capítulo 3

2.9. Desarrollo de la API REST

En este proyecto, el modelo del sistema inteligente es puesto a servicio del usuario mediante dos aplicaciones, una de escritorio y otra móvil, pero la comunicación entre estos dos sistemas es importante, por ello se ha decidido usar una API REST para la comunicación entre cliente y servidor.

Las razones por las que se ha escogido este paradigma son varias y variadas, la primera es el conocimiento de esta tecnología, pues en la asignatura de cuarto Ingeniería web, se vieron y programaron APIs REST en python con Flask. La segunda es la sencillez de desarrollo con este microframework que es Flask. Y la tercera es la facilidad de uso pues se hace todo mediante HTTP.

2.9.1. Flask

Flask es un microframework de python basado en Wekzeug, que es una librería de aplicaciones web. Incluye entre otras cosas un servidor

propio para depuración, manejo de solicitudes REST, plantillas con Jinja2... . A nosotros lo que nos interesa realmente es el manejo de solicitudes REST,

2.9.2. ¿Que es REST?

REST es un acrónimo inglés que responde a REpresentational State Transfer es un estilo de arquitectura software para sistemas hipermedia distribuidos como la World Wide Web. El término se originó en el año 2000, en una tesis doctoral sobre la web escrita por Roy Fielding. Si bien REST se refería a un conjunto de principios de arquitectura, en la actualidad se usa para denominar toda aquella comunicación sobre HTTP entre un cliente y un servidor.

REST tiene las siguientes características:

- Un protocolo cliente/servidor sin estado: cada mensaje HTTP contiene toda la información necesaria para comprender la petición. Como resultado, ni el cliente ni el servidor necesitan recordar ningún estado de las comunicaciones entre mensajes. Sin embargo, en la práctica, muchas aplicaciones basadas en HTTP utilizan cookies y otros mecanismos para mantener el estado de la sesión.

- Un conjunto de operaciones bien definidas que se aplican a todos los recursos de información: HTTP en sí define un conjunto pequeño de operaciones, las más importantes son POST, GET, PUT y DELETE. Con frecuencia estas operaciones se equiparan a las operaciones CRUD (Create, Retrieve, Update y Delete) en bases de datos.

- Una sintaxis universal para identificar los recursos. En un sistema REST, cada recurso es direccionable únicamente a través de su URI.

- El uso de hipermedios, tanto para la información de la aplicación como para las transiciones de estado de la aplicación: la representación de este estado en un sistema REST son típicamente HTML

o XML. Como resultado de esto, es posible navegar de un recurso REST a muchos otros, simplemente siguiendo enlaces sin requerir el uso de registros u otra infraestructura adicional.

2.10. Desarrollo de las aplicaciones de escritorio

En esta sección se entrará mas en detalle en las distintas versiones de la aplicación de escritorio, veremos la interfaz y cómo está estructurada, además de algunos diagramas de actividad para ver como es la navegación dentro de la misma. Aquí se van a ver las 2 implementaciones que se han realizado de la aplicación de escritorio, la primera en la tecnología Jinja2 y la segunda en Angular que se realizó más tarde tras implementar la aplicación móvil en NativeScript y Angular.

2.10.1. Primera versión con Jinja2

Jinja 2 es un motor de plantillas para python parecido a lo que es jsp con java, permite hibridar código python con codigo web, (html, css y js), esto permite hacer cosas tan interesantes como pasar una lista de elementos y mostrarlos en la web a través de iteraciones y generar páginas web diferentes con el mismo código dependiendo sólo de los argumentos que se reciban.

En esta primera versión con Jinja2 lo que se buscaba era la funcionalidad y no tanto la estética de la misma, por tanto es mas sencilla que la que después se terminó desarrollando con Angular

2.10.2. Diagrama de actividad de la primera versión

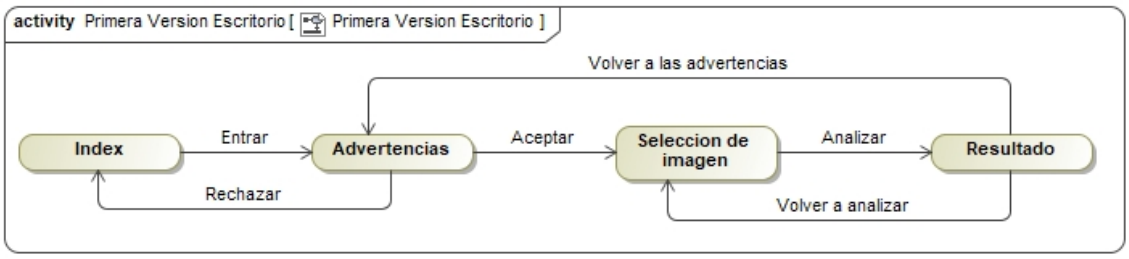


Figura 2.12: Diagrama de actividad de la primera versión de la aplicación de escritorio

2.10.3. Estructura de la navegación

Como hemos visto en el diagrama de navegación, la aplicación se compone de sendas vistas. En primer lugar tenemos la vista inicial, en la que se muestra el logo de la aplicación, que es el símbolo que todos asociamos con el cáncer, que es un lazo y cuyo color es el negro que es el correspondiente al melanoma, que es el representativo del cáncer de piel. Además del logo, lo que nos encontramos son un resumen de las advertencias y un botón grande para acceder a la aplicación.

Figura 2.13: Pantalla inicial

Nada mas acceder lo que nos encontramos son las advertencias legales, que están escritas en esta primera versión tanto en español como en ingles, pues lo que se buscaba era la funcionalidad y era ciertamente complicado la internacionalización de la aplicación web. Junto a esas advertencias nos encontramos un botón de aceptar y acceder, mediante el cual estamos dando por entendido que el usuario ha leído y aceptado las advertencias.

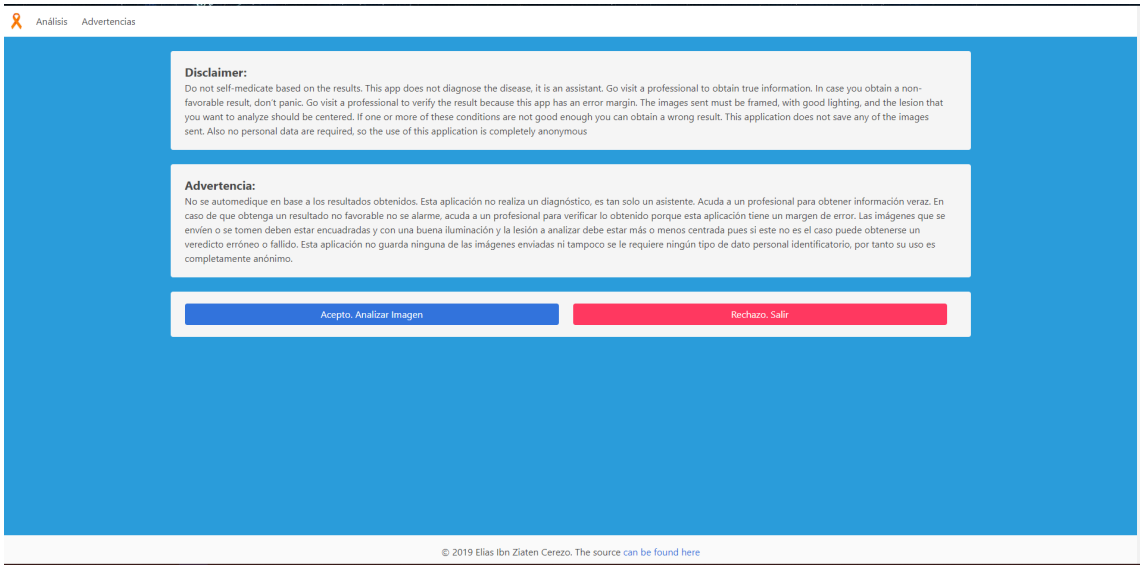


Figura 2.14: Captura de la pantalla de advertencias

Tras aceptar las advertencias accedemos a una página en la cual hay un formulario sencillo con un botón que nos invita a seleccionar un archivo, y un botón de enviar.

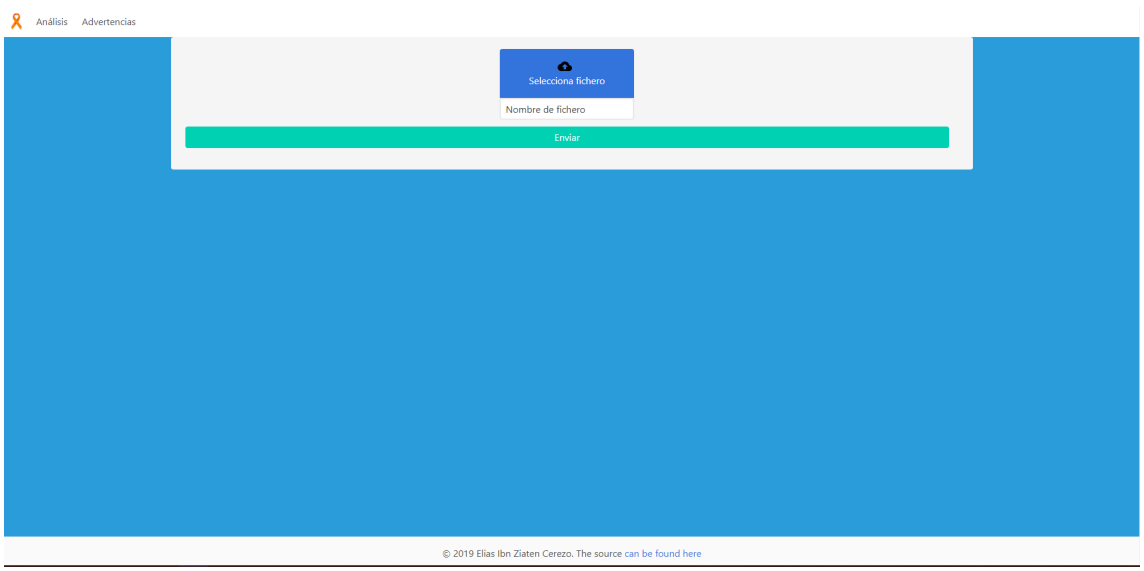


Figura 2.15: Captura del formulario del envío

Cuando hemos seleccionado el archivo y seleccionamos enviar, se nos

pone en espera, mediante la muestra de una animación, dándonos a entender que se está procesando la imagen. Esta animación no tiene por qué ser necesaria una vez que el servidor este desplegado, pues los análisis son instantáneos una vez que el modelo está cargado, pero en caso de que el modelo no esté cargado, el primer usuario que quiera hacer un análisis tendrá que esperar a que éste se cargue, y ahí si es útil la animación para que el usuario no piense que la aplicación está bloqueada.

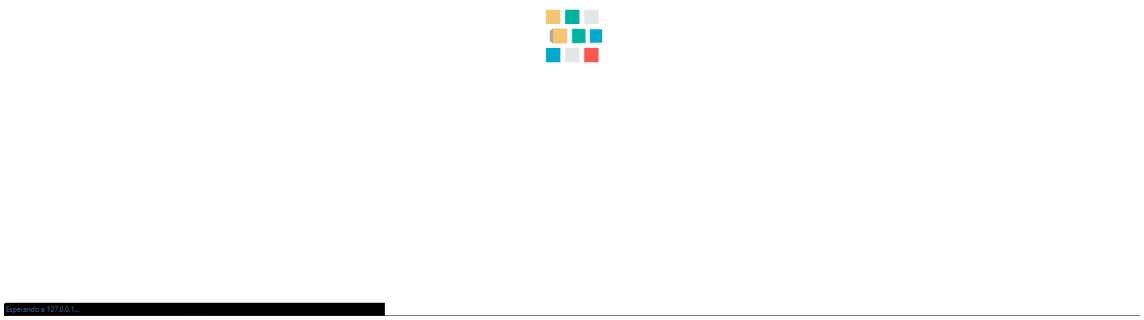


Figura 2.16: Captura de la animación

La pagina a la que somos redirigidos cuando pulsamos enviar y esperamos o no a que la animación termine, es una página sencilla, tiene 2 lugares para imágenes y uno para texto, en los lugares para imágenes, lo que se muestra es la imagen original que se ha enviado, y la que se ha analizado, pues debido a la falta de potencia de computación, no se ha podido trabajar con las imágenes en su resolución nativa y por tanto se muestra la imagen realmente analizada para que el usuario vea que es lo que realmente ha sido procesado y sobre lo que se ha juzgado.

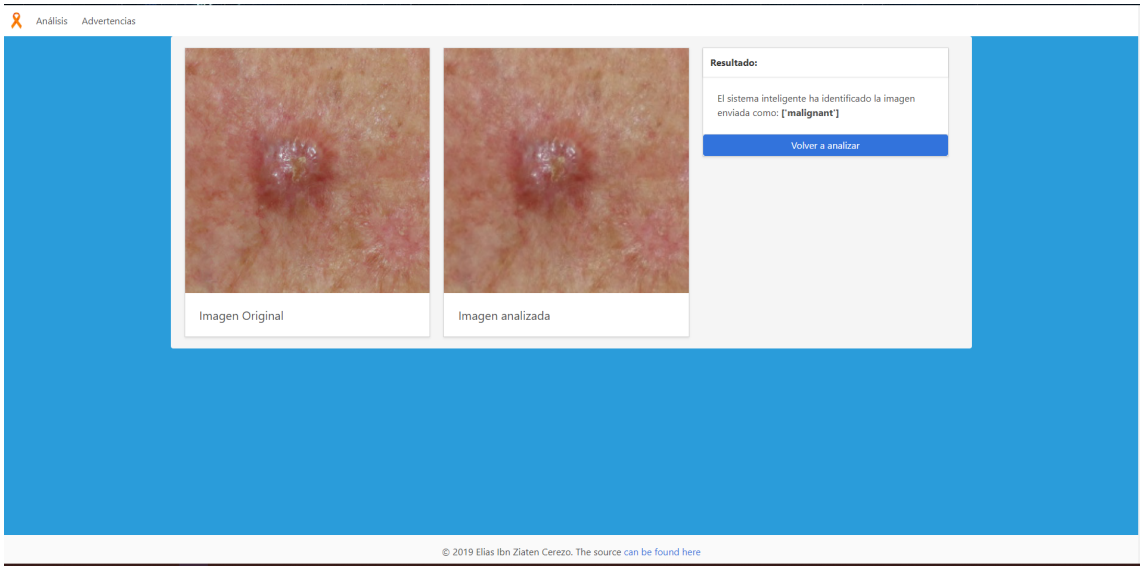


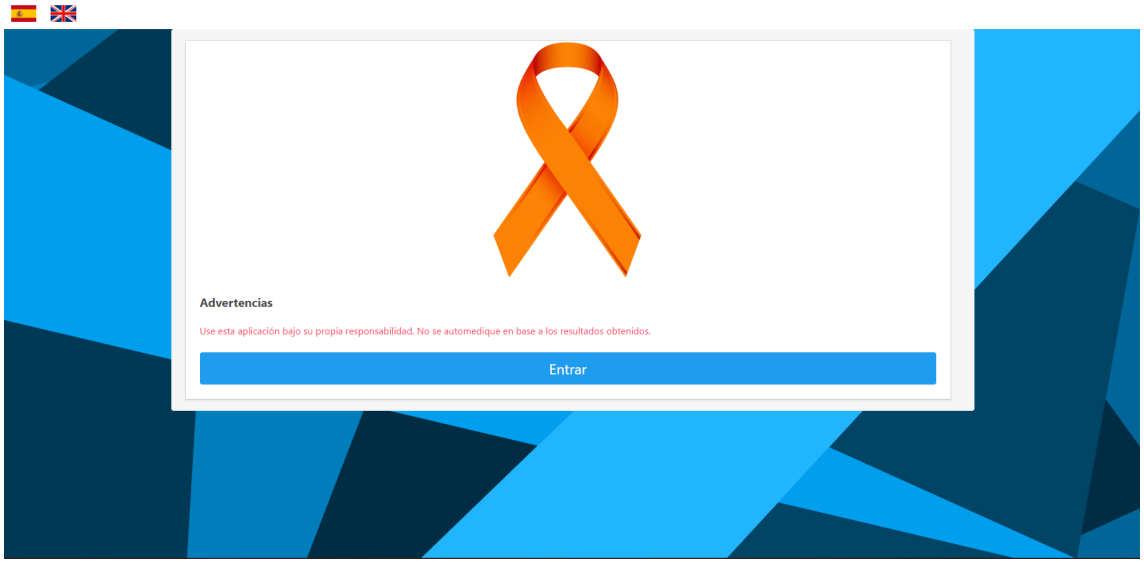
Figura 2.17: Captura de la pantalla de resultados

2.10.4. Veredictos

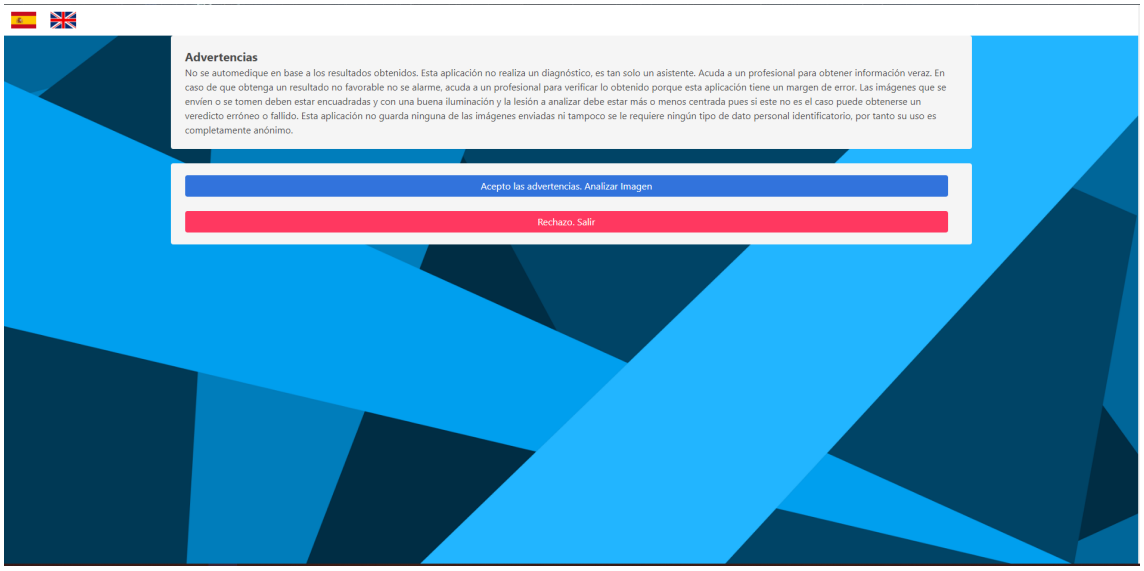
Todas las predicciones hechas por cualquiera de las aplicaciones, están sujetas a un modelo neuronal concreto. Éste se discutirá en el capitulo 3 de la memoria, pues ahí es donde se están comentando las pruebas para ver qué modelo es mejor que otro, y donde se realizan todas las evaluaciones pertinentes. Este tema se retomara en las conclusiones y direcciones futuras.

