



Universidad
Politécnica
de Cartagena



Proyecto fin de grado:

**Sistema de reconocimiento de gestos de la mano
basado en procesamiento de imagen y Redes
Neuronales Convolucionales**

Título en inglés: *CNN-based image processing system for hand gesture recognition*

Autor: Ernesto Ortega Asensio
Director: Francisco Javier Toledo Moreo

Índice general

1. Introducción y retos	4
2. Conceptos básicos sobre imagen digital	6
3. Introducción <i>Machine Learning</i> y <i>Deep Learning</i>	8
3.1. Aprendizaje supervisado	9
3.2. Aprendizaje no supervisado	10
3.2.1. Enfoques	11
3.3. Redes neuronales artificiales	14
3.4. Máquinas de vector de soporte o SVMs	15
3.5. Deep Learning	16
3.5.1. Evolución y progreso	17
4. Introducción a las redes neuronales artificiales y redes neuronales convolucionales	18
4.1. Redes neuronales convolucionales	21
4.1.1. Flujo de trabajo con redes neuronales convolucionales	25
4.2. Funciones y características de las redes neuronales convolucionales	27
4.2.1. Capas de convolución	27
4.2.2. Capas Relu o Unidades de Rectificación Lineal	30
4.2.3. Capas Pooling o de agrupación	31
4.2.4. Capas Fully connected o totalmente conectadas	32
4.2.5. Capas Dropout o de exclusión	33
4.2.6. Capas Cross Channel Normalization o normalización de canales cruzados	35
4.2.7. Capas Softmax y de clasificación	36
5. Diseño de la solución implementada	37
5.1. Conjunto de imágenes o Dataset	38
5.2. Ejemplo de imágenes del Dataset o conjunto de imágenes	45
5.3. Flujo de trabajo 1 detallado	46
5.4. Flujo de trabajo 2 detallado	57
6. Resultados y conclusiones	65

Agradecimientos

A mi familia por apoyarme en todo momento desde el inicio de mi etapa universitaria.

A todos los profesores que me han ayudado y servido de ejemplo en todos estos años de carrera.

A todos mis compañeros de clase que me han ayudado en cada año de carrera.

“Nunca consideres el estudio como una obligación, sino como una oportunidad para penetrar en el bello y maravilloso mundo del saber.” Albert Einstein.

Fuerte abrazo a Julian De la vida.

Ficha del proyecto

Tipo de Proyecto	Específico
Curso Académico	2016-17
Autor del proyecto	Ernesto Ortega Asensio
Director del Proyecto	Francisco Javier Toledo Moreo
Departamento	Electrón,Tecnol Computadoras y Proyectos
Área de conocimiento	Arquitectura y Tecnología de Computadores
Título en castellano	Sistema de reconocimiento de gestos de la mano basado en procesamiento de imagen y Redes Neuronales Convolucionales
Título en inglés	CNN-based image processing system for hand gesture recognition
Objetivos	El Trabajo Fin de Grado se enmarca en una línea de desarrollo de soluciones para la interacción Persona-Computador (<i>Human-Computer Interface</i> , HCI) utilizando los gestos de la mano y basada en la visión por computador. En este contexto, el objetivo del presente Trabajo es evaluar el empleo de una Red Neuronal Convolucional (<i>Convolutional Neural Network</i> , CNN). Ésta es un tipo de red neuronal artificial inspirada en el patrón de conectividad de las células de la corteza visual. Recientemente, y en el marco del Deep Learning, las CNNs han adquirido gran popularidad para el procesamiento de imagen y vídeo.
Fases	Las principales fases de desarrollo del TFE son las siguientes: 1. Familiarización con la arquitectura y características de las <i>CNNs</i> y con las herramientas de trabajo para su diseño e implementación. 2. Creación de bases de datos de imágenes para el desarrollo de la propuesta y para la evaluación de resultados. 3. Estudio de configuraciones <i>CNN</i> atendiendo a las características de nuestra aplicación. 4. Evaluación de resultados y estimación de las prestaciones.

Capítulo 1

Introducción y retos

La visión por computador trabaja en encontrar medios computacionales para interpretar la información visual del mundo que nos rodea, algo en lo que, los seres humanos, nos hemos especializado. El propósito de este proyecto es documentar una forma de resolver un problema específico dentro del ámbito de la visión por computador. Nos centraremos en capturar diferentes gestos de la mano con la cámara del ordenador o con otro dispositivo, además, mediante el uso de diferentes técnicas y recursos, obtendremos una clasificación de las imágenes con el fin de ejecutar una acción (Ej: Abrir navegador, etc) según el gesto capturado. En capítulos posteriores se detallará el funcionamiento y las diferentes soluciones aplicadas. El programa contará con una interfaz de usuario donde se visualizarán las imágenes de la cámara y se clasificarán los diferentes gestos de la mano capturados por el sistema, mostrando la categoría y la confianza en esa predicción en el título de la figura de MATLAB. Para ello se hará uso del lenguaje de programación MATLAB y se implementarán una serie de *scripts* (Programas) que harán uso de las redes neuronales convolucionales, y otros recursos de MATLAB que se explicarán en el capítulo 5, además de los varios flujos de trabajo que utilizaremos para intentar solucionar el problema. Los *scripts* harán uso de diferentes recursos: la memoria gráfica del ordenador (*Graphic Processor Unit*) paquete *Parallel Computing Toolbox* de MATLAB, paquete *Statistics and Machine Learning Toolbox* de MATLAB y el paquete *Neuronal Network Toolbox* de MATLAB, que contienen funciones que usaremos en el diseño de nuestra solución. Según el flujo de trabajo que escojamos realizaremos la clasificación de la imagen mediante una herramienta u otra.

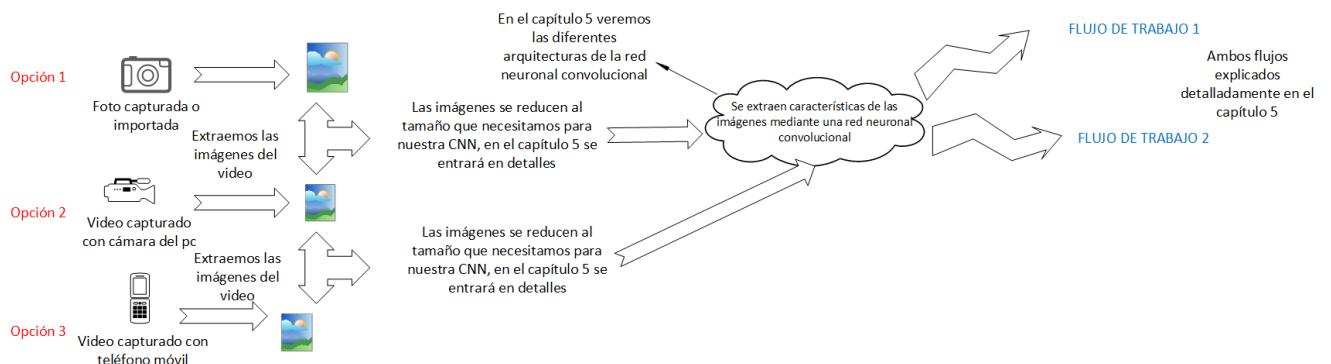


Figura 1.1: Estructura del proyecto

Para los humanos, la visión es un sistema que funciona de forma inconsciente, ni siquiera nos detenemos a pensar lo complejo que es ese proceso, aplicándolo a nuestro problema, si viésemos la mano de un niño y la mano de un adulto podríamos distinguirlas rápidamente, pero una maquina se enfrenta con una cantidad enorme de problemas para diferenciarlas, por ejemplo, ambas manos tienen cinco dedos y prácticamente la misma forma. Otro de los retos es el tiempo de procesamiento de las imágenes adquiridas, una de las formas sencillas, sería comparar las imágenes con otras imágenes, pero esto podría provocar que el sistema fuese lento. Algunos de los retos que nos encontramos en los problemas de visión por computador aplicados a nuestro problema en particular, están representados en la siguiente figura.



Figura 1.2: Algunos de los retos de la visión por computador aplicado a nuestro problema en particular

Algunos de los problemas de la figura 1.2 se intentarán paliar creando un conjunto de imágenes robusto, es decir, que contenga ejemplos variados de las diferentes categorías, con el fin de crear una mejor definición de lo que cada categoría significa. En el capítulo 5 se explica con más detalle el código MATLAB para la obtención del conjunto de imágenes usando varias técnicas, así como varios ejemplos de estas imágenes, además, se comentará también el código MATLAB con el que se procesan las imágenes del *Dataset* o conjunto de imágenes, con el fin de resolver en cierta medida los problemas de la figura anterior. Otro reto a gestionar sería la gran cantidad de datos a procesar, en este caso, imágenes de los diferentes gestos de la mano, por eso se hará uso de la memoria gráfica del ordenador para el proceso de entrenamiento de la red neuronal convolucional. Somos conscientes de la cantidad de retos que nos encontramos en los problemas de visión por computador, en cambio, para muchos de ellos ya se tienen algoritmos y técnicas que resuelven en menor o mayor medida el problema. Haremos uso de estas técnicas mediante el uso de algunas funciones de MATLAB.

Capítulo 2

Conceptos básicos sobre imagen digital

En este texto no se entrará en detalles sobre el proceso de obtención de las imágenes digitales, simplemente sus conceptos básicos. Una imagen digital en color se representa por una matriz tridimensional ($m * n * p$), donde m y n se corresponden con la anchura y la altura de la imagen, mientras que p representa el plano, que puede ser 1 para el rojo, 2 para el verde y 3 para el azul (Colores primarios). La figura 2.1 muestra detalles de estos conceptos.

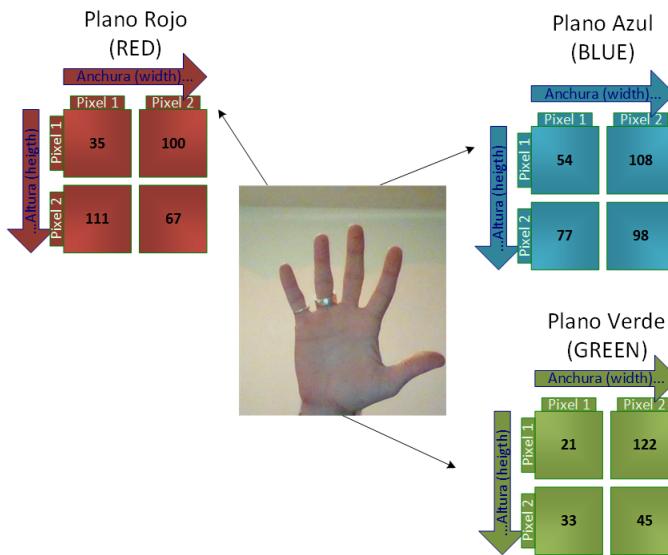


Figura 2.1: Representación de una imagen digital en color *RGB*, los valores de los píxeles para una imagen representada en este espacio de colores son números enteros que oscilan entre 0 y 255

Cada plano p contiene una matriz de valores que corresponden con el grado de intensidad de color sobre un determinado pixel, el primer valor de la matriz corresponde con el valor del primer pixel. Este, es la unidad mínima de visualización de una imagen digital. Si aplicamos *zoom* sobre ella, observaremos que está formada por una parrilla de

CAPÍTULO 2. CONCEPTOS BÁSICOS SOBRE IMAGEN DIGITAL

puntos o píxeles. Las cámaras digitales y los escáneres capturan las imágenes en forma de cuadrícula de píxeles. La figura 2.2 muestra un ejemplo.

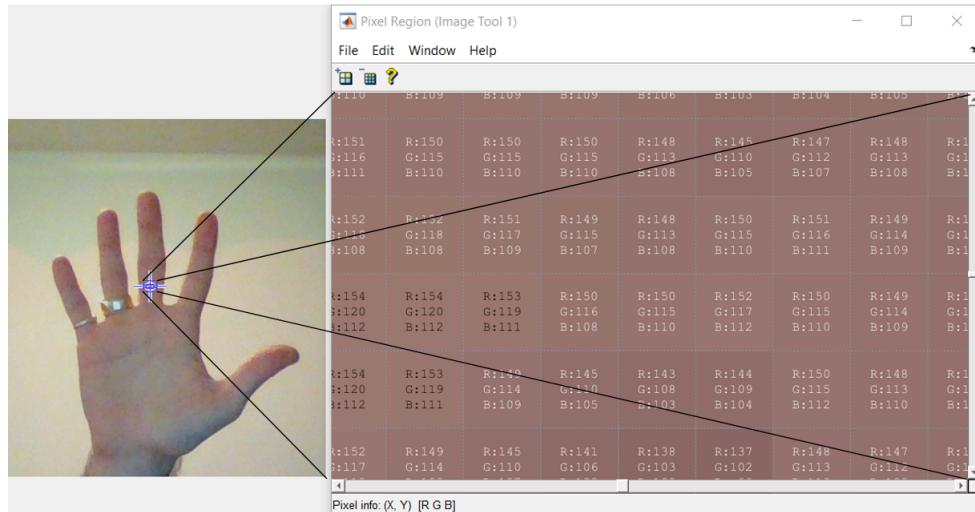


Figura 2.2: Píxeles en imagen digital RGB

En la figura 2.2 tenemos una captura de pantalla de la aplicación *Pixel Region* de MATLAB. Esta aplicación nos permite ver el valor *RGB*, en este caso (La imagen podría estar en otro espacio de colores, por ejemplo, *hsv*), de los píxeles de la región que seleccionemos mediante un cuadro en una imagen cargada desde el espacio de trabajo de MATLAB o desde un archivo.

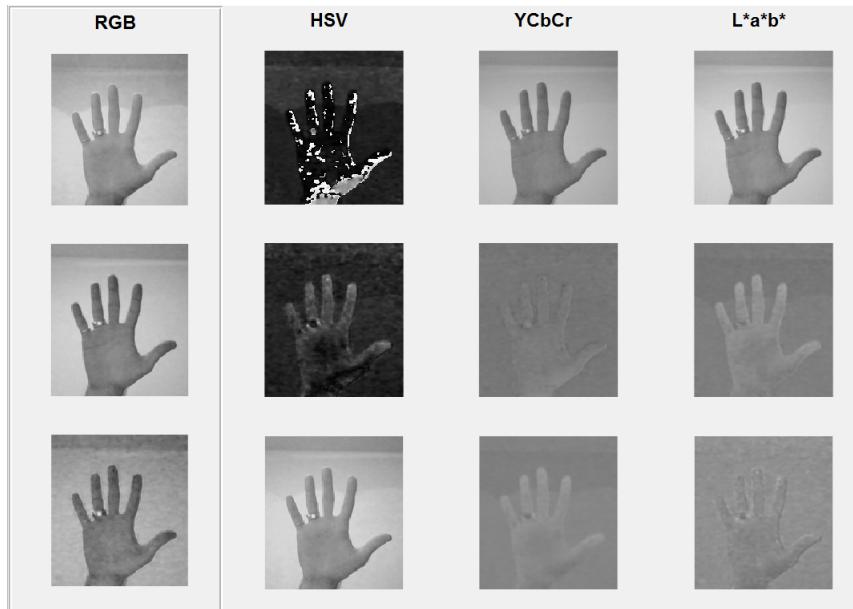


Figura 2.3: Diferentes espacios de colores en la aplicación *Color Thresholder* de MATLAB (Cada imagen de la misma columna representa un plano distinto, en *RGB* la primera imagen de la columna es el plano *RED* de la imagen)

Capítulo 3

Introducción *Machine Learning* y *Deep Learning*

“Se dice que un programa de computación aprende de la experiencia E con respecto a una tarea T y alguna medida de rendimiento P , si es que el rendimiento en T , medido por P , mejora con la experiencia E .” [1]

El aprendizaje de máquinas o aprendizaje automático (*Machine learning* en inglés) es un sub-campo de las ciencias de la computación y una rama de la inteligencia artificial, cuyo objetivo es desarrollar técnicas y algoritmos que permitan a las computadoras aprender mediante la generalización de comportamientos a partir de una información suministrada en forma de ejemplos. Este sub-campo nació por los años 50 cuando ocurrieron diversos acontecimientos: en 1950 *Alan Turing* creó el ”Test de Turing” para determinar si una máquina era realmente inteligente, en 1952 *Arthur Samuel* escribió el primer programa de ordenador capaz de aprender (Jugaba a las damas y mejoraba su juego con las partidas), en 1956 *Martin Minsky* y *John McCarthy*, con la ayuda de *Claude Shannon* y *Nathan Rochester*, organizan la conferencia de Dartmouth de 1956, considerada como el evento donde nace el campo de la Inteligencia Artificial y además en 1958 *Frank Rosenblatt* diseña el *Perceptrón*, la primera red neuronal artificial. En 1967 se escribe el algoritmo *Nearest Neighbor* que está considerado como el nacimiento del campo de reconocimiento de patrones en computadores. A partir de la década de los 90, el trabajo en *Machine Learning* gira desde un enfoque orientado al conocimiento (*knowledge-driven*) hacia uno orientado a los datos (*data-driven*). Los científicos comienzan a crear programas que analizan grandes cantidades de datos y extraen conclusiones de los resultados. En 2006 *Geoffrey Hinton* acuñó el término *Deep Learning* (Aprendizaje Profundo) para explicar nuevas arquitecturas de redes neuronales profundas, que son capaces de aprender mucho mejor modelos más planos.

A partir de esta fecha tenemos un *boom* comercial donde muchas grandes empresas añaden plataformas para trabajar con estos algoritmos o compran *Startups* que se dedican a la investigación y diseño de soluciones empleando estos, por ejemplo: *GoogleBrain*, *Facebook DeepFace*, *DeepMind*, etc. El *Machine Learning* se divide en dos áreas principales: aprendizaje supervisado y aprendizaje no supervisado.

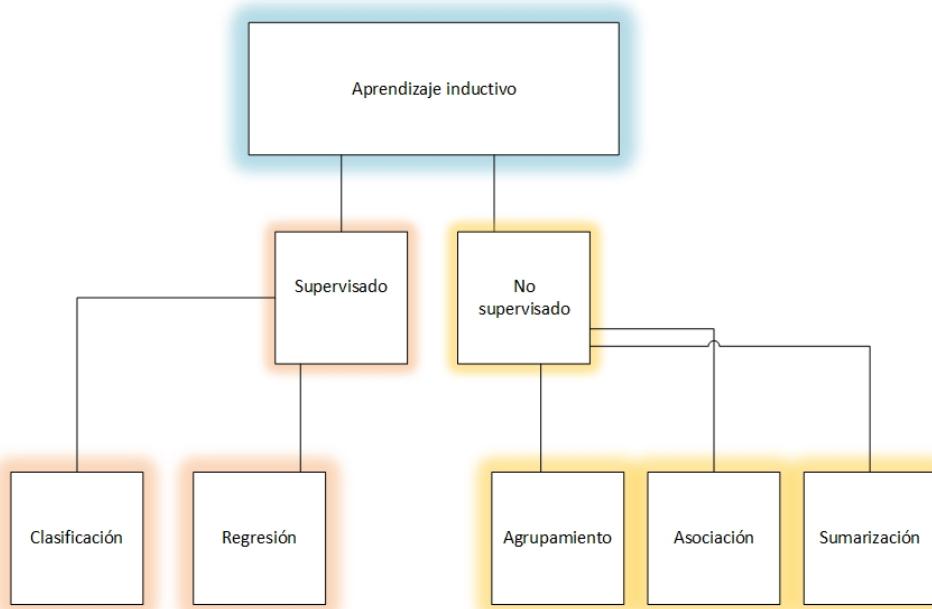


Figura 3.1: Clasificación tipos aprendizaje

Dentro de los diferentes tipos de aprendizaje podemos encontrar muchos otros, por ejemplo: aprendizaje semisupervisado, aprendizaje por refuerzo, aprendizaje multitarea y transducción. No entraremos en detalles sobre todos estos tipos, nos centraremos en los principales. En este proyecto haremos usos de algoritmos y técnicas que entran dentro de la categoría de aprendizaje supervisado.

3.1. Aprendizaje supervisado

En un problema de aprendizaje supervisado, se le muestran al sistema una gran cantidad de ejemplos del tipo de conocimiento con el que se le quiere entrenar, adjuntando a cada ejemplo el detalle de lo que se trata. Cuantos más ejemplos se le den al sistema, éste generará internamente una mejor definición de lo que significa cada concepto. Una vez procesados todos los ejemplos, el sistema ya estará en condiciones de poder clasificar o procesar la nueva información a la que se le exponga, utilizando para esto el conocimiento adquirido durante la fase de entrenamiento en la que ha aprendido diferentes características de los ejemplos. En este tipo de aprendizaje se establece una relación entre las entradas y las salidas deseadas del sistema. Un ejemplo sería el problema de clasificación, donde el sistema trata de etiquetar (clasificar) una serie de vectores utilizando una entre varias categorías (clases). La base de conocimiento del sistema está formada por ejemplos de etiquetados anteriores. Para nuestro entorno en particular, los ejemplos de etiquetados anteriores serán un conjunto de imágenes o (*Dataset*) de los diferentes gestos de la mano y estarán clasificadas en carpetas con el nombre de la clase correspondiente, en el capítulo 5 se hablará sobre este conjunto de imágenes.

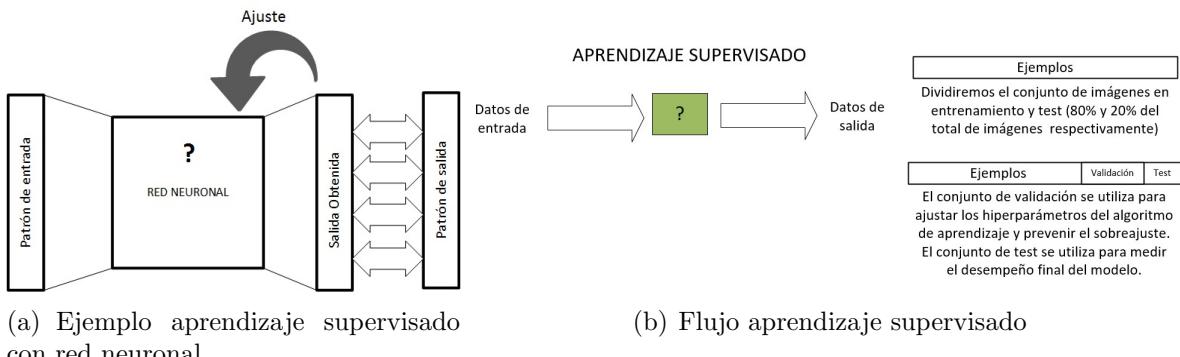


Figura 3.2: Aprendizaje supervisado

3.2. Aprendizaje no supervisado

En un problema de aprendizaje no supervisado, no hay nadie que le diga al sistema de qué trata cada ejemplo, por lo que es el propio *software* quien debe elaborar los conceptos abstractos y asociarlos a cada elemento. Este tipo de sistemas capaces de aprender por sí mismos son los que concentran hoy en día el mayor interés por parte de los investigadores. La razón de esto radica en la búsqueda por lograr mecanismos capaces de aprender de forma similar a como lo haría un niño, simplemente observando e interactuando con su entorno. En este tipo de aprendizaje todo el proceso de modelado se lleva a cabo sobre un conjunto de ejemplos formado tan sólo por entradas al sistema. No se tiene información sobre las categorías de esos ejemplos. Por lo tanto, en este caso, el sistema tiene que ser capaz de reconocer patrones para poder etiquetar las nuevas entradas.

