Universidad de La Laguna
Escuela Superior de Ingeniería y Tecnología

Trabajo de Fin de Grado

Usando Redes Neuronales Convolucionales Para Convertir Características Visuales en Estímulos Sonoros

Using Convolutional Neural Networks to convert visual features into sound stimuli

Pablo Pastor Martín

La Laguna, 3 de septiembre de 2018

D. **Rafael Arnay del Arco**, con N.I.F. 78.569.591-G profesor Ayudante doctor adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

C E R T I F I C A

Que la presente memoria titulada:

"Usando Redes Neuronales Convolucionales Para Convertir Características Visuales en Estímulos Sonoros"

ha sido realizada bajo su dirección por D. **Pablo Pastor Martín**,

con N.I.F 79.084.379-Y.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 3 de septiembre de 2018

Agradecimientos

A Rafael, por su permanente disposición a ayudarme en todo lo posible y haberme ofrecido la posibilidad de hacer este TFG tan interesante, atractivo y bonito.

A mi familia, por su inestimable y constante apoyo a lo largo de toda mi vida, un apoyo que me ha permitido ser quien soy y que nunca me sienta solo, por muy duro que pueda ser el camino. Gracias de corazón, Mamá.

A mis compañeros Ángel, Cristian, Dani, Isaac, Nico y muchos otros, que hicieron que el madrugar a diario no fuera tan duro y que han hecho estos años más llevaderos con su simpatía y apoyo. Son lo mejor que me llevo de estos años, y espero que sólo sean el comienzo.

A Tina, por haberse preocupado siempre por mi TFG, por hacer mucho más amenos estos últimos días de estrés y por enseñarme que el mar no es quién para romper nada.

A María e Isis, que siempre han estado ahí para sacarme una sonrisa por muy estresado que estuviera, por su amistad de tantos años y por los que quedan por venir.

A todos ustedes, tengan mi más sincero agradecimiento.

Licencia

Resumen

Recientemente hemos asistido a un gran auge en el campo de la Inteligencia Artificial, producido en gran medida gracias a las Redes Neuronales. Numerosos campos de investigación han sido beneficiados de este auge, como el procesamiento de lenguaje natural textual y hablado, sistemas de recomendación y visión artificial.

Este trabajo se centra en este último campo y su objetivo ha sido investigar una nueva forma de utilizar las redes neuronales convolucionales para tratar imágenes y generar información útil al usuario.

De esta forma, se intenta, mediante los filtros intermedios de las redes convolucionales, extraer información de sus características y mostrársela al usuario por medio de estímulos sonoros.

Cabe destacar también que algunos de los enfoques que se pueden encontrar en este Trabajo de Fin de Grado son novedosos y forman parte de una línea de investigación más amplia que se pretende desarrollar en el futuro.

En concreto, en este Trabajo de Fin de Grado se han generado imágenes que maximizan la activación de estos filtros, se han creado descriptores de los mismos, los cuales han sido agrupados automáticamente sin supervisión y se ha creado una pequeña aplicación gráfica en Python que sirve de demostración de lo realizado en el desarrollo del trabajo.

Palabras clave: red neuronal, red neuronal convolucional, aprendizaje profundo, agrupamiento, histograma de gradientes orientados, descriptores, capa, filtro, Keras, Python, Keras-Vis, TkInter

Abstract

In the past few years, the Artificial Intelligence industry has gained a major importance because of artificial neural networks. Several active research fields have been benefited from this grow. Some examples of them are: Natural language processing, both textual and spoken; recommendation systems and artificial vision.

This work is focused on the last of them and its goal has been to research a new way of using convolutional neural networks to process images and generate useful information from them.

In this way, we tried to use the activation of some intermediate filters of a convolutional neural network to get some information of the visual features of an image given and show it to the final user via sound stimuli.

It is important to remark that some of the techniques used in this final work are groundbreaking and that this work is the first step on a line research, which is going to be further developed in the future.

In this Final Degree Project, we have generated images that maximize the activation of the above-named filters, created descriptors of them, which have been automatic clustered and developed a small python application with a graphical user interface, which allows the user to use and check the work made in this project.

Keywords: neural network, convolutional neural network, deep learning, clustering, histogram of oriented gradients, descriptors, layer, filter, Keras, Python, Keras-Vis, TkInter.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1
Introducción

En los últimos años hemos asistido a una verdadera revolución en el campo de la inteligencia artificial, produciéndose grandes avances en el reconocimiento de imágenes y voz, análisis de textos, traducción automática, recomendación de contenidos, etc. [1,2]

Estos avances producidos en la última década se deben en gran medida a las redes neuronales (O aprendizaje profundo) y a la mejora del hardware que hace posible su uso a gran escala. Así, las redes neuronales han demostrado ser la mejor opción conocida en una gran variedad de aplicaciones de la inteligencia artificial.

Una de las grandes características de esta tecnología es su capacidad de aprender de forma autónoma, sin ningún tipo de intervención humana a lo largo del proceso. Las redes neuronales, por tanto, son capaces de aprender en base a una información suministrada previamente sin que el desarrollador tenga que supervisar cómo lo hace.

Es por lo dicho en el anterior párrafo que se suele tratar a las redes neuronales como una suerte de caja negra, de la cual sólo importa su salida y no lo que sucede para que ésta se produzca. Por ejemplo: se quiere saber qué hay en la imagen suministrada, pero no cómo lo sabe la red neuronal.

Es aquí donde difiere este Trabajo de Fin de Grado, pues entraremos en el interior de las capas intermedias de una red neuronal convolucional destinada a reconocer objetos en imágenes y, una vez ahí buscaremos características útiles y llamativas para convertirlas en estímulos sonoros que puedan ayudar a mejorar la calidad de vida de personas con algún tipo de discapacidad visual.

En los siguientes capítulos se introducirán los objetivos pormenorizados del trabajo, así como los conceptos teóricos usados para la realización de éste. Posteriormente, se describirá el desarrollo del mismo, así como los resultados y conclusiones obtenidas.

1.1 Objetivos del Trabajo de Fin de Grado

Con la realización del trabajo anteriormente introducido se persiguen los siguientes objetivos:

1. Identificar las redes neuronales convolucionales que mejores resultados están dando en las tareas de clasificación de imágenes.
2. De entre las anteriores, seleccionar una para trabajar en base a criterios de simplicidad y utilidad.
3. Generación de una visualización de los filtros de dicha red.
4. Extraer las características HOG [3] (Histogram of Oriented Gradients) de los filtros obtenidos.
5. Realizar, mediante diferentes técnicas, un proceso de *clustering* entre los anteriores descriptores, buscando aquellos filtros que tengan características comunes.
6. Desarrollar un sistema que permita localizar activaciones de los diferentes filtros con una cierta imagen de entrada.
7. Seleccionar los filtros más interesantes
8. Desarrollar una interfaz que permita a un usuario con un ratón oír las características anteriormente obtenidas en base a los movimientos que haga con él.
9. Realizar una conexión entre dichas activaciones y una librería de generación de sonido.

1.2 Diferentes fases en el desarrollo

En el desarrollo encontramos cuatro fases, las cuales se detallan a continuación:

- **Fase de obtención de características:** Comprende los objetivos 1,2,3 y 4
- **Fase de clustering:** Comprende el objetivo 5
- **Trabajo con los tipos de filtros:** Comprende los objetivos 6 y 7
- **Interfaz de usuario:** Comprende los objetivos 8 y 9
- **Fase de documentación:** Consistente en documentar las tareas realizadas, así como redactar la presente memoria.

Capítulo 2
Antecedentes teóricos y estado del arte

En el presente capítulo se describirán las bases teóricas en las que se basa el Trabajo de Fin de Grado, además, se hablará del estado del arte en este campo.

Hablamos por consiguiente de las redes neuronales, las redes neuronales convolucionales, las técnicas de visualización de las mismas y los últimos avances en los campos citados.

2.1 Redes Neuronales

Antes de entrar en profundidad en las redes neuronales, cabe destacar que éstas son sólo una forma de aprendizaje automático, existiendo diversas técnicas que consiguen también aprender en base a unos datos dados sin la supervisión humana. Ejemplos de estas otras técnicas son: máquinas de soporte vectorial (SVM), árboles de decisión o redes bayesianas.

Las redes neuronales artificiales son un **modelo computacional** que trata de imitar el funcionamiento del cerebro de los animales, si bien las redes funcionan de una manera más abstracta y simple.

Su funcionamiento, en el que entraremos más en detalle, se podría simplificar de la siguiente forma: Al recibir una serie de entradas numéricas, se realizan una serie de cálculos internos que terminan dando como resultado otros datos numéricos que pueden representar cosas totalmente distintas.

En el caso de las imágenes resulta trivial obtener una entrada numérica (Las imágenes ya son datos numéricos internamente), y una salida podría ser la probabilidad de que en dicha imagen haya objetos de diferentes clases.

2.1.1 Historia de las Redes Neuronales

Sus orígenes se remontan a 1943, año en el que W. McCulloch y W. Pitts crearon un primer modelo de redes neuronales basándose en las matemáticas y algoritmos denominados lógica de umbral [4]. Poco después de sus avances, Donald Hebb explicó a grandes rasgos el aprendizaje neuronal, conociéndose su ley como la “regla de Hebb” [5], la cual es precursora de las técnicas de aprendizaje por parte de redes neuronales en la actualidad.

Posteriormente, en 1956, se produjo en Dartmouth la primera conferencia en la que se habló de la capacidad de la computación de simular el aprendizaje de los cerebros biológicos. Al poco de esto (1960) llegó la primera aplicación de las redes a problemas reales: La eliminación de ecos en líneas telefónicas mediante el Adaline (Adaptative

Linear Elements) y el Madaline (Multiple Adaptative Linear Elements).

Sin embargo, en 1969, con la publicación del libro “Perceptrons: An introduction to Computational Geometry” [6] por parte de Marvin Minsky y Seymour Papert, el interés por las redes neuronales cayó repentinamente dado que en el libro se demostraron importantes deficiencias en los modelos neuronales artificiales que se habían desarrollado hasta ese momento, sobre todo del perceptrón. A pesar de dicha caída la investigación no se frenó, destacando en 1977 el trabajo realizado por James Anderson y su extensión “Brain-State-in-a-Box” [7], la cual permitió modelar funciones de mayor complejidad.

El resurgimiento de las redes neuronales se produjo en 1985 con la publicación por parte de John Hopfield del libro “Computación neuronal de decisiones en problemas de optimización” [8]; un año más tarde, en 1986, David Rumelhart introdujeron en su versión actual el algoritmo de propagación hacia atrás, lo cual provocó un nuevo panorama mucho más alentador en las investigaciones y desarrollo de redes neuronales.

Desde entonces observamos numerosos esfuerzos en el campo de las redes neuronales, produciéndose avances tanto en software como en hardware.