2.10.5. Segunda versión con Angular.

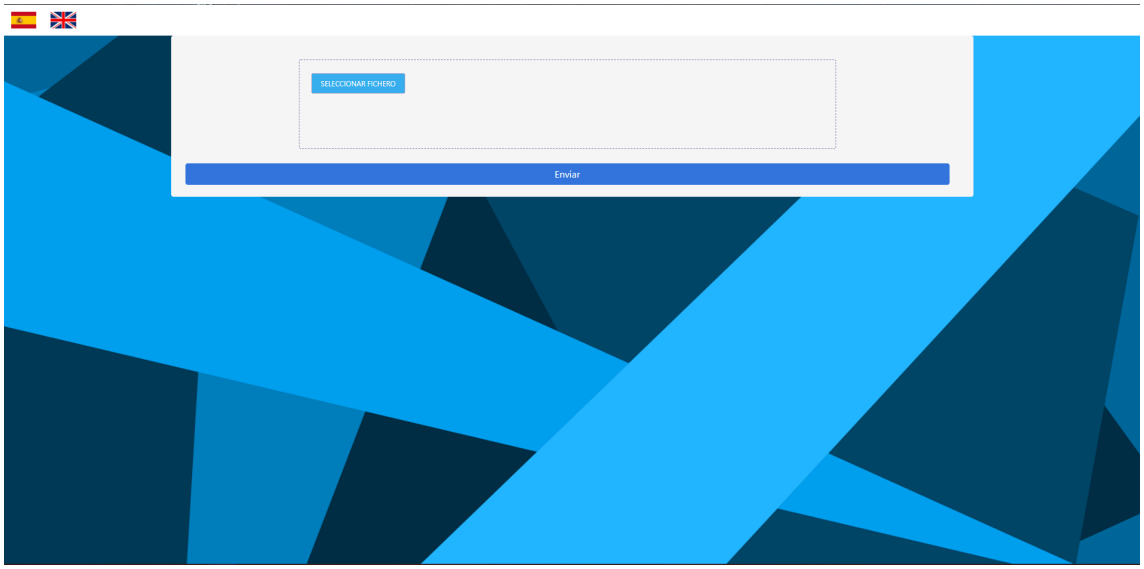
Se ha desarrollado una nueva aplicación de escritorio basada en Angular, framework que se menciona en la sección del desarrollo de la aplicación móvil pues esta segunda versión se realiza a raíz de implementar la aplicación móvil, y ver que tenia tiempo suficiente para desarrollarla. Los cambios principales en la aplicación es el hecho de añadir el inglés como idioma a la misma y mejorar la pantalla del formulario incrustando la pantalla de resultados en la misma.



En esta primera pagina muy similar a la de la versión con Jinja lo que se tiene son unas advertencias simples y un botón grande para entrar



En la captura de advertencias se tienen las mismas advertencias que en la primera versión con Jinja pero con la particularidad de la inclusión de la internacionalización.



Esta vista es la que ha sufrido mas cambios pues ahora es única para enviar las imágenes y visualizar los resultados. Esto ha sido posible gracias a como está estrucutrado Angular y a la actualización a través de JavaScript sin necesidad de recargar la página

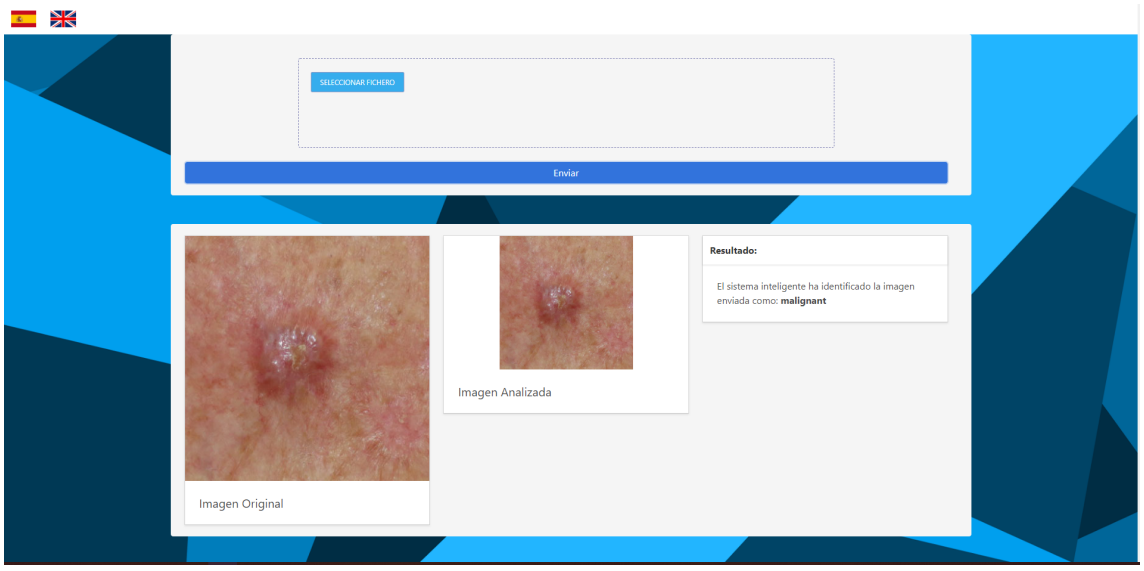


Figura 2.18: Captura del formulario con resultados

Como se puede ver, se ha usado el mismo framework de css para

elaborar el estilo de las vistas. El hecho de que Angular tenga como lenguaje base JavaScript, es mas sencillo elaborar cosas en la misma pagina pudiendo ocultar y mostrar elementos sin cambiar de página, y el soporte tan sencillo del que dispone para internacionalización.

Para la internacionalización simplemente hay que instalar un paquete, crear un par de ficheros .json y poner las traducciones. Una vez incluidas las traducciones simplemente hay que referenciar en el fichero .html el mensaje concreto.

2.11. Desarrollo de la aplicación móvil

En esta sección se tratará con más detalle el desarrollo de la aplicación móvil, hablaremos sobre el framework NativeScript, sobre Angular, componentes y como se construyen las distintas vistas. Además incluiremos capturas y diagramas de navegación.

2.11.1. NativeScript

NativeScript nos permite construir aplicaciones nativas para iOS, Android y la web a partir de un único código JavaScript. Con soporte para TypeScript, CSS y los frameworks de desarrollo mas comunes, como Angular y Vue.js este framework era bastante interesante para esto.

2.11.2. NativeScript y Angular

Angular es un framework bastante conocido para hacer aplicaciones web, su estructura en componentes y servicios no es muy difícil de aprender para realizar aplicaciones que se comuniquen con un servidor, aunque su curva de aprendizaje es muy pronunciada al principio. Tras realizar un par de tutoriales que venían en la documentación de la web se comenzó el desarrollo, y no se tuvieron muchos problemas para elaborar las distintas vistas. Para la parte de la comunicación con la API RESTful hubo que usar algunos paquetes nuevos, pero tras ver un par de ejemplos no hubo mayor problema al desarrollar la aplicación.

2.11.3. Construcción de vistas y componentes de Angular

Las vistas en Angular se generan como componentes, al generar un componente se nos genera una carpeta nueva en la que hay cuatro ficheros, un .js, un .css, un .html y un .ts.

En el fichero .html se aloja la vista y NativeScript (a partir de ahora NS) nos proporciona una buena variedad de componentes con los que podemos construir nuestra vista. Todos los componentes se encuentran en el recurso bibliográfico [NSC19]

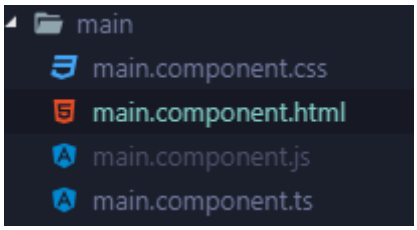


Figura 2.19: Visión de un componente de Angular

El fichero .css se usa para el estilo de esa vista en concreto, aunque también hay un .css para toda la aplicación, en caso de que haya aspectos estilísticos que mantener en la aplicación en conjunto como lo son el color y la imagen de fondo por poner un ejemplo.

El fichero .js es para incluir código JavaScript que se ejecutará en la vista, como por ejemplo animaciones únicas de cada vista o cualquier otro código JavaScript.

Por último en el .ts se incluye la lógica de esa vista, como las distintas redirecciones, cómo se comportan los botones etcétera, todo esto escrito en TypeScript.

Angular dispone de un sistema de etiquetas para enlazar un componente con los ficheros que lo componen.

```
@Component({
  selector: 'ns-main',
  templateUrl: './main.component.html',
  styleUrls: ['./main.component.css'],
  moduleId: module.id,
})
```

Figura 2.20: Estructura que enlaza un componente a sus ficheros correspondientes

Para pasar de una vista a otra tenemos que crear una función para asignarla a un componente de la vista que activará la función y la ejecutará.

La traza de vistas se mantiene automáticamente, con lo cual no debemos preocuparnos de a qué vista vamos cuando el usuario pulsa el botón atrás en el teléfono. Con lo cual si vamos retrocediendo en la navegación llegaremos a la vista inicial, y si pulsamos atrás cerraremos la aplicación y nos devolverá al lugar donde estuviéramos antes de iniciar su ejecución dentro del control del SO.

2.11.4. Diagrama de Navegación

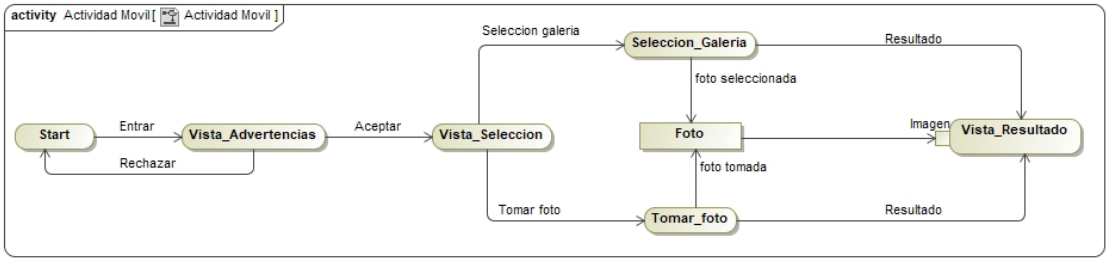


Figura 2.21: Diagrama de navegación de la aplicación móvil

2.11.5. Navegación por la aplicación

En la primera vista lo único que se nos presenta es el logo de la aplicación que es el símbolo clásico del cáncer, es decir un lazo de color naranja en indicación del cáncer de piel.

Junto a este logo se encuentra un botón para entrar, aunque si pulsamos en la imagen también pasamos a la siguiente actividad.

Figura 2.22: Captura de la primera actividad

En la siguiente actividad lo que nos encontramos son las advertencias, que están tanto en español como en ingles, pues esta aplicación está internacionalizada. Es decir, si tenemos el teléfono en español, los textos, los botones etcétera nos saldrán en español. y si tenemos el teléfono configurado en inglés, pues los textos, los botones etcétera estarán todos en inglés. Además de las advertencias nos encontramos dos botones, uno de aceptar y otro de rechazar. Si la persona pulsa sobre ese botón se le lleva a la siguiente actividad, en caso de que pulse el de rechazar se le llevará a la actividad anterior.

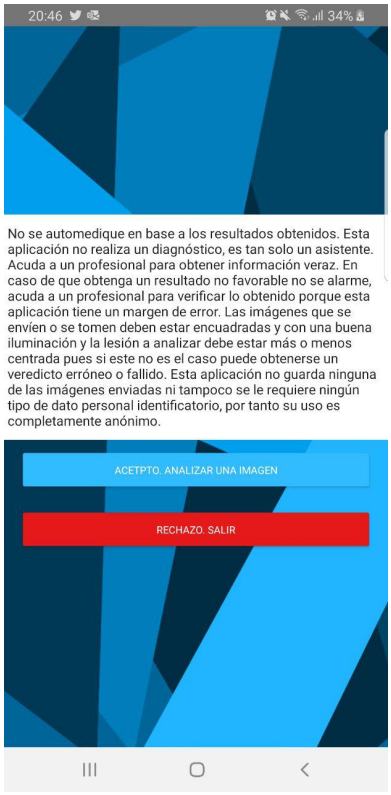


Figura 2.23: Captura de la actividad de advertencias

En la siguiente actividad nos encontramos dos imágenes pulsables junto a dos botones. Cada imagen está asociada a un botón, y cada grupo de estos señalan la galería y la cámara. En otras palabras, es un menú de selección para dar al usuario la opción de escoger entre galería y cámara.

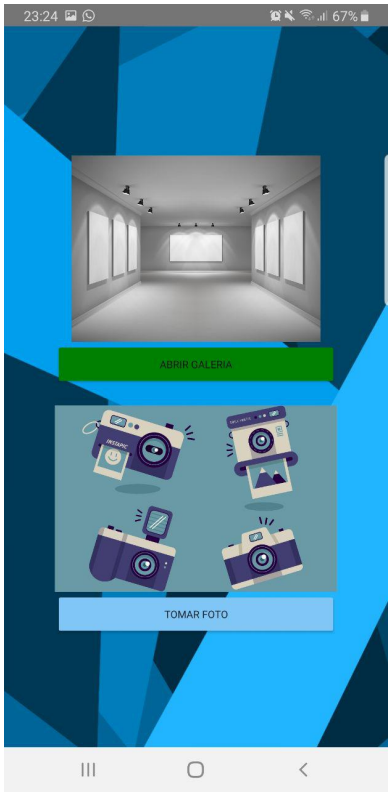


Figura 2.24: Captura del menú de selección

Si el usuario escoge la galería se le abre un menú de selección de su galería para que éste seleccione una foto a enviar para el servidor

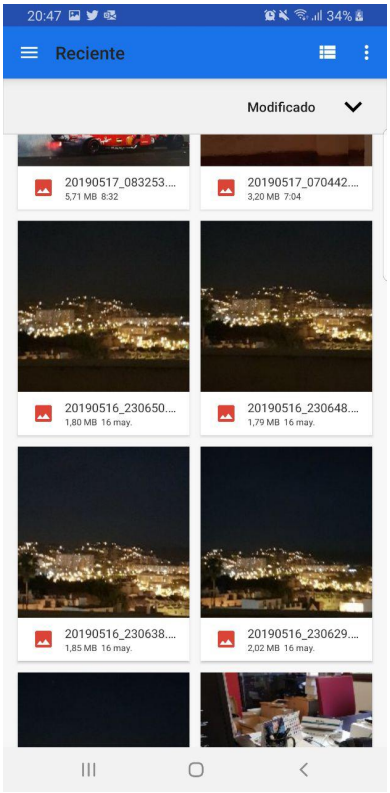


Figura 2.25: Captura de la opción galería

En caso de que el usuario escoja la cámara, se abrirá la cámara trasera del teléfono para tomar una foto y enviarla al servidor. La actividad que nos permite tomar la foto es la aplicación de la cámara del teléfono, por tanto la captura incluida en la memoria no será la misma para todos los teléfonos.

Figura 2.26: Captura de la cámara una vez se ha tomado la foto (puede variar de teléfono a teléfono)

Usemos una u otra opción al final ambas nos reconducirán hasta la actividad de los resultados, donde vamos a ver la foto que hemos enviado, la que hemos analizado y el resultado

Figura 2.27: Captura de la actividad de resultado

En este caso se ha optado por poner un indicador de actividad en vez de una animación propiamente dicha para que el usuario no crea que la aplicación se ha quedado bloqueada y que no va a recibir ningún resultado mientras se carga el modelo.

Además de estar en castellano, también se ha realizado la internacionalización añadiendo el inglés como idioma. La decisión de la aplicación de que idioma escoger dependerá del idioma en el que tengamos configurado nuestro teléfono pues si tenemos el español, nos mostrará la aplicación en español. Sin embargo, si tenemos el teléfono en cualquier otro idioma que no es el español, se nos mostrará en inglés.

Las dos aplicaciones tienen que comunicarse con un servidor, en caso de que ese servidor no esté en linea, en ambas nos va a salir una alerta, de que el envió al servidor ha sido erróneo y que por tanto no vamos a poder recibir respuesta del mismo.

2.11.6. Tensorflow.js

Gracias a que la aplicación móvil se ha desarrollado con NativeScript y Angular, y que ambos frameworks utilizan JavaScript, se investigó la posibilidad de ejecutar el modelo dentro del mismo smartphone. Esto es posible gracias a Tensorflow.js

Tensorflow.js es una versión de TensorFlow para javascript como su nombre indica. Lo único que tenemos que hacer para usarlo en smartphones es instalarlo como si un paquete de npm se tratara. npm es el gestor de paquetes de javascript, como pip lo es a python. Una vez instalado solo hay que hacer una llamada para comenzar la ejecución y tener un modelo precompilado para usar. Como Keras se ejecuta sobre TensorFlow existe una función que te convierte un modelo Keras a un modelo preparado para TensorFlow.js.

Tras convertir el modelo, instalar TensorFlow.js en el smartphone y ejecutar la aplicación para ejecutar in situ, me di cuenta de que era poco viable. En primer lugar se tardaba mucho en realizar la clasificación. En segundo lugar se gastaba bastante batería en el proceso y el smartphone se calentaba bastante, por tanto esta opción se descartó, pero si en un futuro se mejoran los procesadores de los teléfonos, la duración de las baterías o se optimiza esta versión de TensorFlow.js no se descarta ejecutar todo en el smartphone y así eliminar el requisito de la necesidad de conexión haciendo la aplicación más portable y usable en lugares en los que no hay internet o cobertura para enviarlo a través de la conexión de datos del smartphone

Capítulo 3

Pruebas y optimización

Contenido

Sinopsis

En este capitulo se va a hablar sobre las pruebas, tanto las realizadas al sistema inteligente, hablando de la optimización de hiperparámetros, incluyendo resultados de ejecuciones, y finalmente hablando sobre las pruebas realizadas a las aplicaciones cliente, tanto la aplicación móvil como la aplicación de escritorio.

3.1. Pruebas de redes neuronales

En este último capítulo de la memoria, vamos a ver pruebas de caja negra, y pruebas de sistema, que son las más interesantes y las más importantes. Además de esto vamos a tratar lo que de verdad define a una red neuronal y son las métricas y las distintas configuraciones de parámetros que nos van a diferenciar unos resultados de otros, todo esto hablando siempre de la red neuronal.

Por otra parte tenemos las aplicaciones de escritorio y móviles. Éstas si van a tener sus casos de prueba particulares, y si que se verán unitarias, de integración, funcionales y de sistema, pues estas aplicaciones se prestan a ello, pues no son tan complejas y tienen una estructura bien definida, no como las redes neuronales.