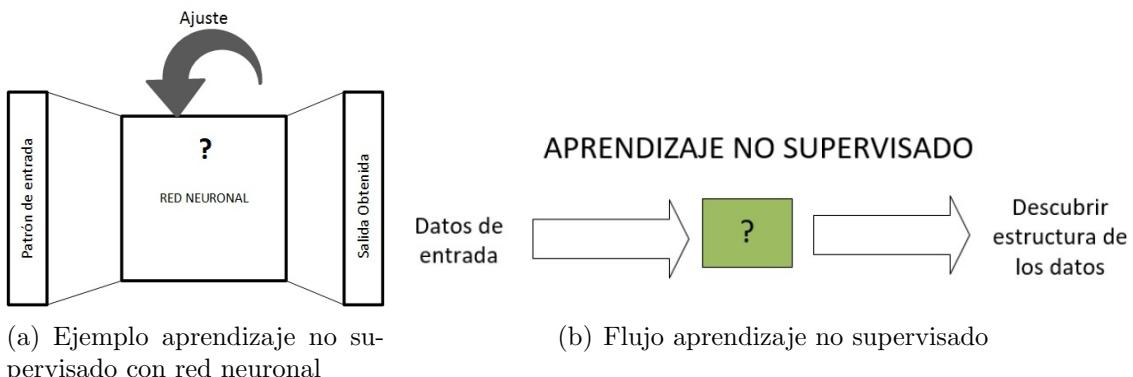


Figura 3.3: Aprendizaje supervisado

3.2.1. Enfoques

Entre los diversos enfoques que se pueden aplicar, comentaremos únicamente los que vamos a utilizar en este proyecto, los demás solo los introduciremos muy brevemente.

- Árboles de decisión: El árbol de decisión es un diagrama que representan en forma secuencial condiciones y acciones; muestra qué condiciones se consideran en primer lugar, en segundo lugar y así sucesivamente.

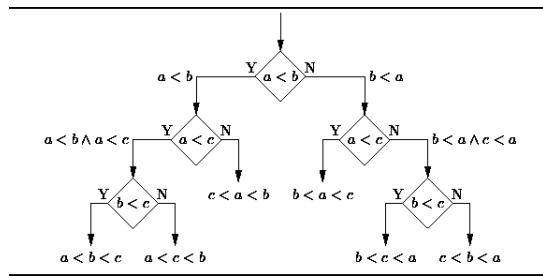


Figura 3.4: Ejemplo Árbol de decisión

- k-vecinos más cercanos (k-NN): El algoritmo k-vecinos más cercanos es un clasificador supervisado basado en el reconocimiento de patrones con criterios de vecindad. Parte de que una nueva muestra será clasificada a la clase a la cual pertenezca la mayor cantidad de vecinos más cercanos del conjunto de entrenamiento más cercano a ésta.

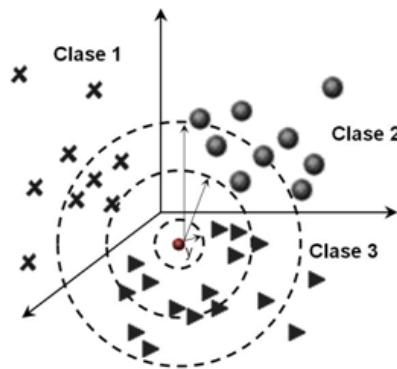


Figura 3.5: Ejemplo K vecinos más cercanos

- Clustering: Es un procedimiento de agrupación de una serie de vectores de acuerdo con criterios por lo general de distancia o similitud.

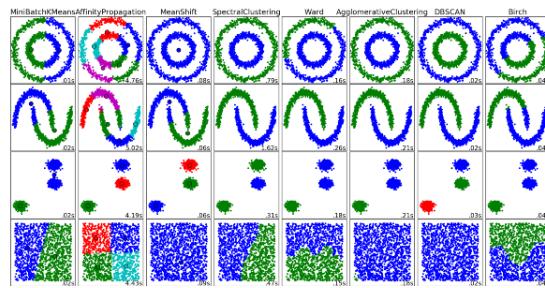


Figura 3.6: Ejemplo de diferentes ejecuciones del proceso de *Clustering*

- Regresión lineal: Permite determinar el grado de dependencia de las series de valores X e Y , prediciendo el valor y estimado que se obtendría para un valor x que no esté en la distribución.

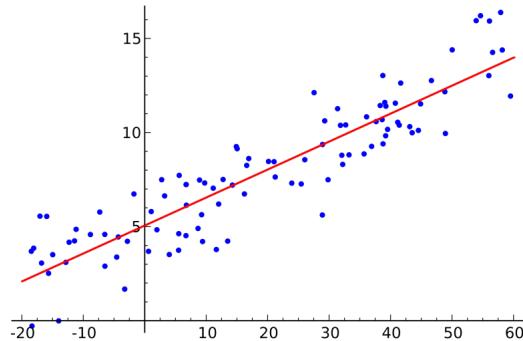


Figura 3.7: Ejemplo Regresión lineal

- Clasificador bayesiano ingenuo: Está basado en el teorema de Bayes con el supuesto de que existe independencia entre los predictores.

$$P(c | x) = \frac{P(x | c)P(c)}{P(x)}$$

Likelihood Class Prior Probability
 ↓ ↑
 Posterior Probability Predictor Prior Probability

$$P(c | X) = P(x_1 | c) \times P(x_2 | c) \times \dots \times P(x_n | c) \times P(c)$$

Figura 3.8: Ejemplo Teorema de Bayes

- Redes neuronales (NN): Son un paradigma de aprendizaje automático inspirado en las neuronas de los sistemas nerviosos de los mamíferos

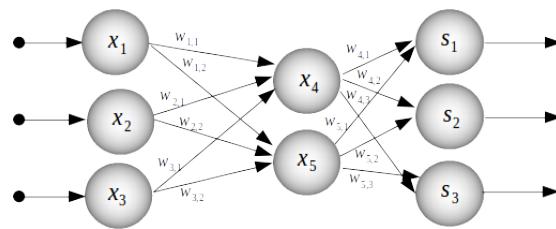


Figura 3.9: Ejemplo Red Neuronal

- Regresión logística: Es un tipo de análisis de regresión utilizado para predecir el resultado de una variable categórica en función de las variables independientes o predictoras.

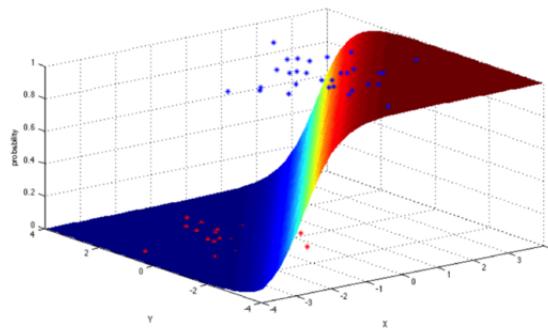


Figura 3.10: Ejemplo Regresión logística

- Máquinas de vector soporte (SVM): Son una serie de métodos y algoritmos de aprendizaje supervisado, usados para clasificación y regresión

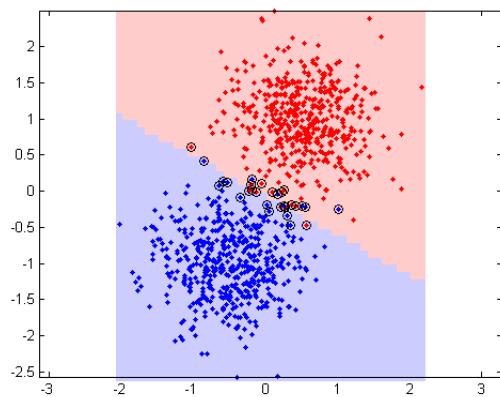


Figura 3.11: Ejemplo Máquina de vector soporte

3.3. Redes neuronales artificiales

Las redes de neuronas artificiales (RNA) son un paradigma de aprendizaje automático inspirado en las neuronas de los sistemas nerviosos de los mamíferos. Se trata de un sistema de enlaces de neuronas que colaboran entre sí para producir un estímulo de salida. Las conexiones tienen pesos numéricos que se adaptan según la experiencia. De esta manera, las redes neuronales se adaptan a un impulso y son capaces de aprender. La importancia de estas redes disminuyó durante un tiempo a causa del desarrollo de las máquinas de vector soporte y clasificadores lineales, pero volvió a surgir a finales de la década del 2000 con la llegada del aprendizaje profundo. En 1949 Hebb definió dos conceptos muy importantes y esenciales, que han sido fundamentales en el campo de las redes neuronales, basándose en investigaciones psicofisiológicas, tales como:

- El aprendizaje se localiza en las sinapsis o conexiones entre las neuronas.
- La información se representa en el cerebro mediante un conjunto de neuronas activas o inactivas.

Las hipótesis de *Hebb*, se sintetizan en la regla de aprendizaje de *Hebb*, que sigue siendo usada en los actuales modelos. Esta Regla nos dice que los cambios en los pesos de las sinapsis se basan en la interacción entre las neuronas pre y postsinápticas. La primera conferencia sobre la inteligencia artificial en la cual se discutió sobre la capacidad de las computadoras para simular el aprendizaje fue en 1956 en Dartmouth como se ha comentado previamente en el inicio del capítulo 3. Los principales asistentes a esta conferencia fueron: *John McCarty*(1927-2011), *Marvin Minsky*, *Nathaniel Rochester*(1919-2001), *Claude Elwood Shannon*(1916-2001), *Ray Solomonoff*(1926-2009), *Herbert Simon*(1916-2001), *Allen Newell*(1927-1992), *Oliver Selfridge*(1926-2008), *Trenchard More* y *Arthur Samuel*(1901 – 1990).



Figura 3.12: 50º Aniversario de la Conferencia de Dartmouth. (*More, McCarty, Minsky, Selfridge y Solomonoff*), de esta conferencia surgieron muchas de las teorías que actualmente usamos y ponemos en práctica.

3.4. Máquinas de vector de soporte o SVMs

Las máquinas de vector soporte (*Support Vector Machine* en inglés) son una serie de métodos y algoritmos de aprendizaje supervisado, usados para clasificación y regresión desarrollados por *Vladimir Vapnik* y su equipo en los laboratorios *AT&T*. Los algoritmos empleados utilizan un conjunto de ejemplos de entrenamiento clasificado en categorías para construir un modelo que prediga si un nuevo ejemplo pertenece a una u otra de dichas categorías.

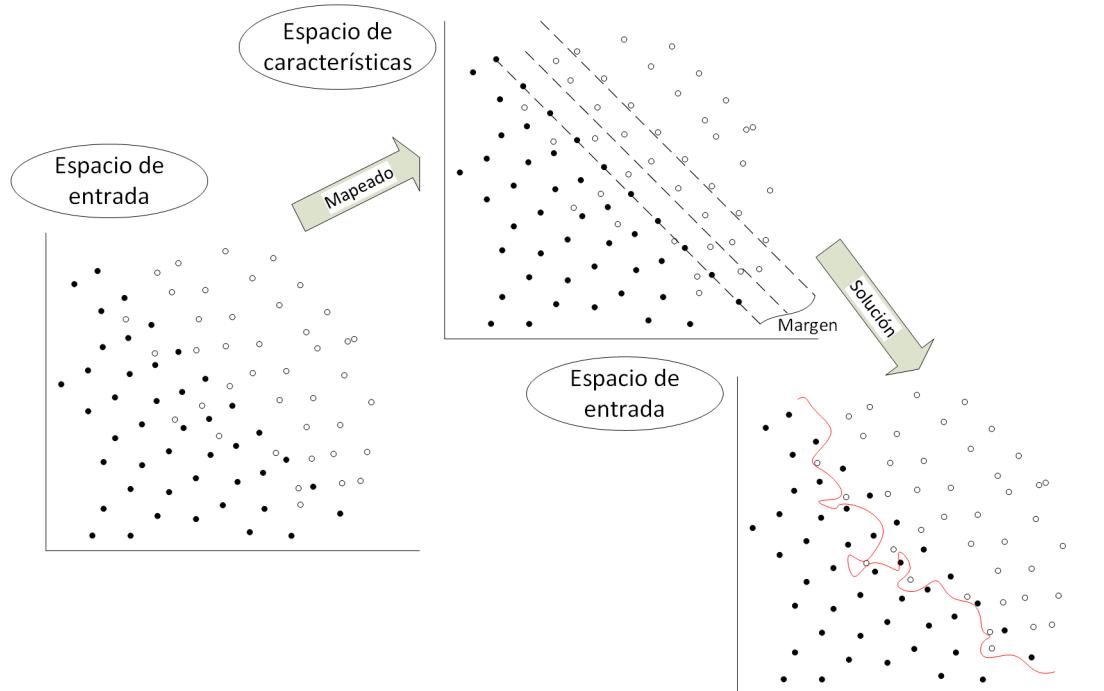


Figura 3.13: Visión general del proceso de clasificación de una máquina de vector soporte.

Estos modelos están estrechamente relacionados con las redes neuronales, de hecho, un modelo *SVM* que utiliza *kernel sigmoid* (Núcleo sigmoide) es equivalente a una red neuronal perceptron de dos capas. En el lenguaje de la literatura *SVM*, una variable predictora es un atributo extraído de los datos, y a este atributo transformado se le llama característica. La máquina de vector soporte o *SVM* busca un hiperplano que separe de forma óptima los puntos de una clase de la de otra, que eventualmente han podido ser previamente proyectados a un espacio de dimensionalidad superior. En ese concepto de separación óptima es donde reside la característica fundamental de las *SVM*: este tipo de algoritmos buscan el hiperplano que tenga la máxima distancia (margen) con los puntos que estén más cerca de él mismo. La manera más simple de realizar la separación es mediante una línea recta, un plano recto o un hiperplano N-dimensional. Desafortunadamente, los universos a estudiar no se suelen presentar en casos idílicos de dos dimensiones, sino que un algoritmo *SVM* debe tratar con varios aspectos: más de dos variables predictoras, curvas no lineales de separación, casos donde los conjuntos de datos no pueden ser completamente separados, clasificaciones en más de dos categorías.

3.5. Deep Learning

Deep Learning es la etiqueta para un conjunto de algoritmos que habitualmente se emplean para el entrenamiento no supervisado de redes neuronales multicapa. No existe una única definición de aprendizaje profundo o *Deep Learning*, en general, se trata de una clase de algoritmos para aprendizaje automático. El aprendizaje profundo o *Deep Learning* es una rama del aprendizaje automático que enseña a las computadoras lo que naturalmente los seres humanos hacen: aprender de la experiencia. Los algoritmos de aprendizaje de máquinas o *Machine Learning* utilizan métodos computacionales para aprender información directamente de los datos sin depender de una ecuación predeterminada como modelo. Este aprendizaje es especialmente adecuado para el reconocimiento de imágenes, lo cual es importante para resolver problemas como reconocimiento facial, detección de movimiento y muchas tecnologías avanzadas de asistencia al conductor, tales como conducción autónoma, detección de carriles, detección de peatones y estacionamiento autónomo. La popularidad del *Deep Learning* se basa en que el método conocido como descenso por gradiente estocástico (*SGD*) puede entrenar eficazmente redes neuronales de entre 10 y 20 capas para resolver problemas que ocurren en la práctica. Lo cual es relevante porque el reajuste de pesos en una red neuronal a partir de los errores es un problema de optimización no convexa NP-Completo.

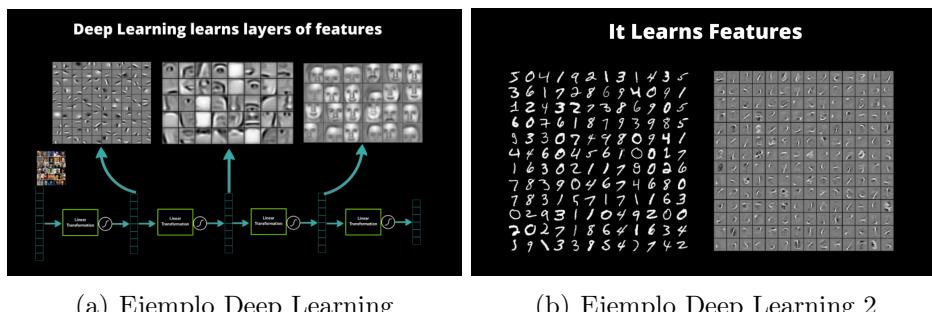


Figura 3.14: Ejemplos de Deep Learning

En el enfoque para el *Deep Learning* se usan estructuras lógicas que se asemejan a la organización del sistema nervioso de los mamíferos, teniendo capas de unidades de procesamiento (neuronas artificiales) que se especializan en detectar determinadas características existentes en los objetos percibidos. La visión artificial es una de las áreas donde el *Deep Learning* proporciona una mejora considerable en comparación con algoritmos más tradicionales. Existen varios entornos y bibliotecas de código de *Deep Learning* que se ejecutan en las potentes *GPUs* modernas tipo *CUDA*, como por ejemplo *NVIDIA cuDNN*.

3.5.1. Evolución y progreso

Nos encontramos ante una serie de problemas que han surgido a partir del nacimiento de la visión por computador y el *Deep Learning* y que se han intentado resolver desde entonces adoptando varias estrategias. En la siguiente figura tenemos una gráfica con el progreso en la precisión de los diferentes sistemas implementados en las últimas 4 décadas.

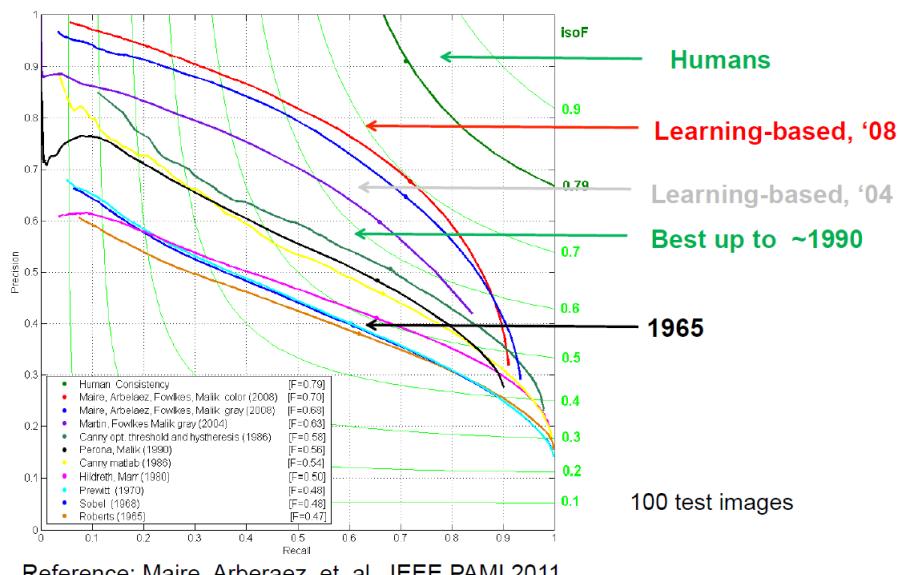


Figura 3.15: Progreso en el desarrollo de soluciones para problemas de visión por computador en las últimas 4 décadas

En un principio no se tenían claras cuales serían las ramas por las que se desarrollaría la inteligencia artificial y las redes neuronales, debido a nuevas necesidades tecnológicas este área se ha desarrollado fuertemente a partir del año 2010 gracias a la introducción del *Deep Learning*.

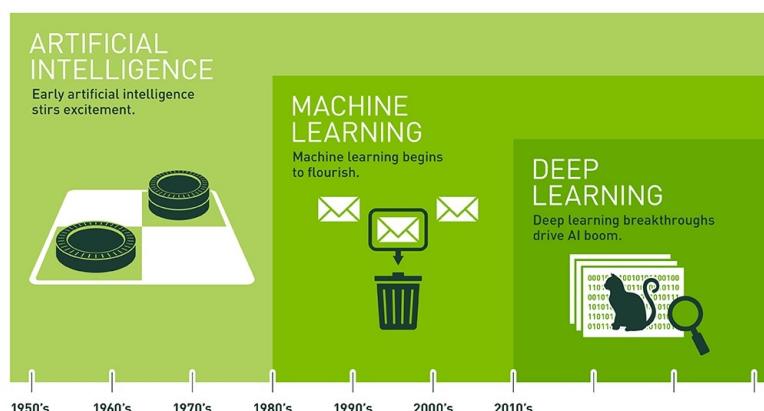


Figura 3.16: Evolución hasta Deep Learning

Capítulo 4

Introducción a las redes neuronales artificiales y redes neuronales convolucionales

Las Redes Neuronales (*Neuronal Networks* en inglés) fueron originalmente una simulación abstracta de los sistemas nerviosos biológicos, constituidos por un conjunto de unidades llamadas neuronas o nodos conectados unos con otros. El primer modelo de red neuronal fue propuesto en 1943 por *McCulloch* y *Pitts* en términos de un modelo computacional de actividad nerviosa. Este modelo era un modelo binario, donde cada neurona tenía un escalón o umbral prefijado y sirvió de base para los modelos posteriores. Una primera clasificación para estos modelos podría ser:

- **Modelos inspirados en la Biología:** Estos modelos comprenden las redes que tratan de simular los sistemas neuronales biológicos, así como ciertas funciones como las auditivas o de visión. Suelen ser empleados para temas relacionados con investigación e interpretación del cerebro, por ejemplo, dentro del ámbito médico.

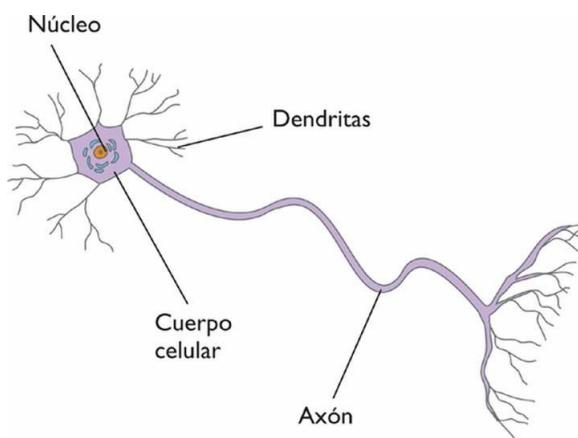


Figura 4.1: Arquitectura básica de una neurona biológica

CAPÍTULO 4. INTRODUCCIÓN A LAS REDES NEURONALES ARTIFICIALES Y REDES NEURONALES CONVOLUCIONALES

- **Modelos artificiales aplicados:** Estos modelos no tienen por qué guardar similitud estricta con los sistemas biológicos. Sus arquitecturas están bastante ligadas a las necesidades de las aplicaciones para las que son diseñadas. Las redes neuronales artificiales y las redes neuronales convolucionales entrarían dentro de esta categoría.