2.1.2 Arquitectura de las redes neuronales

A continuación, se muestra la arquitectura genérica de una red neuronal artificial:

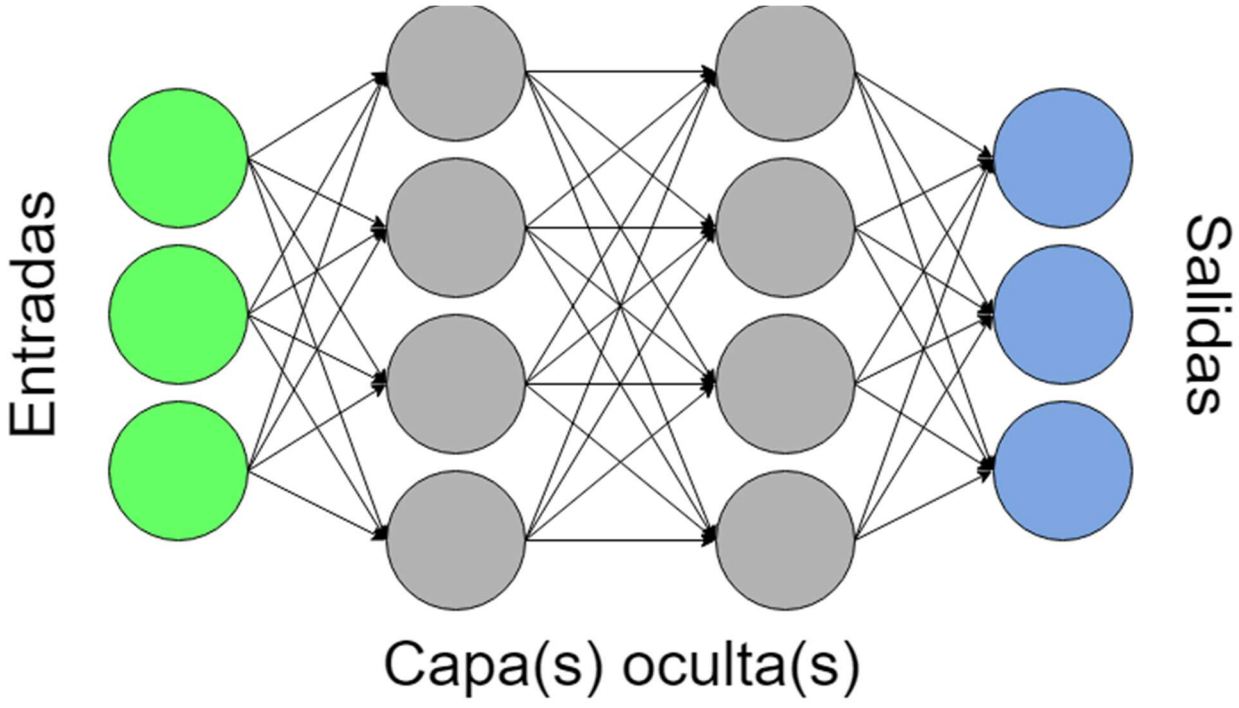

Figura 2.1: Arquitectura típica de una red neuronal artificial

La unidad básica de una red neuronal es **la neurona,** representadas por los círculos en la ilustración anterior. Dichas neuronas se organizan en capas de la siguiente manera:

- **Capa de entrada**: En el documento viene representada por las neuronas verdes. Estas neuronas son las que reciben los datos numéricos que la red procesará.
- **Capa de salida**: Son las neuronas azules en la ilustración anterior. Es la capa encargada de contener el resultado final de las operaciones matemáticas que se hayan producido en el interior de la red.
- **Capas ocultas**: Sus neuronas contienen los cálculos y valores de pesos intermedios de la red. Están representadas en gris en la ilustración.

Normalmente, aunque depende de la implementación concreta de cada red, todas las neuronas de una capa están conectadas con todas las neuronas de la siguiente capa. Es precisamente en estas conexiones entre neuronas donde encontramos los **pesos**, los cuales son utilizados en el cálculo principal de las redes neuronales: multiplicar el valor de salida de una neurona por el peso de una conexión, siendo el resultado de esta operación una entrada para la siguiente neurona.

2.1.3 Funcionamiento de las neuronas artificiales

Simplificando, una neurona no es más que una serie de entradas, un conjunto de pesos y una función de activación. El papel de la neurona es el de transformar dicha serie de entradas en un único valor de salida, el cual puede ser tomado como entrada de otras neuronas en capas superiores.

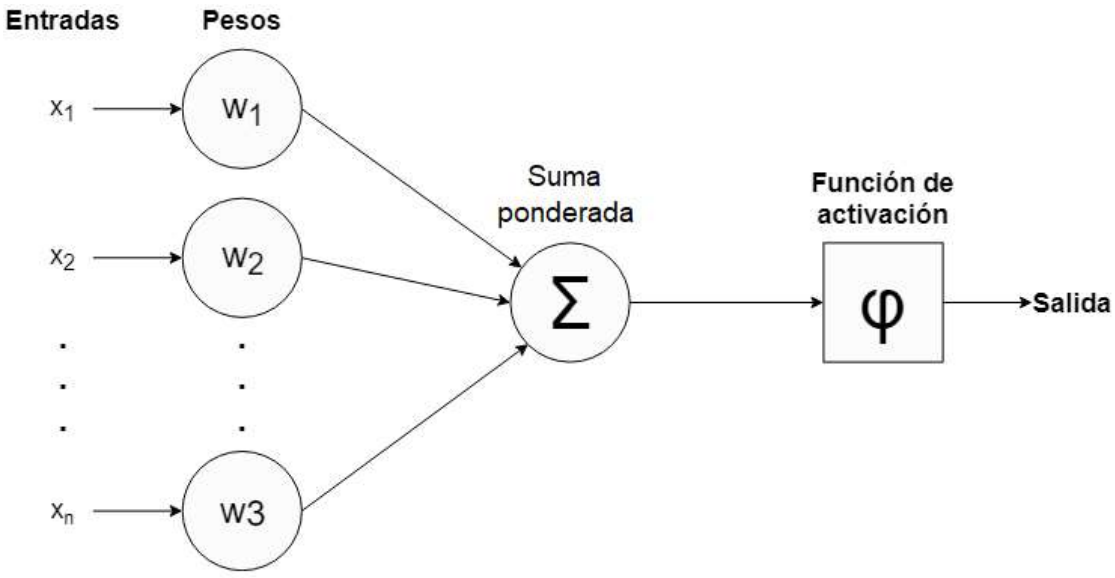

Figura 2.2: Modelo básico de una neurona artificial

Como se puede ver en la figura, nos encontramos con los siguientes elementos principales en la neurona:

- **Entradas:** Pueden ser los datos numéricos iniciales o las salidas (también

numéricas) de otras neuronas.

- **Pesos:** El factor de ponderación por el que se multiplicará la entrada correspondiente, estos pesos son propios de la neurona.
- **Salida:** El valor numérico que la neurona produce tras realizar diferentes cálculos.
- **Suma ponderada:** Cada entrada es multiplicada por su peso correspondiente, y se realiza el sumatorio de todas estas multiplicaciones. El valor resultante es la entrada a la función de activación. Al resultado del sumatorio se añade en ocasiones un **bias**. Que es, esencialmente, una nueva entrada con valor 1 y peso b.
- **Función de activación:** Su principal cometido es mantener los valores producidos por la neurona dentro de un rango razonable (En muchas ocasiones 0 y 1). Existen diferentes funciones de actuación. Separándose en 2 clases principales: Lineales y no lineales. Las lineales dan lugar a modelos sencillos y poco prácticos, por lo que se suelen usar funciones no lineales, destacando entre ellas la **gaussiana** y la **sigmoidal**.

2.1.4 Aprendizaje en redes neuronales

Una de las características más importantes de las redes neuronales, tal y como se dijo, es su capacidad de aprender autónomamente. Por ello, antes de usarlas en problemas reales, encontramos una **fase de aprendizaje** en la que la red aprende los valores (pesos y bias) más adecuados para producir la salida esperada por los usuarios.

El proceso de aprendizaje, también llamado **entrenamiento**, es por tanto un proceso **iterativo** donde los parámetros citados se van refinando paulatinamente partiendo de valores aleatorios con el fin de alcanzar un nivel de operación lo suficientemente bueno.

En la mayoría de los casos, de cara a realizar el entrenamiento, se encuentra una **función de pérdida** que de una medida del rendimiento de la red con unos pesos dados. Es por ello que el entrenamiento puede ser definido como el proceso que busca minimizar la función de pérdida.

En muchos casos, el algoritmo encargado de ayudarnos a buscar estos parámetros es el de **backpropagation** [9].

2.1.5 Algoritmo de Backpropagation

Este algoritmo es en realidad un método de aprendizaje supervisado en el que podemos distinguir 2 fases:

1. **Fase de propagación**: En esta fase se proporciona a la red una entrada determinada para que ésta se propague desde la primera capa hasta la última de las mismas, produciéndose por tanto una salida determinada. Posteriormente se compara dicha salida con la salida esperada mediante la función de pérdida -*también llamada de coste o error*-.
2. **Fase de adaptación:** Con el error calculado en la fase anterior, comienza una propagación inversa -*de la última hacia la primera capa*-. De manera que dicha propagación acaba llegando a todas las neuronas que contribuyen directamente a la salida. Cabe destacar sin embargo que cada neurona sólo recibe una porción del error correspondiente a su contribución relativa a la salida original. Sabiendo la contribución de cada neurona al error, se pueden ajustar sus pesos de cara a disminuir el error en futuras iteraciones. Es por ello que este algoritmo es efectivo entrenando redes neuronales, pues corrige los pesos (inicializados de forma aleatoria) para que se produzcan las salidas deseadas.

Durante el entrenamiento con *backpropagation*, se propaga por la red de forma sucesiva un conjunto de entrenamiento. Cada iteración completa del conjunto de entrenamiento a la red neuronal es llamada **época**. De esta forma, el proceso de aprendizaje se repite época tras época hasta que los parámetros se estabilizan y el rendimiento de la red converge a un valor aceptable.

2.1.6 Otras arquitecturas de Redes Neuronales

A pesar de haber presentado el modelo más común de red neuronal, debemos destacar la existencia de otros modelos con los que también se obtienen grandes resultados.

Un ejemplo de estas otras arquitecturas son las **redes neuronales recurrentes**. La principal característica de este tipo de red es que no tienen una estructura de capas como la presentada anteriormente, sino que permiten conexiones arbitrarias entre todas las neuronas, llegando incluso a permitir la existencia de ciclos. Esta característica aporta a la red incorporar el concepto de temporalidad y memoria, haciendo de este tipo de red la mejor opción cuando queremos realizar análisis de textos y procesamiento de voz, así como de música.

Otra arquitectura ampliamente usada de redes neuronales es la de **redes neuronales convolucionales**, con las cuales se ha trabajado en este Trabajo de Fin de Grado, motivo por el cual se le dedicará un apartado propio donde se introducirán en profundidad.

2.2 Redes Neuronales Convolucionales

Las redes neuronales convolucionales son especialmente utilizadas en el campo de la

visión artificial, puesto que sus neuronas se corresponden a campos receptivos de forma similar a las neuronas de la corteza visual primaria de un cerebro biológico.

Es digno de mención que esta arquitectura de red neuronal es una variación de la vista en el anterior apartado (perceptrón multicapa), sin embargo, gracias a que su aplicación es realizada en matrices bidimensionales, son especialmente efectivas en clasificación y segmentación de imágenes, así como en cualquier tipo de datos donde los mismos estén distribuidos de forma continua a lo largo de la entrada.

2.2.1 Historia y Fundamentos Biológicos de las Redes Neuronales Convolucionales

El origen de este tipo de red se encuentra en el Neocognitron [10], introducido por Kunihiko Fukushima en 1980. Dicho modelo fue mejorado por Yann Lecun en 1998, pues introdujo el aprendizaje basado en *backpropagation* [11]. En el año 2012 este tipo de redes fueron refinadas por Dan Ciresan y fueron implementadas en GPU, consiguiendo un rendimiento computacional mejor a los obtenidos hasta entonces [12]. Además, la calidad de los resultados resultó mucho mayor a la de otras técnicas previas en clasificación de imágenes como máquinas de soporte vectorial o cascadas de Haar.

Como se dijo anteriormente, esta arquitectura tiene una clara inspiración en la corteza visual del cerebro. Esta inspiración se debe en gran medida al trabajo realizado por Hubel y Wiesel en 1959 [13], gracias al cual se comprendió en gran medida el funcionamiento de la corteza visual, sobre todo de las células responsables de la selectividad de orientación y detección de bordes en los estímulos visuales.

Hubel y Wiesel descubrieron dos tipos de neuronas: simples y complejas, las cuales, si bien son distintas, comparten la característica de excitarse si el estímulo visual que reciben está alineado con los patrones que estas células tienen. Ésta será una característica que encontraremos también en las células de este tipo de arquitectura.

2.2.2 Arquitectura de las Redes Neuronales Convolucionales

Las redes neuronales convolucionales consisten en múltiples capas de filtros convolucionales de una o más dimensiones. Tras cada capa se suele añadir una función para realizar un mapeo causal no-lineal.

En su versión más común, dedicada generalmente a la clasificación, encontraremos al principio una fase de extracción de características. Esta fase está compuesta de neuronas convolucionales y de reducción de muestreo. Por el contrario, al final de la red veremos neuronas perceptrón sencillas para realizar la clasificación final sobre las características extraídas.

En la primera de las fases, y a medida que progresan los datos a través de la misma, se disminuye la dimensionalidad, lo cual hace que las neuronas de las capas lejanas sean mucho menos sensibles a las perturbaciones en los datos de entrada, sin embargo, estas neuronas son activadas por características cada vez más complejas.