3.2. Pruebas Funciónales o pruebas de caja negra

Para realizar las pruebas de caja negra, necesitamos un modelo ya entrenado y una partición del conjunto de datos en 3 partes (datos de test, entrenamiento y validación).

Las pruebas de caja negra siempre se realizan al final de cada entrenamiento, así se ha programado, y esto es así para facilitar el visualizado de el factor de mejora del modelo, pues tras estas pruebas y viendo los resultados se puede discernir si son mejores o peores que otras ejecuciones con otra configuración de hiperparámetros. Todo esto de cara a tener una forma concreta y fija de evaluar el modelo, pues si hiciéramos pruebas diferentes para cada modelo, y por diferentes se entiende con distinto numero de imágenes, o con un conjunto de datos diferente para cada una de las distintas ejecuciones, no tendríamos resultados consistentes, y por tanto todos ellos sólo tendrían valor por si mismos y no en el conjunto total de pruebas. Por esto se ha decidido usar las pruebas de caja negra al final del entrenamiento para así hacernos una idea de dónde se coloca esa configuración de hiperparámetros concreta respecto a las demás.

Para mostrar los resultados se usará la matriz de confusión, la cual permite la visualización del desempeño de nuestra red supervisada. Por tanto vamos a mostrar ahora los resultados obtenidos en cada una de las ejecuciones que se han realizado una vez incluida en el programa la función que muestra la matriz de confusión. Además de estas imágenes y sus tablas de hiperparámetros se incluirán todas las trazas de ejecución que se tienen guardadas de entrenamientos anteriores a la inclusión de la función que muestra la matriz de confusión

Antes de comentar los resultados de las distintas ejecuciones, es importante resumir cada uno de los hiperparámetros:

- Numero de épocas: Numero de iteraciones que la red neuronal realiza sobre el número de imágenes definido por el batch size y el numero de pasos por época
- Batch Size: Numero de elementos, en este caso imágenes que se le pasan a red neuronal en cada paso de la época
- Numero de pasos por época: es el número de veces que se llena al generador del batch size para que cargue con imágenes nuevas el tensor que le pasamos a la red neuronal. Por tanto el numero de imágenes que se le pasan a la red neuronal por época es igual a : Numero de pasos por época * Batch size
- Learning rate inicial: Es la tasa de aprendizaje con la que empieza la red neuronal, después se irá modificando en función del callback ReduceLROnPlateau, del que ya se habló en el capítulo 2
- Dropout: Es una capa que hay antes de las capas que conforman el clasificador y que ayuda a mejorar el sobreajuste. Tiene un valor entre 0 y 1 y ese valor es el porcentaje de elementos que componen la salida de la capa anterior que va a poner a 0 aleatoriamente eliminando parte de la información que está contenida en la salida de esa capa.

- Numero de neuronas en la capa de inicio del clasificador: Numero de neuronas de entrada de la capa inicial del clasificador.
- Imagenet: Este parámetro indicará si las capas que preceden al clasificador han sido entrenadas en imagenet o no.

En las tablas que preceden a la matriz de confusión se van a mostrar los distintos hiperparámetros que han sido modificados para esa determinada ejecución.

3.3. Pruebas realizadas tomando como modelo base InceptionResNetV2

Como modelo base inicial hemos tomado la aplicación InceptionResNetV2 disponible en Keras y que tiene la siguiente estructura:

3.3.1. Primera ejecucion

Hiperparámetro	Valor
Modelo base	InceptionResNetV2
Número de neuronas en la capa de inicio del clasificador	256
Número de épocas	40
Número de pasos por época	235
Learning Rate inicial	0.01
Dropout	0.5
Batch size	32
Imagenet	Sí
Optimizador	Adamax
Función de pérdida	Categorical Crossentropy

Tabla 3.1: Hiperparámetros de modelo inicial

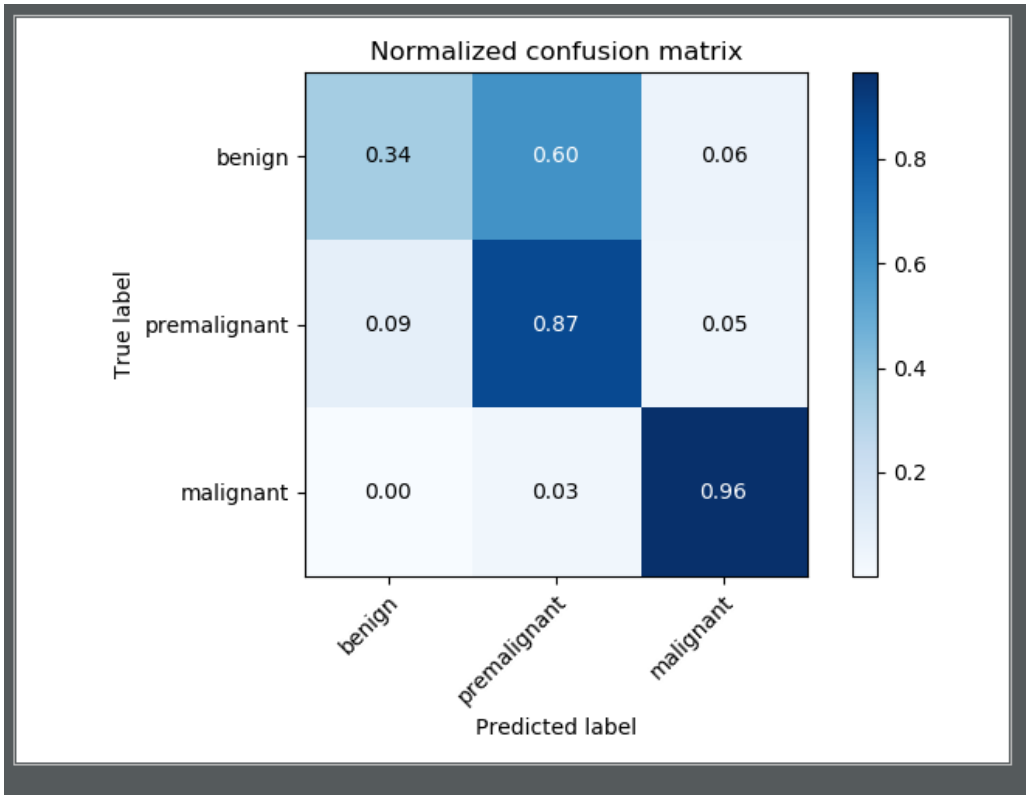


Figura 3.1: Matriz de confusión del modelo inicial

Como se puede observar en esta primera ejecución se tienen problemas para diferenciar las etiquetas premalignas y benignas, muy probablemente por iterar con casi todas las imágenes por toda la red neuronal reitaradas veces, por ello, para la segunda ejecución se cambiaron los hiperparámetros.

3.3.2. Segunda Ejecución

En esta segunda ejecución veremos como un cambio a los hiperparámetros hacen una gran diferencia, pues a partir de este cambio, no veremos ningún otro tan significativo como éste.

Hiperparámetro	Valor
Modelo base	InceptionResNetV2
Número de neuronas en la capa de inicio del clasificador	256
Número de épocas	80
Número de pasos por época	20
Learning Rate inicial	0.01
Dropout	0.5
Batch size	32
Imagenet	Sí
Optimizador	Adamax
Función de pérdida	Categorical Crossentropy

Tabla 3.2: Hiperparámetros de la segunda ejecución

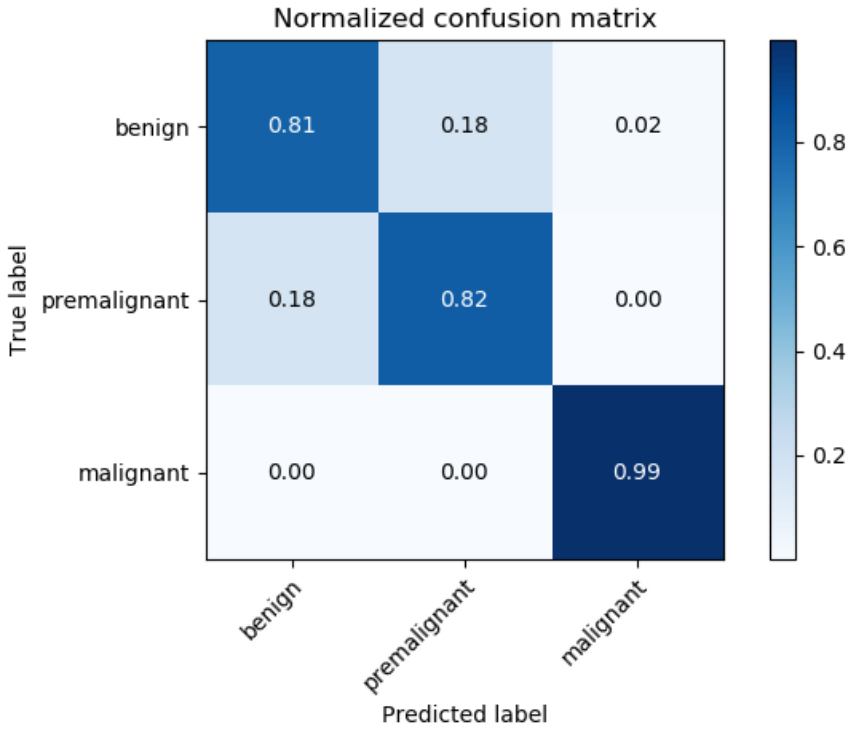


Figura 3.2: Matriz de confusión de la segunda ejecución

Como en este modelo ya se tenían unos resultados bastante aceptables, a partir de aquí comienzan las distintas optimizaciones de cara a

seguir mejorando el modelo, pero no vamos a tener mejoras tan sustanciales como de la primera ejecución a la segunda.

3.3.3. Tercera Ejecución

En esta ejecución ya se han empezado a jugar con los demás parámetros para probar el efecto potencial que tienen en la construcción del modelo

Hiperparámetro	Valor
Modelo base	InceptionResNetV2
Número de neuronas en la capa de inicio del clasificador	256
Número de épocas	40
Número de pasos por época	30
Learning Rate inicial	0.01
Dropout	0.5
Batch size	32
Imagenet	Sí
Optimizador	Adamax
Función de pérdida	Categorical Crossentropy

Tabla 3.3: Hiperparámetros de la tercera ejecución

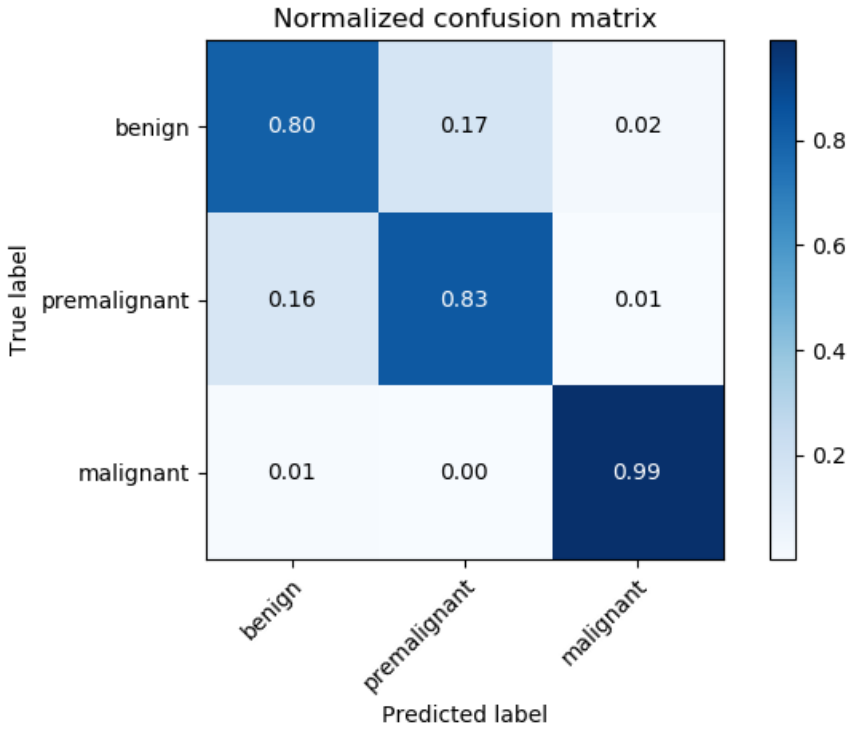


Figura 3.3: Matriz de confusión de la tercera ejecución

3.3.4. Reducción de la complejidad

En esta ejecución se han reducido el numero de neuronas en la primera capa del clasificador de cara a ver si había demasiadas y por tanto podían estar afectando al rendimiento negativamente

Hiperparámetro	Valor
Modelo base	InceptionResNetV2
Número de neuronas en la capa de inicio del clasificador	128
Número de épocas	40
Número de pasos por época	30
Learning Rate inicial	0.01
Dropout	0.5
Batch size	32
Imagenet	Sí
Optimizador	Adamax
Función de pérdida	Categorical Crossentropy

Tabla 3.4: Hiperparámetros de la ejecución

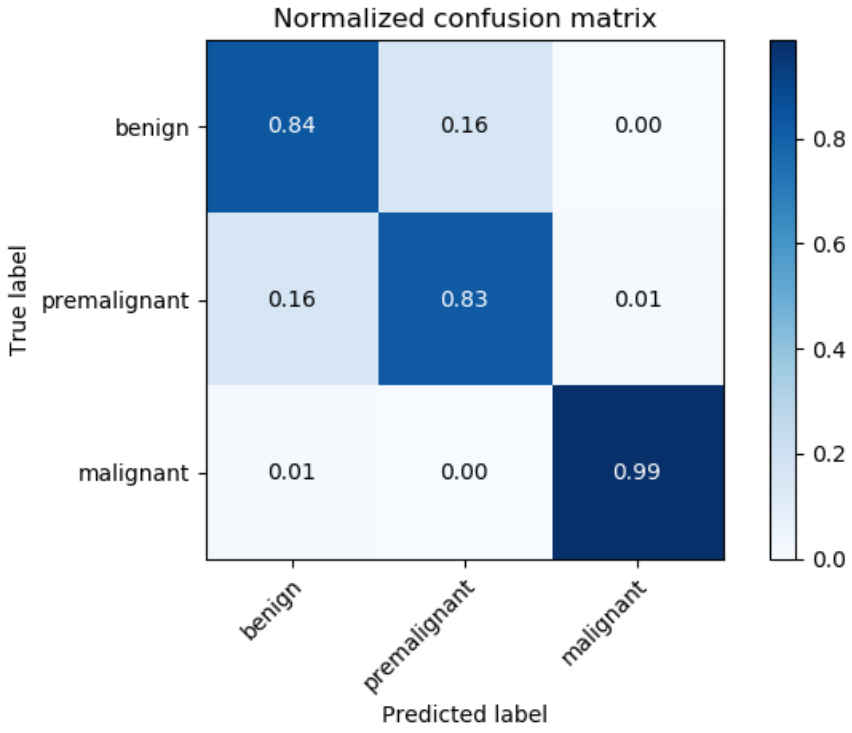


Figura 3.4: Matriz de confusión de la ejecución

Como se puede ver, las mejoras ya no son tan sustanciales, y es mas bien un ajuste mas fino hacia lo que queremos hacer. Tras ver esta

ejecución, me pregunté cuánto empeoraría si le añadíamos complejidad innecesaria a la arquitectura del clasificador

3.3.5. Aumentando la complejidad del clasificador

Hiperparámetro	Valor
Modelo base	InceptionResNetV2
Número de neuronas en la capa de inicio del clasificador	512
Número de épocas	40
Número de pasos por época	30
Learning Rate inicial	0.01
Dropout	0.5
Batch size	32
Imagenet	Sí
Optimizador	Adamax
Función de pérdida	Categorical Crossentropy

Tabla 3.5: Hiperparámetros de la ejecución

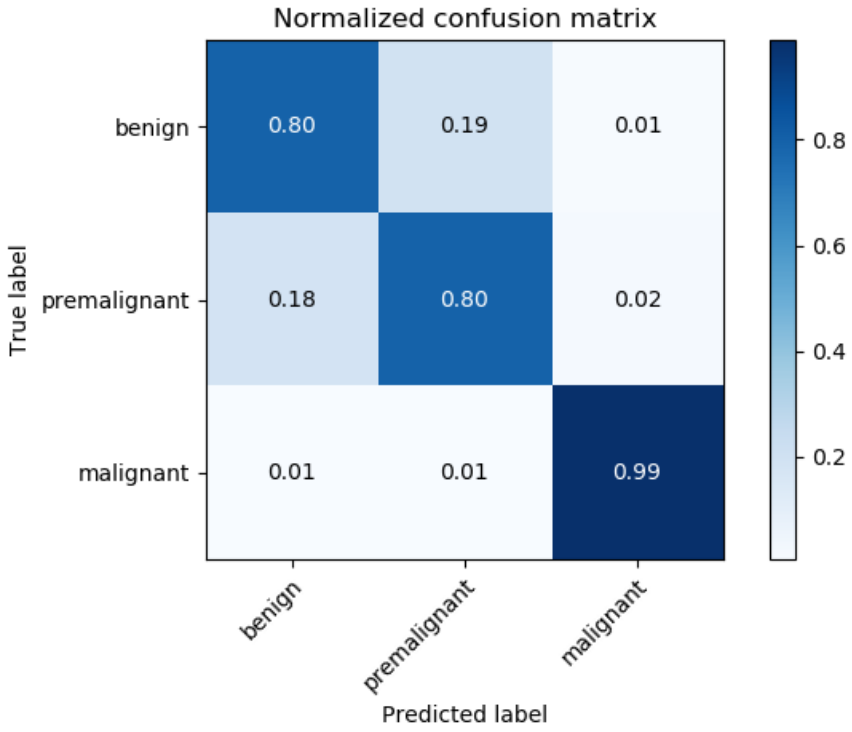


Figura 3.5: Matriz de confusión de la ejecución

Como una de las maneras de luchar contra el sobreajuste que se explicó anteriormente era subir el ratio de Dropout, se investigaron distintas ejecuciones con dropout diferentes

3.3.6. Aumentando el dropout a 0.6

Hiperparámetro	Valor
Modelo base	InceptionResNetV2
Número de neuronas en la capa de inicio del clasificador	128
Número de épocas	40
Número de pasos por época	30
Learning Rate inicial	0.01
Dropout	0.6
Batch size	32
Imagenet	Sí
Optimizador	Adamax
Función de pérdida	Categorical Crossentropy

Tabla 3.6: Hiperparámetros de la ejecución

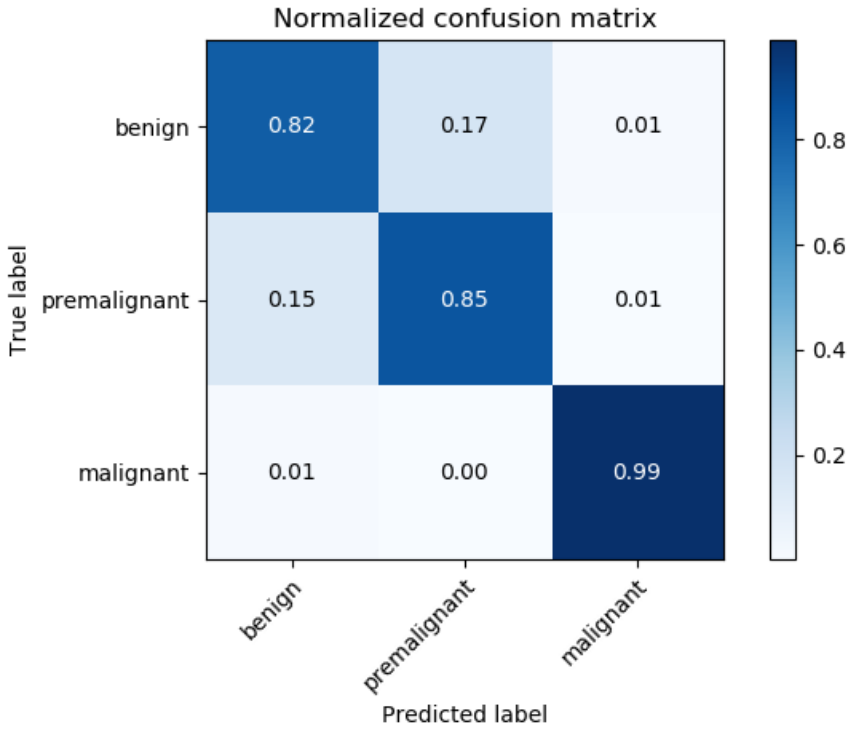


Figura 3.6: Matriz de confusión de la ejecución

Como se puede ver en la tabla de hiperparámetros de esta ejecución se han revertido a los hiperparámetros que consiguieron buenos resultados anteriormente, de cara a ver si se podía mejorar el resultado. Lo siguiente que se hizo fue subir el dropout a 0.7 a ver si se mantenía esa mejora.