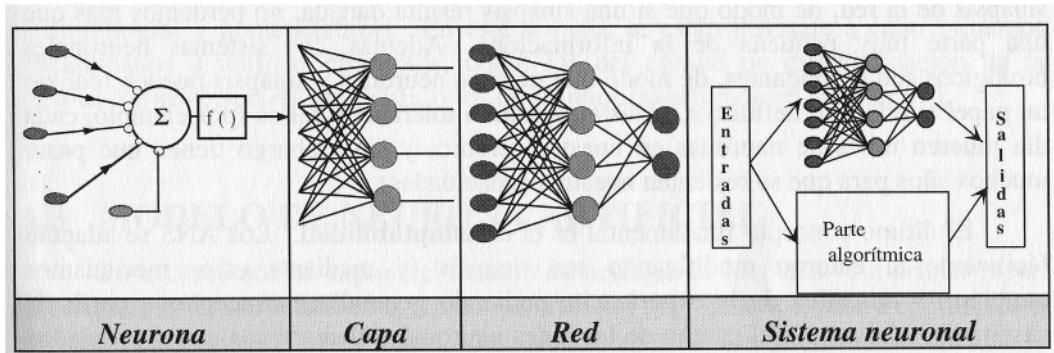


Figura 4.2: Sistema global de proceso de una red neuronal

Una red neuronal artificial clásica se compone de unidades llamadas neuronas. Cada neurona recibe una serie de entradas a través de interconexiones y emite una salida, como muestra la figura 4.2. Esta salida viene dada por tres funciones:

- 1 . Una función de propagación (también conocida como función de excitación), que por lo general consiste en el sumatorio de cada entrada multiplicada por el peso de su interconexión (valor neto). Si el peso es positivo, la conexión se denomina excitatoria y si es negativo, se llama inhibitoria.
- 2 . Una función de activación que modifica a la anterior. Y si no existe, la salida es la misma función de propagación.
- 3 . Una función de transferencia, que se aplica al valor devuelto por la función de activación. Se utiliza para acotar la salida de la neurona y generalmente viene dada por la interpretación que queramos darle a dichas salidas. Algunas de las más utilizadas son la función *sigmoide* (para obtener valores en el intervalo [0,1]) y la tangente hiperbólica (para obtener valores en el intervalo [-1,1]).

CAPÍTULO 4. INTRODUCCIÓN A LAS REDES NEURONALES ARTIFICIALES Y REDES NEURONALES CONVOLUCIONALES

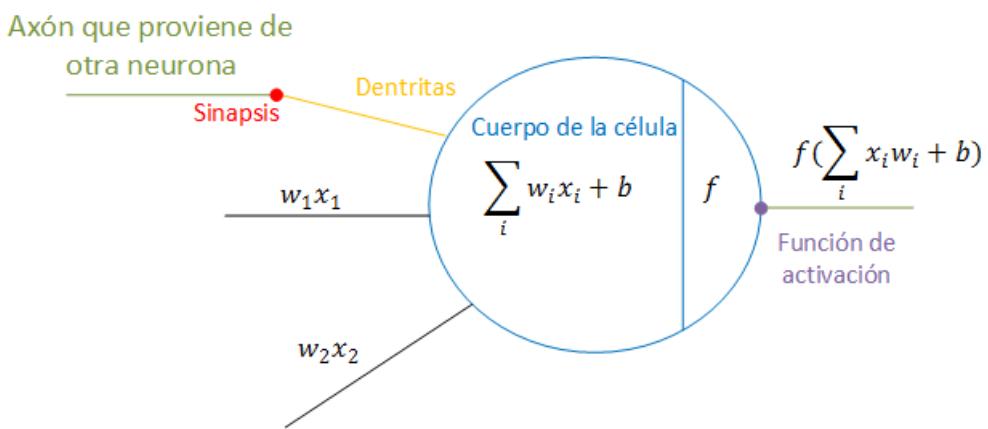


Figura 4.3: Arquitectura básica de una neurona artificial

El funcionamiento detallado de la célula o perceptron está representado en la siguiente figura:

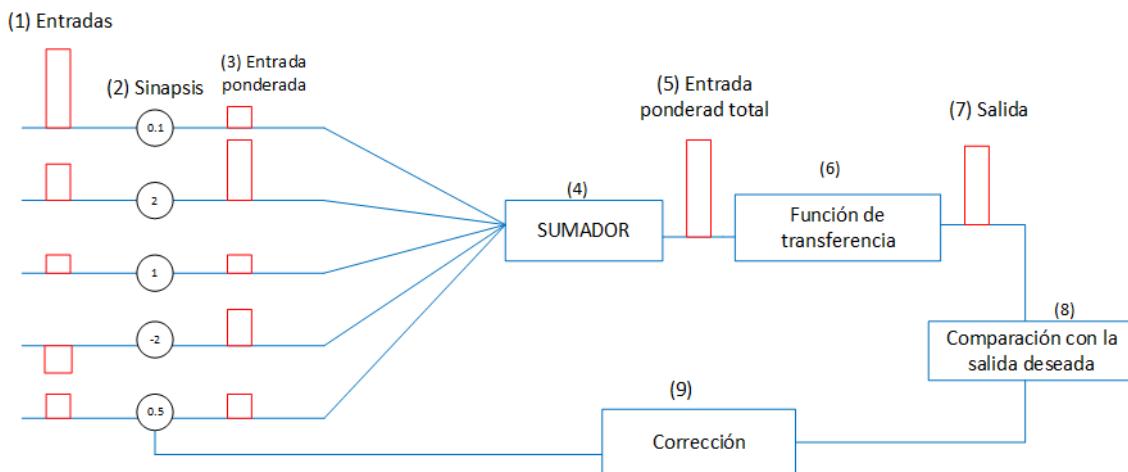


Figura 4.4: Arquitectura básica de una neurona artificial

Como se aprecia en la figura 4.4, la neurona o perceptron básico realiza varios "pasos", primero la entrada se multiplica por el peso de la sinapsis y se obtiene la salida ponderada (paso (3)) para esa entrada, se suma con la función sumador y se obtiene la salida ponderada total (paso (5)). Una obtenida la entrada ponderada total, se le aplica la función de transferencia con la que finalmente tenemos la salida de la red neuronal, es decir, la predicción. Esta predicción se compara con la salida deseada y se hacen las correcciones necesarias en las sinapsis correspondientes y se vuelve a repetir el proceso para otro dato.

4.1. Redes neuronales convolucionales

“Las redes neuronales convolucionales son muy similares a las redes neuronales ordinarias, se componen de neuronas que tienen pesos y sesgos que pueden aprender. Cada neurona recibe algunas entradas, realiza un producto escalar y luego aplica una función de activación. Al igual que en el perceptrón multicapa también tenemos una función de pérdida o coste (por ejemplo, SVM / Softmax) sobre la última capa, la cual estará totalmente conectada. Lo que diferencia a las redes neuronales convolucionales, es que estas son diseñadas especialmente para procesar datos estructurados en 2D como imágenes o señales de voz presentadas en espectrogramas. Sin embargo, se pueden adaptar para procesar datos en arreglos multidimensionales de 1D, 3D o más. Su principal ventaja es que tienen menos parámetros a entrenar que una red multicapa con conexión total del mismo número de capas ocultas, por lo que su entrenamiento es más rápido.” [3]

Este tipo de redes son una variación del perceptrón multicapa y su función es tratar de buscar características locales en pequeños grupos de entradas (en el caso de las imágenes, de píxeles), como puedan ser bordes o colores más o menos homogéneos. Para este tipo de redes, la entrada ya no es un vector, sino una imagen o un audio. Para llevar a cabo esta idea, ponemos un mismo grupo de neuronas por cada grupo de entradas (por ejemplo, un cuadrado de 3x3 píxeles en una imagen o una secuencia de 4 mediciones en un archivo de sonido). La idea es que todos los elementos que metamos en la capa (llamada capa de convolución) tienen los mismos pesos por cada entrada, y se reduce considerablemente el número de parámetros. Si metemos más capas, la red neuronal podrá descubrir más y más características complejas de la imagen: se puede empezar por colores o bordes orientados y acabar con capas que se activan con formas circulares o cuadradas, por ejemplo. Los fundamentos de estas redes se basan en el *Neocognitron*, introducido por *Kunihiko Fukushima* en 1980 [5]. Este modelo fue más tarde mejorado por *Yann LeCun* en 1998 [6] al proporcionar un método de aprendizaje basado en *Backpropagation* para poder entrenar correctamente al sistema. En el año 2012, fueron refinadas por *Dan Ciresan* e implementadas en una *GPU* consiguiendo muy buenos resultados [4].

Actualmente, los algoritmos neuronales avanzan con tal rapidez que las arquitecturas particulares que hoy son efectivas puede que mañana se reemplacen en beneficio de otras, lo que significa que hablamos de un campo en continua evolución, además de tener extensas aplicaciones en reconocimiento de imágenes y vídeo, sistemas de recomendación y procesamiento del lenguaje natural. Por lo que la elección de la arquitectura para nuestra red convolucional es un pilar muy importante para el buen funcionamiento del sistema.

CAPÍTULO 4. INTRODUCCIÓN A LAS REDES NEURONALES ARTIFICIALES Y REDES NEURONALES CONVOLUCIONALES

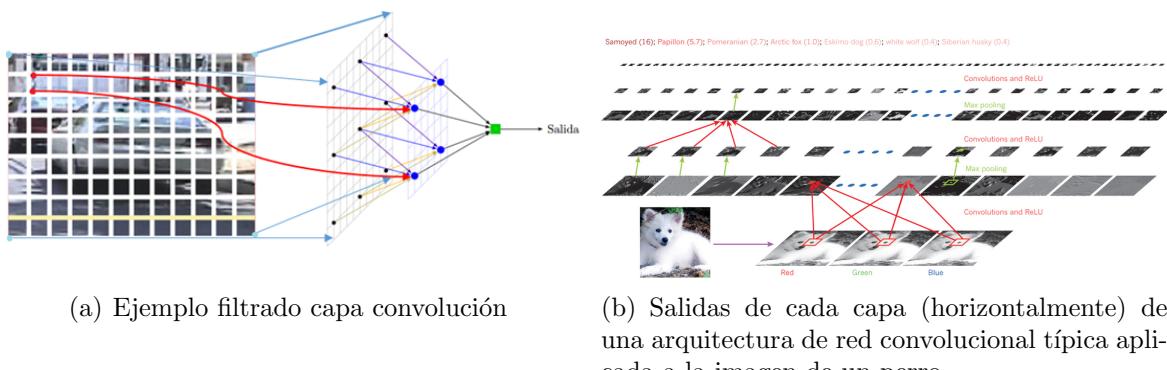


Figura 4.5: Ejemplo capa convolución

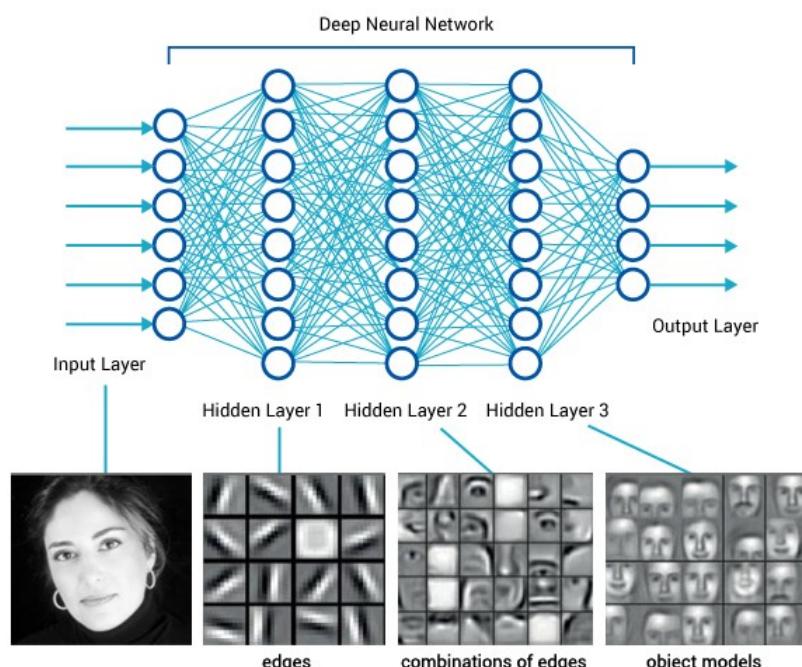


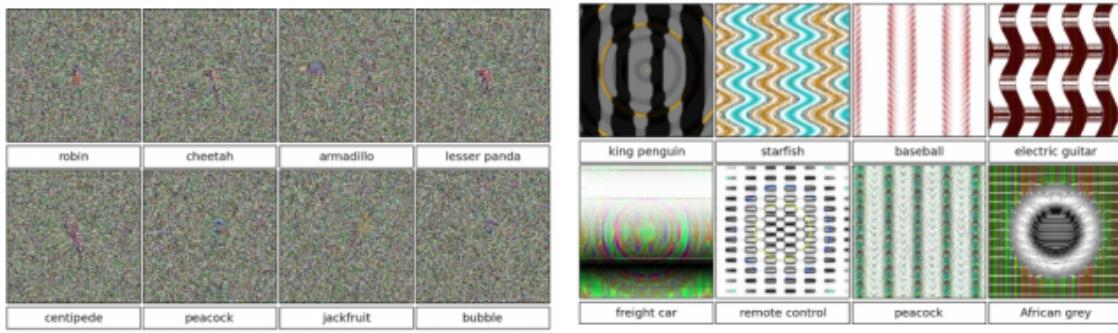
Figura 4.6: Ejemplo de red neuronal convolucional y las salidas de las diferentes capas de convolución

En la figura 4.6 tenemos la representación de las diferentes salidas de las capas de convolución de una red neuronal convolucional que está destinada al reconocimiento facial. Como observamos, la primera capa de convolución se centra en captar patrones bastante sencillos (Rectas y alguna curva), mientras que las posteriores capturan patrones más complejos y en las ultimas capas se juntan para formar la predicción que la capa de clasificación nos otorga.

A modo de curiosidad, unos investigadores usaron una red neuronal para generar imágenes que engañaban a otra red neuronal diseñada para reconocer objetos. Lo que a nosotros nos parece una imagen aleatoria, para la red neuronal es un bikini o un armadillo.

CAPÍTULO 4. INTRODUCCIÓN A LAS REDES NEURONALES ARTIFICIALES Y REDES NEURONALES CONVOLUCIONALES

Es parte del problema del sobreajuste: redes que se comportan muy bien para los datos de ejemplo o parecidos, pero que con datos muy distintos dan resultados absurdos.



(a) Ejemplo 1 imágenes generadas

(b) Ejemplo 2 imágenes generadas

Figura 4.7: Imágenes generadas por una red neuronal convolucional para engañar a otra

Sea como sea, es un campo muy interesante y que promete bastantes avances a corto plazo sobre todo en reconocimiento de imagen y de sonido. Entre las diversas aplicaciones de estas redes destacan las siguientes:

Aplicaciones comerciales:

- LeNet: Lectura de cheques automática usado en U.S.A. y Europa en los 90s en un DSP por AT&T.
- Reconocimiento de números y letras manuscritas incluyendo caracteres arábigos y chinos por Microsoft.
- Detección y borrado automático de rostros y placas vehiculares para protección de la privacidad en Google Street View.
- Identificación del género y la edad de los usuarios de máquinas expendedoras por la empresa NEC Labs en Japón.
- Detección y seguimiento de clientes en supermercados por NEC Labs.
- Detección de intrusos y ladrones por AgilityVideo.
- Para mejorar el reconocimiento de voz de Android y ahorrar electricidad en sus centros de datos por Google.
- Motores de búsqueda.
- Diagnósticos médicos.
- Vehículos autónomos.

CAPÍTULO 4. INTRODUCCIÓN A LAS REDES NEURONALES ARTIFICIALES Y REDES NEURONALES CONVOLUCIONALES

Aplicaciones experimentales:

- Detección de rostros con record en velocidad y precisión con GPU.
- Identificación de expresiones faciales.
- Evasión de obstáculos para robots móviles usando visión.
- Detección de obstáculos a gran distancia.
- Segmentación de imágenes biológicas.
- Restauración de imágenes (eliminación del ruido).
- Reconstrucción de circuitos neuronales a partir de imágenes transversales del cerebro con espesor nanométrico.
- Cuando combinan las redes neuronales con algoritmos evolutivos, una computadora puede aprender a jugar Mario Bros. (MarI/o, enlace a video de demostración: <https://www.youtube.com/watch?v=qv6UVQ0F44>).
- Robot que aprende a hacer tareas.

Uno de los ejemplos más recientes y que parece haber conmocionado a los países asiáticos es el de Alphago, el sistema de inteligencia artificial (IA) desarrollado por Google DeepMind. AlphaGo no sólo ha demostrado que es capaz de jugar al Go, sino que recientemente ha derrotado a Lee Sedol, el campeón del mundo de esta disciplina en un torneo a 5 partidas, con un parcial de 4-1 a su favor.



(a) Agility Video



(b) AttyT



(c) Microsoft



(d) Nec Laboratories america

Figura 4.8: Algunas de las empresas que utilizan redes neuronales convolucionales o neuronales artificiales

4.1.1. Flujo de trabajo con redes neuronales convolucionales

Con este tipo de redes, el flujo de trabajo se divide principalmente en 2 fases :

1. Fase de entrenamiento: Se usa un conjunto de datos de entrenamiento, en este caso imágenes de los diferentes gestos de la mano, para determinar los pesos (parámetros) que definen el modelo de la red neuronal. Se calculan de manera iterativa de acuerdo con los valores de entrenamiento, con el objeto de minimizar el error cometido entre la salida obtenida por la red neuronal y la salida deseada.

2. Fase de Prueba o Test: Para evitar el problema del sobreajuste, es aconsejable utilizar un segundo grupo de datos diferentes a los de entrenamiento, el grupo de validación, que permite controlar el sobreajuste del proceso de aprendizaje.

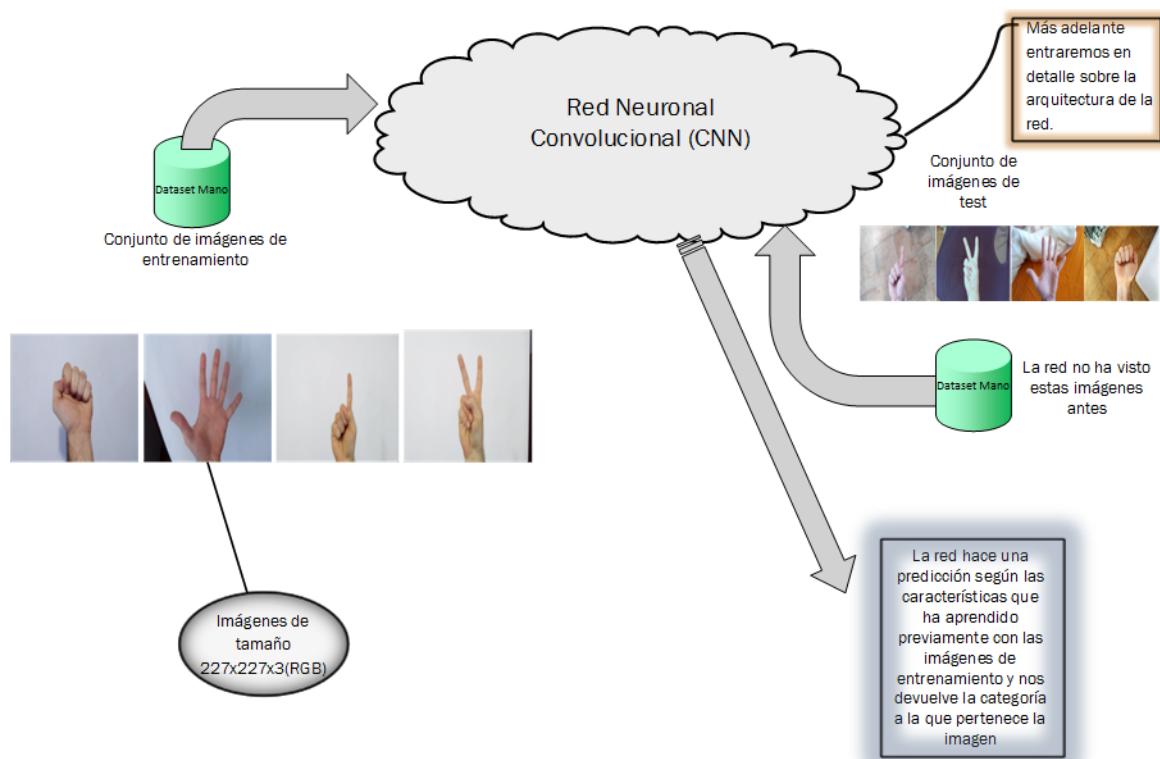


Figura 4.9: Ejemplo flujo de trabajo con redes neuronales convolucionales

3. Como flujo de trabajo adicional usando las redes neuronales convolucionales, uno de los flujos de trabajo implementados en el capítulo 5, no obtendrá una clasificación directa mediante la red neuronal convolucional, si no que se empleará una máquina de vector soporte o *SVM* para clasificar las características de las imágenes, obtenidas con la red neuronal.

CAPÍTULO 4. INTRODUCCIÓN A LAS REDES NEURONALES ARTIFICIALES Y REDES NEURONALES CONVOLUCIONALES

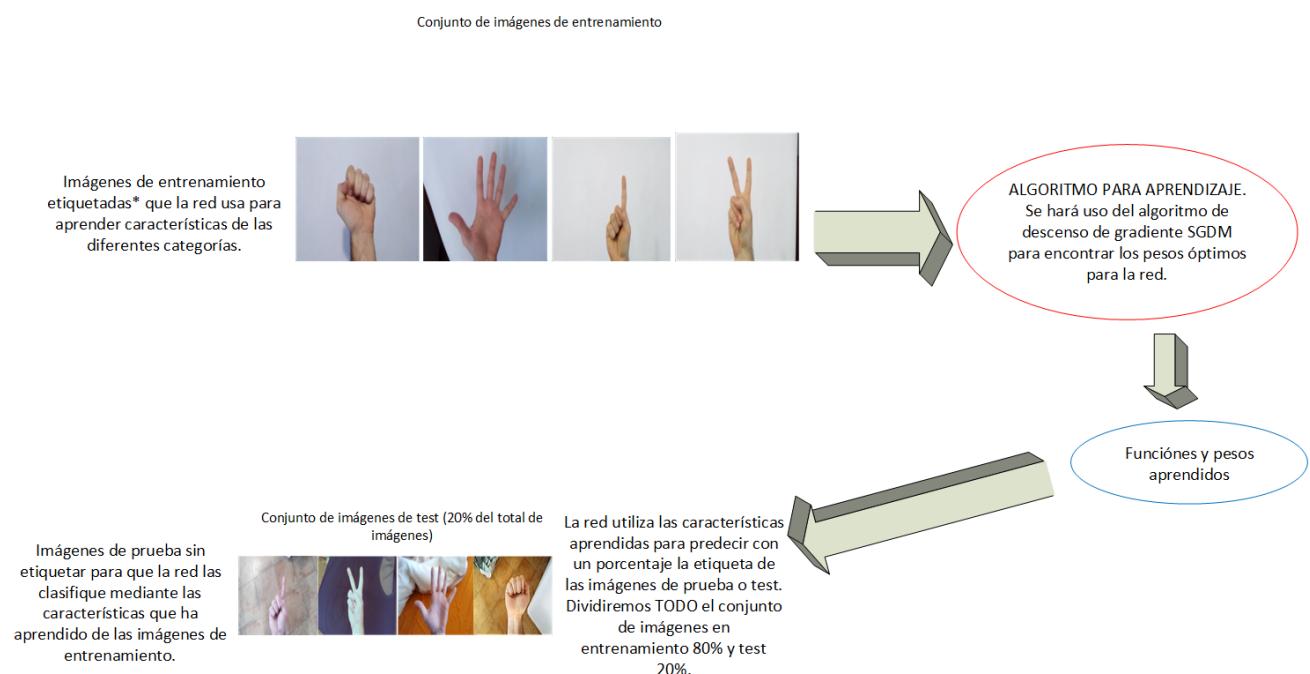


Figura 4.10: Ejemplo flujo de trabajo con redes neuronales convolucionales

Como podemos observar en la figura 4.11, sobre el conjunto de imágenes de entrenamiento y aplicando el algoritmo de descenso de gradiente, se obtienen los pesos (Valor numéricico decimal) de las conexiones entre neuronas, que posteriormente serán los que se aprenderán y usarán para el aprendizaje en capas posteriores. Esta figura es un ejemplo de los pesos aprendidos en una red neuronal, este esquema de pesos en una escala mayor, es el utilizado por las redes neuronales convolucionales. Donde cada neurona debe aprender los pesos de los 3 canales del espacio de color de la imagen de entrada.