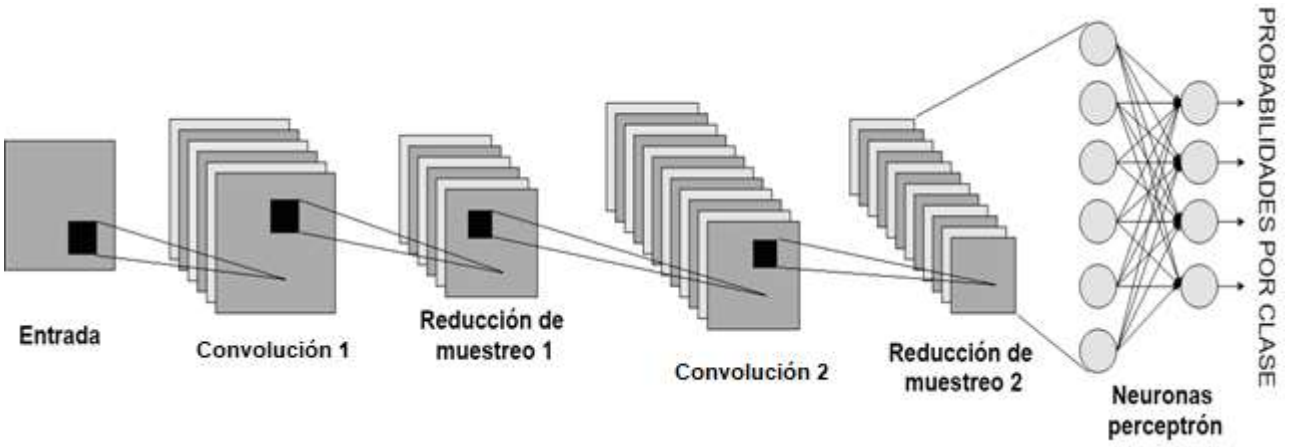

Figura 2.3: Estructura clásica de una red convolucional para clasificación

2.2.3 ¿Qué es una convolución?

El principal objetivo de la operación de convolución aplicada a las redes neuronales convolucionales es el de extraer características de la imagen de entrada.

La convolución, a grandes rasgos, es el proceso de añadir cada elemento de la imagen ponderado por un núcleo a sus vecinos locales.

El núcleo -*kernel*- previamente nombrado es una matriz cuadrada de tamaño impar e igual o menor a la imagen de entrada. Cada kernel es útil para determinadas tareas, como detección de bordes horizontales, verticales, etc.

De cara a realizar la convolución se realizan los siguientes pasos, nótese que se realiza la operación sobre un fragmento de la imagen original del mismo tamaño al del núcleo:

1. Se voltea el núcleo tanto horizontal como verticalmente. Si el núcleo es simétrico, de forma obvia este paso no tiene ninguna consecuencia.
2. Se multiplica cada elemento de la matriz núcleo por su similar local de la imagen.
3. Se suman todos los resultados de las multiplicaciones del paso anterior.
4. El elemento central del fragmento tomado de la imagen obtendrá el valor que se obtuvo en el paso 3.

Este proceso se va repitiendo, colocando el centro del núcleo sobre cada elemento de la imagen.

Sin embargo, se pueden encontrar problemas en los bordes de la imagen, pues se necesitaría multiplicar elementos del kernel con elementos que no existen en la imagen, pues están más allá de los bordes. Para solventarlo tenemos varias alternativas:

- Extender los bordes de la imagen tanto como sea necesario

- Tomar los píxeles necesarios del otro extremo de la imagen+
- Ignorar los cálculos más allá de los bordes.

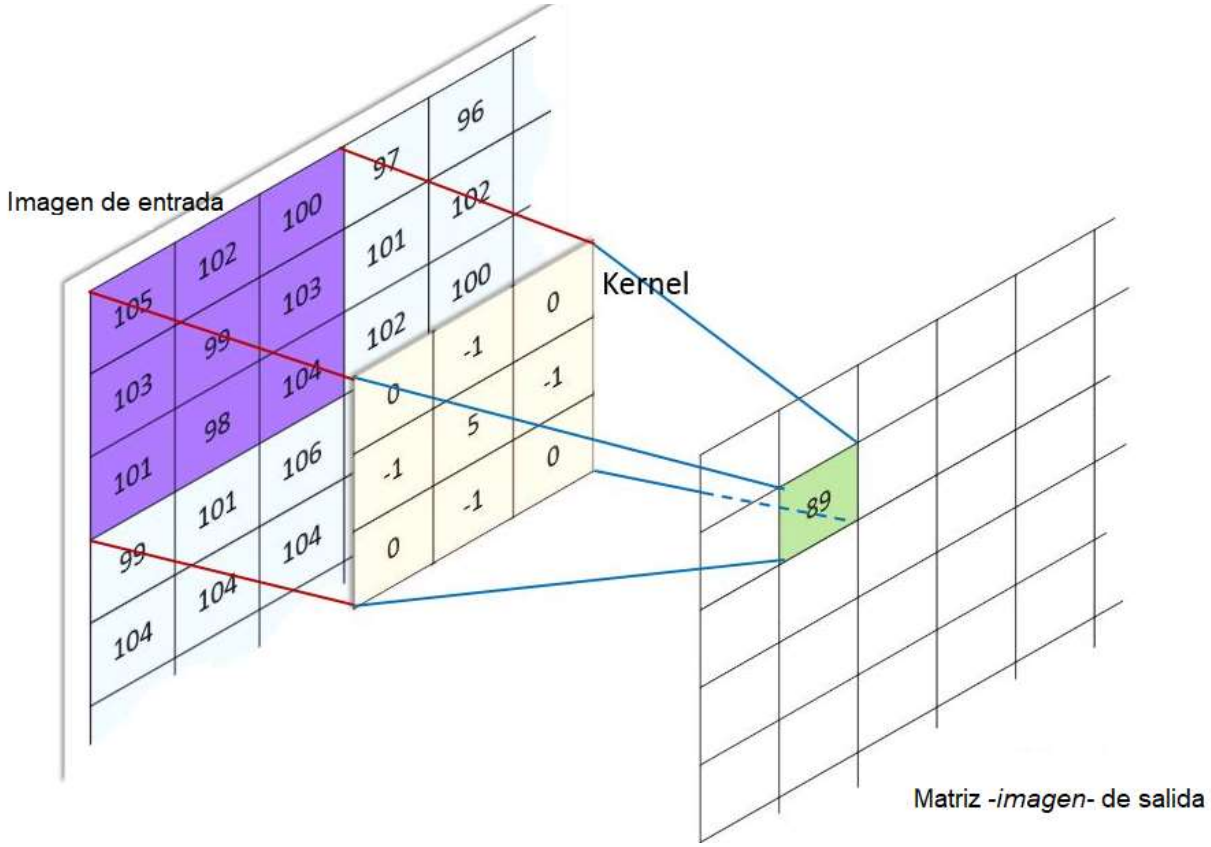

Figura 2.4: Ejemplo de operación de convolución

2.2.4 Tipos de Capas y Neuronas en Redes Convolucionales

Como se introdujo en el apartado *2.2.2*, normalmente encontramos en este tipo de redes 3 capas distintas: de convolución, de reducción de muestreo y de clasificación. A continuación, describiremos cada una de ellas:

De convolución: En estas capas se encuentran las neuronas de convolución, cada una de estas neuronas tienen un kernel -*generalmente distinto*-, y encontramos numerosas de estas neuronas por capa. Adicionalmente, cada neurona aplica la operación descrita en el apartado *2.2.3*, produciendo una nueva imagen como resultado.

Es muy común que en este tipo de neuronas se utilice una función de activación llamada **ReLU [14]**, por las siglas en inglés de unidad lineal rectificada. Y su cometido es el de evitar números negativos en la salida de la neurona.

Esta función de activación, en su versión más simple se define como:

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

Sin embargo, también existe una versión más suave, llamada *softplus* y definida como:

$$f(x) = \ln(1 + e^x)$$

De reducción de muestreo: La razón de la existencia de estas capas es la creencia de que la localización exacta de una característica es menos importante que su localización aproximada con respecto a otras características. De esta forma, la capa de reducción de muestreo permite reducir paulatinamente el consumo espacial de la representación de la imagen, el número de parámetros y el costo computacional de la red. Además, permite reducir el sobreajuste y aumentar la invariancia frente a la traslación.

Hay diversas funciones de reducción de muestreo no lineales entre las que elegir a la hora de implementar este tipo de capas en redes convolucionales, destacando por ser la más utilizada la técnica de *max pooling,* la cual se basa en elegir, de entre la región seleccionada, la activación máxima. Otras formas, también usadas son *average pooling* y *L2-norm pooling*.

La forma más común de este tipo de capas trabaja con filtros 2x2 con un salto de 2 elementos, consiguiendo por tanto reducir el muestreo en un 75% de las activaciones al realizar cada operación sobre 4 elementos.

Cabe mencionar a su vez que la operación de reducción de muestreo -*o pooling*- actúa de forma independiente en cada capa de la entrada.

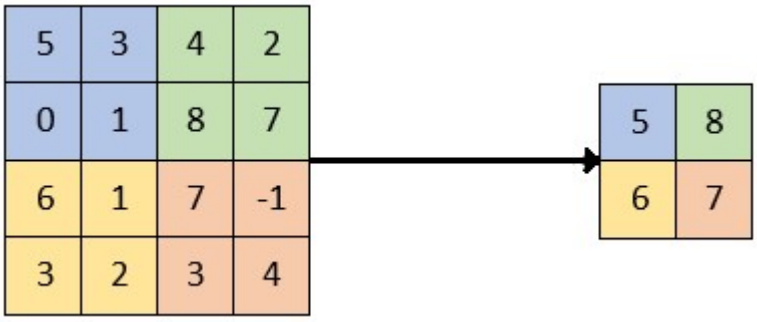

Figura 2.5: Ejemplo de Max Pooling

De clasificación: Este tipo de capas cuentan con neuronas como las vistas en el anterior apartado (2.1), suelen recibir las características de las capas convolucionales y de pooling para decidir qué objetos puede haber en una imagen -*en el caso de la clasificación de objetos en imágenes*-.

Es importante mencionar la tendencia a eliminar este tipo de capas en algunas redes neuronales convolucionales, en este tipo de redes -*llamadas FCN (Fully Convolutional Network)*- el proceso de aprendizaje se hace en su totalidad mediante filtros, incluso

en las capas de clasificación finales. Además, las redes totalmente convolucionales pueden manejar entradas de tamaño variable, mientras que aquellas redes con neuronas perceptrón no son capaces de hacerlo.

2.3 Visualización de características

Dado el gran potencial de las redes neuronales convolucionales, surge la duda de cómo funcionan internamente para obtener unos resultados tan satisfactorios. Ante la necesidad por tanto de que las redes neuronales sean interpretables por seres humanos, surgió el campo de investigación de la interpretabilidad de redes neuronales, el cual con el tiempo se ha dividido en dos grandes áreas: **Visualización de características** y **atribución.**

La **atribución** estudia qué parte de una entrada es responsable de que la red se active de una determinada forma. Mientras tanto, **la visualización de características** responde a la pregunta de qué buscan las diferentes partes de la red para tomar decisiones.

Una primera aproximación muy general a este concepto es entender que sabiendo que las redes neuronales modifican su comportamiento en función de la entrada, podremos utilizar derivadas de los resultados para ir progresivamente optimizando la entrada de cara a maximizar la activación de una neurona o el resultado final de la red. Este proceso es similar al seguido al entrenar cualquier red neuronal.

Además, deberemos seleccionar qué es exactamente lo que queremos entender o maximizar, encontrando diferentes alternativas para ello:

- Neurona: Buscaremos optimizar una neurona (Características individuales) en una región dada de la imagen.
- Canal: Buscamos optimizar la imagen entera.
- Capa: Se optimiza todos los canales (filtros) de una capa simultáneamente.
- Una cierta clase: Busca obtener imágenes que maximizan la probabilidad de salida de una cierta clase.

Cabe mencionar que existen más alternativas, pero las mencionadas son, por norma general, las más comunes.

El hecho de utilizar optimización, y no mirar de entre posibles imágenes de un *dataset* cuáles maximizan una activación determinada radica en la capacidad de la optimización de descubrir qué es lo que realmente está buscando la red, separando lo que busca en realidad de lo que simplemente está correlacionado.

Sin embargo, la optimización plantea también una serie de retos los cuales trataremos a continuación.

2.3.1 Diversidad

La optimización suele darnos sólo un ejemplo tremendamente optimizado, el cual puede no representar todos los posibles escenarios en los que se conseguiría una activación. En contraste a ello, usando varias imágenes de algún conjunto de datos, podríamos conseguir varios de esos escenarios al ver imágenes diferentes pero que activan de forma similar un determinado objetivo.