3.3.7. Aumentando el dropout hasta 0.7

Hiperparámetro	Valor
Modelo base	InceptionResNetV2
Número de neuronas en la capa de inicio del clasificador	128
Número de épocas	40
Número de pasos por época	30
Learning Rate inicial	0.01
Dropout	0.7
Batch size	32
Imagenet	Sí
Optimizador	Adamax
Función de pérdida	Categorical Crossentropy

Tabla 3.7: Hiperparámetros de la ejecución

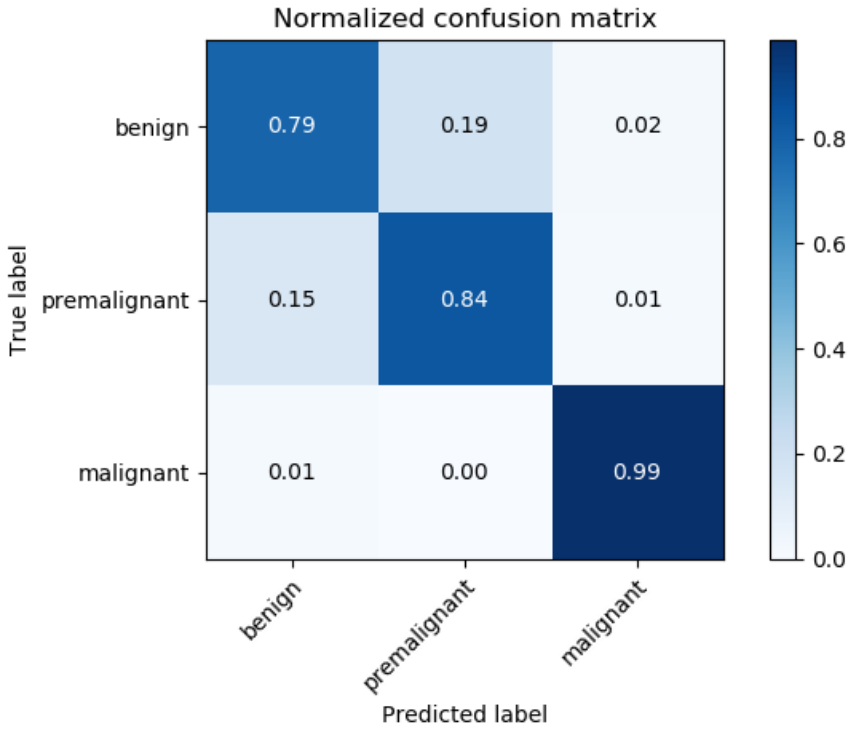


Figura 3.7: Matriz de confusión de la ejecución

3.3.8. Simplificando mas el clasificador

En esta ejecución vamos a intentar simplificar aún mas el clasificador, reduciendo a la mitad el numero de neuronas del clasificador.

Hiperparámetro	Valor
Modelo base	InceptionResNetV2
Número de neuronas en la capa de inicio del clasificador	64
Número de épocas	40
Número de pasos por época	30
Learning Rate inicial	0.01
Dropout	0.6
Batch size	32
Imagenet	Sí
Optimizador	Adamax
Función de pérdida	Categorical Crossentropy

Tabla 3.8: Hiperparámetros de la ejecución

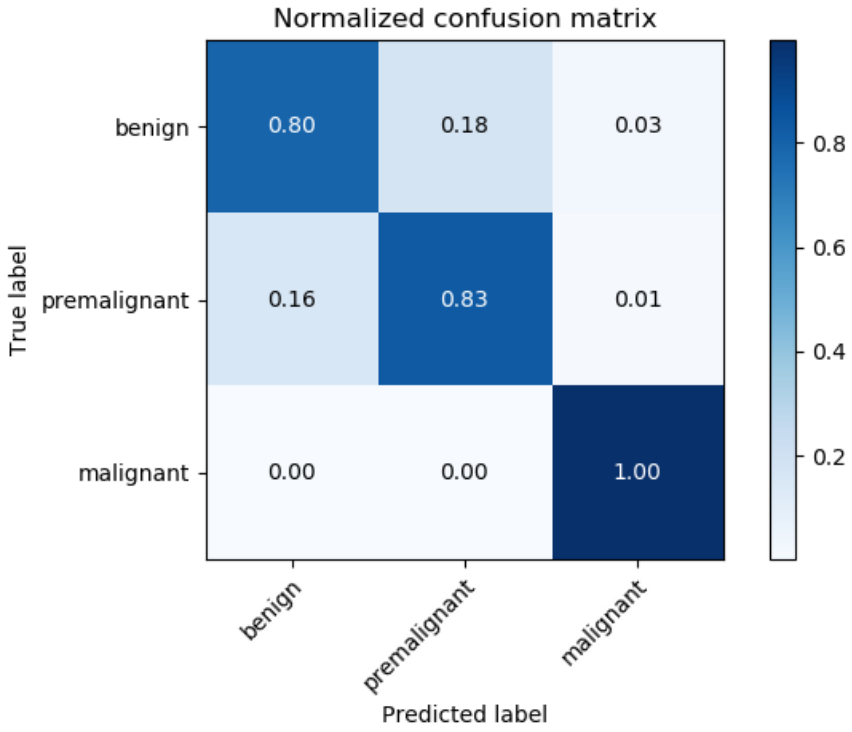


Figura 3.8: Matriz de confusión de la ejecución

3.3.9. Probando el entrenamiento con inicialización aleatoria

En esta prueba vamos a usar parámetros con mejores resultados hasta el momento y vamos a ver como afecta en el modelo el hecho de que los pesos ya no estén inicializados en el dataset imagenet

Hiperparámetro	Valor
Modelo base	InceptionResNetV2
Número de neuronas en la capa de inicio del clasificador	128
Número de épocas	40
Número de pasos por época	30
Learning Rate inicial	0.01
Dropout	0.6
Batch size	32
Imagenet	No
Optimizador	Adamax
Función de pérdida	Categorical Crossentropy

Tabla 3.9: Hiperparámetros de la ejecución

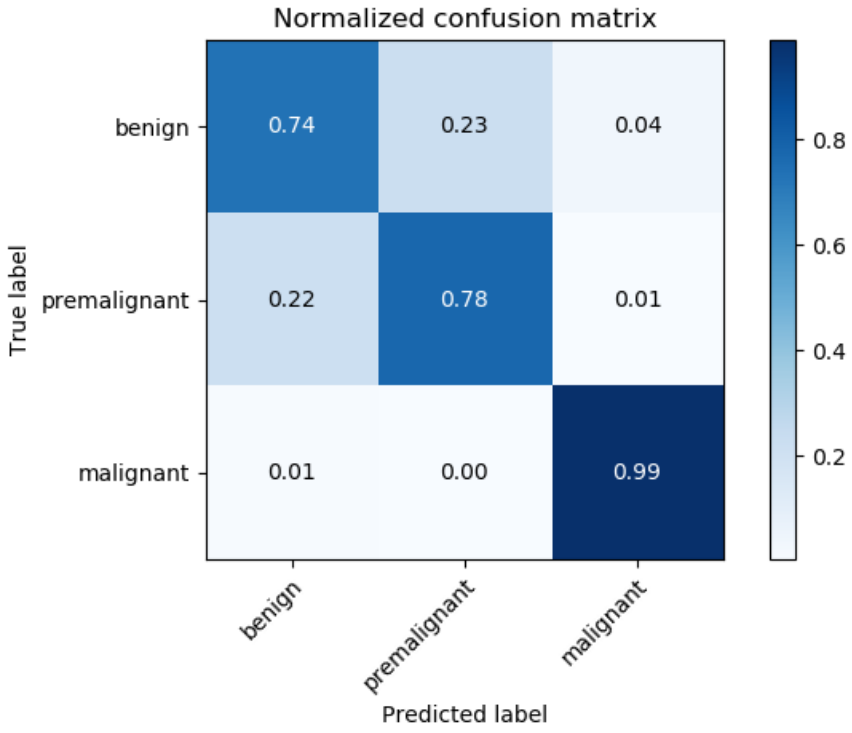


Figura 3.9: Matriz de confusión de la ejecución

3.3.10. Cambiando la tasa de aprendizaje inicial a 0.1

En esta prueba se va a mostrar cómo afecta al modelo que la tasa de aprendizaje sea demasiado grande

Hiperparámetro	Valor
Modelo base	InceptionResNetV2
Número de neuronas en la capa de inicio del clasificador	128
Número de épocas	40
Número de pasos por época	30
Learning Rate inicial	0.1
Dropout	0.6
Batch size	32
Imagenet	Sí
Optimizador	Adamax
Función de pérdida	Categorical Crossentropy

Tabla 3.10: Hiperparámetros de la ejecución

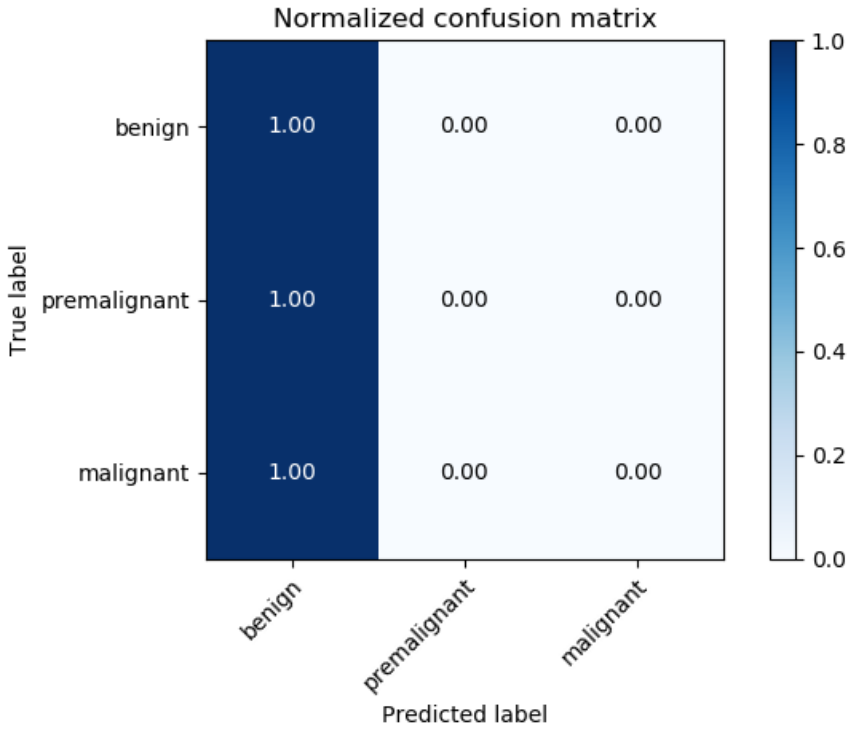


Figura 3.10: Matriz de confusión de la ejecución

Como podemos observar en la matriz de confusión un cambio grande en la tasa de aprendizaje tiene unos resultados desastrosos para el modelo que estamos construyendo, vamos a ver ahora qué pasa si hacemos una aumento más asumible en la tasa de aprendizaje.

3.3.11. Cambiando la tasa de aprendizaje inicial a 0.03

En esta ejecución vamos a cambiar la tasa de aprendizaje a 0.03, siendo este cambio más asumible por el modelo.

Hiperparámetro	Valor
Modelo base	InceptionResNetV2
Número de neuronas en la capa de inicio del clasificador	128
Número de épocas	40
Número de pasos por época	30
Learning Rate inicial	0.03
Dropout	0.6
Batch size	32
Imagenet	Sí
Optimizador	Adamax
Función de pérdida	Categorical Crossentropy

Tabla 3.11: Hiperparámetros de la ejecución

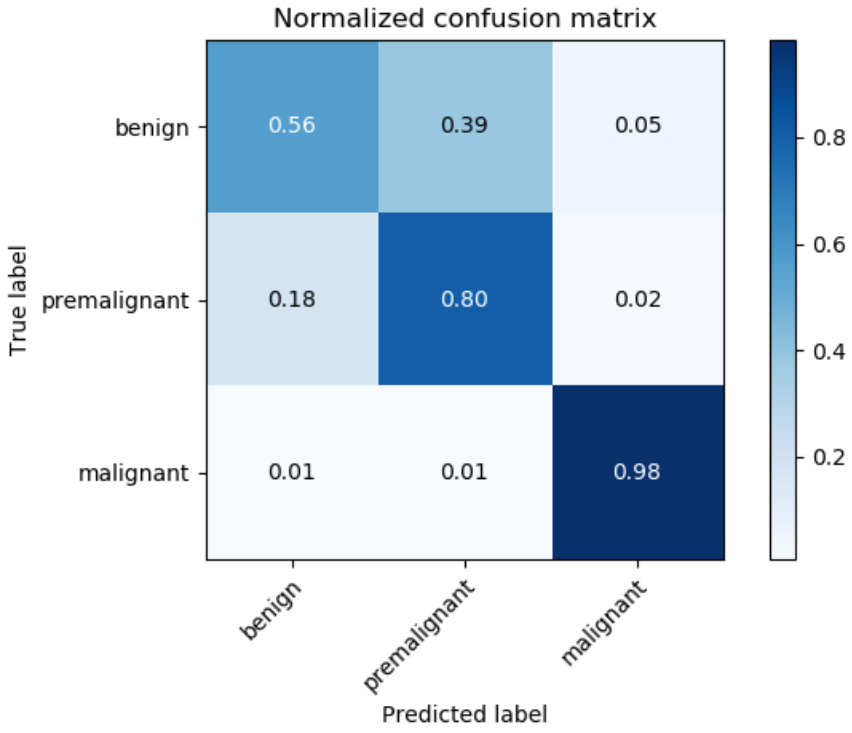


Figura 3.11: Matriz de confusión de la ejecución

3.3.12. Reduciendo la tasa de aprendizaje inicial a 0.005

En esta ejecución veremos que ocurre cuando a la red neuronal le reducimos el learning rate inicial a 0.005

Hiperparámetro	Valor
Modelo base	InceptionResNetV2
Número de neuronas en la capa de inicio del clasificador	128
Número de épocas	40
Número de pasos por época	30
Learning Rate inicial	0.005
Dropout	0.6
Batch size	32
Imagenet	Sí
Optimizador	Adamax
Función de pérdida	Categorical Crossentropy

Tabla 3.12: Hiperparámetros de la ejecución

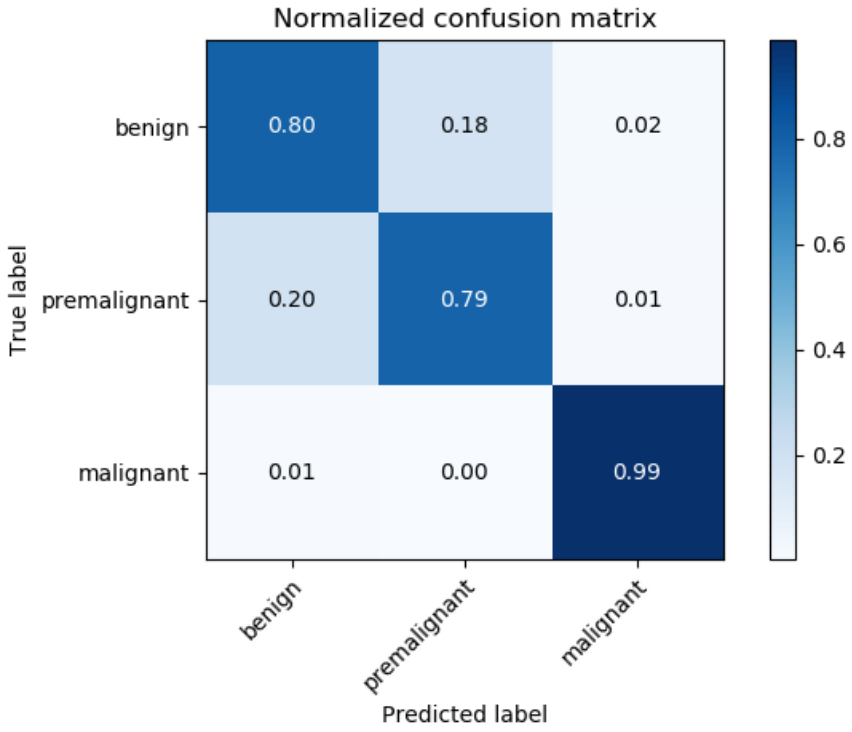


Figura 3.12: Matriz de confusión de la ejecución

3.3.13. Quitando el dropout

En esta ejecución vamos a visualizar qué ocurre cuando no utilizamos dropout antes de pasar los datos al clasificador

Hiperparámetro	Valor
Modelo base	InceptionResNetV2
Número de neuronas en la capa de inicio del clasificador	128
Número de épocas	40
Número de pasos por época	30
Learning Rate inicial	0.01
Dropout	0
Batch size	32
Imagenet	Sí
Optimizador	Adamax
Función de pérdida	Categorical Crossentropy

Tabla 3.13: Hiperparámetros de la ejecución

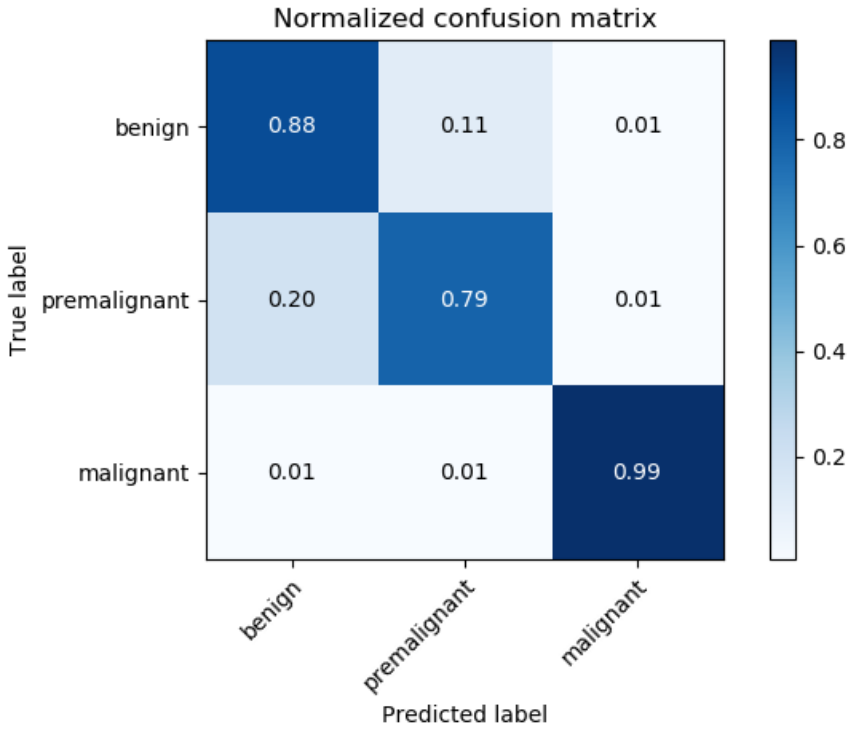


Figura 3.13: Matriz de confusión de la ejecución

Como se puede observar al quitar el dropout la red se desbalancea hacia una de las clases

3.3.14. Cambiando la función de pérdida al error cuadrático medio

Existen muchas funciones de pérdida y muchos optimizadores, ahora vamos a visualizar cómo afectan los cambios de éstos a los resultados de la red neuronal.