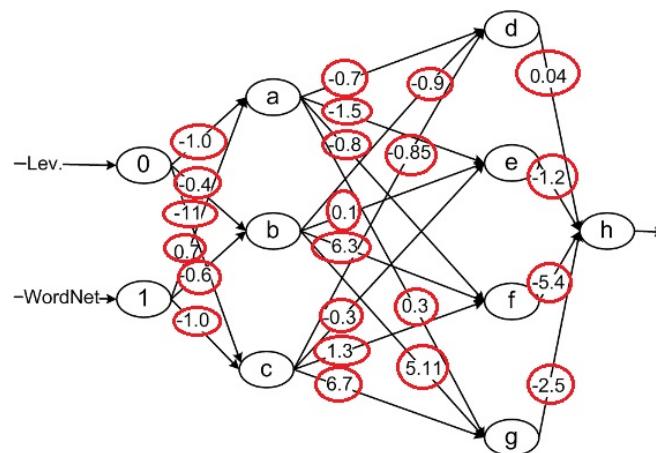


Figura 4.11: Pesos en una red neuronal y red neuronal convolucional (Círculos rojos)

Estos pesos son valores tanto negativos como positivos, un peso positivo indicaría que ese camino tiene más peso sobre los otros, un peso negativo indicaría lo contrario.

4.2. Funciones y características de las redes neuronales convolucionales

En este capítulo se explicarán las principales características y funcionalidades de las diferentes capas que intervienen en la red neuronal convolucional que emplearemos en este proyecto, la mayoría de las arquitecturas de red neuronal convolucional contienen estas capas. Algunas de las ecuaciones expuestas en esta sección corresponden con las usadas en funciones de MATLAB que emplearemos en el código de los diferentes flujos de trabajo.

4.2.1. Capas de convolución

Una capa convolucional consiste en un conjunto de neuronas que se conectan a pequeñas regiones de la entrada o de la capa anterior. Estas regiones se llaman filtros. Para cada región, se calcula un producto escalar de los pesos por la entrada, y luego se agrega un término de polarización (*Bias*). El filtro se mueve a lo largo de la entrada verticalmente y horizontalmente, repitiendo el mismo cálculo para cada región, es decir, convolucionando la entrada. El tamaño del paso con el que se mueve el filtro de convolución se llama *Stride*.

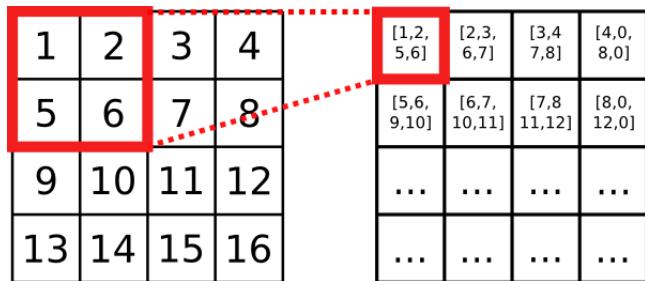


Figura 4.12: Ejemplo convolución tamaño de ventana 2x2 Entrada a la (Izquierda y salida convolución a la derecha)

El número de pesos utilizados para un filtro de convolución es:

$$h * w * c$$

Donde h es la altura, w es el ancho del tamaño del filtro, y c es el número de canales en la entrada (Por ejemplo, si la entrada es una imagen a color, el número de canales será de tres). A medida que un filtro se mueve a lo largo de la entrada, utiliza el mismo conjunto de pesos y sesgos para la convolución, formando un mapa de características. La capa de convolución generalmente tiene múltiples mapas de características, cada uno con un conjunto diferente de pesos y un sesgo. El número de mapas de características está determinado por el número de filtros, es decir, para un número elevado de filtros, tendremos bastantes más mapas que nos ayudarán para el proceso de clasificación.

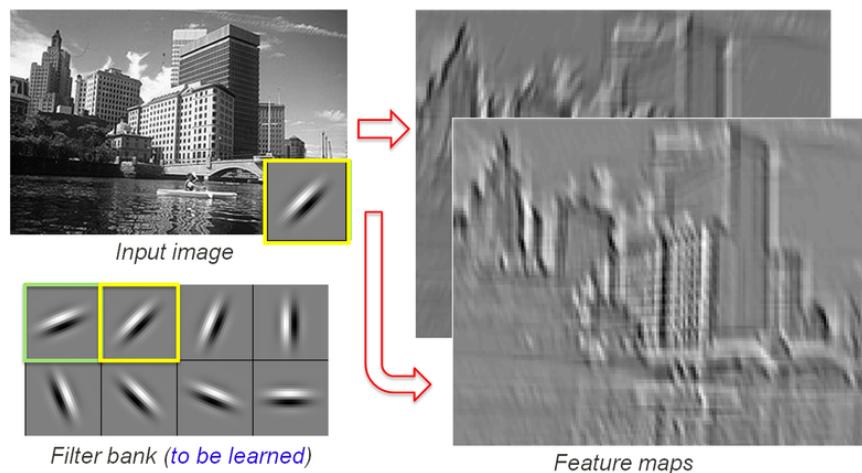


Figura 4.13: Ejemplo de diferentes mapas de características producidos por una capa de convolución, estos mapas suelen ser más pequeños que la imagen de entrada

Dentro de una capa de convolución tenemos una serie de hiperparámetros que se deberán ajustar para el correcto funcionamiento:

1. La profundidad del volumen de salida, que corresponde con el número de filtros que usaremos, cada uno aprendiendo a buscar algo diferente de la entrada. Por ejemplo, si la primera capa convolucional toma como entrada la imagen en crudo, entonces diferentes neuronas a lo largo de las diferentes dimensiones de la imagen pueden activarse en presencia de varios bordes orientados o manchas de color.
2. El paso o *Stride* con el que se desliza el filtro de convolución. Cuando este hiperparámetro es igual a 1 entonces movemos los filtros de un pixel en un pixel. Cuando el paso es igual a 2 (o muy pocas veces 3 o más, aunque esto es raro en la práctica), los filtros saltan de 2 píxeles en 2 píxeles. Esto producirá volúmenes de salida más pequeños espacialmente.

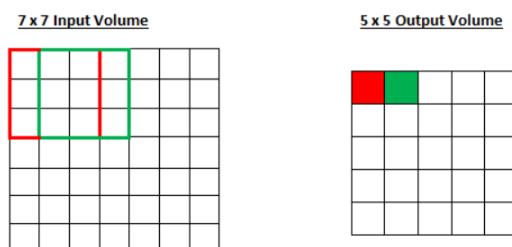


Figura 4.14: Ejemplo paso (*Stride*) igual a 1

3. El tamaño del relleno con ceros o *Zero padding*. Este hiperparámetro nos permitirá controlar el tamaño espacial de los volúmenes de salida. Lo normal será preservar exactamente el tamaño espacial del volumen de entrada de modo que el ancho de entrada y de salida sean los mismos.

CAPÍTULO 4. INTRODUCCIÓN A LAS REDES NEURONALES ARTIFICIALES Y REDES NEURONALES CONVOLUCIONALES

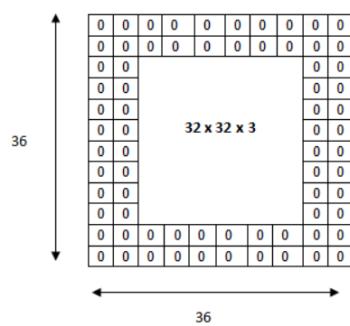


Figura 4.15: Ejemplo relleno con ceros igual a 2 (*Zero padding*) para una imagen de 32x32x3

Se puede calcular el tamaño espacial del volumen de salida como una función del tamaño del volumen de entrada W , el tamaño del campo receptivo de las neuronas de la capa de convolución F , el paso con el que se mueven los filtros de convolución S y la cantidad de relleno con ceros P usada en los bordes. Puede convencerse de que la fórmula correcta para calcular cuántas neuronas encajan está dada por:

$$(W - F - 2P)/S + 1$$

Por ejemplo, para una entrada 7x7 y un filtro 3x3 con *Stride* 1 y *Padding* 0 obtendríamos una salida 5x5. Con el *Stride* igual a 2 obtendríamos una salida de 3x3.

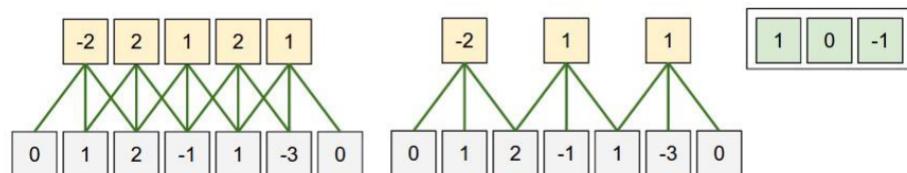


Figura 4.16: Izquierda *Stride*=1 (Salida de 5x5 color carne). Derecha *Stride*=2 (Salida de 3x3 color carne)

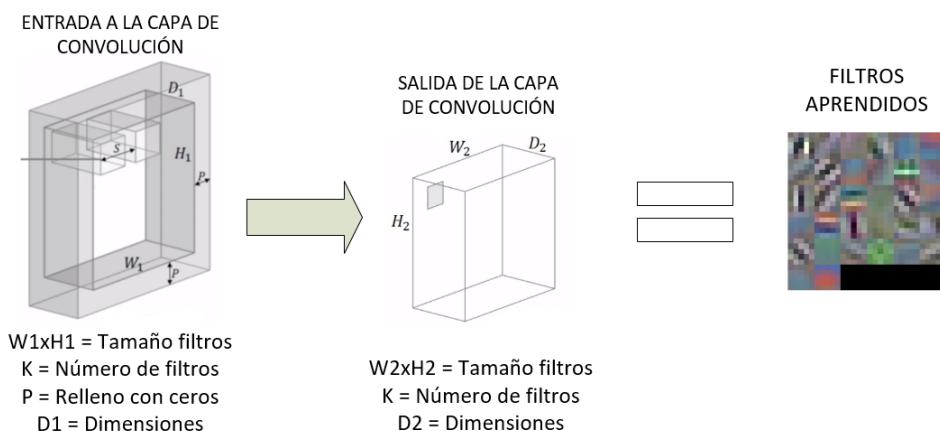


Figura 4.17: Ejemplo convolución (Izquierda entrada)

4.2.2. Capas Relu o Unidades de Rectificación Lineal

En el contexto de las redes neuronales artificiales y las redes neuronales convolucionales, el rectificador es una función de activación definida como:

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

Donde x es la entrada a una neurona. También se llama función de rampa y es análogo a la rectificación de media onda en la ingeniería eléctrica. Esta función de activación se introdujo por primera vez en una red dinámica por *Hahnloser* [7]. El rectificador es, a partir de 2015, la función de activación más popular para las redes neuronales profundas. Una unidad que emplea el rectificador también se denomina unidad rectificada lineal (*ReLU*). Una aproximación suave al rectificador es la función analítica:

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

Que se denomina función *softplus*. La derivada de *softplus* es la función logística:

$$f'(x) = \frac{e^x}{e^x + 1} = \frac{1}{1 + e^{-x}}$$

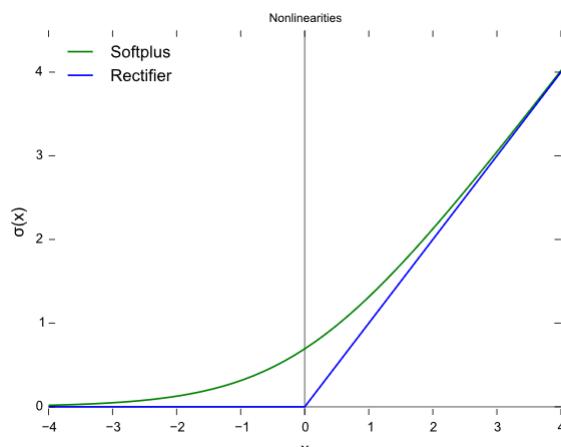


Figura 4.18: Ejemplo Función Rectificador Lineal

Una capa convolucional suele estar seguida por una función de activación no lineal. Esta realiza una operación de umbral para cada elemento, donde cualquier valor de entrada menor que cero se pone a cero, es decir:

$$f(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

4.2.3. Capas Pooling o de agrupación

La función de las capas *pooling* es reducir progresivamente el tamaño espacial de la representación de los datos, para reducir también la cantidad de parámetros y la computación en la red, y por lo tanto también controlar el *overfitting* o sobreajuste. La capa de agrupación redimensiona espacialmente cada segmento de la entrada, utilizando la operación *máximo (max)* o *media (avg)*. La forma más común es una capa de agrupación con filtros de tamaño 2x2 aplicado con un *stride*(paso) de 2.

La capa de agrupación no realiza ningún aprendizaje, simplemente realiza una operación de muestreo descendente. Para regiones que no se superponen (El tamaño de la agrupación y el *Stride* (Paso) deben de ser iguales), si la entrada a la capa de agrupación media es $n*n$, y el tamaño de la región de agrupación es $h*h$, entonces se toman muestras de las regiones por h en ambas direcciones. Es decir, la salida de la capa de agrupación media para un canal de una capa convolucional es :

$$\frac{n}{h} * \frac{n}{h} = \frac{n^2}{h^2}$$

Para regiones superpuestas, la salida de una capa de agrupación es:

$$(Tam_{entrada} - Tam_{agrupacion} + 2 * Zero_{padding}) / (Tam_{stride} + 1)$$

Donde $Tam_{entrada}$ corresponde con el tamaño de entrada de los datos, $Tam_{agrupacion}$ corresponde con el tamaño de la capa de agrupación, $Zero_{padding}$ corresponde con el tamaño del relleno con ceros, y Tam_{stride} corresponde con el tamaño del paso con el que se mueve el filtro de agrupación.

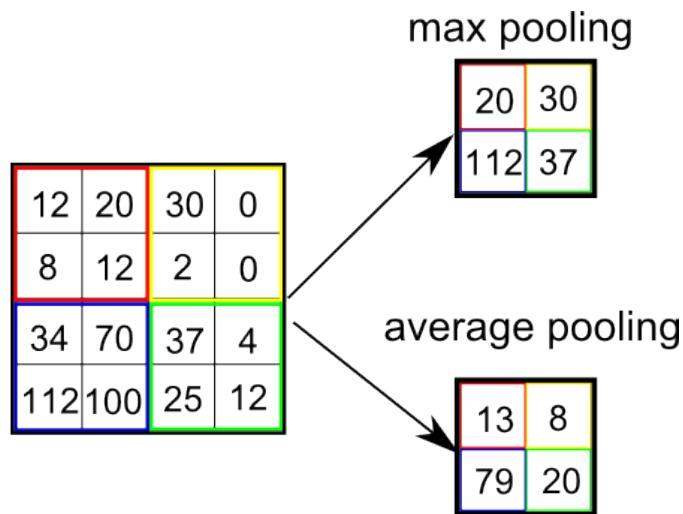


Figura 4.19: Ejemplo Función de agrupación usando distintas máscaras (Media y máximo)

4.2.4. Capas Fully connected o totalmente conectadas

La capa *Fully connected* o capa totalmente conectada, tiene conexiones completas con todas las activaciones en la capa anterior, como se ve en la figura 4.20. Por tanto, sus activaciones pueden calcularse con una multiplicación matricial seguida de un desplazamiento de polarización. En las arquitecturas típicas de redes neuronales convolucionales, nos encontraremos con 2 o más de estas capas, además, las capas convolucionales y de *pooling* suelen estar seguidas por una o más capas de este tipo.

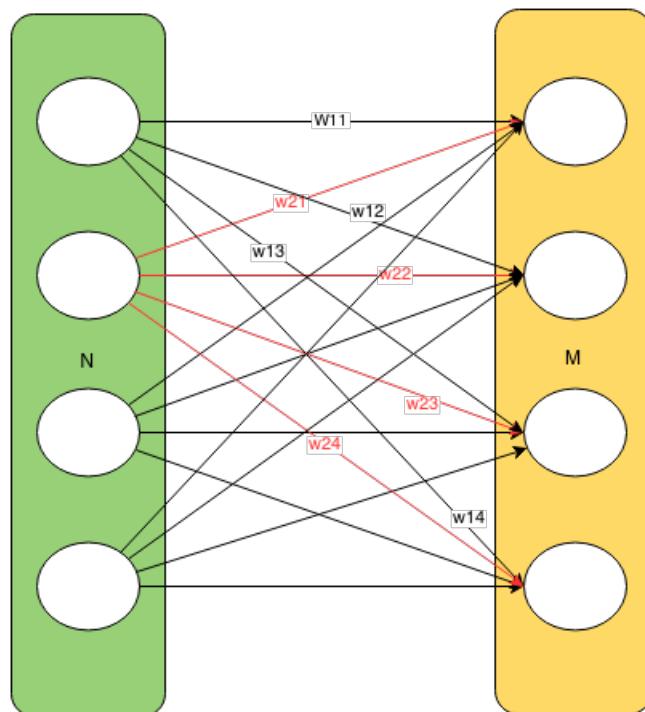


Figura 4.20: Ejemplo Capa totalmente conectada

Como su nombre indica, todas las neuronas en una capa completamente conectada se conectan a las neuronas en la capa anterior, combinando todas las características (información local) aprendidas por las capas anteriores a través de la imagen para identificar los patrones más grandes. Para problemas de clasificación, la última capa completamente conectada los combina para clasificar las imágenes. Es por eso que el parámetro *Output-Size* (Tamaño de salida) en la última capa completamente conectada es igual al número de clases en los datos de destino. Para problemas de regresión, el tamaño de salida debe ser igual al número de variables de respuesta. También se puede ajustar la velocidad de aprendizaje y los parámetros de regularización para esta capa para problemas de *Transfer Learning* (Transmisión de conocimientos).

4.2.5. Capas Dropout o de exclusión

Las capas de exclusión o *Dropout* tienen una función muy específica dentro de las redes neuronales y redes neuronales convolucionales. Se suelen usar para intentar reducir el problema del sobreajuste u *overfitting*, donde después del entrenamiento, los pesos de la red están muy ajustados a los ejemplos de entrenamiento y la red no funciona bien cuando se le dan nuevos ejemplos. La idea de la capa de exclusión es “eliminar” un conjunto aleatorio de activaciones en esa capa estableciéndolas a cero. Esto, de alguna manera, obliga a la red a ser redundante. Con esto se quiere decir que la red debe ser capaz de proporcionar la clasificación o salida correcta para un ejemplo específico incluso si algunas de las activaciones son eliminadas. Una nota importante es que esta capa sólo se utiliza durante el entrenamiento, y no durante el tiempo de prueba.[8][10]

La siguiente figura muestra a la derecha el resultado de aplicar dropout a una red neuronal, lo que obtenemos es que se han desechado algunas de las conexiones entre neuronas.