Un ejemplo de ello podría ser un clasificador de gatos. Para que dicho clasificador funcionara de forma correcta debería detectar tanto primeros planos de estos como imágenes más amplias, aun siendo visualmente distintos ambos tipos de imágenes.

Para conseguir diversidad mediante un proceso de optimización se han realizado diversos enfoques:

Una primera idea fue optimizar teniendo como imagen inicial el centroide de un clúster con las imágenes que produjeron las activaciones necesarias.

Más tarde se intentó iniciar la optimización en diferentes ejemplos de entrada que satisfacían el objetivo a maximizar, lo cual tuvo un éxito limitado. También se ha trabajado con un modelo generativo, el cual ha demostrado dar muy buenos resultados.

Una aproximación más simple es añadir un término de diversidad a la función a optimizar, penalizando por tanto que los resultados sean similares entre sí.

Cabe destacar, de todas formas, que existen neuronas que responden a mezclas de ideas no relacionadas entre ellas, lo cual puede llevar a pensar que la neurona no es el elemento adecuado para entender las redes neuronales.

2.3.2 Problemas para una correcta visualización de las características

La aproximación anteriormente comentada en el punto 2.3 suele no ser suficiente de cara a obtener buenos resultados, pues por norma general se alcanzan unos resultados llenos de ruido y con patrones de alta frecuencia (Los cuales se pueden observar en la ilustración 2.6).

Estos patrones, si bien no está claro el porqué de su formación, parecen estar relacionados con la serie de convoluciones y capas de reducción de muestreo, la cual provoca patrones de alta frecuencia en el gradiente.

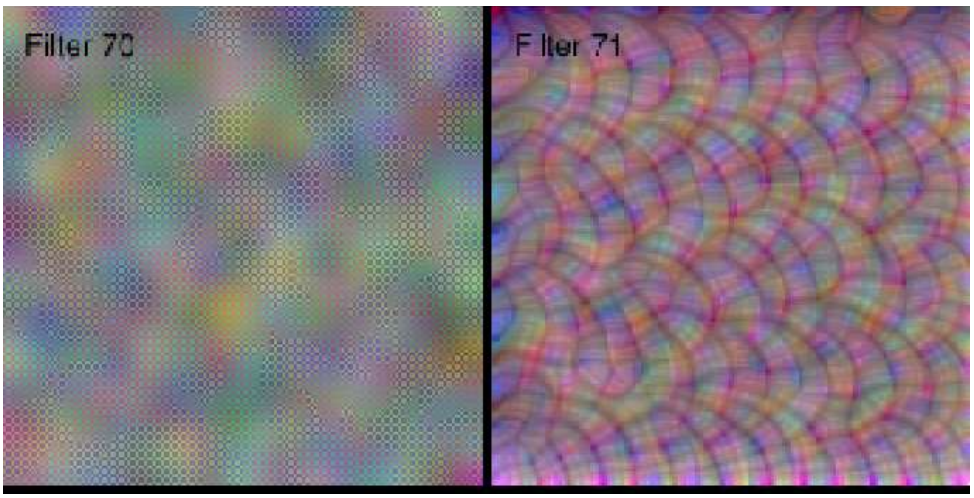

Figura 2.6: Diferencia entre filtro con patrones de alta frecuencia (70) y sin ellos (71)

Evitar estos patrones ha sido uno de los grandes retos de la investigación en este campo, pues de cara a poder interpretar correctamente los resultados obtenidos, es mejor evitar este comportamiento. De hecho, podemos encontrar tres enfoques diferentes:

- **Penalización de frecuencia**: Este método se centra directamente en el problema de la alta frecuencia. Por ello, penaliza explícitamente la variación entre píxeles vecinos o penaliza el ruido de alta frecuencia mediante un emborronamiento en cada paso de optimización. El principal problema de este método es que también penaliza los cambios de intensidad legítimos.
- **Robustez frente a transformaciones**: Se basan en el intento de encontrar ejemplos que activen el objetivo de la optimización, aunque se transformen ligeramente. Incluso una pequeña variación ha demostrado ser efectiva en el caso de las imágenes. Ejemplos de estas variaciones pueden ser: traslaciones, rotaciones y escalados.
- **Características previamente aprendidas:** Bajo este método se busca fabricar un modelo basándose en datos reales para forzar al resultado a ajustarse a dicho modelo. Con ello se suelen conseguir resultados más realistas, pero es difícil distinguir cuál es el producto del modelo y cuál el de la optimización. Diferentes formas de conseguir el modelo utilizado han sido propuestas, pero no entraremos en detalles de las mismas.
- **Precondicionamiento y parametrización:** Consiste en cambiar el gradiente usado, lo cual puede cambiar el ritmo y la dirección de la optimización, pero no la ubicación de los mínimos locales y globales. Es común usar un espacio decorrelacionado para este tipo de problemas.

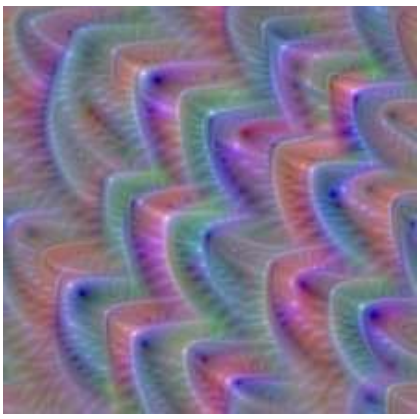

Figura 2.7: Mismo filtro (70), pero sin patrones de alta frecuencia

Como conclusión a este apartado, cabe mencionar que, si bien la visualización de características nunca podrá darnos un entendimiento pleno de interior de las redes neuronales convolucionales, sí pueden ser entendida como un paso inicial de cara a conseguir dicho objetivo.

2.4 Estado del Arte

La inteligencia artificial es un campo en constante evolución, innovación y mejora, es por ello que en los últimos años se hayan sucedido numerosas nuevas técnicas que mejoraban los resultados previos.

Para hablar del estado del arte de los campos que abarca este trabajo nos centraremos en diferentes subcampos.

2.4.1 Redes Neuronales Convolucionales

Las redes neuronales convolucionales han visto su uso extendido en los últimos años. Además, como vimos, sus aplicaciones son variadas y abarcan clasificación de imágenes, detección de objetos, segmentación de los mismos, etc.

En el campo de la clasificación de imágenes se ha avanzado enormemente desde la introducción en 2012 del modelo “AlexNet” [15]. Desde aquel modelo, que conseguía un 57% de precisión en aquel entonces, se ha avanzado a un escenario en el que los principales modelos se mueven en torno al 75% de precisión en el dataset de ImageNet (Top-1).

Más detalles sobre las cifras recién expuestas [16] pueden ser encontrados en la Tabla 1.

Modelo	Desarrollador	Precisión
AlexNet	BVLC	57.1%
SqueezeNet	DeepScale	59.5%
Inception v1 (GoogleLeNet)	BVLC / Google	67.9%
Inception v2	Google	72.2%
Inception v3	Google	76.9%
ResNet 50	Microsoft	75.3%
ResNet 152	Microsoft	77.0%
Inception-ResNet-v2	Google	79.79%
VGG-16	Oxford	70.5%
VGG-19	Oxford	71.3%

La investigación se centra en descubrir nuevas arquitecturas de redes que puedan mejorar los resultados obtenidos hasta la fecha. Fruto de este esfuerzo, tecnologías distintas como Inception o ResNet (De Google y Microsoft respectivamente) han llegado a ser fusionadas en Inception-ResNet [17], lo cual ejemplifica el alto grado de colaboración en la investigación de este campo.

2.4.2 Visualización de características

Esta rama, de más reciente introducción, está viendo centrada su investigación en reducir el ruido de alta frecuencia introducido previamente en este trabajo. Es por ello por lo que en la última década se hayan publicado numerosos artículos buscando combatir este problema.

En 2015 Mahendran y Vedaldi [18] introdujeron un regularizador de la variación total. Asimismo, ese mismo año Nguyen [19] introdujo el desenfoque y explora los contraejmplos. Ambos métodos se basan en la penalización de la frecuencia.

A la penalización de frecuencia se le añadió en años posteriores, a partir de 2016 la robustez contra transformaciones, ejemplos de estos trabajos los encontrados en el trabajo de Tyka y Mordvintsev [20].

Estos esfuerzos no fueron los únicos, ya que como observamos anteriormente, también existe la opción de trabajar con características ya aprendidas. Esta opción se vio explorada en los trabajos de Nguyen principalmente.

Sin embargo, y puesto que el problema sigue estando presente, aunque con menor gravedad, se siguen investigando posibles soluciones al mismo.

2.4.3 Sistemas similares al propuesto

Puesto que la inteligencia artificial ha demostrado un gran potencial a la hora de mejorar la vida de las personas, no es de extrañar que aparezcan en el mercado diversos productos que la usen con fines de paliar algunas de las dificultades que viven a diario las personas discapacitadas.

Un ejemplo de lo anterior lo encontramos en los discapacitados visuales, a los que diversos productos intentan ayudar desde diversos enfoques. Desgraciadamente la gran mayoría de estas soluciones son de software propietario, por lo que no podemos conocer en profundidad la tecnología que utilizan.

Una muestra de estos productos es *Seeing AI* [21] de Microsoft, el cual es una app que narra al usuario diferente información en base a una inteligencia artificial subyacente de la cuales no se dan más detalles.

Otro ejemplo, más aproximado al objetivo de este trabajo es *HORUS* [22], de la StartUp suiza *Eyra*. El producto es un wereable, el cual usa un procesador Tegra K1 de Nvidia para procesar la imagen proveniente de sus cámaras y dar información sonora al usuario a través de vibraciones directamente trasladadas a los huesos del mismo. Lamentablemente, no comparten los detalles de su tecnología.

Capítulo 3
Herramientas Utilizadas

De cara a cumplir los objetivos mencionados y las actividades que lo permitan, se han utilizado herramientas que han facilitado el desarrollo del trabajo.

En este capítulo se introducirán dichas herramientas y se dará una visión global de las mismas, con el fin de poder explicar de mejor manera la metodología empleada en la realización del trabajo.

3.1 Keras

Keras [23] es un framework de Python que facilita enormemente la creación de modelos de *Deep Learning*. Su funcionamiento se apoya en librerías subyacentes, entre las cuales el usuario puede decidir (*TensorFlow [24] o Theano [25], principalmente)*.

Entre las bondades de este framework está su forma de aunar simplicidad con versatilidad, puesto que poner en funcionamiento una red neuronal requiere de muy pocas líneas de código, pero permite entrar en profundidad en el modelo y da una gran libertad al desarrollador para adaptar el funcionamiento a sus necesidades concretas.

El éxito de este framework ha llevado a que se haya visto integrado en el núcleo de *TensorFlow*, lo cual demuestra su valía y popularidad.

3.2 Keras Visualization Toolkit (Keras-vis)

Este conjunto de herramientas [26] es una librería que funciona encima de Keras y proporciona a los usuarios algunas funcionalidades que Keras no incluye por defecto y que sería tedioso añadir manualmente.

Algunas de las funcionalidades que aporta son:

- Visualización de filtros convolucionales
- Visualización de capas densas (Neuronas tradicionales)
- Mapas de atención

3.3 OpenCV

OpenCV [27] es la librería de análisis y tratamiento de imágenes y vídeo por antonomasia, y su abanico de algoritmos implementados y funciones es

enormemente amplio. Entre este abanico encontramos funciones tan variadas como binarización de imágenes, generación de descriptores SURF y HOG, sustractores de fondo, etc.

Aunque tradicionalmente esta librería ha estado enfocada para ser usada con C++ (Lenguaje en el que está escrita), debido al auge de Python en el campo de la inteligencia artificial, encontramos una interfaz para Python de OpenCV, que es la cual hemos usado para este trabajo.

3.4 Scikit-learn (Sklearn)

Scikit-learn [28] es una librería para Python que incluye una serie de herramientas de minería y análisis de datos. Subyacentemente usa NumPy (Una conocida librería de cálculos matemáticos escrita en C para ser usada en Python). Los trabajos que se pueden realizar con esta librería son los siguientes:

- Clasificación
- Regresión
- Clustering
- Reducción de dimensionalidad
- Preprocesamiento

Capítulo 4
Desarrollo y Metodología

Gracias a los anteriores capítulos han quedado definidos los cimientos en los que se sustenta el trabajo de fin de grado. En este capítulo describiremos los trabajos realizados y los resultados que se han ido obteniendo por el camino, así como las decisiones que se han ido tomado para afrontar diversos problemas y las razones que llevaron a dichas decisiones.