Hiperparámetro	Valor
Modelo base	InceptionResNetV2
Número de neuronas en la capa de inicio del clasificador	128
Número de épocas	40
Número de pasos por época	30
Learning Rate inicial	0.01
Dropout	0.6
Batch size	32
Imagenet	Sí
Optimizador	Adamax
Función de pérdida	Mean Squared Error

Tabla 3.14: Hiperparámetros de la ejecución

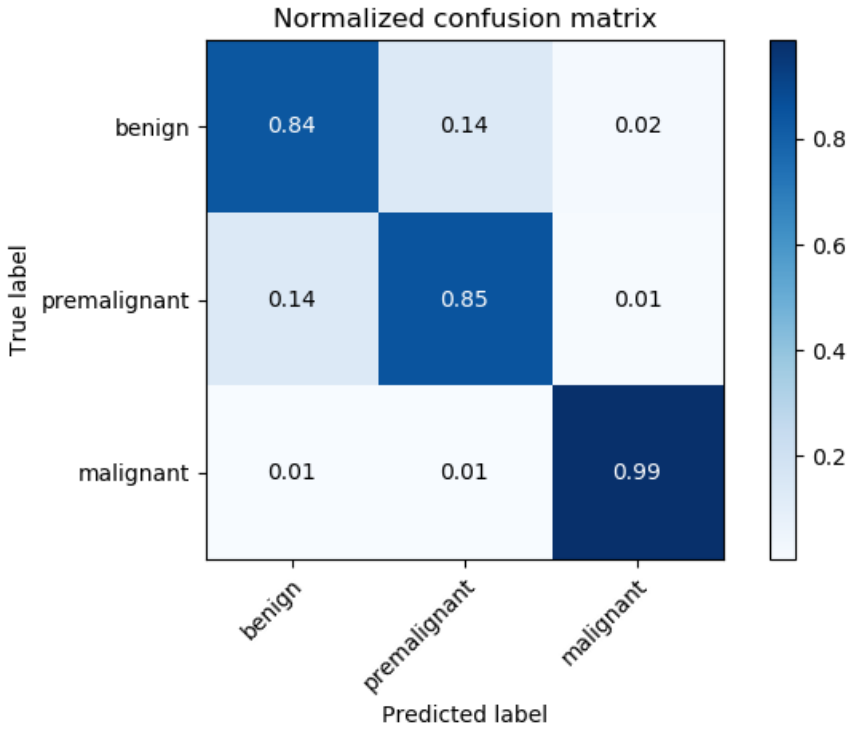


Figura 3.14: Matriz de confusión de la ejecución

Parece que con este optimizador hemos mejorado los resultados globales pero hemos empeorado los falsos positivos del modelo, a partir de ahora se trabajara conjuntamente con ellos, realizando 2 ejecuciones del modelo, una con Categorical Crossentropy y otra con mean squared error, y en función de los resultados, se incluirá una ejecución o las dos

3.3.15. Cambiando el optimizador del modelo a RMSProp

RMSProp es un optimizador para redes neuronales recurrentes, y por tanto en esta demostración vamos a obtener resultados diferentes a los obtenidos hasta ahora.

Hiperparámetro	Valor
Modelo base	InceptionResNetV2
Número de neuronas en la capa de inicio del clasificador	128
Número de épocas	40
Número de pasos por época	30
Learning Rate inicial	0.01
Dropout	0.6
Batch size	32
Imagenet	Sí
Optimizador	RMSProp
Función de pérdida	Categorical Crossentropy

Tabla 3.15: Hiperparámetros de la ejecución

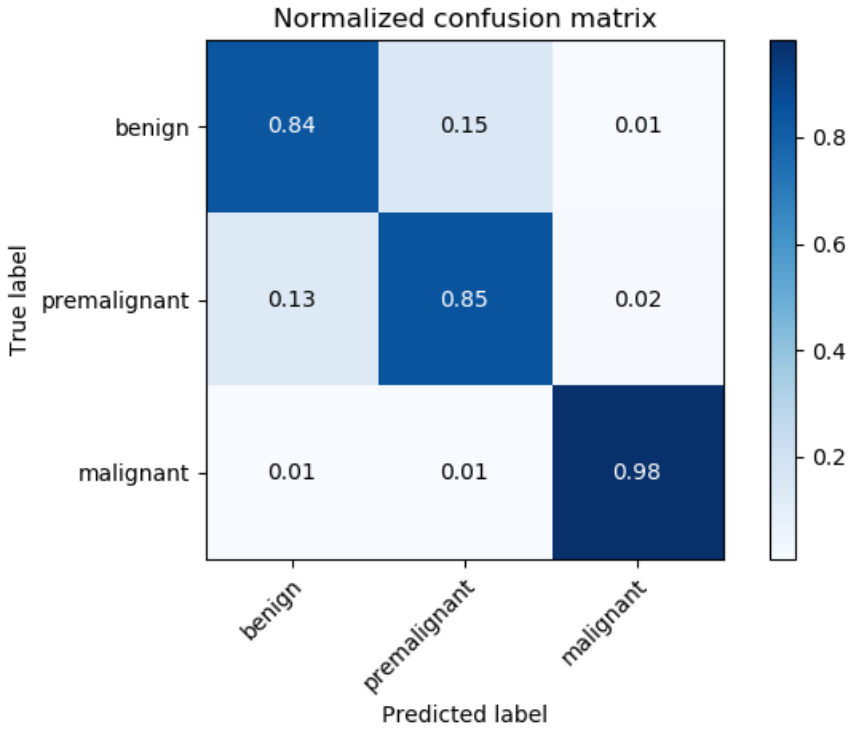


Figura 3.15: Matriz de confusión de la ejecución

3.4. Cambiando el modelo base que se usa para el clasificador a InceptionV3

En esta sección lo que veremos serán las pruebas realizadas al modelo usando como modelo base el Inception V3.

Hiperparámetro	Valor
Modelo base	InceptionResNetV2
Número de neuronas en la capa de inicio del clasificador	128
Número de épocas	40
Número de pasos por época	30
Learning Rate inicial	0.01
Dropout	0.6
Batch size	32
Imagenet	Sí
Optimizador	Adamax
Función de pérdida	Categorical Crossentropy

Tabla 3.16: Hiperparámetros de la ejecución

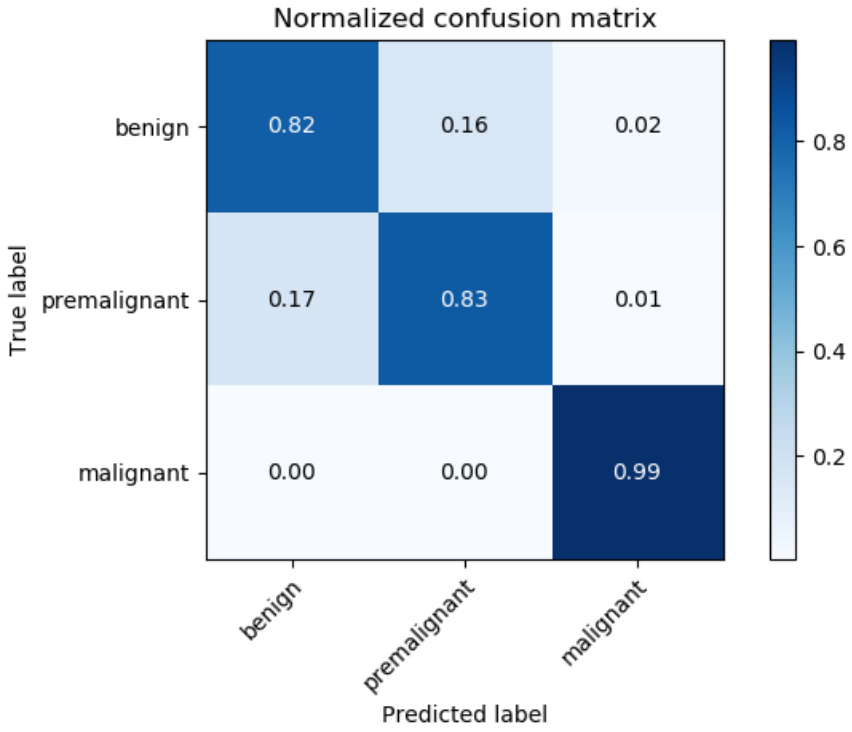


Figura 3.16: Matriz de confusión de la ejecución

3.4.1. Ejecución con los mejores hiperparámetros hasta el momento obtenidos con InceptionResNetV2

Para la primera ejecución con este cambio de modelo, vamos a usar la configuración de hiperparámetros que ha dado mejor resultado en el modelo del Inception ResNet v2

Hiperparámetro	Valor
Modelo base	InceptionResNetV2
Número de neuronas en la capa de inicio del clasificador	128
Número de épocas	40
Número de pasos por época	30
Learning Rate inicial	0.01
Dropout	0.6
Batch size	32
Imagenet	Sí
Optimizador	Adamax
Función de pérdida	Categorical Crossentropy

Tabla 3.17: Hiperparámetros de la ejecución

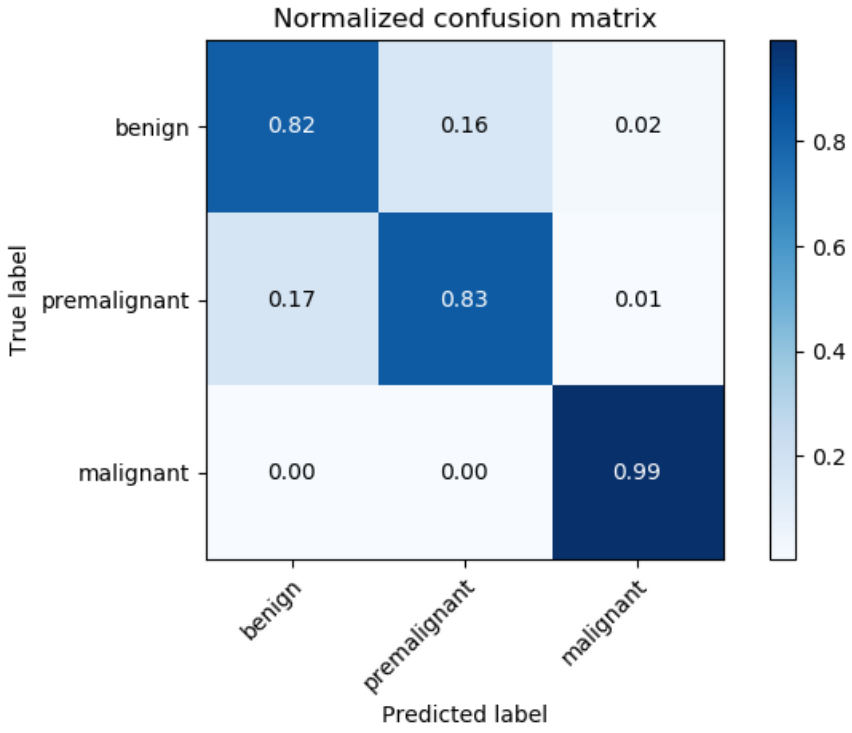


Figura 3.17: Matriz de confusión de la ejecución

3.4.2. Aumentando la complejidad de la arquitectura del clasificador.

Vamos a aumentar la complejidad del clasificador duplicando la cantidad de neuronas en la capa de entrada de éste, para ver como se comporta, tal como hicimos anteriormente. Lo que debería pasar es que la matriz de confusión sea peor, pues al añadir complejidad innecesaria se produce un bajo ajuste.

Hiperparámetro	Valor
Modelo base	InceptionResNetV2
Número de neuronas en la capa de inicio del clasificador	256
Número de épocas	40
Número de pasos por época	30
Learning Rate inicial	0.01
Dropout	0.6
Batch size	32
Imagenet	Sí
Optimizador	Adamax
Función de pérdida	Categorical Crossentropy

Tabla 3.18: Hiperparámetros de la ejecución

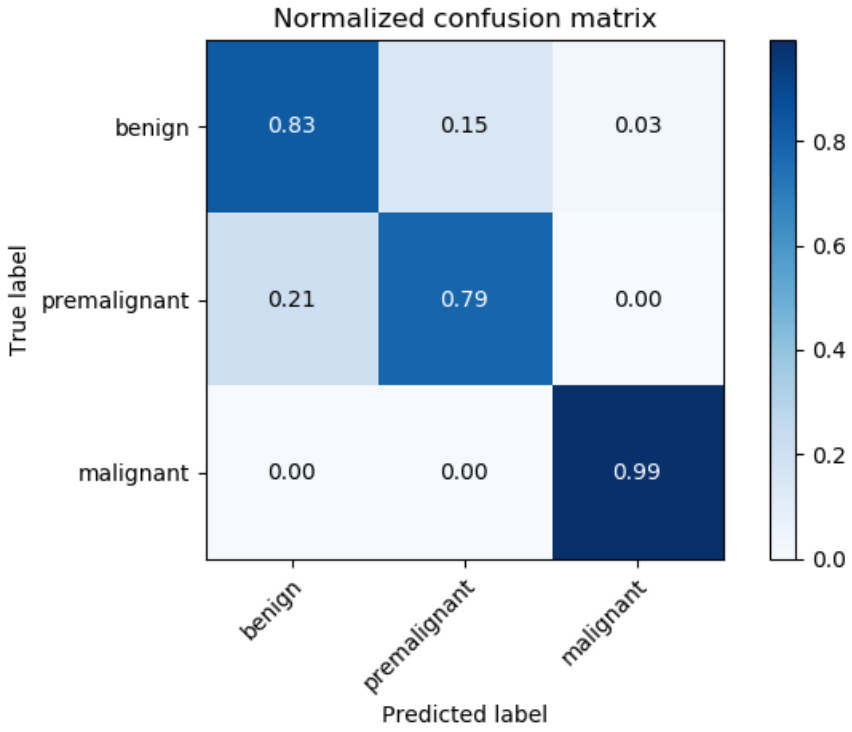


Figura 3.18: Matriz de confusión de la ejecución

3.4.3. Aumentando el dropout maneteniendo complejidad

En la ejecución anterior hemos visto cómo al aumentar la complejidad de la primera capa del clasificador hemos empeorado la matriz de confusión, pero hemos conseguido reducir a cero los falsos positivos de premaligno, como esto es interesante, vamos a ver si subiendo el dropout que debería ayudar a la generalización obtenemos un menor número de falsos positivos en benigno.

Hiperparámetro	Valor
Modelo base	InceptionResNetV2
Número de neuronas en la capa de inicio del clasificador	256
Número de épocas	40
Número de pasos por época	30
Learning Rate inicial	0.01
Dropout	0.7
Batch size	32
Imagenet	Sí
Optimizador	Adamax
Función de pérdida	Categorical Crossentropy

Tabla 3.19: Hiperparámetros de la ejecución

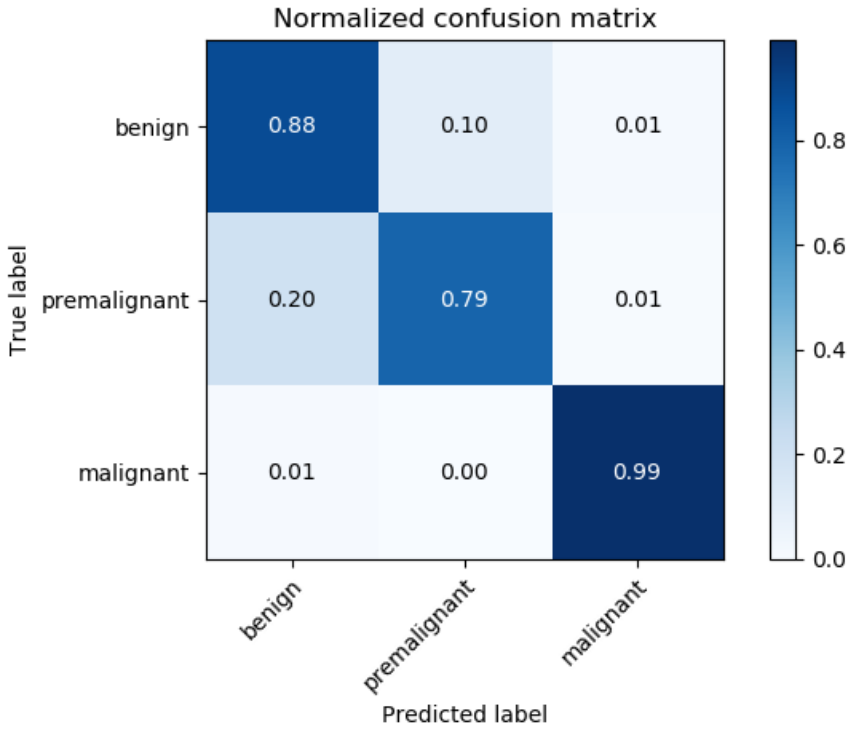


Figura 3.19: Matriz de confusión de la ejecución

No se han obtenido los resultados esperados, por tanto vamos a ver la degradación al eliminar el dropout

3.4.4. Eliminando el dropout

Al eliminar el dropout debemos ver una degradación en la capacidad para generalizar del sistema.