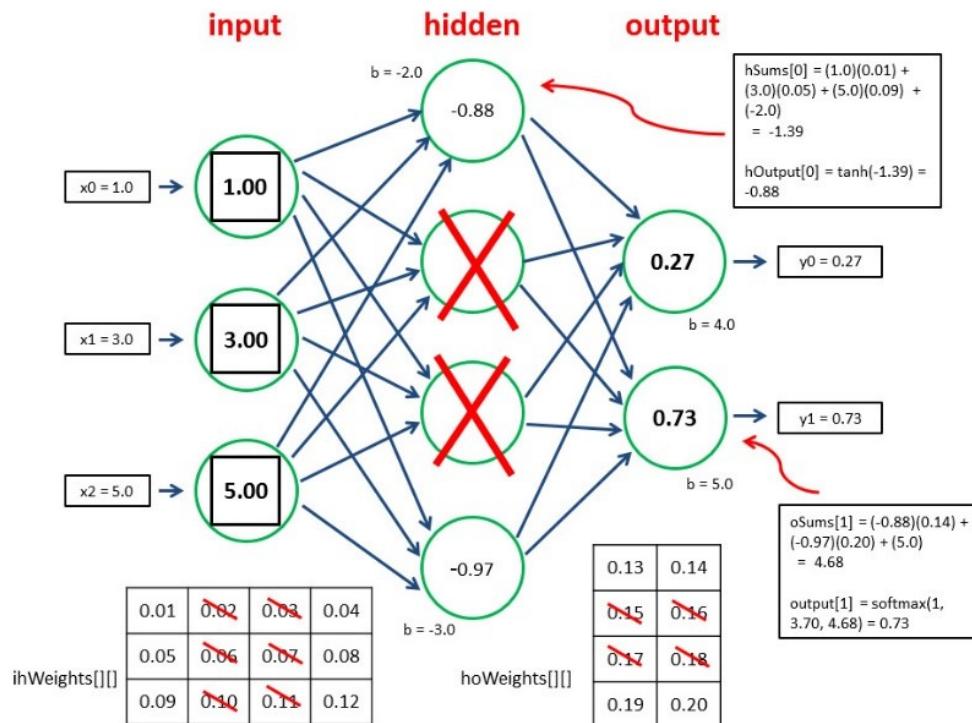


Figura 4.21: Ejemplo eliminación de activaciones (pesos) usando *Dropout*

CAPÍTULO 4. INTRODUCCIÓN A LAS REDES NEURONALES ARTIFICIALES Y REDES NEURONALES CONVOLUCIONALES

A modo de curiosidad vamos a ver las tasas de error para las diferentes arquitecturas y como este error va variando en función de si añadimos capas *Dropout* o no [10]. Para el Dataset “*The MNIST database of handwritten digits*” tenemos las siguientes arquitecturas:

Method	Unit Type	Architecture	Error %
Standard Neural Net (Simard et al., 2003)	Logistic	2 layers, 800 units	1.60
SVM Gaussian kernel	NA	NA	1.40
Dropout NN	Logistic	3 layers, 1024 units	1.35
Dropout NN	ReLU	3 layers, 1024 units	1.25
Dropout NN + max-norm constraint	ReLU	3 layers, 1024 units	1.06
Dropout NN + max-norm constraint	ReLU	3 layers, 2048 units	1.04
Dropout NN + max-norm constraint	ReLU	2 layers, 4096 units	1.01
Dropout NN + max-norm constraint	ReLU	2 layers, 8192 units	0.95
Dropout NN + max-norm constraint (Goodfellow et al., 2013)	Maxout	2 layers, (5 × 240) units	0.94
DBN + finetuning (Hinton and Salakhutdinov, 2006)	Logistic	500-500-2000	1.18
DBM + finetuning (Salakhutdinov and Hinton, 2009)	Logistic	500-500-2000	0.96
DBN + dropout finetuning	Logistic	500-500-2000	0.92
DBM + dropout finetuning	Logistic	500-500-2000	0.79

Figura 4.22: Ejemplo de diferentes simulaciones con y sin capas *dropout*

Obtenemos la siguiente gráfica que nos ilustra bastante bien las diferentes tasas de error en función de la arquitectura

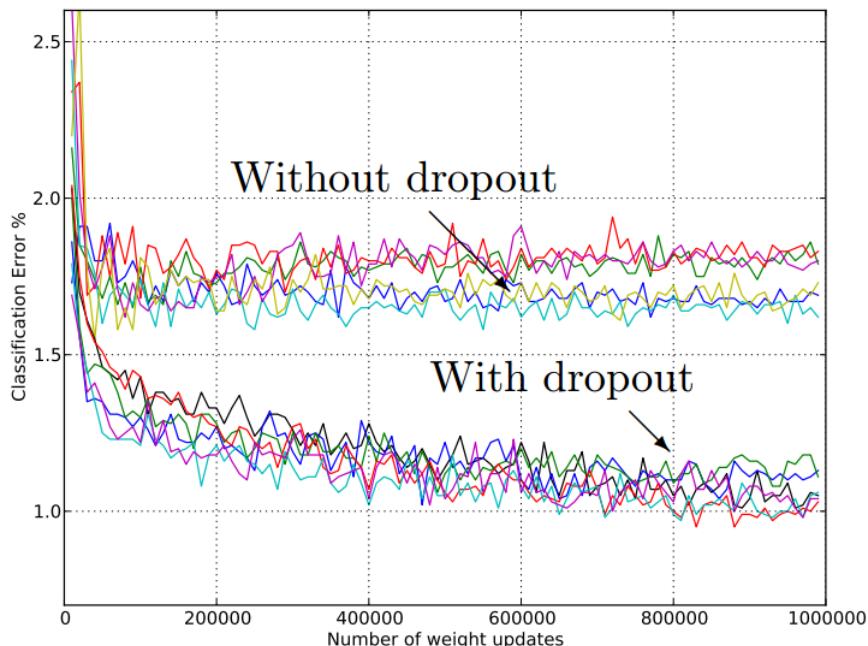


Figura 4.23: Ejemplo diferentes arquitecturas para testear con y sin capas *dropout*

4.2.6. Capas Cross Channel Normalization o normalización de canales cruzados

La capa *Cross Channel Normalization* normalmente se utiliza después de una capa de activación *ReLU*, debido a que las neuronas de estas tienen activaciones ilimitadas y necesitamos *LRN* (*Local Response Normalization*) para normalizar eso. Esta capa atenuará las respuestas que son uniformemente grandes en cualquier vecindario local dado. Si todos los valores son grandes, entonces la normalización de esos valores disminuirá todos ellos. Así que, básicamente, queremos fomentar algún tipo de inhibición y estimular las neuronas con activaciones relativamente mayores. Esto se ha discutido muy bien en la Sección 3.3 del documento original de *Alex Krizhevsky* [8]. Es decir, esta capa reemplaza cada elemento con un valor normalizado que obtiene utilizando los elementos de un cierto número de canales vecinos (elementos en la ventana de normalización). Es decir, para cada elemento x en la entrada, se calcula un valor normalizado x' usando:

$$x' = \frac{x}{K + (\frac{a*ss}{TamVentanaCanal})^{\beta'}}$$

Donde K , α , y β son los hiperparámetros en la normalización, y ss es la suma de los cuadrados de los elementos en la ventana de normalización [8].

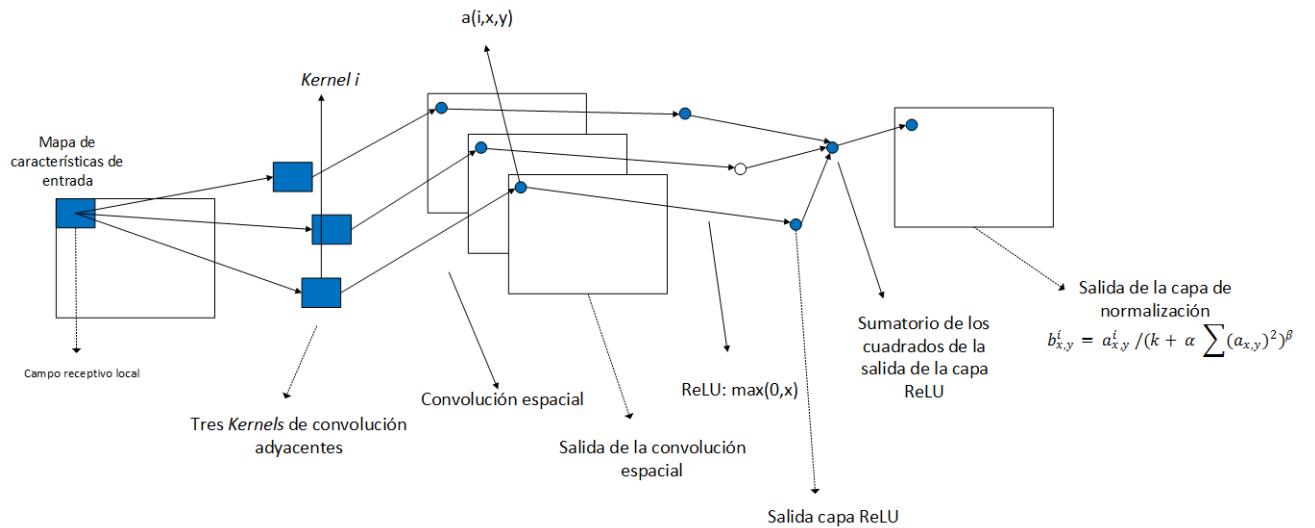


Figura 4.24: Ejemplo red neuronal convolucional con capa de normalización

4.2.7. Capas Softmax y de clasificación

Para problemas de clasificación, una capa *softmax* y luego una capa de clasificación deben seguir la última capa totalmente conectada. La función de activación de la unidad de salida para esta capa es la función *softmax*:

$$y_r(x) = \frac{\exp(a_r(x))}{\sum_{j=1}^k \exp(a_j(x))}, \quad \text{Donde } 0 \leq y_r \leq 1$$

Para problemas de clasificación de varias clases, debemos ajustar esta función mediante:

$$P(c_r|x, \theta) = \frac{P(x, \theta|c_r)P(c_r)}{\sum_{j=1}^k P(x, \theta|c_j)P(c_j)} = \frac{\exp(a_r(x, \theta))}{\sum_{j=1}^k \exp(a_j(x, \theta))}$$

La función *softmax* también se conoce como la exponencial normalizada, y puede considerarse como la generalización multi-clase de la función sigmoide logística. [2]

En la salida de la capa de clasificación, la función *trainNetwork* de MATLAB, que es la empleada para el entrenamiento de la red (En el capítulo 5 se explicará con detalles), toma los valores de la función *softmax* y asigna cada entrada a una de las k clases mutuamente exclusivas usando la función de entropía cruzada para un esquema de codificación *1-of-k* [2]:

$$E(\theta) = - \sum_{i=1}^n \sum_{j=1}^k t_{ij} \ln(y_j(x_i, \theta))$$

Donde t_{ij} es el indicador de que la i -ésima muestra pertenece a la j -ésima clase, θ es el vector de parámetros. $y_j(x_i, \theta)$ es la salida de la muestra i , que en este caso es el valor de la función softmax. Es decir, es la probabilidad de que la red asocie la entrada con la clase j :

$$P(t_{ij} = 1 | x_i)$$

Capítulo 5

Diseño de la solución implementada

Después de explicar las diferentes capas que intervienen en una red neuronal convolucional y sus funcionalidades, pasaremos a detallar los diferentes flujos de trabajo empleados en la solución, así como los procedimientos y clases empleadas. Como se ha comentado previamente en el capítulo 1, tendremos 2 flujos de trabajo, en ambos se harán uso de las redes neuronales convolucionales. El primer paso será descargarnos una red ya entrenada de internet o crear y entrenar una desde cero, por tanto, estos serán nuestros puntos de partida para los diferentes flujos. Como hemos comentado anteriormente, la elección de la arquitectura de red es muy importante a la hora de desarrollar una solución aceptable, por lo que se han investigado varias arquitecturas principalmente en el siguiente enlace <http://www.vlfeat.org/matconvnet/> y se ha escogido la arquitectura *AlexNet*, en el capítulo de conclusiones se explicará detalladamente las razones de esta elección.

1. Para el flujo de trabajo 1 se empleará la red *AlexNet*, que es una red neuronal convolucional ya entrenada que fue propuesta por *Alex Krizhevsky* para el reto *ILSVRC2012 (Imagenet Large Scale Visual Recognition Challenge 2012)*, usaremos esta red para obtener las características de las imágenes en un vector y posteriormente las clasificaremos en función de este vector mediante una máquina de vector soporte o *SVM*.

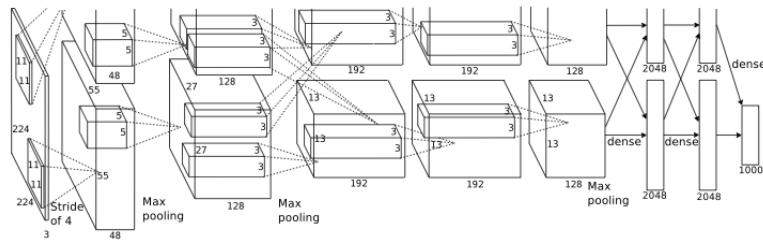


Figura 5.1: Ejemplo arquitectura de la red convolucional *AlexNet*

2. Para el flujo de trabajo 2 se implementará la arquitectura de red neuronal convolucional *AlexNet* desde cero y se entrenará para clasificar imágenes de nuestro problema particular. Se realizará una comparativa de los diferentes flujos de trabajo en cuanto a precisión y velocidad.

5.1. Conjunto de imágenes o Dataset

Debido a que nos encontramos en un problema de aprendizaje supervisado, para ambos flujos de trabajo necesitaremos un conjunto de ejemplos de las diferentes categorías, para ello se han realizado unas funciones de MATLAB con el fin de obtener imágenes, ya sea de la cámara del ordenador o de un video hecho con cualquier dispositivo y ajustarlas al tamaño que necesitamos para nuestra red. Se mostrarán y explicarán los fragmentos de código más relevantes (código y explicación).

```

1      function[rows,cols,dimen1,dimen2] = video(numIm,directIm,vec,
2          train,tamIm)
3      %% OBTENER IMAGENES DE LA CAMARA
4      % Informacion del dispositivo
5      info = imaqhwinfo('winvideo',1);
6      % Camara integrada OS Windows
7      vidobj = videoinput('winvideo',1,'MJPG_1280x720');
8      % Opciones del video
9      src = getselectedsource(vidobj);
10     src.Brightness = -45; % ajuste brillo
11     vidobj.ROIPosition = [427 61 377 419]; % ajuste region interes
12     set(vidobj,'Timeout',500); % timeout para getdata
13     % Numero de Imagenes por trigger
14     vidobj.FramesPerTrigger = numImages;
15     % Se repite 1 vez el trigger manual
16     vidobj.TriggerRepeat = Inf;
17     % Modo de trigger manual
18     triggerconfig(vidobj, 'manual');
19     % Inicio adquisicion de imagenes
20     start(vidobj)
21     % Se abre la ventana de previsualizacion del video
22     preview(vidobj)

```

La función *video* se ha implementado con el fin de obtener imágenes de la cámara del ordenador, esta función recibe como parámetros respectivamente: el número de imágenes de cada categoría, el nombre de la carpeta en la que se encuentran las imágenes de cada categoría, el número de categorías, un 1 si es para crear la carpeta *Train* o 0 para *Test* y por último el tamaño de las imágenes. En este fragmento de código se configura la *ROI* (Región de interés) para no obtener imágenes con una resolución muy alta y que al redimensionarlas se pierda mucho la escala, el modo de funcionamiento (Configuraremos *trigger manual* y repeticiones infinitas, ya pararemos mediante bucles), el número de imágenes por *trigger*, es decir, por cada vez manualmente que escribimos el comando *trigger()* y se inicializa la variable *i*, que será la que controle las veces que repetimos el proceso

```

1      while i<=veces
2          % TRIGGER MANUAL —> se guardan en memoria FramesPerTrigger imagenes
3          trigger(vidobj);
4          cd('..');
5          if(train==1)
6              cd('Train');

```

```

7      else
8          cd('Test');
9      end
10         while get(vidobj,'FramesAvailable')<1
11             % Esperar hasta primera img disponible.....
12         end
13         % Obtenemos las imagenes de la memoria
14         [data time meta] = getdata(vidobj,vidobj.FramesPerTrigger);
15         % dimension datos
16         [rows cols dimen1 dimen2] = size(data);
17         % comprobar carpetas
18
19         if(exist(directorioImagen{1,i}, 'dir')==7)
20             cd(directorioImagen{1,i});
21         else
22             mkdir(directorioImagen{1,i});
23             cd(directorioImagen{1,i});
24         end

```

Para comenzar la adquisición de imágenes ejecutamos la función *trigger()* de MATLAB tantas veces como categorías tengamos, lo controlaremos mediante el bucle *while* y la variable *veces*, una vez empezada esperamos hasta que la primera imagen está disponible y comenzamos a obtener los datos de la memoria mediante la función *getdata()* de MATLAB todas las imágenes de esa categoría. Hacemos las comprobaciones necesarias para la configuración de los diferentes directorios y nos movemos a ellos con la función *cd()* de MATLAB. Las funciones *tic* y *toc* de MATLAB son para medir el tiempo de ejecución.

```

1     tic; % inicio timer
2
3     for j = 1:dimen2
4         % imagen j de la memoria
5         image = data(:,:,:,:,j);
6         % Redimensionamos imagen
7         imagenBuena=imresize(image, [tamImg tamImg]);
8         imwrite(imagenBuena,strcat('image',int2str(j),'.png'));
9     end
10    toc;
11    % Movemos el buffer y vaciamos las imagenes del trigger anterior
12    flushdata(vidobj,'triggers');
13    pause(2); % pausamos 2 segundo
14    i=i+1;
15    cd('..');
16
17
18    % vaciamos buffer TOTALMENTE
19    flushdata(vidobj);
20    % Parar adquisicion de imagenes
21    stop(vidobj)
22    % Timer stop
23    toc;
24    cd('..');

```

Después de comprobar los directorios, empezamos a recorrer con un bucle *for* todas las imágenes del objeto anterior y las redimensionamos con la función *imresize()* de MATLAB con el tamaño adecuado. Una vez tenemos todas las imágenes de esa categoría, ejecutamos la función *flushdata()* de MATLAB que vacía el buffer de datos, en este caso imágenes, para la siguiente categoría, esperamos 2 segundos con la función *pause()* de MATLAB para poder cambiar de gesto con la mano y que no capture imágenes que no contienen el gesto en cuestión y paramos la adquisición de imágenes. Este proceso se repite tantas veces como categorías tengamos.

```

1 %% ELIMINAR DE MEMORIA
2 delete(vidobj);
3 clear vidobj

```

Una vez finalizado el proceso anterior, se ejecuta la función *delete()* de MATLAB y se vacía totalmente el buffer de datos borrando la variable *vidobj*.

El segundo *script* que se usa para obtener el *Dataset* o conjunto de imágenes, extrae estas de un video realizado con cualquier dispositivo con cámara y las guarda en la carpeta indicada.

```

1 function [ ] = videoAFOTOS( video,carpeta)
2 nombre = video;
3 video = VideoReader(nombre);
4 carpetaSalida = fullfile(cd, carpeta);
5
6 if ~exist(carpetaSalida, 'dir')
7 mkdir(carpetaSalida);
8 end
9
10 numImagenes = video.NumberOfFrames;
11 numImgEscritas = 0;
12
13 for t = 1 : numImagenes
14 imgActual = read(video, t);
15 buena=imresize(imgActual,[227 227]);
16 BaseNombre = sprintf(' %3.3d.png', t);    %nombre foto
17 FullNombre = fullfile(carpetaSalida, BaseNombre);
18 imwrite(buena, FullNombre, 'png');    %escribir foto
19 progIndication = sprintf('Imagen %4d de %d.', t, numImagenes);
20 disp(progIndication);
21 numImgEscritas = numImgEscritas + 1;
22 end
23
24 prog=sprintf('Escritas %d imagenes en carpeta "%s"',...
25 numImgEscritas, carpetaSalida);
26 disp(progIndication);
27
28 end

```

Con la función *VideoReader()* de MATLAB creamos un objeto de la clase *VideoReader* para leer archivos de video, una vez creado, comprobamos el directorio de salida, recorremos todas las imágenes del vídeo (desde 1 a *numImagenes*) con el bucle *for*, las redimensionamos con la función *imresize()* de MATLAB y las guardamos en la carpeta indicada con la función *imwrite()* de MATLAB. Mediante la función *sprintf()* de MATLAB y además propia de otros lenguajes de programación le damos título a esa imagen con la opción *%3.3d.png*.

```
1 %% OBTENER IMAGENES DE UN VIDEO
2 videoAFOTOS( 'cero.mp4' , 'cero' );
```

Estos vídeos se pueden realizar con cualquier dispositivo que tenga cámara.

Para la gestión de las imágenes usaremos la clase *ImageDataStore* de MATLAB que nos permite crear nuestro propio conjunto de imágenes. Un objeto de la clase *ImageDatastore* administra una colección de imágenes, donde cada imagen individual se ajusta a la memoria, pero puede que la colección completa no necesariamente se ajuste. Este objeto *mapea* todas las rutas de las imágenes en el atributo *Files* y el objeto categorico de MATLAB que contiene la categoría en el atributo *Labels*. Con la opción ('*LabelSource*', '*foldernames*') de la función *imageDatastore()* de MATLAB creamos un conjunto de imágenes donde cada categoría es el nombre de la carpeta donde se encuentran las imágenes de dichas categorías.

Property	Value
Files	28992x1 cell
Labels	28992x1 categorical
ReadFcn	@(filename)leerYpreprocesarImagenAlexCNN(filename)

Figura 5.2: Ejemplo de los atributos de la clase *ImageDataStore* para el flujo de trabajo 1 (Más adelante se entrará en detalles sobre este flujo), *Files* contiene un *string* con la ruta de la imagen en el disco duro, *Labels* contiene la categoría a la que pertenece en una variable tipo *categorical* y el atributo *ReadFcn* es la función que se ejecuta para la lectura y comprobación de cada imagen

imds.Files	
	1
1	C:\Users\████████\Desktop\DeepLearning\ComputerVision\vid\cero\024.png
2	C:\Users\████████\Desktop\DeepLearning\ComputerVision\vid\cero\025.png
3	C:\Users\████████\Desktop\DeepLearning\ComputerVision\vid\cero\026.png
4	C:\Users\████████\Desktop\DeepLearning\ComputerVision\vid\cero\027.png
5	C:\Users\████████\Desktop\DeepLearning\ComputerVision\vid\cero\028.png
6	C:\Users\████████\Desktop\DeepLearning\ComputerVision\vid\cero\029.png
7	C:\Users\████████\Desktop\DeepLearning\ComputerVision\vid\cero\030.png

imds.Labels				
	1	7849	uno	18486 dos
1	cero	7850	uno	18487 dos
2	cero	7851	uno	18488 dos
3	cero	7852	uno	18489 dos
4	cero	7853	uno	18490 dos
5	cero	7854	uno	18491 dos
6	cero	7855	uno	18492 dos
7	cero	7856	uno	18493 dos
8	cero	7857	uno	18494 dos

(a) Ejemplo atributo *Files*

(b) Ejemplo atributo *Labels*

Figura 5.3: Atributos de la clase *ImageDataStore*

Con el fin de entrenar ambos flujos de trabajo (Red neuronal convolucional y máquina de vector soporte) con una gran variedad de características diferentes, procesaremos las imágenes con el objetivo de mejorar el desempeño final del sistema. Para ello se harán uso de varias técnicas bastante conocidas de procesado digital de imágenes con la siguiente función de MATLAB implementada en el siguiente código:

```

1      function [ ] = Procesado(carpetas,carpeta)
2
3      cd(carpeta);
4      h = msgbox('Procesando imagenes... ');
5      set(h, 'position', [400 150 150 50]);
6      tic
7      cont=0;
8
9      for i=1:size(carpetas,2)
10
11         archivos = dir(carpetas{1,i});
12         nombres = {archivos.name};
13         cd(carpetas{1,i});
14
15         for v=1:size(nombres,2)
16
17             png = strfind(nombres{1,v}, '.png');
18             jpeg = strfind(nombres{1,v}, '.jpg');
19
20             if(size(png,1) == 1 || size(jpeg,1) == 1) % .png o .jpg
21                 % Leemos la imagen
22                 cont=cont+1;
23
24                 % Procesamos 1 de cada 20 imagenes con el
25                 % fin de no introducir mucho sobreajuste en la red.
26                 if(mod(cont,20)==0)
27                     I = imread(nombres{1,v});
28
29                     % Rotacion
30                     IRotada90 = imrotate(I, 90);
31                     imwrite((IRotada90), strcat('Rotada90', nombres{1,v}));
32                     IRotada180 = imrotate(IRotada90, 90);
33                     imwrite((IRotada180), strcat('Rotada180', nombres{1,v}));
34                     IRotada270 = imrotate(IRotada180, 90);

```

```

35     imwrite((IRotada270),strcat('Rotada270',nombres{1,v}));  

36  

37     %Filtrado realce contraste  

38     H=fspecial('unsharp');  

39     IRealzada=imfilter(I,H);  

40     imwrite((IRealzada),strcat('Realzada',nombres{1,v}));  

41  

42     % Filtrado guiado de MATLAB  

43     range = [min(I(:)) max(I(:))];  

44     nhoodSize = 3;  

45     smoothValue = 0.01*diff(range).^2;  

46     B = imguidedfilter((I), 'NeighborhoodSize',nhoodSize,...  

47     'DegreeOfSmoothing',smoothValue);  

48     imwrite((B),strcat('GuidedFilter',nombres{1,v}));  

49  

50     end  

51     end  

52     end  

53     cd('..');  

54     end  

55     toc  

56     cd('..');  

57     delete(h);  

58     end

```

Las técnicas empleadas son:

- Rotado de imágenes con la función *imrotate()* de MATLAB. Se rotan las 90 grados y se guardan como imágenes nuevas. Se repite 3 veces.