Los códigos programados para realizar las distintas tareas que se exponen en este capítulo, así como gran parte de los resultados de los mismos (Visualizaciones de lis filtros, visualizaciones de clases, resultados de clusterings, etc.) pueden ser encontrados en el repositorio de GitHub de este Trabajo de Fin de Grado, encontrado en [31].

4.1 Fase de obtención de características

Esta primera fase comprende las acciones necesarias para reunir el conocimiento y archivos necesarios para realizar el trabajo en fases posteriores.

Tras las labores de documentación llevados a cabo en la sección anterior, fue momento de utilizar dichos conocimientos para realizar diversas acciones:

4.1.1 Seleccionar un modelo para trabajar en base a criterios de simplicidad y utilidad

Debido a que el objetivo del trabajo no abarca el crear un nuevo modelo de red neuronal, se decidió utilizar un modelo ya conocido para, sobre él, realizar el trabajo. Cabe destacar que últimamente, a la par que sus resultados mejoran, las redes neuronales están viendo aumentado su número de capas y complejidad.

Dicha complejidad presentaba un reto, puesto que nuestro objetivo es analizar los diferentes filtros y capas de la red. Por ello, nuestro objetivo fue encontrar un modelo que tuviese una precisión suficientemente buena, pero sin entrañar una extraordinaria complejidad.

Después de un análisis de las alternativas, nos decantamos por el modelo **VGG16**, desarrollado por el equipo de VGG para la competición ILSVRC-2014.

El motivo que nos llevó a tomar esta decisión es un rendimiento razonablemente alto (70.5% de precisión) aunado con una simplicidad elevada comparado con otros modelos como InceptionV3.

Para ilustrar esta diferencia, podemos ver las ilustraciones *4.1 y 4.2*.

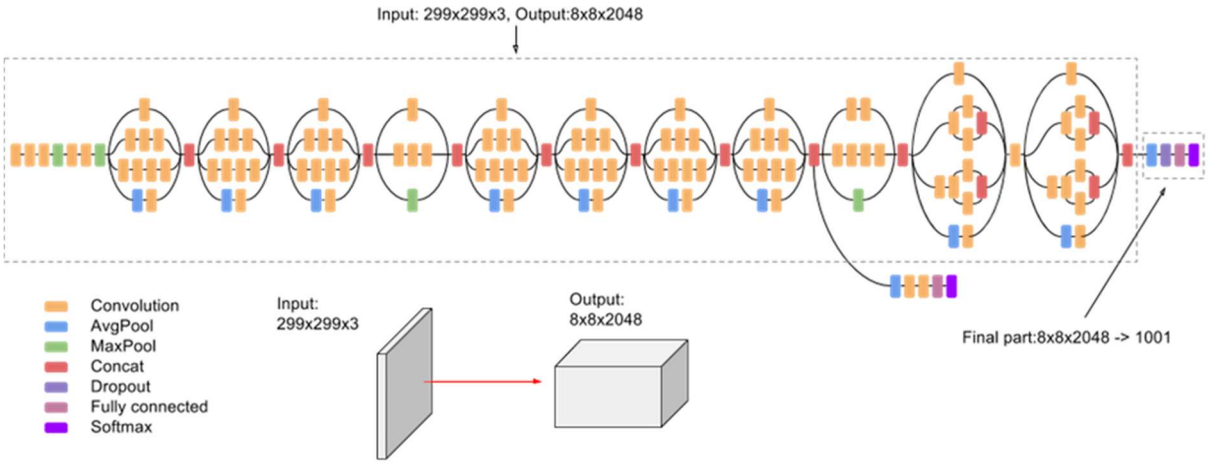

Figura 4.1. Modelo de InceptionV3

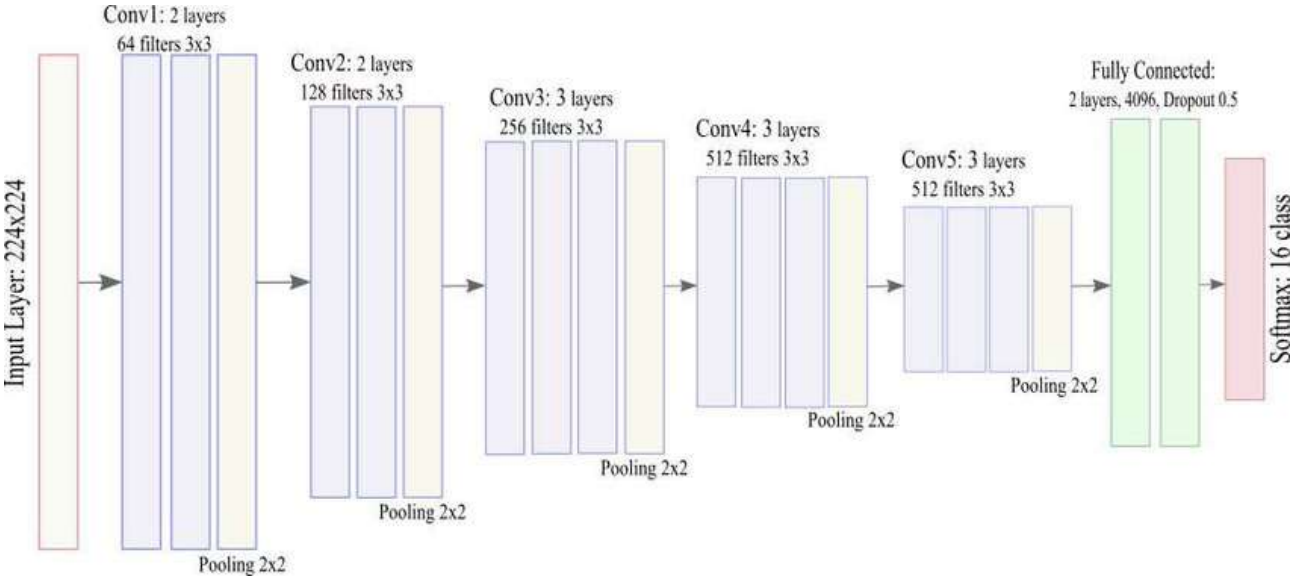

Figura 4.2. Modelo de VGG16

Como se puede observar, la diferencia en cuanto a simplicidad es abismal a favor de VGG16.

Al ser el modelo escogido, pasaremos a explicar su estructura. Encontramos en esencia 3 tipos de capas, correspondientes a las clásicas (Convolucionales, reducción de muestreo y tradicionales), además, vemos 6 bloques principales. Los 5 primeros de ellos contienen capas de convolución con una capa de reducción de muestreo, mientras que el último contiene 2 capas sucesivas de neuronas tradicionales.

Cabe mencionar también que todos los filtros son de tamaño 3x3 y se distribuyen de la siguiente manera:

Número de bloque	Número de capas convolucionales	Número de filtros por capa
1	2	64
2	2	128
3	3	256
4	3	512
5	3	512

Tabla 1. Estructura de VGG16

Múltiples son las ventajas de que un modelo sea menos complejo. Por un lado, simplifica el proceso de obtención de visualizaciones correctas y de trabajo con los resultados, pero también reduce drásticamente el tiempo de cómputo requerido para generar dichas visualizaciones que, aun ejecutándose en GPU, como veremos más adelante, consumen una gran cantidad de tiempo.

4.1.2 Generación de una visualización de los filtros de dicha red

Con el objetivo de conseguir unos primeros resultados del proyecto, ejecutamos mediante el framework *Keras-Vis* una primera optimización de los filtros. Esta primera ejecución evidenció un problema que nos acompañaría el resto del trabajo: el elevado tiempo de ejecución de dicha optimización.

Bloque	Tiempo de proceso por capa
1	3 minutos
2	10 minutos
3	40 minutos
4	120 minutos
5	280 minutos

Tabla 2. Tiempo de ejecución de la optimización por capa

Como se puede calcular mediante los tiempos apreciables en la Tabla 2, obtener todos los filtros de la red supone un tiempo total por encima de las 23 horas. Este elevado

tiempo de ejecución hizo necesario reducir el número de ejecuciones, así como el asegurarse antes de las mismas de lo modificado previamente.

Tras esta primera toma de contacto con el framework, obtuvimos resultados razonablemente buenos. Sin embargo, encontramos, como se podría prever tras lo dicho en la introducción teórica, con algunos filtros que no convergieron correctamente, lo cual se puede observar en la *Figura 4.3*.

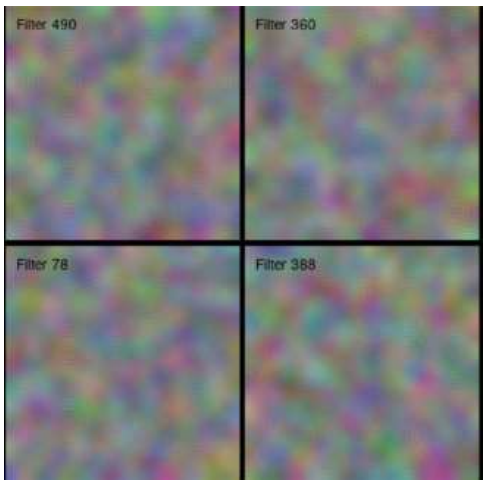

Figura 4.3. Malas convergencias en filtros de la tercera capa del filtro número 5

Para corregir esta situación, se adoptaron dos medidas principales, las cuales fueron:

- Añadir un pequeño *Jitter* (Vibración) para mejorar la robustez frente a transformaciones. Mediante esta técnica conseguimos mejorar los resultados anteriores.

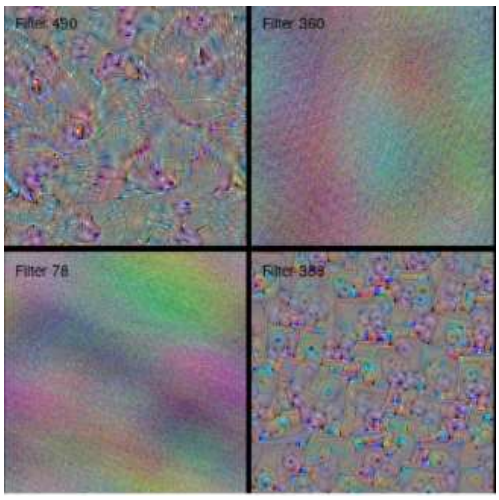

Figura 4.4. Mejoras tras añadir vibración

- Además, se decidió añadir un segundo proceso de maximización, el cual parte

de las imágenes generadas en la primera iteración. Esto, obviamente, aumenta enormemente (duplica) el tiempo de ejecución, pero también los resultados obtenidos.

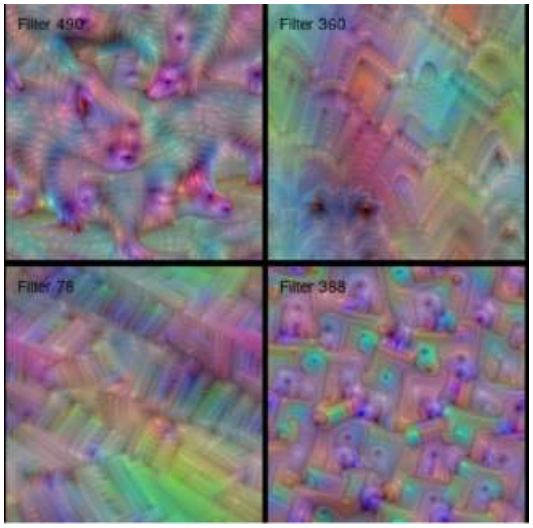

Figura 4.5. Resultados tras una segunda iteración

Analizando las imágenes obtenidas, pudimos empezar a observar patrones en la mayoría de las capas de los bloques intermedios (2 y 3), si bien existen algunos otros filtros interesantes en otros lugares de la red. Podemos distinguir, además, numerosos patrones que tienen que ver con la orientación (diagonales en ambos sentidos, horizontales y verticales), ejemplos de los cuales vemos en la ilustración 4.6.