Hiperparámetro	Valor
Modelo base	InceptionResNetV2
Número de neuronas en la capa de inicio del clasificador	128
Número de épocas	40
Número de pasos por época	30
Learning Rate inicial	0.01
Dropout	0.6
Batch size	32
Imagenet	Sí
Optimizador	Adamax
Función de pérdida	Categorical Crossentropy

Tabla 3.20: Hiperparámetros de la ejecución

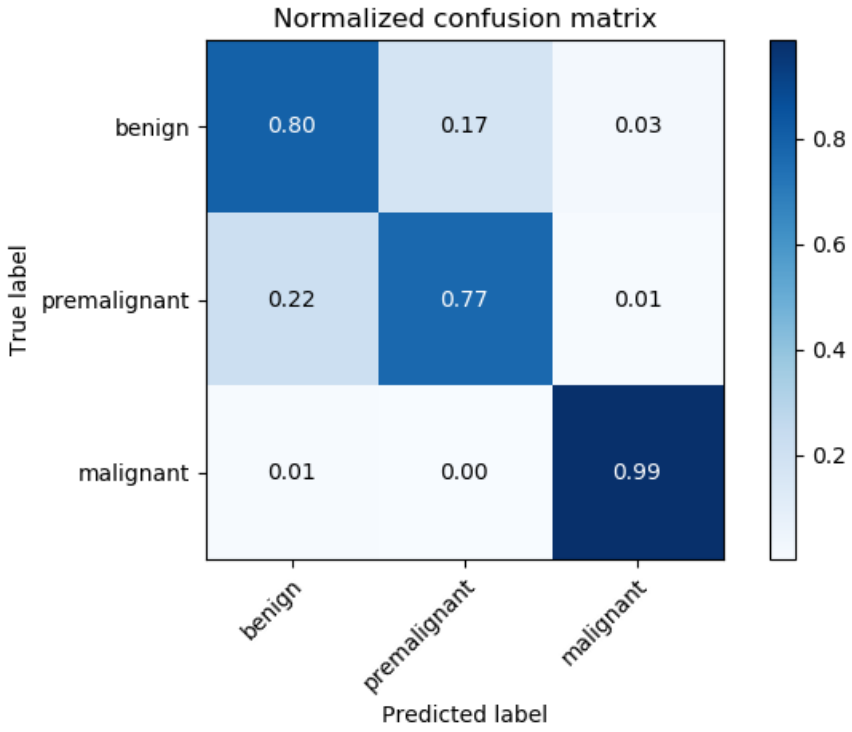


Figura 3.20: Matriz de confusión de la ejecución

Si miramos la matriz de confusión los límites entre benigno y premaligno se han difuminado bastante y han aumentado los casos de falsos positivos de benigno así como de premaligno.

3.4.5. Cambiando la función de pérdida al error cuadrático medio

Al cambiar la función de pérdida deberíamos ver un cambio en la matriz de confusión no muy grande, pero debería ser notorio.

Hiperparámetro	Valor
Modelo base	InceptionResNetV2
Número de neuronas en la capa de inicio del clasificador	128
Número de épocas	40
Número de pasos por época	30
Learning Rate inicial	0.01
Dropout	0.6
Batch size	32
Imagenet	Sí
Optimizador	Adamax
Función de pérdida	Mean Squared Error

Tabla 3.21: Hiperparámetros de la ejecución

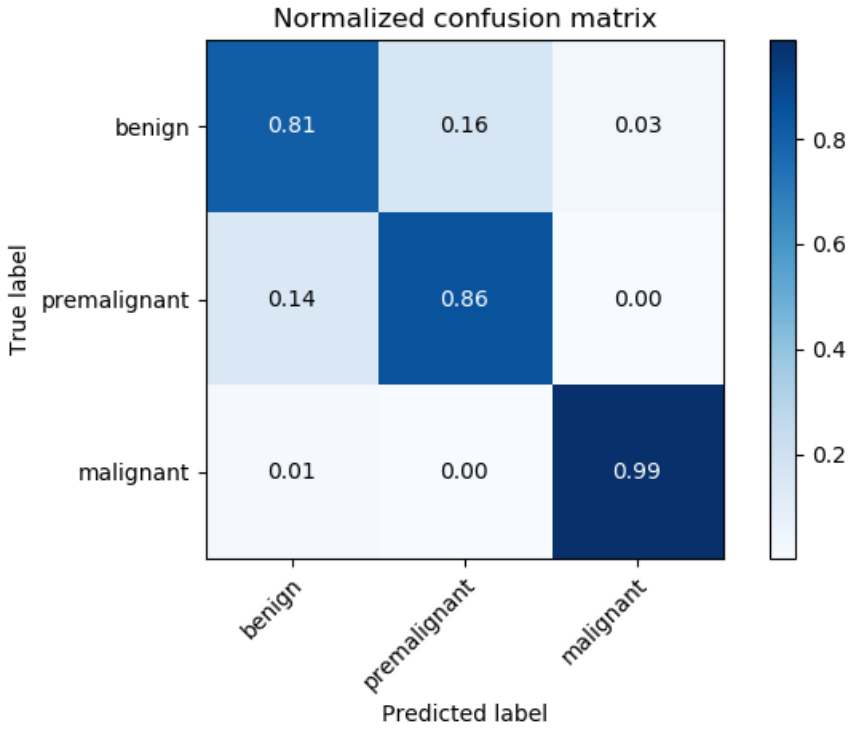


Figura 3.21: Matriz de confusión de la ejecución

En este caso parece que se consigue distinguir mejor las clases problemáticas (benignas y premalignas) pero sin embargo han aumentado los falsos positivos de los casos en los que la clasificación es benigna

3.4.6. Manteniendo la función de pérdida y cambiando el optimizador a RMSProp

En ejecuciones anteriores con este optimizador se obtuvieron resultados dispares, la idea es ver que resultados da y contemplar si es mejor que Adamax.

Hiperparámetro	Valor
Modelo base	InceptionResNetV2
Número de neuronas en la capa de inicio del clasificador	128
Número de épocas	40
Número de pasos por época	30
Learning Rate inicial	0.01
Dropout	0.6
Batch size	32
Imagenet	Sí
Optimizador	RMSProp
Función de pérdida	Mean Squared Error

Tabla 3.22: Hiperparámetros de la ejecución

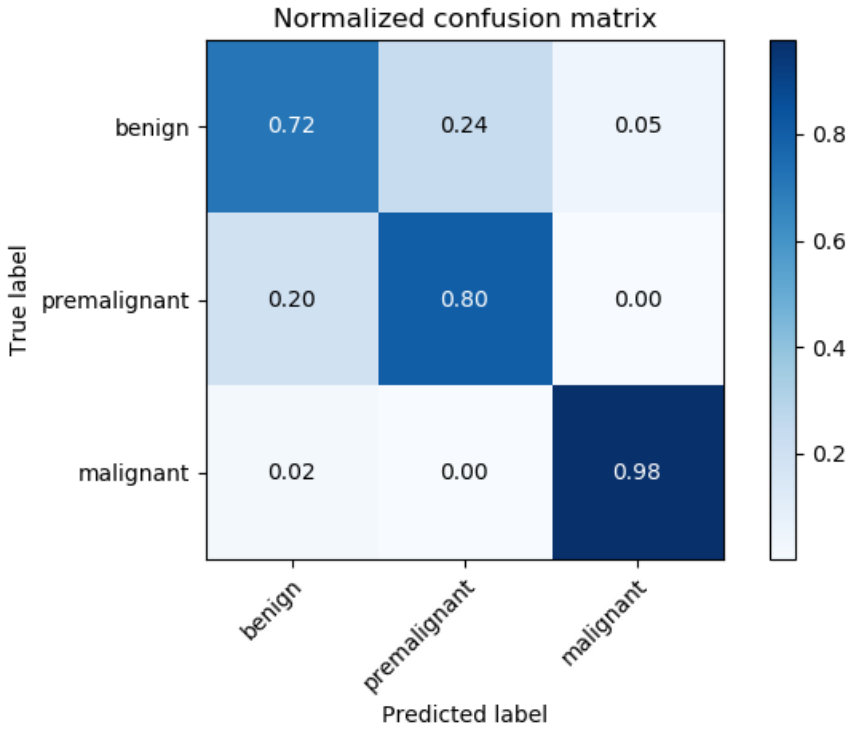


Figura 3.22: Matriz de confusión de la ejecución

3.4.7. Conclusiones de las pruebas

Tras realizar todas las ejecuciones ajustando los distintos hiperparámetros hasta el punto óptimo se han llegado a los siguientes resultados:

Hiperparámetro	Valor
Modelo base	InceptionResNetV2
Número de neuronas en la capa de inicio del clasificador	128
Número de épocas	40
Número de pasos por época	30
Learning Rate inicial	0.01
Dropout	0.6
Batch size	32
Imagenet	Sí
Optimizador	Adamax
Función de pérdida	Categorical Crossentropy

Tabla 3.23: Hiperparámetros de la ejecución

Con esta configuración de hiperparámetros y tras el proceso de entrenamiento, los resultados que se obtienen son:

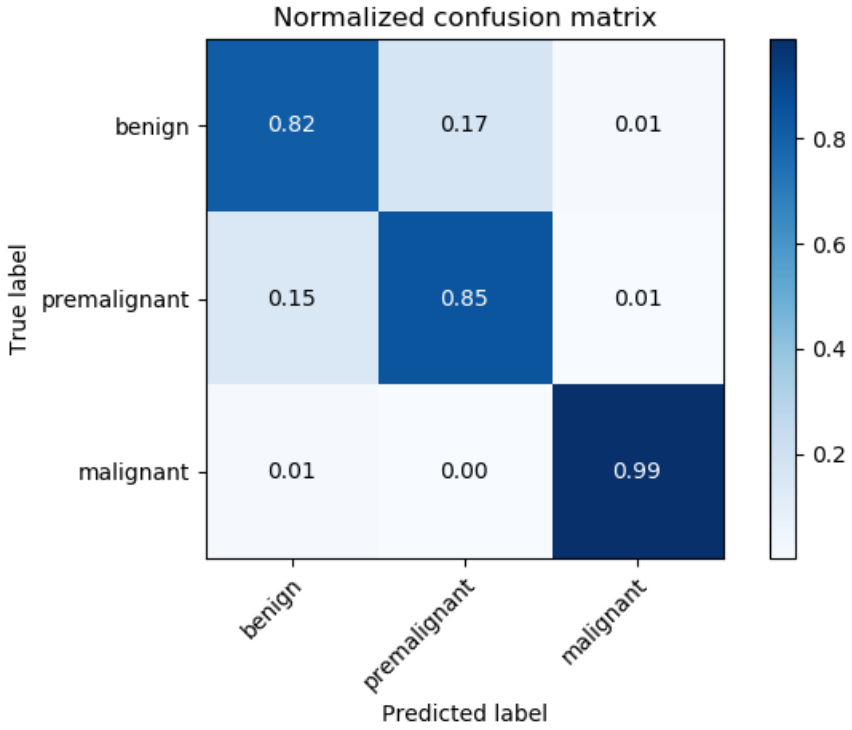


Figura 3.23: Matriz de confusión del modelo final

Gracias a la optimización de los hiperparámetros hemos pasado de

un estado inicial en el que el sistema no clasificaba correctamente casi nada hasta uno en el cual lo hace de forma bastante correcta, siempre teniendo en cuenta que el sistema es conservador y nos hemos centrado en dar el mínimo número de falsos positivos posible.

3.5. Pruebas Estructurales

Las pruebas estructurales en lo que respectan al sistema inteligente, se tienen varias funciones creadas pero todas siguen un único flujo salvo la función de entrenamiento llamada model_training() que tiene un condicional que se evalúa a verdadero o a falso en función del tipo de csv que le pasemos, pues para permitir la inclusión de técnicas de data augmentation sin eliminar la posibilidad de entrenamiento normal se añadió una columna artificial al dataset llamada 'mods'. Esta columna tenia un número, si era 0 la imagen no recibía modificaciones, sin embargo si era un numero mayor que 0 se usaba como semilla para el generador aleatorio de modificaciones de cara a obtener imágenes modificadas usando la clase ImageDataGenerator, y no se ha eliminado del código final porque es una de las líneas de investigación futuras.

3.6. Pruebas de integración

Debido a que el desarrollo de la red neuronal en un primer momento fue prioritario sobre el desarrollo de las aplicaciones cliente, se desarrollaron las funciones de carga de modelo llamable desde otra clase y la función de predicción sobre una imagen pasada por parámetros antes de que las aplicaciones estuvieran desarrolladas, por tanto cuando estas se desarrollaron, sobre todo la primera versión de la aplicación de escritorio, fue el momento de realizar las pruebas de integración.

Se prepararon dos casos de prueba, uno en el que se realizaba la predicción de la imagen cuando el sistema tenía en memoria el modelo del sistema inteligente y otro cuándo no se tenía el modelo cargado, pues es cuando en la interacción entre sistema inteligente y aplicación clien-

te podía haber problemas, y los dos casos de prueba fueron superados satisfactoriamente.

3.7. Pruebas de sistema

Para esta parte de pruebas de sistema se han tenido varias soluciones aplicadas a todo el desarrollo del proyecto

3.7.1. Pruebas Incrementales

Se han realizado pruebas incrementales a lo largo del desarrollo cuando se perseguía alcanzar metas concretas. Algunas de estas metas en las que se realizaban pruebas incrementales pueden ser: La creación de la primera red neuronal, Generación de un modelo a partir de una aplicación de Keras, Consecución del primer entrenamiento exitoso o Consecución de la primera predicción exitosa. Aquí nos estamos refiriendo a pruebas del sistema inteligente, las pruebas de las aplicaciones cliente se desarrollarán a continuación.

3.7.2. Pruebas de usabilidad

Para la parte de pruebas de usabilidad se ha contado con cinco usuarios a los cuales se les ha presentado la aplicación móvil y mientras realizaban el uso de la misma se tomaban apuntes para mejorar las interfaces. Los resultados de la primera batería de pruebas fueron satisfactorios salvo un usuario que preguntó si se podían cambiar el color de las letras de algunos botones por unas más claras pues no se leían bien. Se realizaron los cambios pertinentes y en una segunda batería de pruebas la opinión de todos los usuarios fue satisfactoria

3.7.3. Pruebas de estrés

Para la parte de pruebas de estrés se ha utilizado un bot de chrome que permite realizar peticiones a una API de manera automática.

Se probó a realizar peticiones cada segundo y el sistema fue capaz de soportarlo, eso si, una vez cargado el modelo en memoria. En el caso de que el modelo no esté cargado no se pueden realizar peticiones mientras este no esté cargado

3.8. Pruebas de las aplicaciones cliente.

En esta sección vamos a ver las pruebas realizadas a las aplicaciones cliente, aunque su funcionalidad es bastante sencilla.

3.8.1. Pruebas unitarias a lo largo del desarrollo

En primer lugar tenemos la navegación pero es bastante sencilla de probar, pues cada vez que terminaba una vista, creaba la siguiente vacía, y como NativeScript en el caso del smartphone y a Angular en el caso de la aplicación de escritorio permiten ejecutar las aplicaciones mientras de desarrollan y se van guardando los cambios cada vez que guardas cualquier fichero, era tan sencillo como esperar a que la aplicación en cuestión se recargara, ir a la vista en cuestión y probar si al pulsar el botón correspondiente, o enviar la imagen la aplicación pasaba a la siguiente vista, o mostraba otra parte de la vista embebida en esa misma pagina web.

Además de las pruebas comentadas anteriormente, se realizaron pruebas al cambiar el idioma para probar la internacionalización de las dos aplicaciones. En el caso de la aplicación de escritorio se dispone de dos imágenes pulsables, una con la bandera de España y la otra con la bandera de Reino Unido, y en función de la que se pulse tanto esa vista como las siguientes que se muestren estarán en el idioma seleccionado, sea cual sea la navegación realizada a partir de ese momento. En el caso de la aplicación móvil como la comprobación se realiza al principio de la ejecución de la aplicación si cambiamos el idioma del teléfono durante de la ejecución tendremos que reiniciar la aplicación para que esta cambie de idioma. Se tomó esta decisión de diseño para no sobrecargar todas las vistas pues normalmente nadie cambia de idioma el teléfono a cada

rato. En un uso normal las personas y los potenciales usuarios tienen su teléfono configurado en un idioma único, y por esto se ha decidido sólo comprobar una única vez al inicio el idioma del dispositivo.

3.8.2. Pruebas Funciónales

En cuanto a las pruebas funcionales, vamos a tomar los diagramas de navegación de ambas aplicaciones como si maquinas de estado finitas se trataran. Una vez vemos los distintos nodos, vemos que solamente hay 2 posibles caminos sin entrar en bucle, y al recorrer esos dos caminos cubrimos el 100 % de los nodos, por tanto extraemos los dos casos de prueba. En primer lugar accedemos a la aplicación, leemos las advertencias y las rechazamos volviendo así a la pantalla de inicio.

En el segundo flujo cumplimos el segundo caso de uso. Accedemos a la aplicación, leemos las advertencias, aceptamos las advertencias, seleccionamos una imagen para analizar, y obtenemos el resultado.

Con estos dos casos de prueba cubrimos todos los nodos, con lo cual hasta aquí estarían la sección de pruebas funcionales

3.8.3. Pruebas Estructurales

En cuanto a pruebas estructurales, si vemos el flujo de ejecución de las aplicaciones cliente, son cíclicas, con lo cual las posibles ejecuciones son 2, representadas por los casos de uso.

En la primera ejecución accedemos a la aplicación, leemos las advertencias y las rechazamos volviendo así a la pantalla de inicio.

En el segundo flujo cumplimos el segundo caso de uso. Accedemos a la aplicación, leemos las advertencias, aceptamos las advertencias, seleccionamos una imagen para analizar, y obtenemos el resultado.

En las pruebas estructurales no se tienen en consideración los ciclos del mismo por tanto teniendo estos dos caminos dentro del grafo de flujo de control.

En cuanto a los grafos de llamadas interprocedurales, la aplicación es totalmente lineal y solo hay dos condiciones en ella. La primera se

encuentra a la hora de aceptar las advertencias, y la segunda al enviar la imagen al servidor, en caso de que este no esté disponible. Al ser aplicaciones tan lineales es sencillo alcanzar la cobertura del código de las llamadas pues no hay muchas condiciones y las que hay son condiciones simples, de manera que solo tendríamos que generar 4 pruebas distintas en el caso de estas aplicaciones particularmente.