(a) Rotada 90º derecha (b) Rotada 180º derecha (c) Rotada 270º derecha

Figura 5.4: Ejemplo de imágenes de la clase cero rotadas en los 3 sentidos

- Enfoque automático de la imagen con la función *imfilter()* de MATLAB usando la mascara *unsharp*.

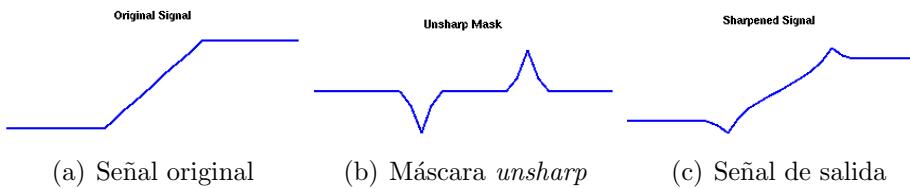


Figura 5.5: Ejemplo proceso de filtrado usando la mascara *unsharp*

- Suavizado de imágenes usando un filtro guía mediante la función *imguidedfilter()* de MATLAB, esta función tiende a suavizar los bordes de los objetos de la imagen que estén pixelados.

Para no introducir demasiado sobreajuste en la red (Esto significa que la red se ajusta demasiado a las imágenes de entrenamiento y produce malas predicciones, lo vemos en la figura 5.6), procesaremos 1 imagen cada 20 imágenes de entre el conjunto total de imágenes de entrenamiento. Esta operación la realizaremos con una variable contador *cont* y un *if* que compruebe que el resto de la división entre la imagen actual (Variable *v* que controla el segundo *for*) y la variable *cont*.

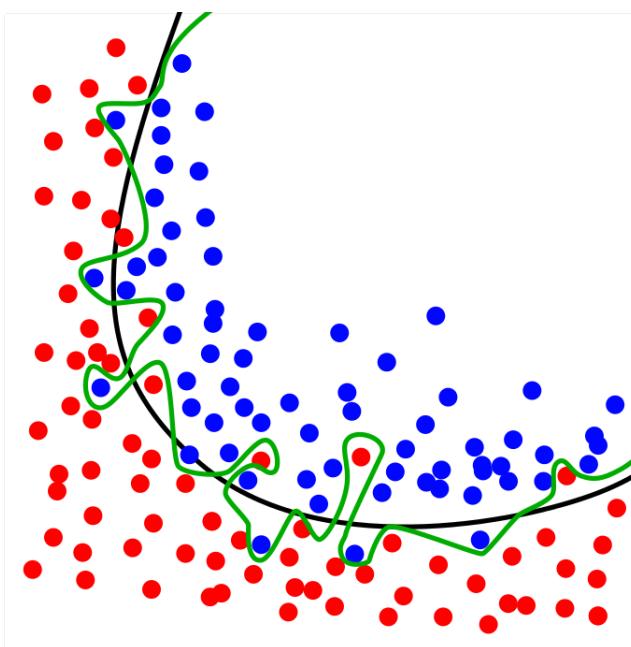


Figura 5.6: Usar la línea verde como clasificador, se adaptaría mejor a los datos con los que hemos entrenado el clasificador, pero está demasiado adaptado a ellos, de forma que ante nuevos datos probablemente nos otorgará más errores que la clasificación usando la línea negra.

5.2. Ejemplo de imágenes del Dataset o conjunto de imágenes

En las siguientes figuras nos encontramos con 4 ejemplos de cada categoría. Estas imágenes se han realizado con diferentes dispositivos: cámara integrada *MSI-GL626QD* y cámara Smartphone *Huawei Y-625*.



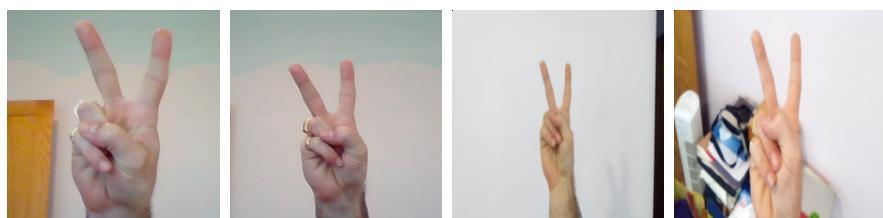
(a) Ejemplo 1 (b) Ejemplo 2 (c) Ejemplo 3 (d) Ejemplo 4

Figura 5.7: Algunos de los ejemplos de la clase cero



(a) Ejemplo 1 (b) Ejemplo 2 (c) Ejemplo 3 (d) Ejemplo 4

Figura 5.8: Algunos de los ejemplos de la clase cinco



(a) Ejemplo 1 (b) Ejemplo 2 (c) Ejemplo 3 (d) Ejemplo 4

Figura 5.9: Algunos de los ejemplos de la clase dos

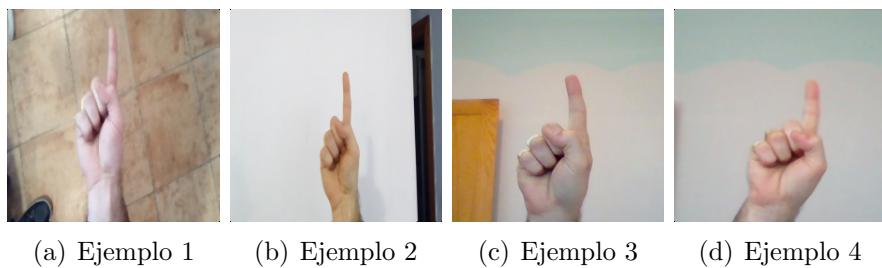


Figura 5.10: Algunos de los ejemplos de la clase uno

5.3. Flujo de trabajo 1 detallado

Como se ha comentado anteriormente, para este flujo de trabajo emplearemos la red neuronal convolucional ya entrenada *AlexNet*, que está entrenada con más de 1.000.000 de imágenes. Para obtener las características del conjunto de imágenes de entrenamiento que usaremos para nuestro problema en particular de capturar varios gestos de la mano emplearemos esta red. Estas características serán un vector de tamaño igual al número de imágenes por 4096 posiciones (que es el número de características que finalmente obtendremos de la imagen a analizar, además corresponde con el tamaño de la última capa totalmente conectada). Una vez tenemos el vector, entrenaremos un clasificador *SVM* o máquina de vector soporte con estas características y posteriormente probaremos su desempeño mediante la cámara del ordenador con imágenes en “directo”.

El primer paso es descargar la red neuronal convolucional AlexNet e importarla al formato necesario.

```

1    %% Descarga de la CNN pre-entrenada (convnet)
2    % Importamos la red en MATLAB con Neural Network Toolbox,
3    % Y mostrarmos la arquitectura de la CNN.
4    % El objeto SeriesNetwork representa la CNN.
5
6    alexCNN = fullfile(pwd,'imagenet-caffe-alex.mat');
7    cnnURL = 'http://www.vlfeat.org/matconvnet/...
8    models/beta16/imagenet-caffe-alex.mat';
9    if ~exist(alexCNN,'file') % descargar solo una vez
10    disp('Descargando archivo "imagenet-caffe-alex.mat" de 234 Mb... ');
11    websave(alexCNN, cnnURL);
12    else
13    disp('Ya esta descargado');
14    end
15
16    % Importamos la red descargada y vemos la arquitectura
17    convnet = helperImportMatConvNet(alexCNN);
```

Mediante la función *websave()* de MATLAB, descargamos el archivo *.mat* de la página web www.vlfeat.org que contendrá las capas de la red neuronal convolucional. Una vez descargada, con la función *helperImportMatConvNet()* de MATLAB importamos la red *AlexNet* con formato *SeriesNetwork* de MATLAB. Mediante esta función creamos un

vector de capas concatenadas y con sus respectivos pesos aprendidos.

```

1      %% Pesos primera capa de convolucion (Patrones simples)
2
3      w1 = convnet.Layers(2).Weights;
4
5      % Escalado y escala de grises
6      w1 = mat2gray(w1);
7      w1 = imresize(w1,5);
8
9      % figura y montage de los filtros
10     figure
11     montage(w1)
12     title('Pesos aprendidos primera capa convolucion')

```

La red neuronal convolucional ya entrenada *AlexNet* cuenta con una gran cantidad de filtros aprendidos en sus neuronas, estos filtros se traducen en patrones de bits que mediante el código anterior visualizaremos en la primera capa de convolución *Layers(2)*. Los patrones de bits aprendidos en esta capa de convolución serán los mas básicos de la red (Líneas rectas, alguna curva simple, mallas, etc), en capas posteriores estos patrones se combinan para obtener otros más complejos y poder aprender características mucho mas enrevesadas. Mediante la función *mat2gray()* de MATLAB, convertimos la matriz de pesos en una imagen en escala de grises, posteriormente la redimensionamos con la función *imresize()* de MATLAB a un tamaño adecuado para la correcta visualización de todos los pesos.



Figura 5.11: Filtros o patrones de bits aprendidos por la red *AlexNet* ya entrenada

```

1      %% Preparar imagenes de entrenamiento en
2      % conjuntos con el mismo numero de elementos
3
4      CarpetaRaiz = 'vid227Proc\';
5      categorias = {'cero','uno','dos','cinco'};
6      imds = imageDatastore(fullfile(CarpetaRaiz, categorias),...
7      'LabelSource', 'foldernames');
8
9      % contamos imagenes

```

```

10    numImagenes = countEachLabel(imds)
11    minimo = min(numImagenes{:,2});
12
13    % usamos el metodo splitEachLabel para recortar el conjunto.
14    imds = splitEachLabel(imds, minimo, 'randomize');
15
16    % Contamos despues de recortar
17    countEachLabel(imds)

```

Como se ha comentado anteriormente en la sección de conjunto de imágenes o *Dataset* emplearemos la clase *Image datastore* de MATLAB para la gestión del conjunto de datos. Definimos el nombre de la carpeta raíz que es donde se encuentran las diferentes carpetas que contienen las imágenes de cada categoría. Con la función *imageDatastore()* de MATLAB creamos el conjunto de imágenes que usaremos para el entrenamiento del clasificador. Mediante la función *fullfile()* de MATLAB se crea el *string* con la ruta del sistema operativo *D:/....vid/cero*, *D:/....vid/uno*, etc. Una vez tenemos el conjunto de imágenes creado, debemos ecualizar el número de ejemplos de cada categoría mediante la función *splitEachLabel()* de MATLAB, obteniendo un nuevo conjunto de imágenes con el mismo número de imágenes en cada categoría.

```

1    %% Leer y preprocessar imagenes
2    % Funcion de lectura de imagedatastore
3
4    imds.ReadFcn = @(filename) leerYpreprocesarImagenAlexCNN(filename);

```

Después de ecualizar el conjunto de imágenes, debemos indicar a la red la función que se ejecutará para la lectura de las imágenes, mediante el símbolo @ indicamos que es una función anónima de MATLAB y que tiene como parámetro *filename*.

```

1    %% Extraer caracteristicas
2
3    tic;
4    trainingFeatures = activations(convnet, trainingSet, 'fc7', ...
5        'MiniBatchSize', 32, 'OutputAs', 'columns');
6    toc;

```

Obtenemos las características de las imágenes de entrenamiento usando la función *activations()* de MATLAB, que nos otorga una matriz de tamaño 4096 (Tamaño salida última capa *fully connected*) x Número de imágenes de entrenamiento. Estas características serán valores numéricos entre [-39.4193 a 31.9667] para nuestro conjunto de datos en concreto.

```

1    %% Entrenar un clasificador SVM multiclas
2    % Etiquetas imagenes entrenamiento
3    trainingLabels = trainingSet.Labels;
4    classifier = fitcecoc(trainingFeatures, trainingLabels, ...
5        'Learners', 'svm', 'Coding', 'onevsall', 'ObservationsIn',...
6        'columns');

```

El siguiente paso es obtener la máquina de vector soporte o *SVM* usando la opción '*Learners*', '*svm*' de la función *fitcecoc()* de MATLAB. Una vez obtenida, la usaremos para clasificar las nuevas imágenes que obtendremos de la cámara del ordenador o del móvil. Mediante la función *fitcecoc()* de MATLAB obtenemos un modelo multiclase entrenado ECOC (*error-correcting output codes model*), en este caso una máquina de vector soporte. La función *fitcecoc()* de MATLAB recibe como parámetro de entrada las etiquetas verdaderas de las imágenes en la matriz *trainingLabels* (Esta es una matriz de tipo categorica de tamaño 1 x Número de imágenes de entrenamiento). Usando la opción ('*ObservationIn*', '*columns*') de la función *fitcecoc()* obtenemos esta matriz de 1 x Número de imágenes de entrenamiento. Después de entrenar el clasificador con nuestro conjunto de datos, probaremos el desempeño del mediante nuevas imágenes de test que obtendremos ejecutando el la función *video* anteriormente comentada.

```

1      %% Evaluar el clasificador
2      % Crear conjunto de validacion o test
3
4      numFotosCategoria=100;
5      carpetas={'cero','uno','dos','cinco'};
6      numCategorias=4;
7      entrenamoentoOtest=0;
8      tamaImg=227;
9      [ rows, cols, dimen1, dimen2 ] =video(numFotosCategoria,carpetas,... 
10     numCategorias,entrenamoentoOtest,tamaImg);
11
12     % Caracteristicas de las imagenes
13     testFeatures = activations(convnet, testSet,... 
14     'fc7', 'MiniBatchSize',32);
15
16     % Hacer una prediccion de las imagenes de test
17     predicciones = predict(classifier, testFeatures);
18
19     % Etiquetas de las imagenes de test
20     testLabels = testSet.Labels;
```

Emplearemos la función *video* para extraer 100 imágenes de cada categoría con el fin de validar el clasificador. Las imágenes que emplearemos tendrán el fondo en blanco y la mano con los diferentes gestos un poco modificados. Otra vez mediante la función *activations()* de MATLAB conseguimos las características de las imágenes y las guardamos en una matriz de tamaño 4096 x Número de imágenes de test. Para ver como se ha desenvuelto el clasificador en cifras, dibujamos la matriz de confusión que es una herramienta que permite la visualización del desempeño de un algoritmo, usaremos el siguiente código para ello:

```

1      %% Dibujar resultados en matriz de confusion
2
3      cm = confusionmat(testLabels, predicciones)
4      figure
5      imagesc(cm);
6      colorbar;
7      title('Matriz de confusion para datos de test')
```

```

8     ylabel('Categorías Reales')
9     xlabel('Categorías Predichas por la red')
10
11    % Convert confusion matrix into percentage form
12    cm = bsxfun(@rdivide,cm,sum(cm,2))

```

Con la función *confusionmat()* de MATLAB obtenemos la anteriormente comentada matriz de confusión, y mediante la función *imagesc()* de MATLAB la convertimos a una matriz de colores para mejor visualización. Una vez tenemos la matriz en una figura individual usando la función *figure* de MATLAB, ponemos el título de los ejes de coordenadas mediante la función *xlabel()* e *ylabel()* de MATLAB y traducimos la precisión total del sistema mediante la función *bsxfun()* de MATLAB. Como vemos, el clasificador se comporta bastante bien para estos ejemplos de validación, usando otro conjunto de imágenes podría variar la tasa de acierto del clasificador. Como proceso de validación adicional, más adelante se comentará el código para imágenes en “directo”.

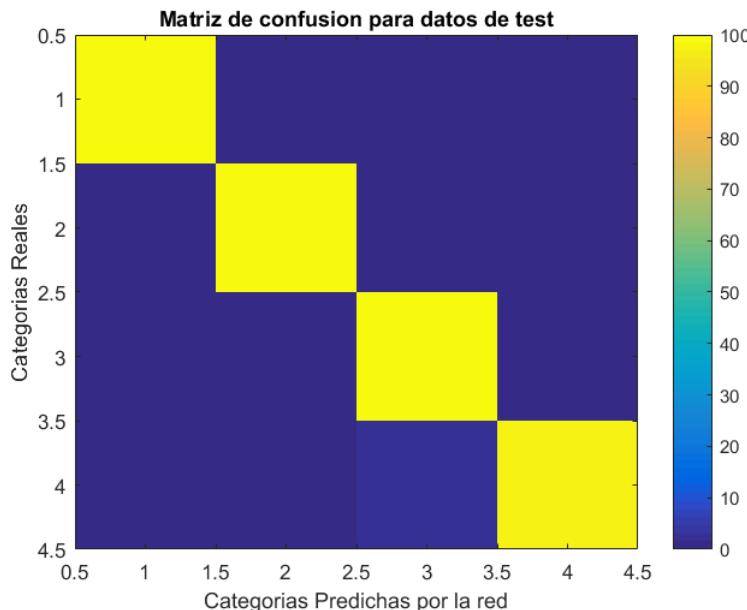


Figura 5.12: Matriz de confusión para 100 imágenes de test. Las categorías corresponden con: 1=cinco, 2=cero, 3=uno, 4=dos

A modo de comprobación final, descargamos un par de imágenes de internet para clasificarlas mediante la máquina de vector soporte o *SVM* para ver su eficiencia:

```

1 %% Probamos en una imagen bajada de internet
2 newImage = 'matest.jpg';
3 newImage2 = 'mctest.jpg';
4 newImage3 = 'm2test.jpg';
5 newImage4 = 'm1test.jpg';
6 img = leerYpreprocesarImagenAlexCNN(newImage);
7 img2 = leerYpreprocesarImagenAlexCNN(newImage2);
8 img3 = leerYpreprocesarImagenAlexCNN(newImage3);
9 img4 = leerYpreprocesarImagenAlexCNN(newImage4);

```

```

10    imageFeatures = activations(convnet, img, featureLayer);
11    imageFeatures2 = activations(convnet, img2, featureLayer);
12    imageFeatures3 = activations(convnet, img3, featureLayer);
13    imageFeatures4 = activations(convnet, img4, featureLayer);
14    [label,score] = predict(classifier, imageFeatures)
15    [label2,score2] = predict(classifier, imageFeatures2)
16    [label3,score3] = predict(classifier, imageFeatures3)
17    [label4,score4] = predict(classifier, imageFeatures4)

```



(a) Ejemplo 1

(b) Ejemplo 2

(c) Ejemplo 3

(d) Ejemplo 4

Figura 5.13: Algunas imágenes descargadas de internet para clasificar con la máquina de vector soporte o *SVM*

<i>Label</i>	<i>Score</i> Clases(0,5,2,1) respectivamente	Imagen en cuestión
cinco	-1.8614 0 -0.9272 -2.2527	<i>matest.jpg</i>
cero	0 -1.2290 -1.6554 -0.9643	<i>mctest.jpg</i>
dos	-1.3898 -0.9243 0 -0.8317	<i>m2test.jpg</i>
uno	-0.6211 -1.1194 -0.9509 0	<i>m1test.jpg</i>

Tabla 5.1: Salida en la consola de MATLAB para el código anterior

El vector *score* nos indica el error en la predicción de la siguiente manera: las posiciones del vector ordenadas ascendenteamente son: cero, cinco, uno, dos. Un valor negativo en este vector indica que la imagen no pertenece a esa categoría, es decir, el valor corresponde con el error, por ejemplo, un error de 0 significa una confianza del 100 %. Como vemos es bastante buena la predicción que nos ha dado para estas imágenes. Con la función *leerYpreprocesarImagenAlexCNN()* implementada, leemos la imagen con el tamaño adecuado para la red.

```

1      function Iout = leerYpreprocesarImagenAlexCNN(filename)
2      I = imread(filename);
3      % Algunas imágenes pueden ser en escala de grises.
4      % Replicar la imagen 3 veces para crear una imagen RGB.
5      if ismatrix(I)
6          I = cat(3,I,I,I);
7      end
8      % Dimension capa entrada CNN
9      Iout = imresize(I, [227 227]);

```

```
10    end
```

Mediante la función *ismatrix()* de MATLAB, comprobamos que sea una imagen en color, es decir, que tenga 3 dimensiones la matriz que representa a la imagen, es decir, que sea de tamaño 227x227x3 en este caso, si no las tiene replicamos la que tiene por 3 canales y conseguimos la imagen en color. Una vez tenemos la imagen en color la redimensionamos con el tamaño necesario para la red *AlexNet*: 227x227x3. Después de probar el clasificador con unas imágenes de internet y ver que el sistema es capaz de predecir la categoría de una nueva imagen ahora usaremos la *webcam* para clasificar imágenes en “directo”:

```

1      %% Clasificar imagenes de la webcam
2      wcam = videoinput('winvideo',1,'MJPEG_1280x720');
3      wcam.ROIPosition = [427 61 377 419];
4      % src = getselectedsource(wcam);
5      % src.Brightness = -45; %ajuste brillo
6      clasi=figure;
7      while ishandle(clasi)
8          picture = getsnapshot(wcam);           % Hacer Foto
9          picture = imresize(picture,[227,227]); % size adecuado
10         imageFeatures = activations(convnet, picture, featureLayer);
11         [label,score] = predict(classifier, imageFeatures);
12         titulo= strcat(char(label), ' Confianza(Clases:0,5,2,1):', '—>', num2str((1.+score));
13         image(picture);        % Mostrar la foto
14         title(titulo); % Mostrar la etiqueta
15         drawnow;
16
17         isopen1=false;
18         isopen4=false;
19
20         switch label
21             case 'cero'
22                 if(score(1)==0 && isopen1==false)
23                     % ACCION 1 : NAVEGADOR PAGINA GOOGLE
24                     [stat,h]=web('www.google.com');
25                     isopen1=true;
26                 else
27                     %close(h);
28                     isopen1=false;
29                 end
30             case 'uno'
31                 if(score(4)==0 )
32                     % ACCION 2 : REPRODUCIR AUDIO
33                     [y,Fs] = audioread('go.mp3');
34                     sound(y,Fs);
35                 end
36             case 'dos'
37                 if(score(3)==0 )
38                     % ACCION 3: DIRECTORIO ACTUAL DE MATLAB
39                     system('cd');
40                 end
41             case 'cinco'
42                 if(score(2)==0 && isopen4==false)
43                     % ACCION 4 : EXPLORADOR DE ARCHIVOS

```

```

44     channel = ddeinit('folders', 'appproperties');
45     ddeexec(channel, '[ViewFolder("%1", d:\, 5)]');
46     isopen4=true;
47     else
48     isopen4=false;
49     end
50   end
51
52
53 end

```

Con la función *activations()* de MATLAB obtenemos las características de la nueva imagen y mediante la función *predict()* de MATLAB obtenemos la predicción del clasificador y un vector con el error para cada categoría. Cuando se captura cada gesto y la probabilidad de error sea 0 (Confianza de 100 %), entonces realizaremos una acción u otra, por ejemplo: abrir navegador para el gesto cero, abrir un archivo de audio para gesto uno, etc. Para que no se abran muchas ventanas del explorador de archivos usaremos la función *ddinit()* y *ddexec()* de MATLAB con la configuración anterior. Se han escogido estas acciones para ejecutar, aunque podrían ser otras cualquiera que usen comandos de MATLAB.