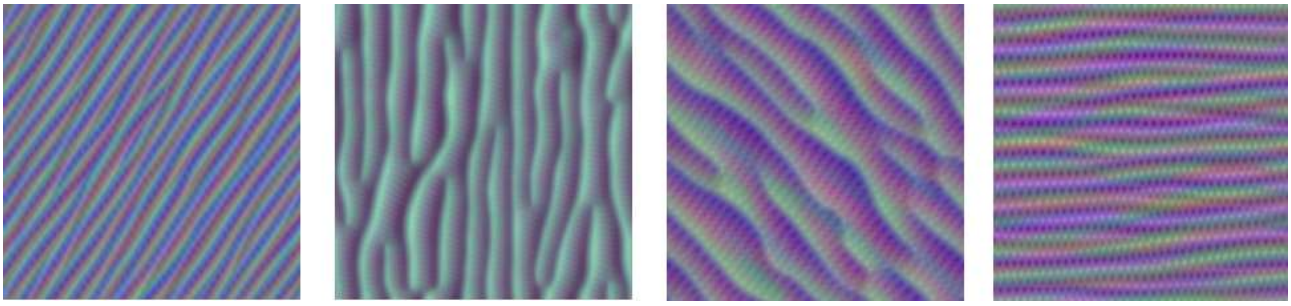

Figura 4.6. Ejemplos de diferentes orientaciones. De izq. a dcha.: B3C1F224, B2C2F127, B3C2F156, B3C3F11 (B: Bloque, C: Capa, F: Filtro)

Tras los pasos recién descritos, tuvimos ya un material de la suficiente calidad como para poder empezar otros trabajos necesarios para realizar el objetivo final de este proyecto, tal y como se describirá en siguientes secciones.

4.1.3 Comprobación de los filtros generados

De cara a estar seguros de que los resultados obtenidos tienen coherencia con lo que la red busca, decidimos generar la visualización de los filtros de la primera capa del

primer bloque. Esta capa es la que permite visualizarla de forma más directa, puesto que basta con visualizar sus pesos. Cabe destacar además que esta capa parece buscar sobre todo diferentes colores, pero sobre ello abundaremos más en las conclusiones.

Para generar dichas visualizaciones de los pesos, tuvimos que acceder a los mismos y normalizarlos, pues sus valores son demasiado pequeños. De esta forma, seleccionamos su valor mínimo y máximo y aplicamos la siguiente operación antes de generar las imágenes.

```
pesos = modelo.primera_capa.pesos
minimo_valor = min(pesos)
maximo_valor = max(pesos)
pesos = pesos + abs(minimo_valor)
pesos = pesos * 255 / (máximo_valor + abs(min_valor))
```

Cabe destacar también, que al tener 3 canales los pesos de esta red son muy fáciles de trasladar a una imagen. Además, como los filtros son de tamaño 3x3 píxeles, se aplicó a la imagen una interpolación proximal (*nearest interpolation*) de cara a obtener imágenes más visibles sin conservar una clara frontera entre los píxeles.

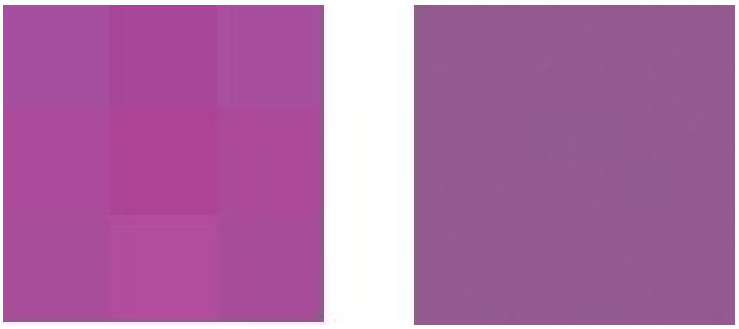

Figura 4.7 Filtro 8

Como se puede ver en la ilustración 4.8, al ser todos los píxeles de un color similar, observamos un resultado de entrenamiento de un color similar y liso.

Estos hallazgos avalan la correcta generación de la maximización de activación de los filtros y nos permiten estar seguros de que trabajamos con datos fiables.

4.1.4 Visualización de máximas activaciones de clases finales

Usando también Keras-Vis, se quiso probar a visualizar las imágenes que pudieran maximizar el valor de confianza para diferentes clases finales de la red VGG16

(Recordemos que está entrenada para ImageNet), si bien el proceso también requirió un alto tiempo de computación, la calidad de las imágenes obtenidas para las clases que elegimos (al azar), así como lo esclarecedoras que son, hacen que el proceso valga la pena.

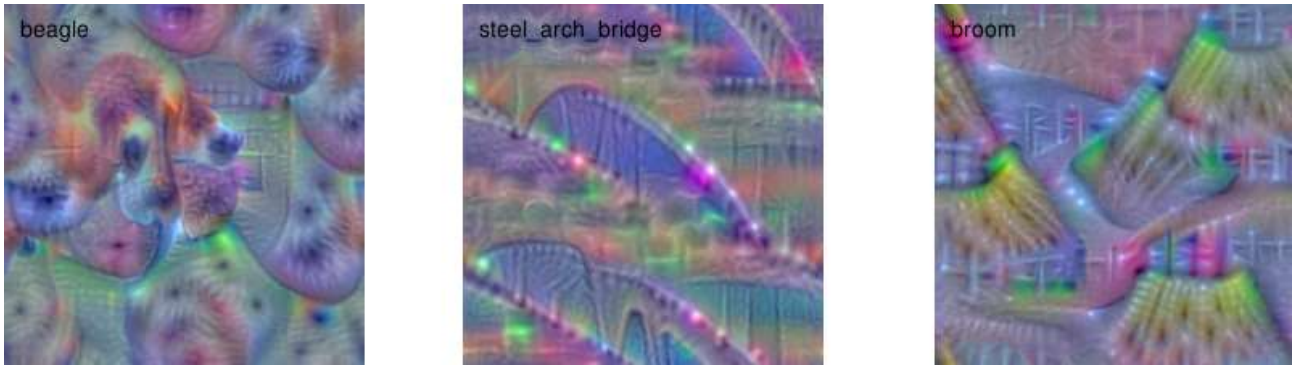

Figura 4.8. Representación ideal de Beagle, puente de acero con arcos y escobas

Figura 4.9. Imágenes que maximizan los ejemplos anteriores

Como se observa en la Ilustración 4.8, a través de este tipo de imágenes podemos deducir qué partes de los objetos son las más importantes cuando la red neuronal escoge una categoría. De esta forma observamos que con el Beagle se centra en la forma de la cara y los ojos, en los puentes en la forma de su arco y en las escobas en la parte inferior de las mismas. En la ilustración 4.9 observamos imágenes de ejemplo que provocan una alta confianza en la clasificación de estas clases. En concreto 0.872 para el Beagle, 0.989 para el puente y 0.915 para la escoba.

4.2 Fase de clustering

4.2.1 Extracción de Características HOG

Tras obtener las diferentes maximizaciones de filtros, era momento de intentar aprovechar los patrones que habíamos sido capaces de detectar en la sección anterior.

Independientemente del color de las diferentes imágenes, la información más valiosa

que podemos extraer es que la red, en ciertos momentos de su estructura, busca ciertos patrones en las imágenes de entrada con el fin de, tras sucesivas operaciones, poder discernir entre los tipos de objetos que está entrenada para detectar.

Como vimos, los patrones se encuentran bastante enfocados hacia la orientación que se pueda encontrar en la imagen de entrada, motivo por el cual, para el posterior **clustering**, quisimos centrarnos en esa información y descartar otras informaciones que pudiesen ser más superfluas. Es por ello que decidimos usar un descriptor de la imagen, en concreto: el conocido Histograma de gradientes orientados (*Histogram of oriented gradients)* (HOG), este descriptor, sin querer entrar en detalles, cuenta las ocurrencias de las diferentes orientaciones de los gradientes en porciones de una imagen, en nuestro caso, los filtros.

Es por ello que decidimos usarlo en este proyecto, pues la información que más utiliza es la orientación de los bordes detectados.

La longitud de cada descriptor es de 26244 elementos usando un tamaño de entrada de 224x224 píxeles, un maño de bloque de 16x16, un salto de bloque de 8x8, un tamaño de celda de 8x8 y un número conjuntos de orientaciones de 9.

4.2.2 Agrupamiento automático de los filtros

Basándonos en lo comentado y aprendido en secciones anteriores, el siguiente paso lógico es distinguir los diferentes tipos de filtros existentes entre los aprendidos previamente. Una opción válida sería realizar un trabajo manual de agrupamiento, en el cual una persona fuese separando los diferentes filtros en función de su orientación, color, forma, etc.

Sin embargo, y éste es uno de los puntos novedosos de este trabajo, creímos más conveniente aprovechar la información contenida en los descriptores de histograma de gradientes orientados y delegar el proceso de agrupamiento en un algoritmo especializado en ello.

Mediante el framework *Sklearn* se probaron diferentes algoritmos de este tipo, entre ellos podemos encontrar *MeanShift, K-Means, Affinity Propagation* y *Ward hierarchical clustering.* Como se puede haber notado, algunos de ellos determinan por sí solos el número de conjuntos que deben crearse. Otros, sin embargo, reciben este valor como parámetro. Los resultados de los diferentes algoritmos han sido dispares, si bien como veremos, hay algunos realmente prometedores.

Affinity Propagation y Meanshift dieron un mal resultado, obteniéndose más de 50 clusters distintos, motivo por el cual su resultado no fue tenido en cuenta.

De entre todos los probados, K-Means resultó ser el que mejor resultado dio, pues al tener un número fijo de clusters sus resultados fueron fácilmente manejables. Además, sus resultados son bastante entendibles desde el punto de vista humano, agrupando a los filtros con una orientación determinada. Se probaron también diferentes cantidades de conjuntos (4, 5, 6 y 7).

De entre las cantidades probadas, aquella que nos arrojó mejores resultados fue la de 6. Los grupos (en líneas generales) son los siguientes:

- **Patrones simples**: En este conjunto encontramos patrones simples, como los de colores en la primera capa.
- **Patrones complejos**: Aquí encontramos a algunos filtros aparentemente complejos de las primeras capas (Quizás por una mala convergencia), así como la gran mayoría de los patrones de las últimas capas.
- **Patrones horizontales**: Patrones que muestran una gran cantidad de líneas horizontales
- **Patrones verticales**: Patrones con una gran variedad de líneas verticales.
- **Patrones diagonales (1)**: Patrones diagonales en dirección ascendente a la derecha.
- **Patrones diagonales (2)**: Patrones diagonales en dirección ascendente a la izquierda.

Esto no quita que algunos de los filtros que no convergieron de forma correcta sean clasificados sin un motivo aparente junto con otros filtros con patrones más claros, así como otro tipo de filtros clasificados de una forma no obvia.

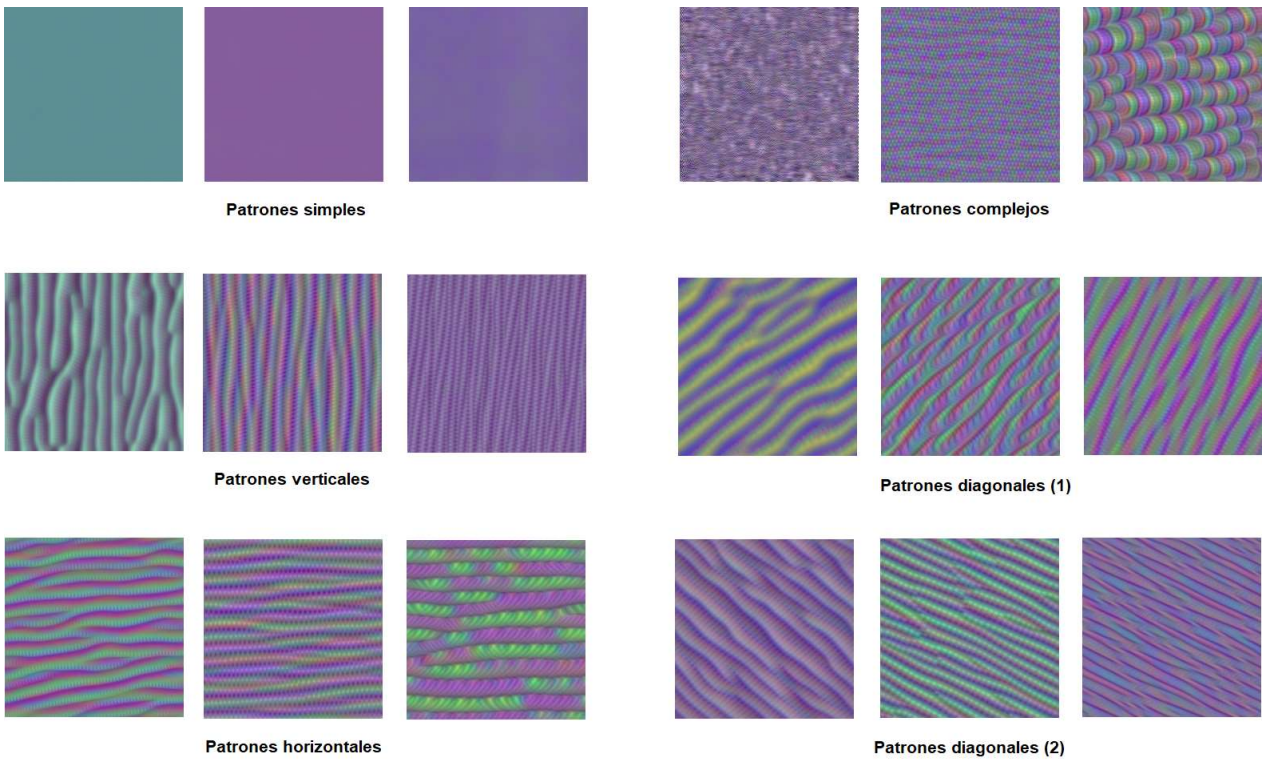

Figura 4.10. Diferentes ejemplos de los diferentes conjuntos

4.3 Trabajo con los tipos de filtros

4.3.1 Detectar las activaciones de las capas convolucionales

Keras no aporta ninguna forma sencilla de detectar las activaciones de las diferentes capas convolucionales de forma eficiente. Por ello, para realizar dicho trabajo, tuvimos que crear una función de keras que devolviera las diferentes capas, así como sus valores para una cierta imagen de entrada.