3.8.4. Pruebas de integración

El sistema completo se basa en el paradigma cliente-servidor, por tanto la integración de ambas partes se realiza cuando la aplicación tiene comunicación con el servidor. En el caso de la primera versión de la aplicación de escritorio realizada con Jinja2, al ser Jinja2 un motor de plantillas necesitaba comunicarse con el servidor constantemente, tanto para realizar la comunicación como para realizar el análisis de la imagen enviada. Sin embargo tanto en la segunda versión de la aplicación de escritorio como en la aplicación móvil desarrollada con Nativescript y Angular, la única comunicación que se realiza con el servidor es a la hora de enviar la imagen para que ésta sea analizada por el sistema inteligente.

Para el caso de la primera versión de la aplicación de escritorio, se ha de desarrollar un endpoint para cada una de las interacciones con el servidor. En este caso se han implementado cinco endpoints para cuatro vistas, pero como la navegacion es cuasi lineal con dos casos de prueba nos bastan, pues en un mismo caso de prueba probamos tanto la navegación como el tratamiento por parte del servidor de la imagen enviada.

Por otra parte tenemos las aplicaciones cliente que sólo se comunican con el servidor cuando han de enviar la imagen para su correspondiente análisis.

Para probar la funcionalidad de esto se creó un endpoint en el servidor que sirviera para recibir solo peticiones y no contestar nada. Así podía verse que datos recibía el servidor. Para esto se hicieron un par de casos de prueba. En el primero se enviaba una imagen, y si el servidor

conseguía decodificarla enviaba un mensaje confirmando la recepción y en la aplicación esto se mostraba mediante un cuadro de texto también conocido como alerta.

En el otro caso de prueba se enviaba cualquier otra cosa que no fuera una imagen y el servidor no devolvería nada. Una vez pasado el tiempo de espera de la respuesta de la petición la aplicación mostraría al usuario una alerta diciendo que no había sido posible analizar lo enviado. Este caso estaba mas enfocado a la aplicación de escritorio pues aunque en el explorador de archivos que se abre en principio solo puedes seleccionar imágenes pues así lo filtra, el usuario puede escoger cualquier fichero. Sin embargo como en la aplicación móvil tienes la opción de abrir la galería en la que solo se muestren imágenes este caso no hace falta. Este caso se diseño con la seguridad de la aplicación en mente pues al ser una API REST desplegada en internet cualquiera puede enviar lo que crea conveniente, por ejemplo un malware, por tanto se hace un prefiltro y después se intenta leer como imagen y si no se consigue, no se responde.

3.8.5. Pruebas de sistema

Para la parte de pruebas de sistema se ha estado trabajando sobretodo en la parte de eficiencia y rendimiento. Sobretodo intentando optimizar el tiempo que tiene esperar el usuario pues se quería suprimir la sensación de espera para el usuario, y al llegar a esta parte de las pruebas fue cuando se determinó que la opción de TensorFlow.js dejaba de ser una opción, tanto por el tiempo que conllevaba de espera como la batería consumida. Además de descartar esto se corrigió la vista de resultados en la aplicación móvil pues en un principio se enviaba la imagen en el mismo momento en el que el usuario pulsaba sobre una de la galería y hasta que no se recibía la respuesta no se cambiaba de vista. Esto en ocasiones podía llevar varios segundos en los que el usuario podría pensar que la aplicación se ha quedado bloqueada, por tanto lo que se hace ahora es pasar a la siguiente vista y cuando se recibe el veredicto

inyectarlo en la misma.

Parte III

Parte tercera.

Conclusiones y líneas futuras

En este proyecto se ha tratado el análisis de imágenes con deep learning, en concreto el reconocimiento y clasificación de lesiones de la piel en 3 categorías. Para la puesta a disposición del público de este sistema inteligente se han usado dos aplicaciones, una para smartphone y otra para escritorio.

Para la elaboración de este proyecto se han usado TensorFlow y Keras , que son dos librerías para python, una es un framework de Google para labores de aprendizaje automático, y la otra es una API que se ejecuta sobre TensorFlow para facilitar todo el proceso de crear, entrenar, probar y poner en producción modelos. Se han escogido estas dos librerías en primer lugar por el conocimiento que se tenía previo a la elaboración de este TFG de las mismas, en segundo lugar por la amplia documentación que hay sobre las mismas así como su disponibilidad y por último porque a haber una comunidad que trabaja con estas librerías podría encontrar la forma de solventarlos posibles problemas que surgieran durante el desarrollo.

El uso de páginas web y de las redes sociales para consultar cuestiones relacionadas con el diagnóstico o tratamiento de enfermedades está más en auge que nunca, desde síntomas, hasta qué medicamento usar para tratar una determinada enfermedad. El mundo digital alberga una cantidad ingente de información, muchas veces útil, pero otras tantas errónea o, sencillamente, no aplicable a la enfermedad o a los síntomas que consultamos. Este proyecto intenta dar respuesta a este fenómeno poniendo a disposición del público una herramienta para guiar a los usuarios, pero como bien se dice en las advertencias de las aplicaciones,

en ningún caso es sustitutivo de una consulta al dermatólogo.

El sistema inteligente se desarrolló usando como base la arquitectura Inception ResNet V2 que a Google le dió tantos buenos resultados. Tras un estudio exhaustivo de dicho modelo y su posterior adecuación en cuanto arquitectura y selección de parámetros, se ha conseguido muy buenos resultados de generalización. Tras realizar todas las pruebas de las que se habla en el capítulo 3 finalmente se ha elegido el modelo con la arquitectura InceptionResNetV2 y la siguiente configuración de hiperparámetros:

Hiperparámetro	Valor
Modelo base	InceptionResNetV2
Número de neuronas en la capa de inicio del clasificador	128
Número de epocas	40
Número de pasos por época	30
Learning Rate inicial	0.01
Dropout	0.6
Batch size	32
Imagenet	Sí
Optimizador	Adamax
Funcion de pérdida	Categorical Crossentropy

Tabla 3.24: Hiperparámetros de la ejecución

Con esta configuración de hiperparámetros y tras el proceso de entrenamiento, los resultados que se obtienen son:

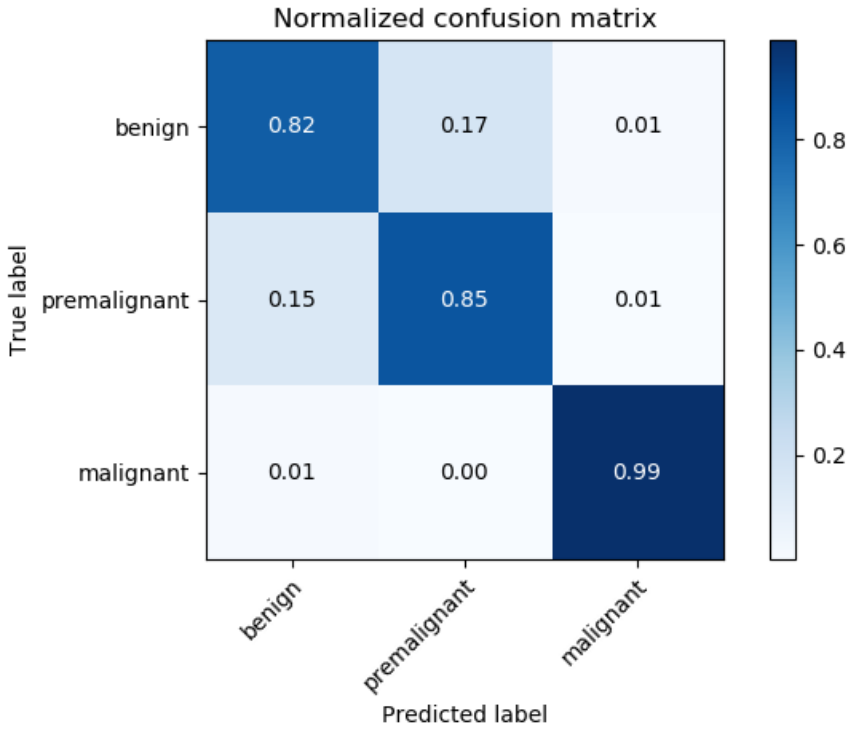


Figura 3.24: Matriz de confusión del modelo final

Como se puede comprobar en la matriz de confusión, los malignos están muy bien clasificados pero sin embargo entre las otras dos clases la diferenciación no es tan clara. Estos son los mejores resultados que se han podido obtener tras realizar toda una serie de ejecuciones y optimizando los hiperparámetros con los que el sistema inteligente se encontraba configurado y teniendo en cuenta los datos de los que se disponían.

Dado que las redes neuronales aprenden a partir de ejemplos, para la elaboración de este proyecto se ha necesitado buscar una base de datos apropiada, y que estos estuvieran bien clasificados. Este punto ha resultado ser conflictivo y la búsqueda, peticiones y preprocesamiento de datos se ha llevado bastante horas de trabajo.

El sistema inteligente desarrollado ha sido dotado de un enfoque conservador, es decir, reduce al mínimo los falsos positivos y los falsos nega-

tivos pues estos son los más peligrosos. La idea es que cuando el usuario muestra una imagen de piel claramente sana el sistema sepa detectarlo y en otro caso, reciba la advertencia de que solo un profesional puede, con pruebas complementarias, diagnosticar con claridad.

En cuanto a las aplicaciones, éstas se han desarrollado teniendo en cuenta los 10 principios de usabilidad de J.Nielsen, haciendo el diseño intuitivo y usable, buscando que cualquier usuario sea capaz de utilizarla con eficacia. Con este fin se ha añadido el idioma inglés para que los usuarios de otros países que conozcan este idioma sean capaces de usar la aplicación.

Hasta ahora se han hablado de las conclusiones extraídas del proyecto, ahora es el turno de las lineas futuras de investigación

En primer lugar una de las lineas futuras más interesantes a tratar sería completar el conjunto de datos con el que se trabaja de cara a ser capaces de identificar qué tipo de lesión se encuentra en la imagen, de entre las más comunes. De cara a esto, se han dejado preparados los archivos .csv con una columna en la que tenemos el diagnóstico por si más tarde se continúa el proyecto tener ese escenario cubierto.

Otra linea futura de investigación consiste en tener la cámara del teléfono abierta todo el rato en vez de forzar al usuario a enviar una foto, para ello tendríamos que hacer un filtrado en tiempo real de la imagen, usar OCR para detectar la región de la foto a investigar y tras eso enviar esa región del espacio al modelo para que devolviera la respuesta a lo que hemos enviado.

Una cosa que se podría hacer es actualizar la parte del servidor a TensorFlow 2.0 pues la alpha se publicó cercana a la entrega de este TFG, también para ver si alguna de las funcionalidades nuevas que incluye TensorFlow puede ser utilizable en este proyecto.

Otra posible trabajo futuro consiste introducir el modelo en el teléfono y ejecutarlo todo in situ. Para este fin TensorFlow tiene una variante para javascript llamada TensorFlow.js, lo que se haría sería una vez guardado el modelo que queremos usar en un fichero .h5, cargarlo en memoria, convertirlo y guardarlo en otro formato, todo esto explicado en la

web de TensorFlow. Sin embargo, uno de los problemas más grandes de este método es que consume mucha batería porque como dijimos antes, el Deep Learning se basa en operaciones tremendamente complejas y los procesadores actuales no pueden manejarlas con un consumo energético eficiente. Por eso se incluye en esta sección de líneas futuras a la espera de procesadores más potentes y con menor consumo, pues ahora sería impracticable por el consumo de batería tan elevado que tendría como contrapartida.

En este punto es interesante resaltar que en el futuro, se debería realizar una aplicación para dispositivos iOS de cara a llegar al máximo público posible. Esta tendría la misma navegación descrita en el capitulo 2 de la memoria en el apartado en el que se habla de la aplicación móvil y la interfaz general no sería muy diferente de la de allí dispuesta para la aplicación Android.

Elias Ibn Ziaten Cerezo
22 de junio de 2019

Parte IV

Apéndices

Apéndice A

Apéndice

Contenido

A.1. Glosario

Machine Learning: El machine learning o aprendizaje automático es una rama de la inteligencia artificial cuyo objetivo es desarrollar una serie de algoritmos con los que se pretende que las maquinas aprendan. Concretamente, se trata de crear programas capaces de generalizar comportamientos a partir de una información suministrada en forma de ejemplos.

Deep Learning: El deep learning o aprendizaje profundo es una rama del machine learning basada en asimilar representaciones de datos.

Feature Compression: Feature Compression es una técnica para reducir el numero de variables que se toman de los valores de entrada a la red neuronal de cara a simplificar los cálculos.

Overfitting: Overfitting es el nombre que recibe la situación en la que una red neuronal se especializa demasiado en las clases que les pa-

samos como entrenamiento y provoca una mala generalización en datos que la red no ha visto nunca. Esto es malo pues el fin ultimo de toda red neuronal es que al ver datos nuevos sea capaz de identificarlos correctamente.

Dataset: Un dataset, literalmente traducido del ingles es un conjunto de datos, y eso es. En esencia es una colección de datos que están clasificados en las distintas clases que se le quieren enseñar a la red neuronal, pero no necesariamente tiene que estar relacionado con el campo de las redes neuronales, en el resto de disciplinas de ML también está presente así como en Data Science en general.

A.2. Manual de Usuario de la aplicación móvil

En esta sección se va a elaborar el manual de usuario de las aplicación móvil

En primer lugar, para iniciar la aplicación pulsamos sobre su icono

Figura A.1: Icono de la aplicación

Tras pulsar sobre el icono de la aplicación se nos abre la aplicación y nos carga la primera página

Figura A.2: Primera actividad

En esta actividad lo que deberemos realizar es una pulsación ya sea en la imagen del lazo o bien sobre el botón entrar. Al pulsar se nos carga la siguiente actividad.

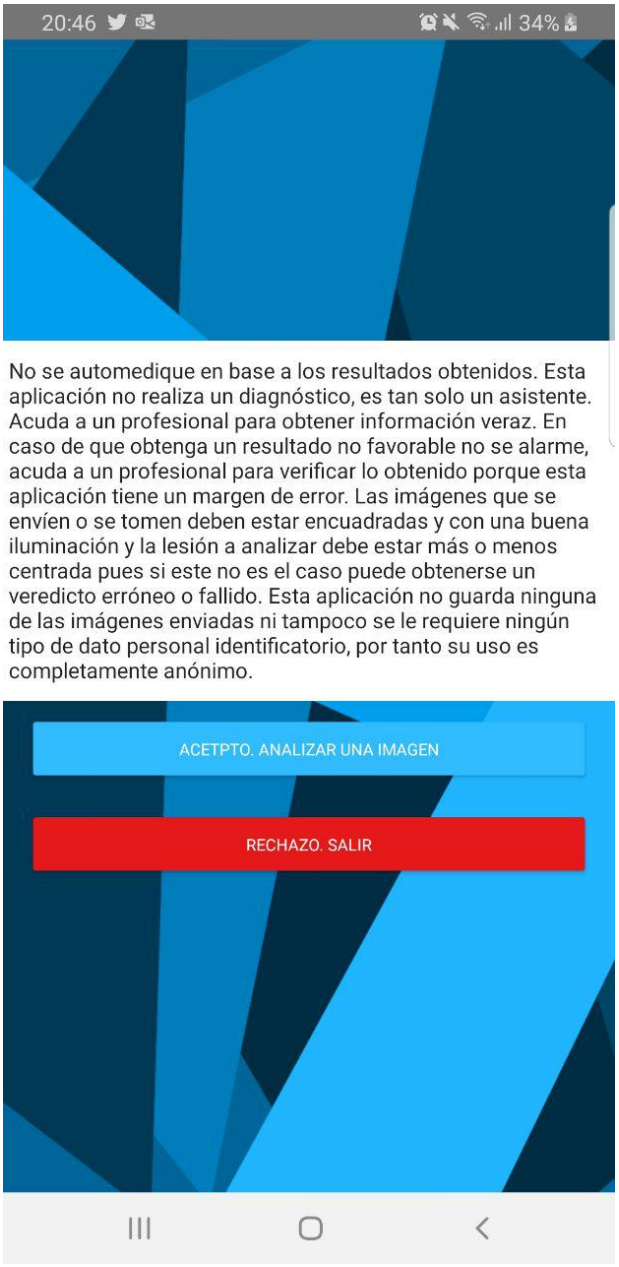


Figura A.3: Captura de actividad de advertencias

En esta vista se pueden leer las advertencias, si queremos realizar un análisis de una imagen deberemos pulsar sobre aceptar, sin embargo si las advertencias no nos parecen correctas o no queremos aceptarlas, podemos pulsar sobre salir, acción que nos llevará a la vista anterior.

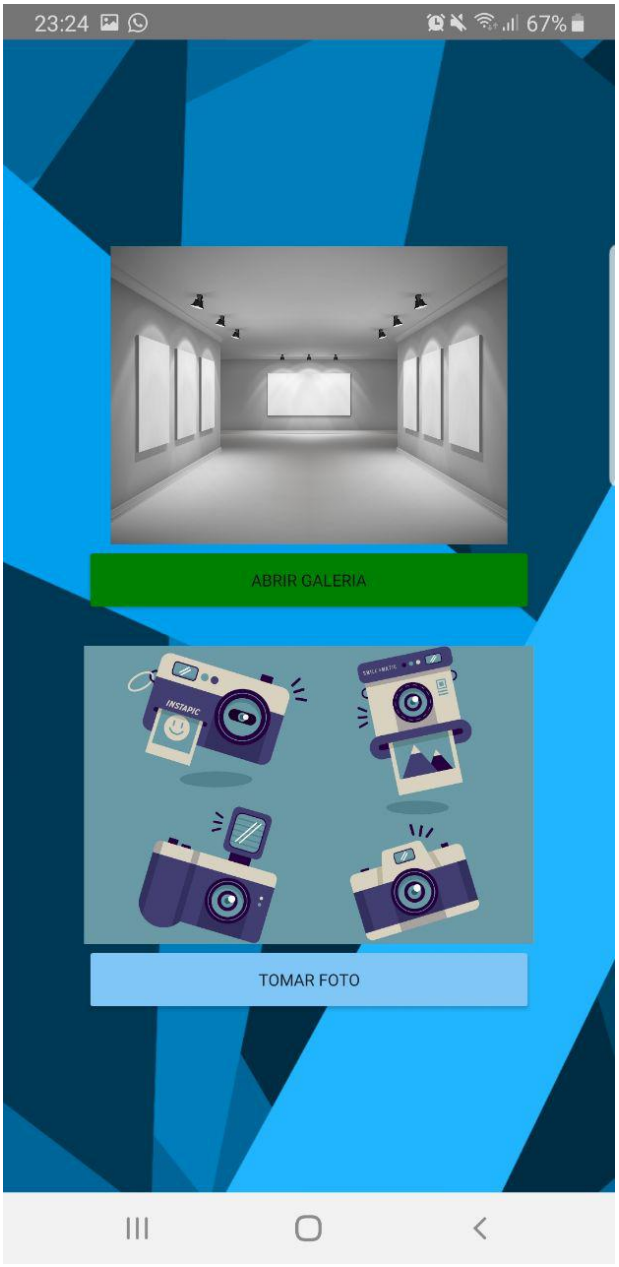


Figura A.4: Vista de selección de método de envío de imagen

En esta vista podemos seleccionar uno de los dos posibles métodos de envío de imágenes al servidor. En primer lugar tenemos una imagen y un botón que nos dan acceso a la galería del teléfono y en la cual nos vamos a encontrar las fotografías ordenadas de las mas recientes a las

más antiguas, desde ahí podremos pulsar sobre una de ellas y se enviará al servidor de manera automática.