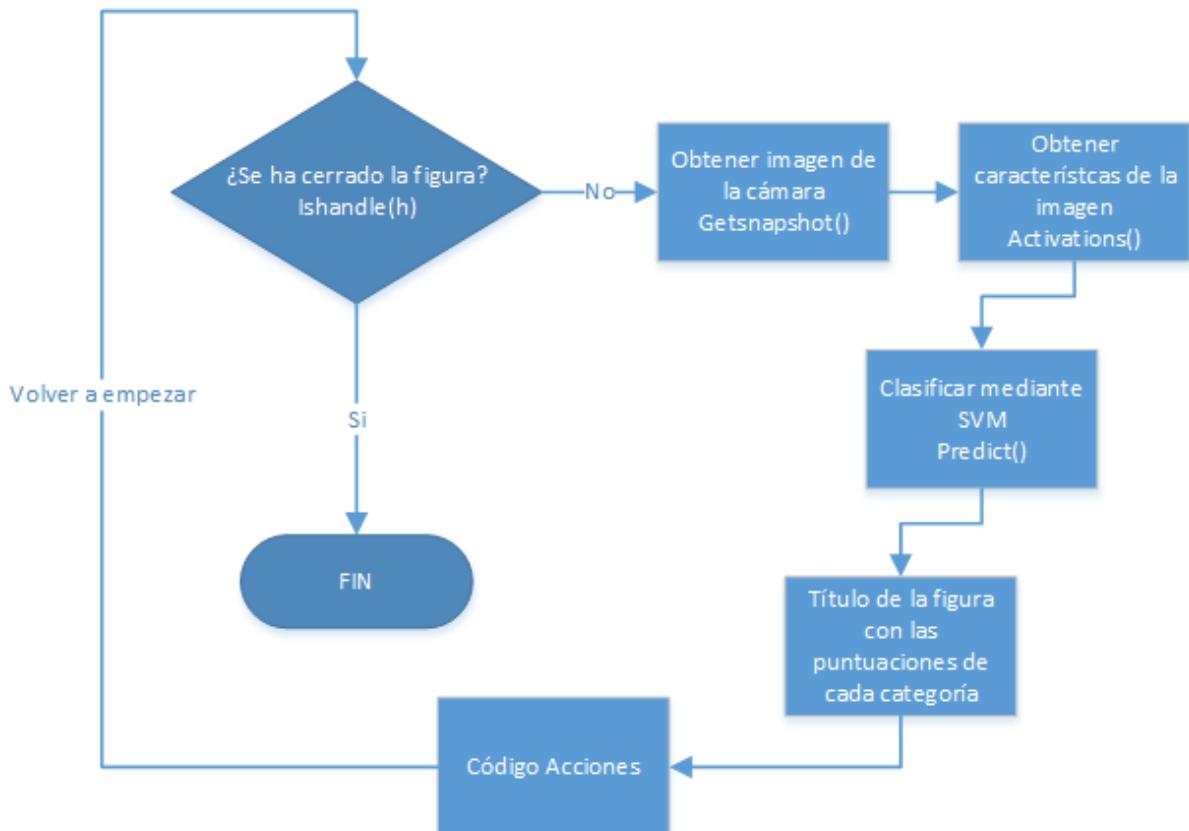


Figura 5.14: Flujograma para clasificación de imágenes mediante el código anterior

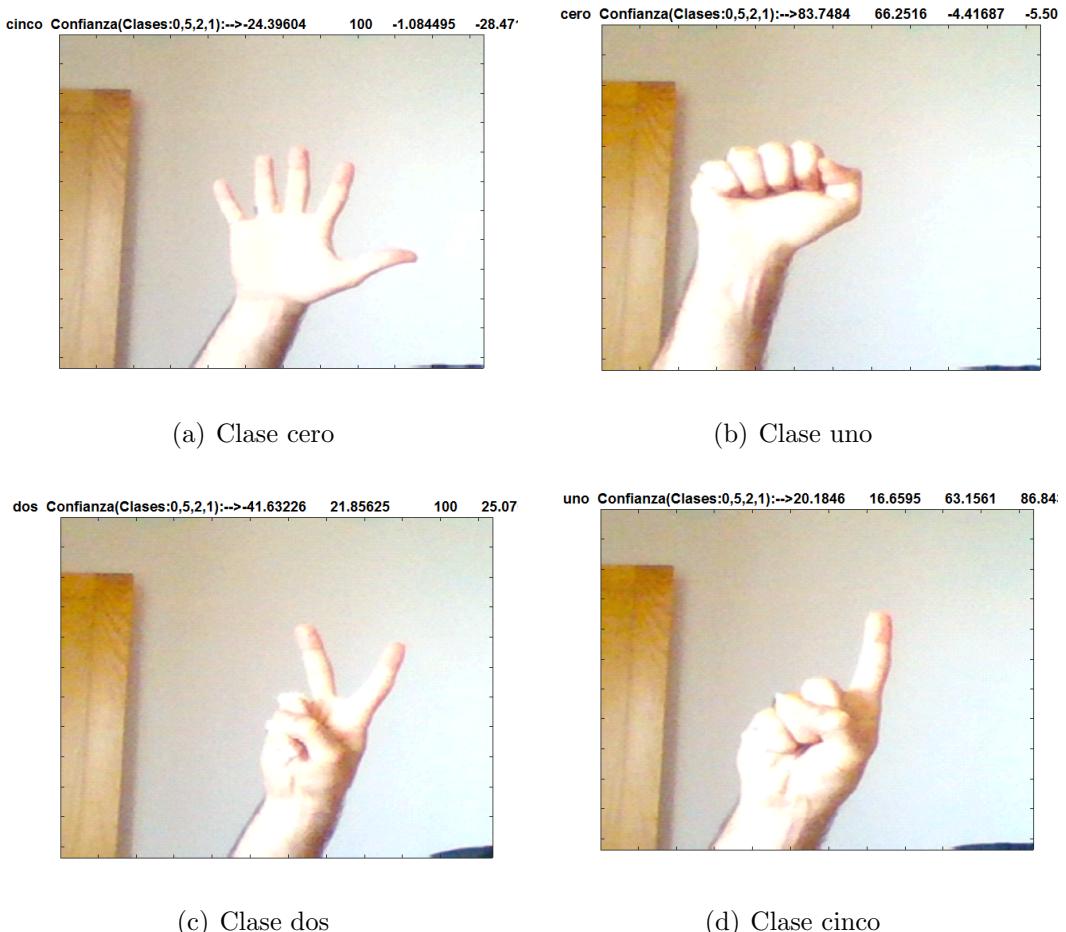


Figura 5.15: Clasificación de imágenes mediante la *webcam* del ordenador

La salida de la función *predict()* de MATLAB es un *string* tipo *categorical* de MATLAB que contiene la categoría predicha y un vector de 1 x Número de categorías que contiene el error de predicción de cada categoría. Para pasar de % de error a % de confianza usamos el fragmento de código anterior:

```

1      % Fragmento de código
2      num2str((1.+score).*100));

```

Sumamos uno a cada posición del vector y luego multiplicamos por 100. Cuando nos encontramos que la probabilidad de una clase es un número negativo, esto quiere decir que la imagen no pertenece a esa clase. También podemos utilizar la cámara del móvil para capturar las imágenes en “directo”, redimensionarlas y clasificarlas mediante la red neuronal convolucional. Para ello descargaremos y usaremos la aplicación IP Webcam de la app Store en el teléfono:



Figura 5.16: Aplicación de la *App Store* para servidor de cámara

Una vez descargada e instalada la *app* en el teléfono, debemos conectarnos a la misma red *lan* que el ordenador con MATLAB y darle a la opción iniciar servidor (retransmisión de video):

```

1      %% CAMARA DEL MOVIL
2      url = 'http://192.168.1.110:8080/shot.jpg';
3      ss = imread(url);
4      h=figure;
5      fh = image(ss);
6
7      while ishandle(h)
8          ss = imread(url);
9          picture = imresize(ss,[227,227]); % Dimension adecuada
10         set(fh,'CData',ss);
11         imageFeatures = activations(convnet, picture, 'fc7');
12         [label,score] = predict(classifier, imageFeatures);
13         titulo= strcat(char(label),' Confianza(Clases:0,5,2,1):',...
14             , '--->', num2str((1.+score).*100)); %
15         title(titulo); % Mostrar la etiqueta
16
17         drawnow;
18     end

```

En la variable *url* tenemos la dirección ip y la configuración para ejecutar una captura de pantalla en el teléfono. En el caso de la red en la que se ejecutó el programa, tenemos la ip 192.168.1.110 del servidor de video, nos conectamos por el puerto 8080 y mediante la opción */shot.jpg* ejecutamos una captura de pantalla. Mediante la función *imread()* de MATLAB creamos la matriz tridimensional que representa a la imagen leyendo la imagen de la variable *url*, que es la captura de pantalla anteriormente comentada y con la función *activations()* de MATLAB obtenemos las características de la imagen. Una vez obtenidas, le damos título a la figura con los porcentajes de confianza para cada categoría mediante la fórmula de la página anterior. Por último, usando la función *set()* de MATLAB y con la opción '*CData*', le damos a la variable *ss* que es la imagen leída el formato de imagen digital en la figura de MATLAB que es la variable *fh*.

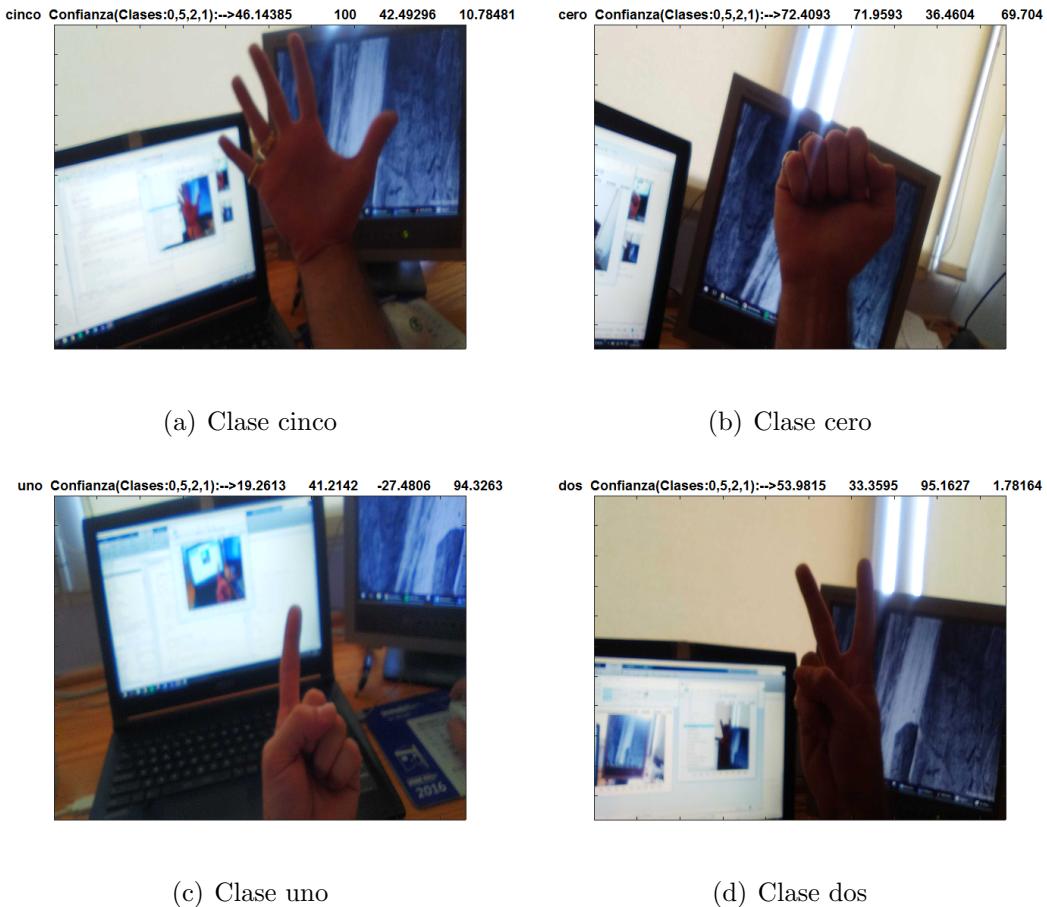


Figura 5.17: Clasificación de imágenes mediante la cámara del móvil

Como podemos ver en la figura 5.17, hemos capturado las imágenes desde la cámara del *smartphone* y las hemos enviado mediante la app de servidor de cámara al ordenador, una vez las hemos recibido, se clasifican mediante la máquina de vector soporte. Para la clasificación de los diferentes gestos, emplearemos cualquiera de las 2 manos, en este caso con la mano derecha los gestos 5,0 y 2 y con la mano izquierda el 1.

5.4. Flujo de trabajo 2 detallado

Para este flujo de trabajo implementaremos la red convolucional *AlexNet* desde cero y la entrenaremos para nuestro problema en particular con el *dataset* antes comentado. La red ya entrenada *AlexNet* contaba con unos filtros aprendidos que serán distintos de los que obtendremos mediante este flujo de trabajo. Para poder ejecutar las funciones necesarias para este flujo de trabajo necesitamos varios Toolbox de MATLAB instalados, además de una capacidad mínima de computo para la tarjeta gráfica del ordenador.

```

1 %% Requerimientos previos CNN
2 clc;clear;
3 modeloGPU = gpuDevice();
4
5 if (modeloGPU.ComputeCapability < 3.0)
6 disp('La capacidad de computo de tu gpu es menor del minimo');
7 exit
8 else
9 disp('Los datos de tu GPU son: ');
10 disp(modeloGPU);
11 disp('La version de Matlab es: ');
12 disp(version);
13 end

```

Mediante la función *disp()* de MATLAB mostramos en la consola los datos de la memoria *GPU* del ordenador. Comprobamos que la capacidad de computo es mayor de 3 y continuamos o cerramos el programa. Una vez comprobados los requerimientos mínimos, configuramos la clase *ImageDatastore* con nuestro *Dataset* con el mismo número de ejemplos de cada categoría.

```

1 %% Preparar imagenes de entrenamiento en conjuntos con el mismo numero
2 CarpetaRaiz = 'vid\';
3 categorias = {'cero','uno','dos','cinco'};
4
5 imds = imageDatastore(fullfile(CarpetaRaiz, categorias),...
6 'LabelSource', 'foldernames');
7
8 % contamos imagenes
9 numImagenes = countEachLabel(imds)
10 minimo = min(numImagenes{:,2});
11
12 % usamos el metodo splitEachLabel para recortar el conjunto.
13 imds = splitEachLabel(imds, minimo, 'randomize');
14
15 % Contamos despues de recortar
16 countEachLabel(imds)

```

Una vez tenemos el conjunto de imágenes creado con la función *imageDatastore()* de MATLAB que se ha comentado anteriormente en la sección de conjunto de imágenes o *Dataset* y las categorías en una *cell* de MATLAB, procedemos con la construcción de la arquitectura de red.

```

1    %% Arquitectura de la red CNN
2    tamImg=227;
3
4    layers = [ ...
5        % data augmentation para conseguir mas imagenes
6        imageInputLayer([tamImg tamImg 3], 'DataAugmentation', 'randcrop');
7        convolution2dLayer(11, 96, 'Padding', 0, 'Stride', 4);
8        reluLayer();
9        crossChannelNormalizationLayer(5);
10       maxPooling2dLayer(3, 'Stride', 2);      % filtros de 3x3 y ...
11       % saltos(stride) de 2 en 2
12       convolution2dLayer(5, 256, 'Padding', 2, 'Stride', 1);
13       reluLayer();          % Rectificador Lineal ---> Genera valores 0 y 1
14       crossChannelNormalizationLayer(5);
15       maxPooling2dLayer(3, 'Stride', 2);
16       convolution2dLayer(3, 384, 'Padding', 1, 'Stride', 1);
17       reluLayer();
18       convolution2dLayer(3, 384, 'Padding', 1, 'Stride', 1);
19       reluLayer();
20       convolution2dLayer(3, 256, 'Padding', 1, 'Stride', 1);
21       reluLayer();
22       maxPooling2dLayer(3, 'Stride', 2);
23       fullyConnectedLayer(4096);
24       reluLayer();
25       dropoutLayer(0.5);
26       fullyConnectedLayer(4096);
27       reluLayer();
28       dropoutLayer(0.5);
29       fullyConnectedLayer(4);
30       softmaxLayer();
31       classificationLayer()];

```

Como se ha visto en la figura 5.1, la red AlexNet está entrenada para clasificar entre 1000 clases, se ha implementado acorde a nuestro problema, es decir, con 4 clases(*fully connected layer* Línea 29) que corresponderán con los 4 gestos a capturar. Con la opción ('*DataAugmentation*', '*randcrop*') en la capa de entrada(*imageInputLayer*) conseguimos algunas imágenes más cortando aleatoriamente la imagen de entrada. A continuación, se explicarán las opciones para el entrenamiento de nuestra red convolucional.

```

1    %% Opciones del solver
2    opts = trainingOptions('sgdm', ...
3        'Momentum', 0.9, ...
4        'InitialLearnRate', 0.001, ...
5        'LearnRateSchedule', 'piecewise', ...
6        'LearnRateDropFactor', 0.1, ...
7        'LearnRateDropPeriod', 1, ...
8        'L2Regularization', 0.004, ...
9        'MaxEpochs', 10, ...
10       'MiniBatchSize', 16, ...
11       'Verbose', true);

```

'sgdm': El algoritmo de descenso de gradiente estocástico actualiza los parámetros (pesos y sesgos) para minimizar la función de error tomando pequeños pasos en la dirección del gradiente negativo de la función de pérdida:

$$\theta_{l+1} = \theta_l - \alpha \nabla E(\theta_l)$$

Donde l representa el número de iteración, $\alpha > 0$ es la tasa de aprendizaje, θ es el vector de parámetros y $E(\theta)$ es la función de pérdida. El gradiente de la función de pérdida, $\nabla E(\theta)$, se evalúa utilizando todo el conjunto de entrenamiento, y el algoritmo de descenso de gradiente estándar utiliza todo el conjunto de datos a la vez. El algoritmo de descenso gradiente estocástico evalúa el gradiente, por lo tanto, actualiza los parámetros, utilizando un subconjunto del conjunto de entrenamiento. Cada evaluación del gradiente que utiliza el *MiniBatch* o mini lote en español es una iteración. En cada iteración, el algoritmo da un paso en busca de la minimización de la función de pérdida. El paso completo del algoritmo de entrenamiento sobre todo el conjunto de entrenamiento usando mini-lotes es un *epoch*.

'Momentum': Debido a que el algoritmo de descenso de gradiente tiene problemas con los barrancos, es decir, áreas donde las curvas de superficie son mucho más pronunciadas en una dimensión que en otra, que son comúnmente alrededor del optima local. Por tanto usaremos el algoritmo de *Momentum* para reducir este hecho. En la figura 5.18 tenemos un ejemplo.

$$\theta_{l+1} = \theta_l - \alpha \nabla E(\theta_l) + \gamma(\theta_l - \theta_{l-1})$$

Donde γ determina la contribución del paso del gradiente anterior a la iteración actual.

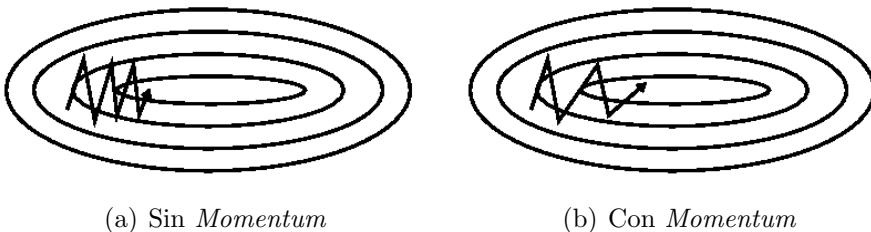


Figura 5.18: Ejemplo de los pasos para la búsqueda de una solución, con y sin *Momentum*

En la siguiente figura tenemos una comparativa del error y la precisión del proceso de entrenamiento al usar el algoritmo de *Momentum* con un valor de 0.9 y al no usarlo.

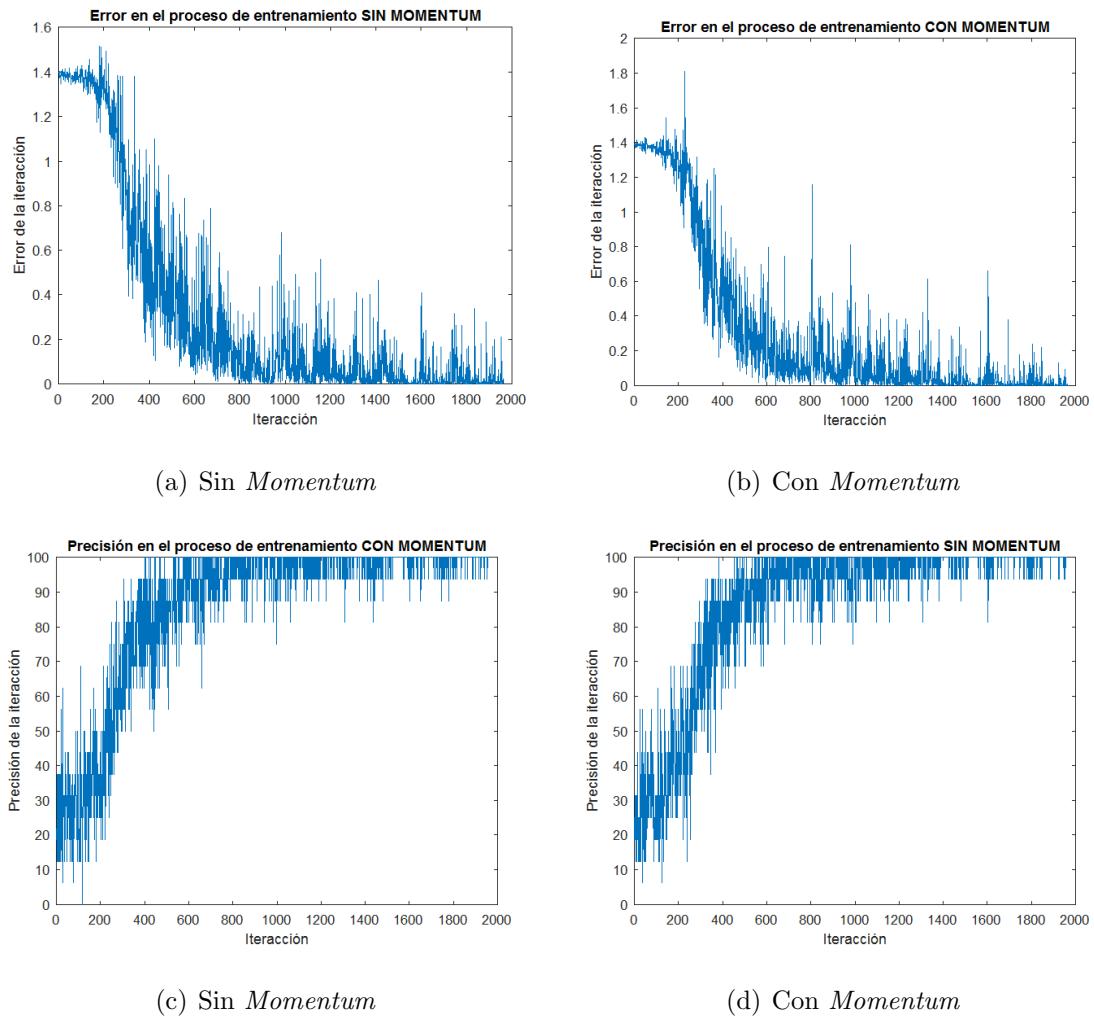


Figura 5.19: Ejemplo error y precisión del entrenamiento, con y sin *Momentum* para 1 paso de entrenamiento (En este caso son mas de 1900 iteraciones)

Se puede apreciar levemente que los cambios sin *Momentum* son más bruscos, mientras que con *Momentum* son mas suaves.