Cabe destacar que el código empleado está pensado para un correcto rendimiento, de manera que se ejecute en GPU y no haya transferencias innecesarias de datos entre GPU y CPU. Es importante saber también que, después de la primera capa convolucional, el resto de las salidas de cada filtro es de un solo canal, por lo que se pueden visualizar como una imagen en blanco y negro. Al querer obtener las imágenes de estas activaciones nos encontramos un problema: VGG16 usa una función de activación del tipo RELUX, la cual no está acotada superiormente.

Este hecho acarreaba de por sí un problema mayúsculo, pues a la hora de normalizar los valores de las activaciones de 0 a 255 para generar una imagen con las mismas, el valor máximo al cual asignar 255 es incierto. Otro problema es que los valores de activaciones varían entre filtros, incluso dentro de la misma capa. Además, generalmente son mayores en capas intermedias.

La solución propuesta para no tener que alterar el modelo de red neuronal consiste en obtener la máxima activación de ese filtro (entre todos los píxeles del mismo) y normalizar la capa entera en base a ese valor. Obviamente esto nos permite saber qué parte de la imagen está activando en mayor medida al filtro. Sin embargo, perdemos la capacidad de saber cuánto, pues en el hipotético caso de un filtro en el que no hay activaciones altas, su máximo será bajo y resultará en una imagen con muchas activaciones marcadas como altas.

Esto además supuso la renuncia a comparar diferentes filtros y sus activaciones dentro de un mismo clúster, quedando esta mejora pendiente tal y como se comentará en la sección de conclusiones.

4.3.2 Selección de filtros interesantes

Con los conjuntos creados y una correcta visualización de las activaciones de los mismos, fue momento de seleccionar qué filtros eran los más adecuados para detectar las características deseadas en las imágenes.

Este trabajo fue completamente manual, realizándose simplemente con ayuda de la visualización de activaciones y de unas imágenes de prueba muy simples con diferentes características, como se puede observar en las ilustraciones que acompañan a esta sección.

También esta técnica sirvió para descubrir que hay filtros que se centran en una cierta

orientación en un solo sentido (Orientación vertical con la parte clara a la derecha, por ejemplo). Este fenómeno encuentra también un ejemplo en su ilustración correspondiente.

Figura 4.11. Imágenes de prueba con diferentes patrones verticales, horizontales y diagonales

También esta técnica sirvió para descubrir que hay filtros que se centran en una cierta orientación en un solo sentido (Orientación vertical con la parte clara a la derecha, por ejemplo). Este fenómeno encuentra también un ejemplo en su ilustración correspondiente.

Figura 4.12. Dcha.: Filtro que reconoce los bordes horizontales en ambos sentidos. Izq.: Filtro que sólo reconoce bordes horizontales en un sentido

Los filtros que resultaron más prometedores del bloque 2 fueron todos de su segunda capa convolucional, siendo los que se observan en la siguiente lista:

- Horizontal: Filtro 25
- Vertical: Filtro 37
- Diagonal 1: Filtro 113
- Diagonal 2: Filtro 105

4.4 Interfaz de usuario

4.4.1 Interfaz gráfica de usuario

Entrando en la recta final del desarrollo del proyecto debemos hablar de la interfaz de usuario que permita interactuar con lo desarrollado, así como probar su funcionamiento de una manera rápida y cómoda.

En una primera versión, y sobre todo por motivos de eficiencia, decidimos que no funcione en tiempo real, pues para cada imagen, aún con una Nvidia GTX-1050 Ti, se requieren 3 segundos por imagen. Por ello, decidimos trabajar con imágenes estáticas, realizándose todos los cálculos necesarios en el momento de la carga de la imagen, no mientras el usuario interactúa con la misma.

Además, los objetivos que la interfaz de usuario tiene que cumplir son:

- Seleccionar una imagen desde el sistema de archivos
- Clasificar dicha imagen y mostrar el resultado de dicha clasificación
- Obtener las activaciones de cierto filtro en cierta capa para la imagen seleccionada (No permitiendo seleccionar una combinación de filtro y capa no válida)
- Poder seleccionar el tipo de patrón que queremos oír
- Poder seleccionar qué región (y el tamaño) de la imagen queremos “oír”

La interfaz contiene un *menu-bar,* el cual hace de puerta de entrada a las diferentes funciones de la misma, este menú contiene las siguientes secciones:

- **Archivo:**
    - Abrir imagen
- **Visualización**
    - Visualizar filtro
- **Ayuda**
    - Acerca de…

Las secciones tienen un nombre tremendamente explicativo, pero a continuación definiremos su comportamiento más en detalle:

Al querer abrir una imagen se desarrollarán diferentes acciones. Por un lado, se mostrará al usuario el cuadro de diálogo estándar para seleccionar un archivo, para que una vez seleccionado, y al confirmar la acción, se muestre la misma en la ventana principal. La imagen mostrada es de 448x448 píxeles (Sea cual sea su tamaño de origen). Adicionalmente, todas las subventanas que haya abiertas en la aplicación actualizarán su contenido a la nueva imagen. Esto último es debido a que todos los cálculos de la red neuronal son realizados al cargar la imagen, permitiendo que sus resultados estén disponibles para ser mostrados.

Figura 4.13. Ventana principal mostrando una imagen

En la segunda de las funciones, la de visualización de filtros, se pide al usuario que seleccione en un combo-box la red donde se ubica el mismo (De entre todas las de la red, excluyendo la de entrada y las totalmente conectadas), así como el número de este. Cabe destacar también que se muestra al usuario los posibles números de filtros que tiene la capa seleccionada (su rango), y que se impide al usuario pulsar el botón *OK*, si los caracteres introducidos por el mismo no se corresponden con un número de filtro válido.

Las activaciones mostradas son siempre de tamaño 224x224, reescalándose lo necesario para que así sea.

Figura 4.14. Diálogo de selección de filtro a visualizar

Al seleccionar el filtro deseado, se mostrará la activación del mismo para la imagen de entrada, esta visualización se muestra en un tamaño de 224x224 píxeles, por lo que las activaciones de los filtros de bloques superiores serán rescaladas a dicha resolución. El título de la ventana recién creada se corresponde al filtro que está representando.

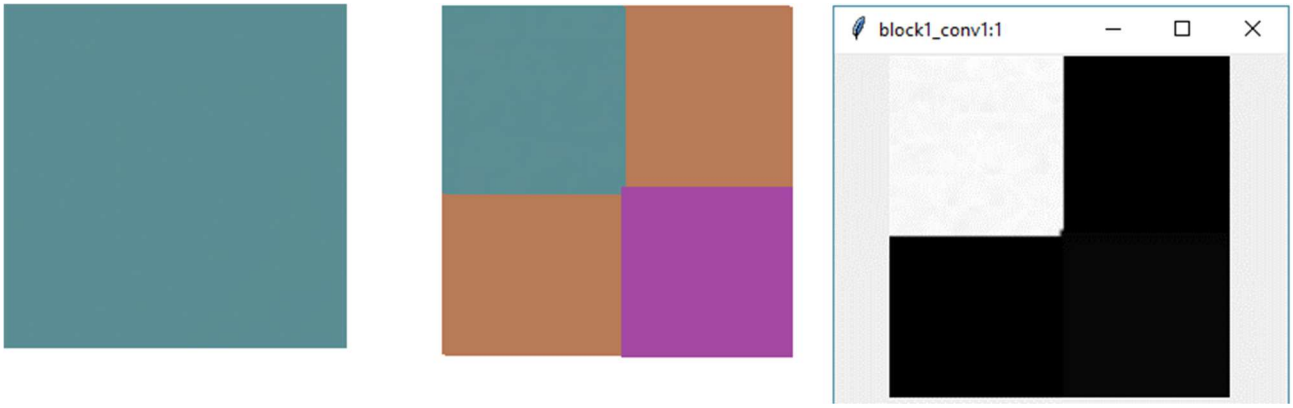

Figura 4.15. De izq. a dcha.: Representación ideal del filtro, imagen de prueba y resultado de la visualización

Como se ve en la ilustración 4.13, al buscar el filtro colores azules, los tonos marrones no lo activan, si bien un color morado (al tener cierto azul) sí.

En la parte inferior de la ventana principal se muestran 2 *combo-boxes*. El primero de ellos permite seleccionar con qué bloque de la red queremos trabajar (Usando los filtros definidos por el usuario o los por defecto, nombrados en la sección 4.3.1), así como si queremos clasificar la imagen en su conjunto en clases de ImageNet. Mientras, el segundo permite seleccionar el tipo de característica que queremos detectar (Horizontal, vertical o 2 tipos de diagonales).

En el caso de querer clasificar la imagen, una vez pulsado el botón izquierdo del ratón sobre cualquier parte de la misma, se abrirá una nueva ventana, mostrando un diagrama de barras (usando matplotlib) con las 5 clases con mayor confianza, así como su porcentaje de confianza, tal y como se puede ver a continuación.

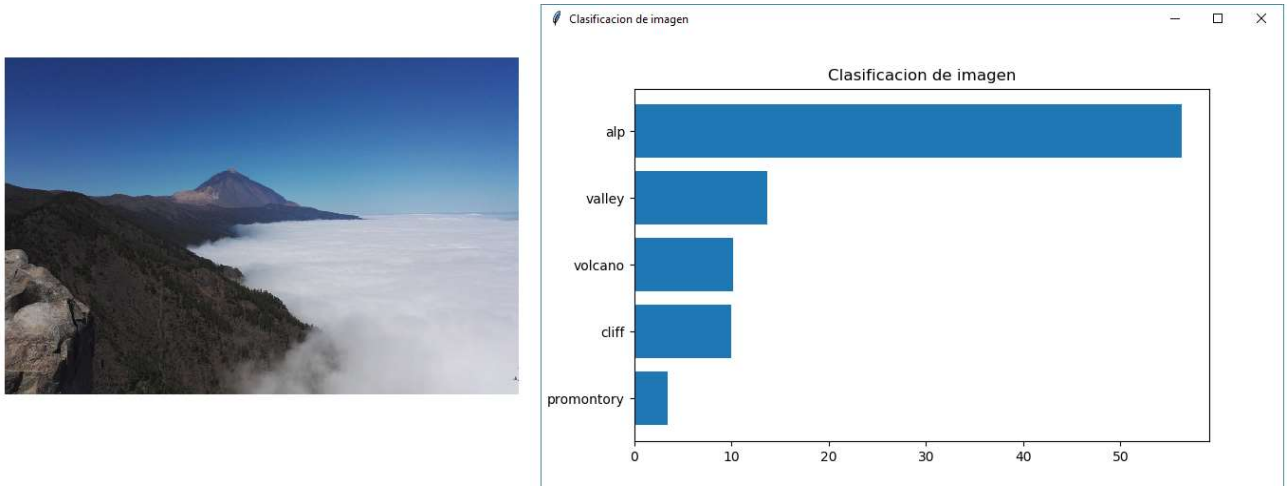

Figura 4.16. Imagen original y su clasificación

En otro caso (Con un bloque y filtro determinado seleccionado) sucederán varias cosas. Por un lado, se mostrará alrededor del ratón un recuadro negro que representa el tamaño del campo receptivo de las neuronas en dicho bloque.