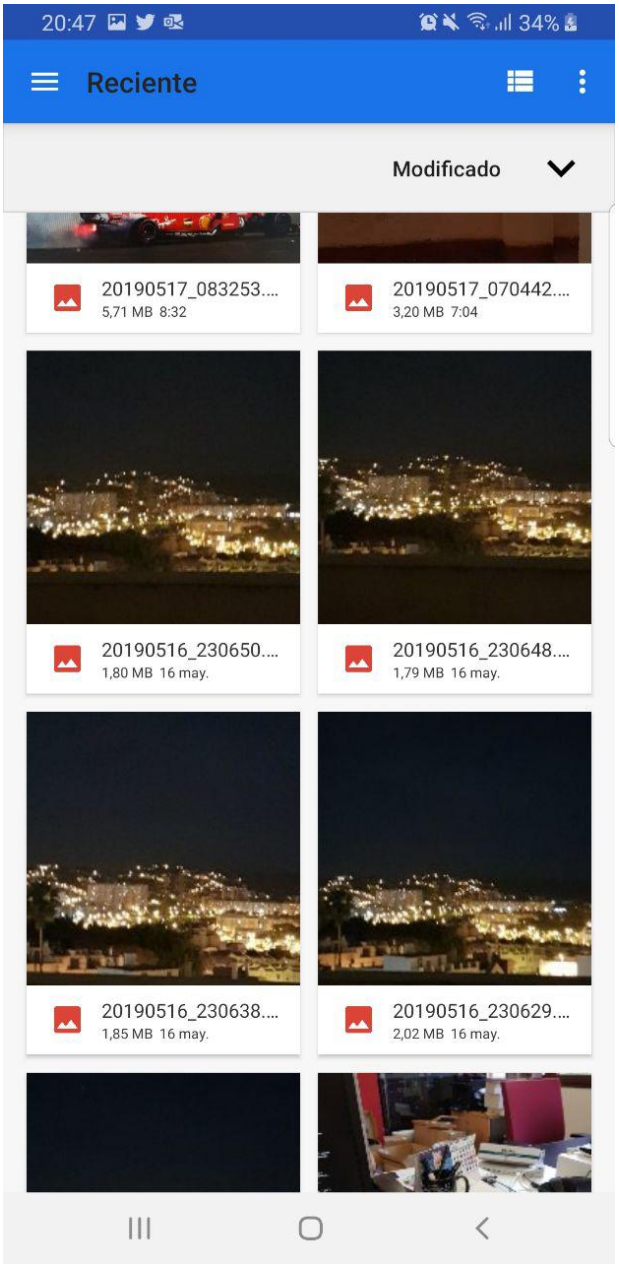


Figura A.5: Galería del teléfono móvil

El otro método de envío de imágenes al servidor es mediante la cámara trasera del teléfono, para acceder a la misma lo que tenemos es

que pulsar o bien sobre la imagen, o bien sobre el botón de la cámara en el menú de selección. Al pulsar sobre uno de esos dos elementos se nos abrirá un cuadro de diálogo preguntando si queremos dar los permisos de ejecución sobre la cámara, al aceptarlos se nos abrirá la aplicación de la cámara del teléfono. Tras esto disparamos una foto, y el teléfono nos pregunta si deseamos enviar la foto. AL pulsar sobre aceptar nos lleva a la vista de resultados.

Figura A.6: Galería del teléfono móvil

Por último tenemos la vista de los resultados.

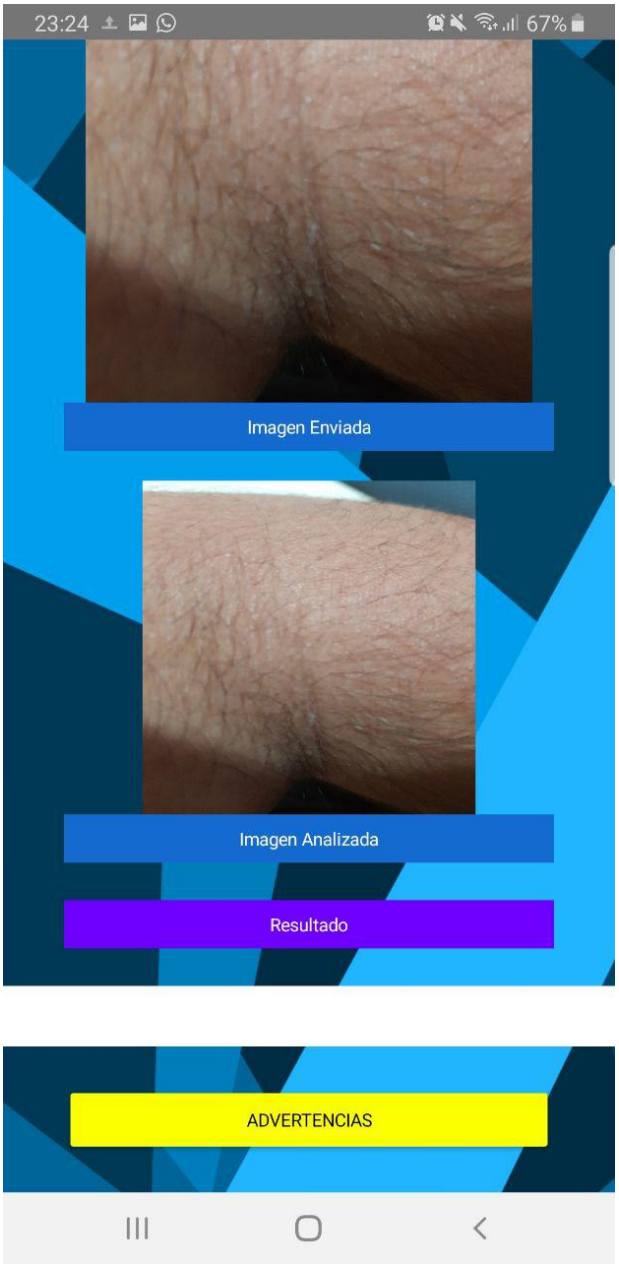


Figura A.7: Galería del teléfono móvil

En esta vista lo que nos encontramos es la foto que hemos enviado al servidor, la foto que el servidor ha analizado pues la foto sufre un rescalado, y justo debajo nos encontramos el veredicto. Este veredicto suele tardar un tiempo en aparecer, en caso de que el servidor no se encuentre

disponible por algún motivo, tras un tiempo en el cual la aplicación está intentando enviar la foto al servidor, aparece un cuadro informándonos de que no ha sido posible realizar el envío.

En caso de que el servidor no se encuentre disponible por algún motivo, bien podemos desplegarlo en nuestra red, o bien ponernos en contacto con una persona que lo tenga desplegado en internet

A.3. Manual de usuario de la aplicación de escritorio

En esta sección se va a elaborar el manual de usuario de las aplicación de escritorio.

En primer lugar, para iniciar la aplicación tecleamos sobre el navegador https://localhost:80 Tras escribir esto en el navegador se nos abre la aplicación y nos carga la primera vista

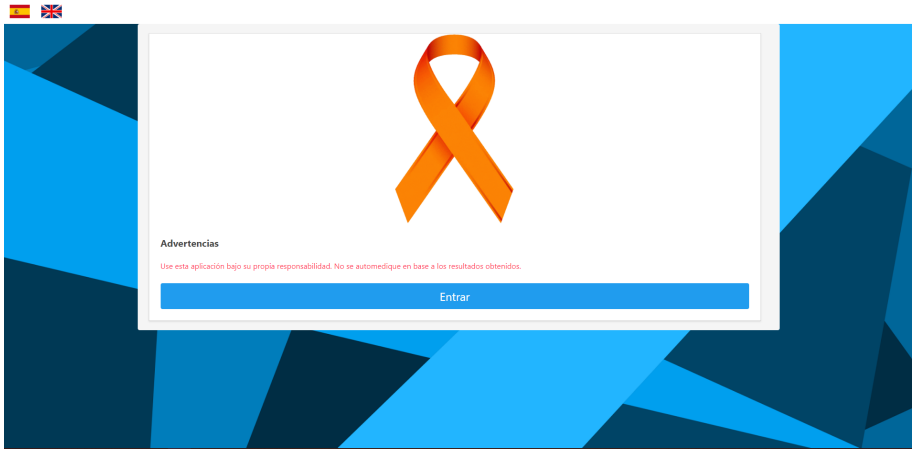


Figura A.8: Primera vista

En esta vista lo que deberemos realizar es una pulsación sobre el botón entrar para acceder a la vista de las advertencias

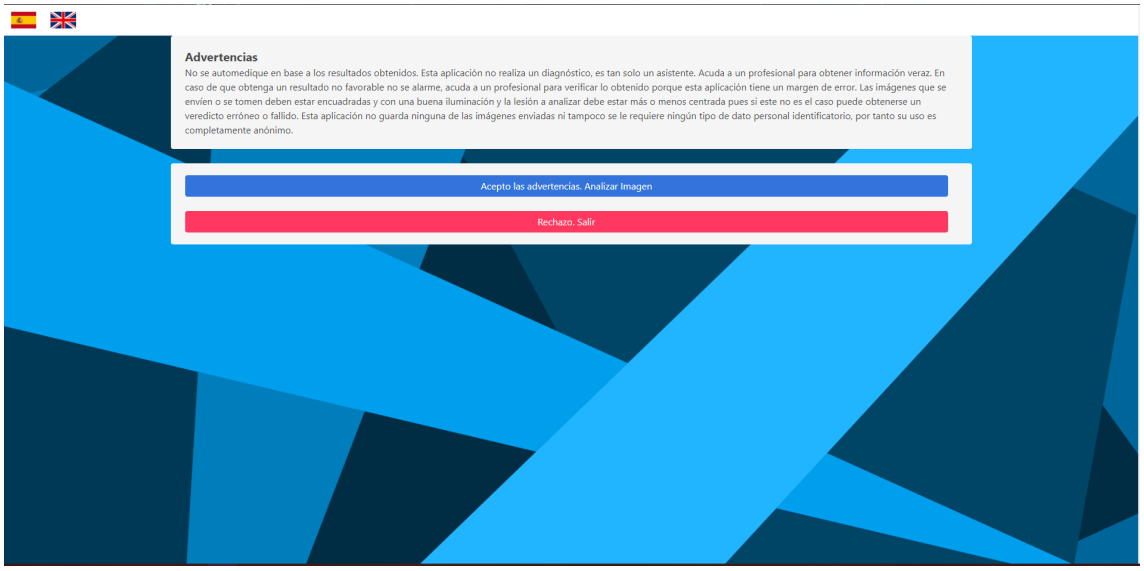


Figura A.9: Captura de la vista de advertencias

En esta vista se pueden leer las advertencias, si queremos realizar un análisis de una imagen deberemos pulsar sobre aceptar, sin embargo si las advertencias no nos parecen correctas o no queremos aceptarlas, podemos pulsar sobre salir, acción que nos llevará a la vista anterior.

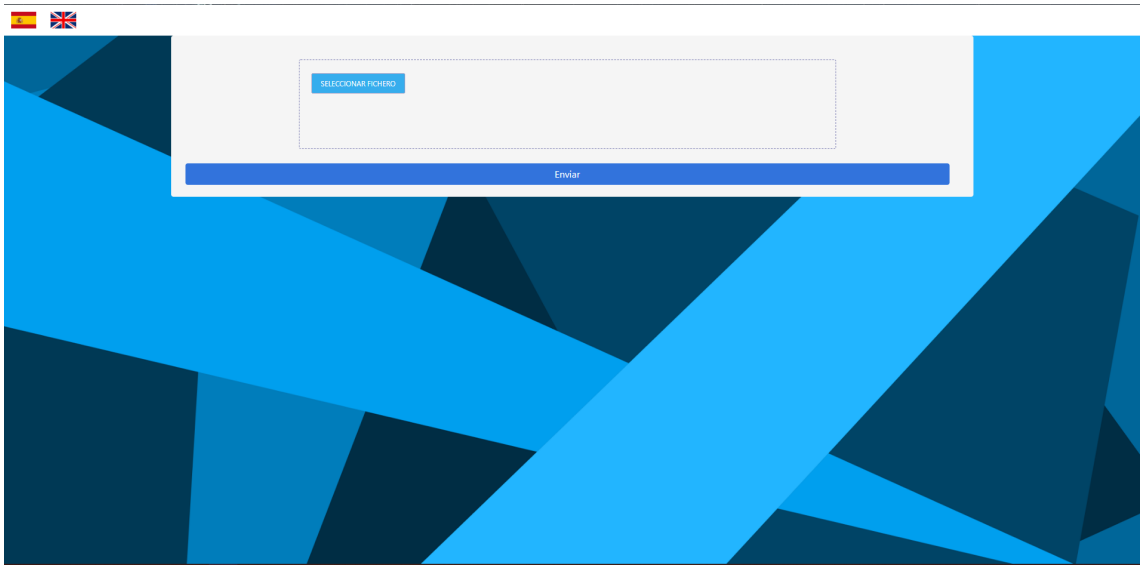


Figura A.10: Vista de análisis de imágenes

En esta vista podemos encontrar un formulario de estilo arrastrar y soltar en el cual si soltamos una imagen se subirá para ser analizada por el servidor, sin embargo, también tenemos un botón sobre el cual podemos pulsar y el cual nos abre un gestor de ficheros para que podamos seleccionar la imagen que queramos de nuestro sistema de ficheros para ser enviada al servidor. Una vez que se envíe la imagen se nos mostrará otra parte de la vista en la que podremos observar la imagen original, la imagen que ha sido analizada por el servidor y además de ello, el resultado. En caso de que el servidor no esté disponible se nos mostrará una advertencia de que el servidor no está disponible.

A.4. Manual de instalación

En esta sección se va a tratar todo lo referente a la instalación del proyecto.

Los requisitos para ejecutar el proyecto de una forma correcta son:

- Disponer de python 3.6+ instalado
- Cumplir los requisitos para usar tensorflow sobre GPU, en su defecto, se incluirá un fichero de requisitos para aquellos que no puedan ejecutar tensorflow-gpu

En primer lugar vamos a descargarnos el proyecto, para esto deberemos tener git instalado en nuestro ordenador, o en su defecto un programa capaz de descomprimir archivos .zip. El comando de git para descargar el proyecto es:

```
$ git clone https://github.com/MagicElyas/TFG.git
```

Tras clonar el repositorio, tendremos que crear un entorno virtual para el proyecto, para ello ejecutamos lo siguiente, una vez llevemos la terminal a la carpeta donde esté contenida el proyecto y ejecutaremos

```
$ pip3 install virtualenv
$ python3 -m venv venv
$ .\venv\Scripts\activate //en caso de windows
```

```
$ source venv/bin/activate //en caso de linux
```

Tras activar el entorno virtual instalamos todos los paquetes necesarios para la ejecución del proyecto. Sabremos si tenemos el entorno virtual activado cuando veamos (venv) antes de la linea de ejecución del terminal

```
$ pip install -r requirements.txt //con bpu
$ pip install -r requirements-no-gpu.txt //sin gpu
```

Tras esto, solo nos quedará ejecutar el siguiente comando

```
$ python start.py
```

Con este comando ya tendremos el servidor ejecutando en nuestra red local. El servidor viene con una versión de la aplicación de escritorio instalada, con lo cual podremos analizar imágenes desde el momento de la ejecución del servidor.

Para la instalación de las aplicaciones de escritorio y móviles la instalación es de la misma forma. Los requisitos para las 2 aplicaciones clientes son tener Node.JS instalado en el sistema. Una vez tenemos Node instalado, ejecutamos lo siguiente

```
$ git clone https://github.com/MagicElyas/TFG\
_apps.git //apps moviles
$ git clone https://github.com/MagicElyas/Desktop\
AppV2.git //app de escritorio
```

Tras bajarnos los repositorios de las aplicaciones, tendremos que mover la terminal usando cd hasta la ruta donde tengamos guardadas las aplicaciones, y una vez aquí ejecutaremos lo siguiente:

```
$ npm install
```

Con este comando lo que estamos haciendo es instalar todos los paquetes que necesitamos para ejecutarlo. Una vez se instalen los paquetes tendremos que abrir el fichero /src/app/shared/result.service.ts en el caso de la aplicación móvil y en la linea 80 tenemos que modificar la IP que se encuentra ahí y escribir la del servidor que se ejecuta dentro

de nuestra red para que la aplicación funcione correctamente, una vez que realicemos ese pequeño cambio, volvemos a la raíz del proyecto y ejecutamos lo siguiente

```
$ tns run
//app movil con el tel fono conectado v a USB
$ ng serve --open //app escritorio
```

Con esto ya tendríamos las aplicaciones ejecutándose, tanto en el teléfono, pues se nos instalaría la aplicación para no depender de estar conectado al ordenador para poder ejecutarse, y en el caso de la aplicación de escritorio tendríamos la aplicación ejecutándose en el navegador.

Bibliografía

[CPCA13] Eduardo Casilari Pérez and José Antonio Cortés Arrabal. *Breves notas de estilo para la redacción de Proyectos Fin de Carrera y Trabajos Fin de Grado.* ETSIT, Universidad de Málaga, 2013. `http://www.uma.es/media/files/Manual_de_Estilo_TFG_ETSIT.pdf`.

[Ent05] J.T. Entrambasaguas. *Ingeniería de Desarrollo de sistemas de Telecomunicación,* apuntes de clase. ETSIT-UMA, Curso 2004-2005.

[ETS01] ETSIT. *Programación docente.* ETSIT-UMA, Curso 2000-2001.

[ETS08] ETSI. MTS: Methods for Testing Specifications. `http://portal.etsi.org/mbs/Testing/testing.htm`, 2008.

[Lam99] Leslie Lamport. *LaTeX: a Document Preparation System.* AddisonWesley Longman, Inc, 2nd edition, August 1999. It includes user's Guide and Reference manual. ISBN: 0-201-52983-1.

[min14] Minted: a highlighted source code for latex. `https://code.google.com/p/minted/`, 2014.

[nnc17] Neural networks chart from asiimov fundation. `http://www.asimovinstitute.org/neural-network-zoo/`, 2017.

[NSC19] Native Script components guide. `https://docs.nativescript.org/ui/components`, 2019.

[Tsc18] Philipp Tschandl. The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions, 2018.

[Wik18] Wikipedia. Transferencia de estado representacional — wikipedia, la enciclopedia libre, 2018. [Internet; descargado 30-abril-2019].