'InitialLearnRate': Es la tasa de aprendizaje inicial de la red.

'LearnRateSchedule': Con la opción *piecewise* se actualiza la tasa de aprendizaje cada cierto número de iteraciones multiplicando esta por un factor.

'LearnRateDropFactor': es el factor que multiplicaremos por la tasa de aprendizaje inicial cada periodo determinado por *learnDropPeriod*.

'LearnDropPeriod': Número de iteraciones para aplicar el factor de descenso de la tasa de aprendizaje.

'L2Regularization': Penaliza la magnitud de todos los parámetros añadiendo $\frac{1}{2}\lambda w^2$ en la función objetivo, donde λ es la intensidad de la regularización. Esta tiene el efecto de preferir pesos distribuidos más difusamente.

Se hará uso de la configuración de estos parámetros relacionados con el aprendizaje, con el fin de suavizar el error en iteraciones posteriores. El número de iteraciones y observaciones por iteración se configuran con *MaxEpochs* y *MiniBatchSize* respectivamente. La adición de un término de regularización de los pesos a la función de pérdida $E(\theta)$ es una forma de reducir el sobreajuste, de ahí la complejidad de una red neuronal. El término de regularización también se denomina pérdida de peso. La función de pérdida con el término de regularización toma la forma:

$$E_r(\theta) = E(\theta) + \lambda\Omega(w)$$

Una vez tenemos las capas de la red convolucional, las imágenes en su respectiva clase, las categorías en su vector correspondiente y las opciones del *solver* definidas, ya podemos entrenar la red con la función *trainNetwork()* de MATLAB.

```
1      %% Entrenar la CNN
2      tic; [net, info] = trainNetwork(XTrain,TTrain, layers, opts); toc;
```

En la variable *info* tenemos una estructura con 3 elementos: El error de *mini-batch* (Este será igual al error en el proceso de entrenamiento), La precisión de *mini-batch* (Esta será igual a la precisión en el proceso de entrenamiento) y la tasa de aprendizaje de la red para cada iteración. Obtenemos la siguiente salida gracias al comando *Verbose* de las opciones de entrenamiento:

===== Epoch Iteration Time Elapsed Mini-batch Mini-batch Base Learning (seconds) Loss Accuracy Rate =====
1 1 50 48.65 1.3964 25.00% 0.001000
1 1 100 94.45 1.3684 25.00% 0.001000
1 1 150 139.85 1.3133 37.50% 0.001000
1 1 200 186.83 0.8075 75.00% 0.001000
1 1 250 232.64 0.7043 81.25% 0.001000

Figura 5.20: Tabla de progreso del entrenamiento, se ha omitido la tabla completa debido al tamaño

Y así continuaría hasta el número de iteraciones configurado anteriormente, se ha omitido la tabla restante debido al tamaño. Una vez terminado el proceso de entrenamiento obtenemos la red entrenada en la variable *net*. Para ver como se ha comportado la red en términos de precisión y error en el proceso de entrenamiento, graficamos la precisión de *mini-batch* y el error de *mini-batch* en la siguiente figura:

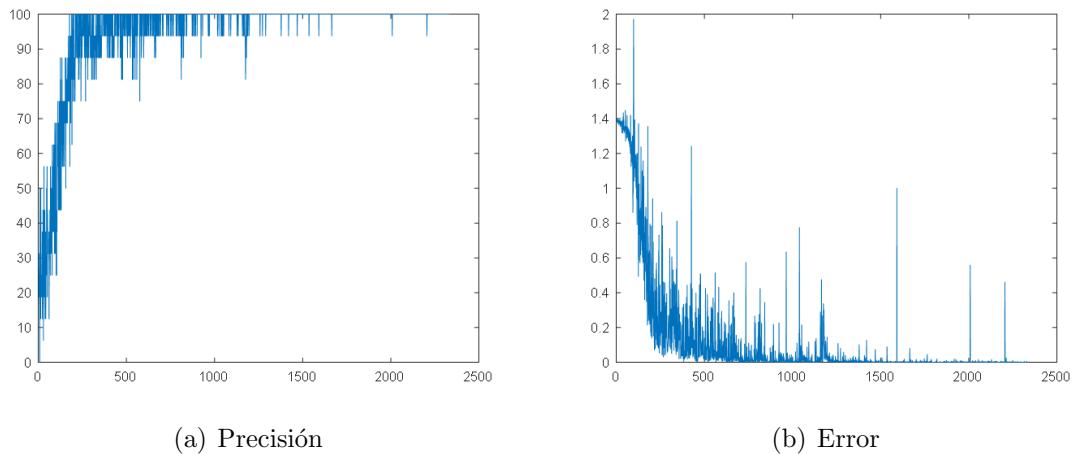


Figura 5.21: Precisión y error en el proceso de entrenamiento completo (10 pasos)

Ahora solo falta usar la red entrenada para clasificar imágenes en “directo” de la cámara del ordenador con el siguiente código:

```

1      %% TEST CON IMAGENES DE LA CAMARA
2      wcam = videoinput('winvideo',1,'MJPG_1280x720');
3      wcam.ROIPosition = [427 61 377 419];
4      h=figure;
5      while ishandle(h)
6          img = getsnapshot(wcam);
7          I = imresize(img,[227 227]);
8          [label, scores] = classify(net, I);
9          titulo= strcat(char(label), ' Confianza ', num2str(scores));
10         image(img);      % Mostrar la foto
11         title(titulo); % Mostrar la etiqueta
12         drawnow;
13     end

```

En este caso emplearemos la función *classify()* de MATLAB para clasificar las imágenes de la cámara del ordenador. Igual que en el flujo 1, la salida de la clasificación es la etiqueta predicha y el vector *score* que es el error de la predicción para cada categoría. Con el atributo *wcam.ROIPosition* configuramos una región de interés para las imágenes con el fin de no alterar mucho la escala al redimensionar imágenes de resolución alta. Para mostrar la imagen actual en la figura como tal, y no como matriz de MATLAB usamos la función *image()*. La función *drawow* se usa para modificar objetos gráficos y ver las actualizaciones en la pantalla inmediatamente.

En la siguiente figura tenemos 4 capturas de pantalla de la clasificación de los diferentes gestos usando el código anterior:

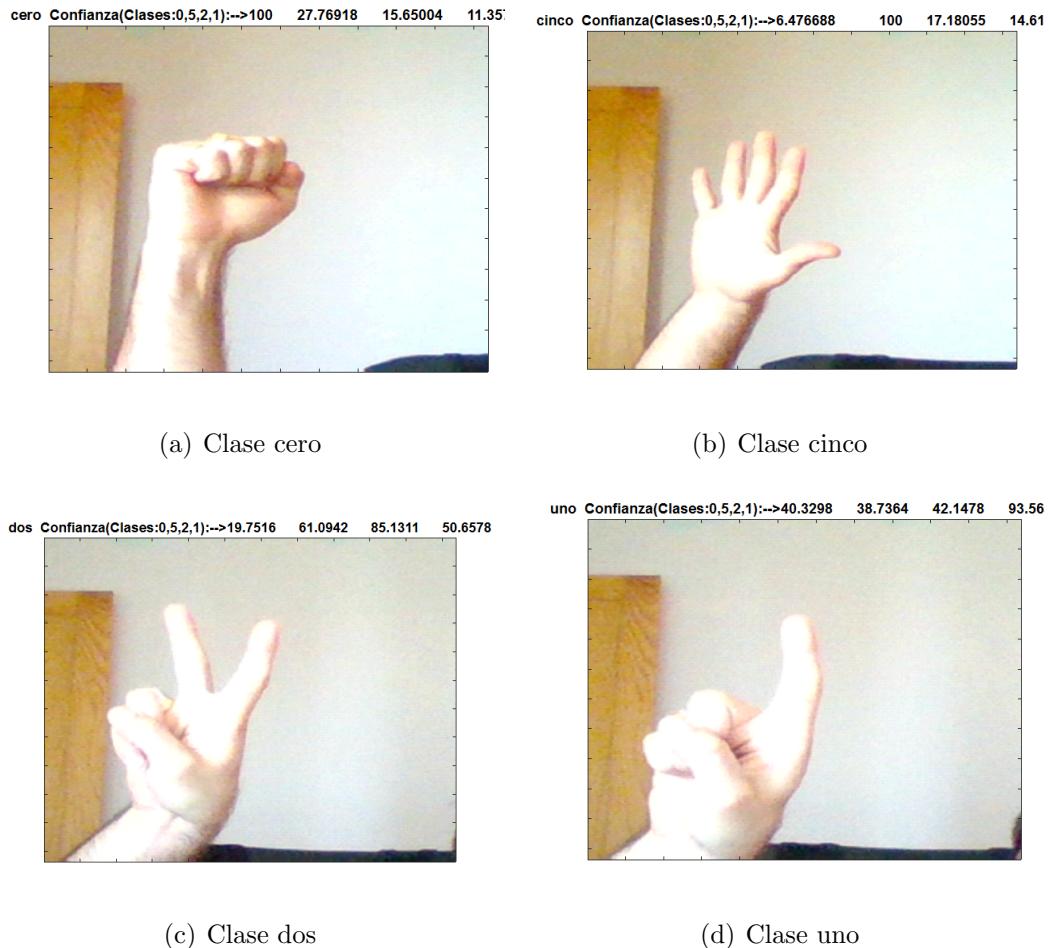


Figura 5.22: Clasificación de imágenes mediante la red *AlexNet* implementada desde cero y la *webcam* del ordenador

Como en el flujo de trabajo 1, se puede utilizar la cámara del móvil para clasificar las imágenes con el mismo código que anteriormente. Para este flujo de trabajo hemos implementado la arquitectura de red *AlexNet* y la hemos aplicado al problema de clasificación de diferentes gestos de la mano, otorgándonos unos resultados de clasificación entre bien y regulares. En la siguiente figura tenemos la arquitectura de la red implementada explicada paso a paso con una breve descripción de cada capa, así como un glosario del numero de capas y del tipo de cada una, además de una representación visual de las primeras capas y como se interconectan entre ellas.

CAPÍTULO 5. DISEÑO DE LA SOLUCIÓN IMPLEMENTADA

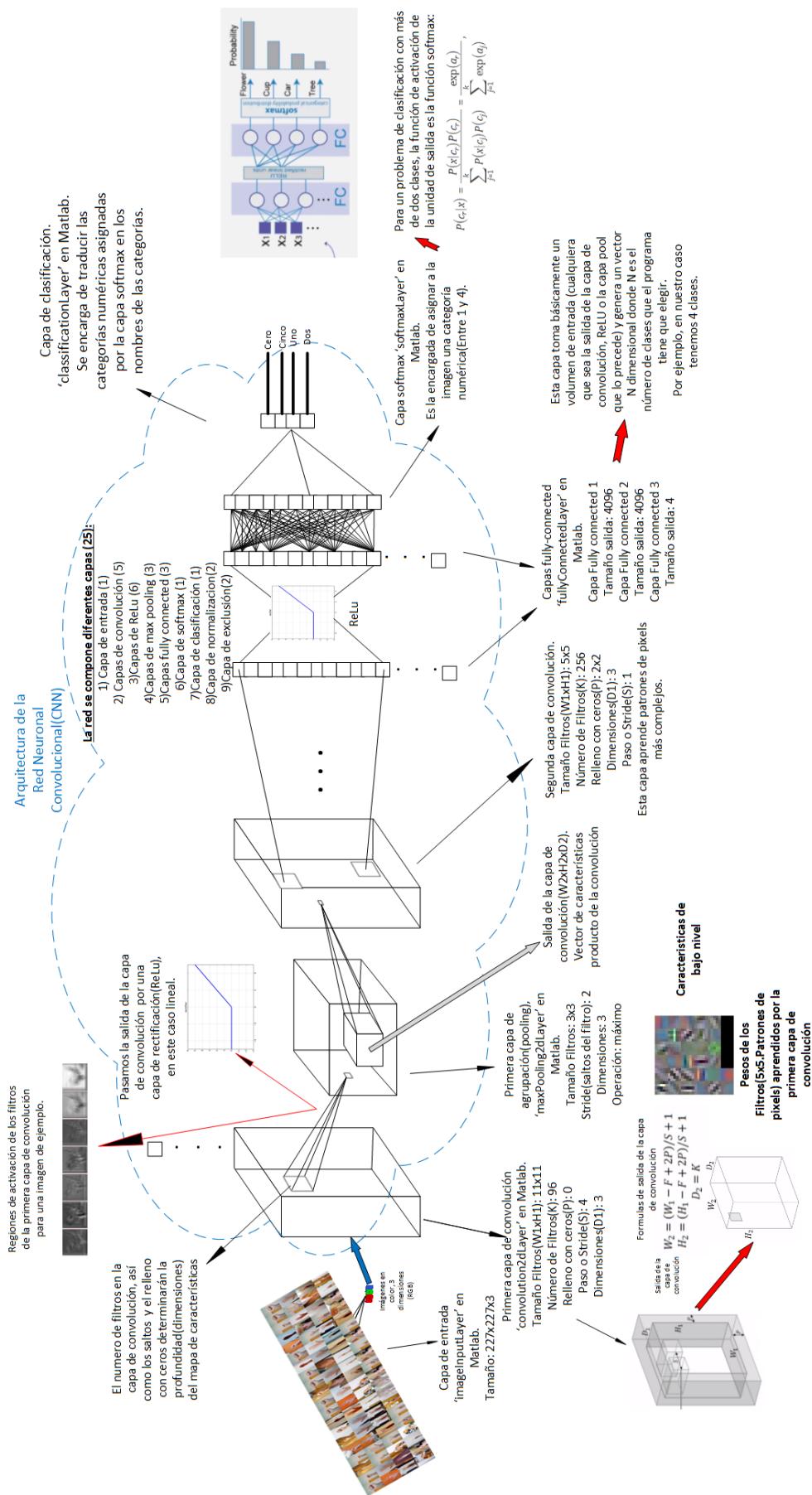


Figura 5.23: Detalles de la arquitectura *AlexNet* implementada

Capítulo 6

Resultados y conclusiones

A lo largo del proyecto se han implementado varios flujos de trabajo, ambos usando las redes neuronales convolucionales para extraer las características de las imágenes y posteriormente clasificar esas imágenes ya sea con la misma red o con un clasificador multiclase *SVM* en función de las características previamente obtenidas. Como punto de inicio se ha implementado la arquitectura de la red *AlexNet* para ambos flujos de trabajo, una descargada de internet y pre entrenada con 23 capas y otra implementada desde cero y con 25 capas (Añadiendo 2 capas de *Dropout*). Se ha visto un ejemplo del efecto de añadir o no estas capas(figura 4.23) a la red.

Para la clasificación de imágenes en “directo”, se han probado a fondo los dos flujos de trabajo y podemos concluir diciendo que el flujo de trabajo 1, se comporta mejor en términos de precisión, es decir, le cuesta menos clasificar gestos de manos diferentes así como gestos “un poco modificados”. Este flujo de trabajo que usa la red ya entrenada *AlexNet*, produce mejores mapas de características que las del flujo 2. Estos mapas, como se ha comentado a lo largo del proyecto, son patrones de bits que las capas de convolución generan. La cantidad de mapas de características almacenados por el flujo 1 (La red ya entrenada *AlexNet*) así como la complejidad de estos, producen que el flujo 1 funcione mejor que el 2, aún habiendo sido entrenado con el mismo conjunto de imágenes. Como conclusión final sobre ambos flujos de trabajo, podemos decir que cuantas más imágenes tenga la red para entrenarse, más características habrá aprendido, lo que se traducirá en mejores y más complejas características obtenidas a la hora de ejecutar la función *activations()* de MATLAB. Pero tenemos que tener claro que estamos ante un problema de aprendizaje supervisado, por lo que el conjunto de imágenes de entrenamiento será clave para el correcto funcionamiento de la red para ejemplos que nunca ha visto. La velocidad con la que se clasifican las imágenes en “directo”, ya sean obtenidas con la cámara del ordenador o con la cámara del teléfono, no es la mejor, ya que no tenemos rachas muy altas de *FPS* (Frames Per Second), teniendo un sistema que funciona bastante bien en un entorno estático, pero es medianamente lento (Tasa de 15-20 FPS) en cuanto a tasa de imágenes por segundo. Esto hace que no debamos mover la mano muy deprisa debido a que la imagen se congela un poco.

Otro aspecto sobre el que podemos concluir, es que hemos visto que el flujo de trabajo 1 es una solución que para un entorno estacionario funciona bastante bien (Rápido reconocimiento del gesto), y por ejemplo podría ser empleada como línea futura para aplicaciones de traducción de lenguaje de signos en “directo”. Obviamente, debemos usar un conjunto de imágenes apropiado para el problema y un correcto procesado de estas, podríamos aplicar esta solución para intentar resolver el problema de traducción de este lenguaje.

El proyecto se ha realizado con un MSI con las siguientes especificaciones técnicas: *GPU GTX-950M* (2GB de *DDR5*) y procesador *i7-6700HQ*. Debido a las especificaciones técnicas del ordenador en el que se ha implementado el código del proyecto, una de las pautas seguidas para la elección de la arquitectura de la red neuronal convolucional empleada (*AlexNet*), ha sido los requerimientos mínimos que esta necesita para ejecutarse, es decir, la capacidad de cómputo mínima que la memoria gráfica del ordenador necesita para poder utilizar principalmente la función *trainNetwork()* de MATLAB, que es la empleada para el entrenamiento de la red y se ejecuta en la *GPU* del ordenador. Se han probado distintas arquitecturas, como por ejemplo: *imagenet-vgg-verydeep-16*, *imagenet-vgg-verydeep-19* y *imagenet-resnet-152-dag*, obteniendo errores de *out of gpu* en la consola de MATLAB, y haciéndonos imposible la ejecución del código. Todas estas redes se han descargado del enlace <http://www.vlfeat.org/matconvnet/pretrained/>. Por tanto, la línea de futuro principal para este proyecto, sería la implementación de las arquitecturas de red anteriormente comentadas para la aplicación a un problema específico de visión por computador, aunque se necesitaría un ordenador mucho más potente que el empleado en este proyecto.

El campo de estudio de las redes neuronales y las redes neuronales convolucionales (Incluidos dentro de la inteligencia artificial), se ha desarrollado bestialmente en los últimos años, ya sea por iniciativa privada de las empresas o con fines competitivos para diferentes concursos y torneos de visión por computador, por ejemplo. Por eso a día de hoy, las empresas del sector tecnológico están invirtiendo en la compra de plataformas o *Startups* que se dedican a la investigación sobre este campo, además de en el desarrollo de soluciones y mejora de los servicios que actualmente ofrecen mediante el uso de este tipo de redes y tecnologías. Actualmente son bastante populares, aunque nadie sabe si en un futuro de 5 a 10 años seguirán siendo empleadas o habrán sido reemplazadas por otro tipo de redes o algoritmos más eficientes.

Bibliografía

- [1] T.M MITCHELL,
- [2] BISHOP, C. M. *Pattern Recognition and Machine Learning*. Springer, New York, NY, 2006.,
- [3] DR. ERIK ZAMORA, *Redes neuronales convolucionales, basado en: Deep Learning Tutorial Stanford University, <http://ufldl.stanford.edu/tutorial/>, Yann LeCun et al. Convolutional Networks and Applications in Vision 2010 y CS231n: Convolutional Neural Networks for Visual Recognition, <http://cs231n.stanford.edu/>,*
- [4] CIRESAN, DAN; UELI MEIER; JONATHAN MASCI; LUCA M. GAMBARDELLA; JURGEN SCHMIDHUBER (2011), *Flexible, High Performance Convolutional Neural Networks for Image Classification. Proceedings of the Twenty-Second international joint conference on Artificial Intelligence- Volume Volume Two 2: 1237–1242,*
- [5] FUKUSHIMA, KUNIHIKO (1980), *Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position.* Biological Cybernetics 36 (4): 193–202,
- [6] LECUN, YANN; LÉON BOTTOU; YOSHUA BENGIO; PATRICK HAFFNER (1998), *Gradient-based learning applied to document recognition”.* Proceedings of the IEEE 86 (11): 2278–2324
- [7] HAHNLOSER, R.H; SARPESHKAR, R; MAHOWALD, M.A; DOUGLAS, R.J. AND SEUNG, H.S (2000), *Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit.* Nature 405, 947-951,
- [8] KRIZHEVSKY, A., I. SUTSKEVER, AND G. E. HINTON (2012), *ImageNet Classification with Deep Convolutional Neural Networks.* Advances in Neural Information Processing Systems. Vol 25,
- [9] NITISH SRIVASTAVA, GEOFFREY HINTON, ALEX KRIZHEVSKY, ILYA SUTSKEVER Y RUSLAN SALAKHUTDINOV.(2013), *Dropout: A Simple Way to Prevent Neural Networks from Overfitting ,*
- [10] NITISH SRIVASTAVA, GEOFFREY HINTON, ALEX KRIZHEVSKY, ILYA SUTSKEVER, RUSLAN SALAKHUTDINOV (2014), *Dropout: A Simple Way to Prevent Neural Networks from Overfitting ,*

Enlaces consultados

<http://ingenierobeta.com/que-es-la-vision-artificial/>

<http://cleverdata.io/conceptos-basicos-machine-learning/>

https://es.wikipedia.org/wiki/Aprendizaje_profundo

<https://es.wikipedia.org/wiki/AprendizajeAutomatico>

https://es.wikipedia.org/wiki/Maquinas_de_vectores_de_soporte

<http://halweb.uc3m.es/esp/Personal/personas/jmmarin/esp/DM/tema3dm.pdf>

https://es.wikipedia.org/wiki/Neurona_de_McCulloch-Pitts

<https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf>

<http://www.synergicpartners.com/una-breve-historia-del-machine-learning/>

<https://es.mathworks.com/help/nnet/ug/layers-of-a-convolutional-neural-network.html#bvk8642>

<https://www.bbvaopenmind.com/que-es-el-aprendizaje-profundo/>