Lo importante, sin embargo, es que tanto al pulsar el ratón en algún lugar de la imagen como al arrastrar el mismo mientras se mantiene apretado, se obtiene el nivel de activación del filtro seleccionado en dicha área para poder mostrar la información sobre la misma en forma de sonido.

4.4.2 Generación de estímulos sonoros

Con las activaciones localizadas como mostramos previamente, era momento de convertirlas en estímulos sonoros. La forma utilizada para hacerlo fue la reproducción de sonidos con frecuencias variables.

Estos trabajos han sido realizados gracias a una librería para *Python* llamada *PySine* [29].

A diferencia del objetivo inicial, y como resultado de los problemas encontrados a la hora de poder normalizar y comparar los niveles de activación de diferentes filtros, decidimos usar el mismo rango de frecuencias para todos los tipos de características, siendo el usuario quien seleccione en todo momento qué tipo de características desea “oír”.

Además, se estableció un umbral mínimo de activación (10% con respecto al máximo del filtro) a partir del cual reproducir algún tipo de sonido, de cara a evitar molestias

de un sonido continuo a los usuarios finales. De igual manera, las frecuencias reproducidas comprenden desde los 540Hz para una activación del 10% hasta los 1800Hz de una activación del 100%, siendo una diferencia suficientemente grande para percibir los cambios en las mismas.

Cabe destacar, además, que los estímulos de sonido reproducidos tienen una duración de 0.05 segundos.

Se meditó también utilizar el volumen como una forma de aportar esta información al usuario. A pesar de ello, se consideró, que al estar oyendo únicamente un tipo de filtro cada vez, las diferencias de frecuencia de los sonidos escuchados son suficientes, si bien no se descarta en un futuro explorar otras soluciones de cara a poder advertir al usuario de varios tipos de características simultáneamente.

En el caso de trabajar con varios tipos de características a la vez, la idea inicial antes de encontrarnos con los problemas anteriormente comentados fue la de asignar un rango de frecuencia (sin intersecciones entre ellos) a cada uno de los tipos de características a detectar, añadiendo, esta vez sí, el volumen como una forma de indicar con qué intensidad está siendo detectada dicha característica.

Capítulo 5
Conclusiones y líneas futuras

Múltiples son las conclusiones que se pueden extraer de este Trabajo de Fin de Grado, así como todos los conocimientos adquiridos durante el mismo. A lo largo de su desarrollo, así como de la presente memoria, se han ido nombrando conceptos y conclusiones que ahora se detallarán en mayor medida.

Como principal conclusión, podemos hablar de que ésta nueva rama de investigación ha demostrado en sus primeras etapas ser merecedora de más trabajos futuros, pues tiene la habilidad y el potencial de mejorar la vida de muchas personas, así como de ayudar a entender mejor la naturaleza de este tipo de redes neuronales.

Además, hemos podido comprobar cómo en este modelo, el color sólo cobra importancia en la primera de sus capas, siendo el resto de las mismas bastante invariables al mismo. De la misma manera, hemos explorado una rama casi inexplorada en este campo, como es el hecho de clasificar los filtros convolucionales de la red mediante técnicas de clustering sin supervisión.

Sin embargo, y como en cualquier proyecto de investigación, surgen nuevos problemas y ramas a explorar en el futuro de cara a mejorar los resultados aquí presentados.

En futuros proyectos relacionados con este trabajo sería conveniente probar a modificar ligeramente el modelo de la red, sustituyendo las funciones de activación ReLu por alguna acotada superiormente, como podría ser SoftMax [30]. Esto permitiría realizar una comparación directa y adecuada entre las activaciones de diferentes filtros a lo largo de la red, eliminando la necesidad de encontrar filtros representantes de su respectivo conjunto.

Otra actividad que resulta una continuación natural de lo desarrollado en este Trabajo de Fin de Grado es la de probar aún más técnicas de clustering, así como realizar estas técnicas con subconjuntos de los filtros de la red y no con todos ellos.

Aun así, seguirá habiendo retos y problemas que ir resolviendo, como la forma de comparar de forma correcta activaciones de bloques con diferentes resoluciones, así como indicarle al usuario que hay varios tipos de características coexistiendo en una misma región de la imagen.

Con todo ello, lo que resulta innegable es que el campo de las redes neuronales y la visión artificial están destinados a seguir refinándose y mejorándose, en una constante evolución que puede mejorar en gran medida la vida de millones de personas alrededor del mundo.

Capítulo 6
Summary and Conclusions

Several conclusions can be drawn of this Final Degree Project, and so the knowledge obtained with it. During the development of this project, some concepts and conclusions have been enumerated, and they will be described in detail in the following lines.

The first and the most important conclusion is that we have proved that this brand-new research area must be further developed in the future. The reason for that is that it has been proved that it has the potential and the ability to improve the life of many visual-impaired people, and to increase the knowledge that we have about the nature of artificial neural nets.

In addition, we have seen that in VGG16 the colors are only important in the first of its layers, being the rest of them pretty color-independent. We have also explored a new technique, based on clustering the convolutional filters of a neural net with unsupervised clustering algorithms.

Nevertheless, as this has been a research problem, new questions and challenges have appeared. They all should be investigated in the future, so that the results that we have obtained can be improved.

In future projects related with the work that we've done here it would be convenient to slightly modify the model, changing the ReLu activation function with another activation function that is bounded above, like SoftMax. That would allow us to compare directly the activation levels of different filters of the net, and to not be limited to use a single filter that represents its whole cluster.

Another activity that is a natural continuation of what we have done, is to try some other clustering algorithms and to do it with subsets of the filters and not with all of them.

However, they will be another challenges and problems to be solved. An example of them is finding a way to compare the activated regions of filters that don't have the same resolution as well as telling the user that several features are detected in the very same region of the image.

Despite all this, what can't be doubt is that the field of Deep Learning and Artificial Vision is going to become more refined in a short period of time and that will improve a big way the life quality of millions of people all around the globe.

Capítulo 7
Presupuesto

A la hora de elaborar el presupuesto de este Trabajo de Fin de Grado, diferenciaremos diferentes apartados: Coste de la mano de obra y el coste de servicios de internet y electricidad.

7.1 Coste de la mano de obra

Estimamos las horas de trabajo dedicadas a este TFG en 345. De esta forma, y asignando un salario de 18€ por hora, el coste resultante es el siguiente:

Trabajador	Salario por hora	Horas	Total
Pablo Pastor Martín	18	345	6.210€

Nota: En este salario por hora se incluye el coste de amortización del equipo informático utilizado para el desarrollo del trabajo de fin de grado.

7.2 Coste de conexión a internet y electricidad

El coste en servicios necesarios para realizar el trabajo es el detallado a continuación:

Servicio	Precio	Detalles	Coste
Internet	55€	Fibra óptica 300mb	165€
Electricidad	0.12€/kWh	Ordenador de 600W de potencia	25€

7.3 Coste total del desarrollo

El presupuesto estimado para el desarrollo de este Trabajo de Fin de Grado es el observado en la siguiente tabla:

Concepto	Coste
Mano de obra	6210€
Servicios	190€
Coste total antes de impuestos	6400€
IGIC (7%)	450€
Presupuesto total	6850€

Bibliografía

[1] "OpenAI Five", OpenAI Blog, 2018. [Online]. Available: https://blog.openai.com/openai-five/. [Accessed: 02- Sep- 2018].

[2] H. Haenssle, C. Fink, R. Schneiderbauer, F. Toberer, T. Buhl, A. Blum, A. Kalloo, A. Hassen, L. Thomas, A. Enk and L. Uhlmann, "Man against machine: diagnostic performance of a deep learning convolutional neural network for dermoscopic melanoma recognition in comparison to 58 dermatologists", Annals of Oncology, vol. 29, no. 8, pp. 1836-1842, 2018.

[3] Navneet Dalal, Bill Triggs, “Histograms of Oriented Gradients for Human Detection”, 2005

[4] McCulloch WS, Pitts W., “A logical calculus of the ideas immanent in nervous activity”, *Bull Math Biol.*, vol. 5, pp.115-133, Diciembre 1943.

[5] D. O. Hebb, “*The Organization of Behavior: A neuropsychological theory”,* Nueva York: Wiley, 1949.

[6] M. Minsky and S. Papert, Perceptrons: An Introduction to Computational Geometry. Cambridge, MA, USA: MIT Press, 1969.

[7] J. A. Anderson, J. W. Silverstein, S. A. Ritz, and R. S. Jones, “Distinctive features, categorical perception, and probability learning: Some applications of a neural model.,” Psychological Review, vol. 84, no. 5, pp. 413–451, 1977.

[8] Hopfield, J.J., & Tank, D.W. “Neural computation of decisions in optimization problems”. Biological Cybernetics, vol.52, pp.141-152, 1985

[9] D. Rumelhart, G. Hinton and R. Williams, "Learning representations by back-propagating errors", Nature, vol. 323, no. 6088, pp. 533-536, 1986.

[10] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition", Biological Cybernetics, vol. 36, no. 4, pp. 193-202, 1980.

[11] Y. LeCun, L. Bottou, G. Orr and K. Müller, "Efficient BackProp", Lecture Notes in Computer Science, pp. 9-50, 1998.

[12] D. Cireşan, U. Meier, J. Masci and J. Schmidhuber, "Multi-column deep neural network for traffic sign classification", Neural Networks, vol. 32, pp. 333-338, 2012.

[13] D. Hubel and T. Wiesel, "Receptive fields of single neurones in the cat's striate cortex", The Journal of Physiology, vol. 148, no. 3, pp. 574-591, 1959.

[14] R. Hahnloser, R. Sarpeshkar, M. Mahowald, R. Douglas and H. Seung, "Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit", Nature, vol. 408, no. 6815, pp. 1012-1012, 2000.

[15] A. Krizhevsky, I. Sutskever, & G.E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. NIPS, vol. 1, pp. 1095-1105, 2012
[16] J. Fu and Y. Rui, "Advances in deep learning approaches for image tagging", APSIPA Transactions on Signal and Information Processing, vol. 6, 2017.
[17] C. Szegedy S. Ioffe V. Vanhoucke "Inception-v4 inception-resnet and the impact of residual connections on learning" arXiv preprint arXiv:1602.07261 2016.
[18] A. Mahendran and A. Vedaldi, "Understanding deep image representations by inverting them", 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015.
[19] A. Nguyen, J. Yosinski and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images", 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015.
[20] A. Mordvintsev and M. Tyka, "Inceptionism: Going Deeper into Neural Networks", Google AI Blog, 2018. [Online]. Available: https://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html. [Accessed: 02- Sep- 2018].
[21] "Seeing AI | Talking camera app for those with a visual impairment", Microsoft.com, 2018. [Online]. Available: https://www.microsoft.com/en-us/seeing-ai. [Accessed: 02- Sep- 2018].
[22] "Home - Horus", Horus, 2018. [Online]. Available: http://horus.tech/?l=en_us. [Accessed: 02- Sep- 2018].
[23] Keras Documentation", Keras.io, 2018. [Online]. Available: https://keras.io. [Accessed: 02- Sep- 2018].
[24] "TensorFlow", TensorFlow, 2018. [Online]. Available: https://www.tensorflow.org/?hl=es. [Accessed: 02- Sep- 2018].
[25] "Theano 1.0.0 documentation", Deeplearning.net, 2018. [Online]. Available: http://deeplearning.net/software/theano/. [Accessed: 02- Sep- 2018].
[26] "Keras-Vis", GitHub, 2018. [Online]. Available: https://github.com/raghakot/keras-vis. [Accessed: 02- Sep- 2018].
[27] "OpenCV library", Opencv.org, 2018. [Online]. Available: https://opencv.org. [Accessed: 02- Sep- 2018].
[28] "Scikit-learn", Scikit-learn.org, 2018. [Online]. Available: http://scikit-learn.org/stable/. [Accessed: 02- Sep- 2018].
[29] “Pysine", GitHub, 2018. [Online]. Available: https://github.com/lneuhaus/pysine. [Accessed: 02- Sep- 2018].
[30] K. Priddy and P. Keller, Artificial neural networks. Bellingham, Wash. (1000 20th St. Bellingham WA 98225-6705 USA): SPIE, 2005, pp. 16-17.
[31] “TFG", GitHub, 2018. [Online]. Available: https://github.com/olbapmar/TFG. [Accessed: 03- Sep- 2018].