Universidad de Alcalá
Escuela Politécnica Superior

Máster Universitario en Ingeniería Industrial

Trabajo Fin de Máster

Estudio e implementación de un sistema de detección de puntos significativos en objetos estructurados con CNNs

Autor: Antonio Hernández Martínez

Tutor: Ignacio Parra Alonso

2019

Universidad de Alcalá

Escuela Politécnica Superior

MÁSTER UNIVERSITARIO EN INGENIERÍA INDUSTRIAL

Trabajo Fin de Máster

Estudio e implementación de un sistema de detección de puntos significativos en objetos estructurados con CNNs

Autor: Antonio Hernández Martínez

Tutor: Ignacio Parra Alonso

TRIBUNAL:

Presidente: D. Iván García Daza

Vocal 1º: D. Roberto López Sastre

Vocal 2º: D. Ignacio Parra Alonso

FECHA: 26/09/2019

"The saddest aspect of life right now is that science gathers knowledge faster than society gathers wisdom."
Isaac Newton

Agradecimientos

Quiero dedicar este trabajo a mi madre, una de las personas más fuertes que conozco.

A Bianca, que ha estado a mi lado apoyándome durante los últimos 6 años.

A mis hermanos, sin su ayuda y consejos nos hubiese podido llegar hasta aquí.

A mis compañeros del grupo INVETT, en especial a Héctor, por la ayuda prestada para la realización de este trabajo.

Por último, me gustaría dedicar este trabajo a mi padre.

Índice general

Índice de figuras

Índice de tablas

Lista de acrónimos

ANN	*Artificial Neural Network.*
AP	Average Precision.
CAD	*Computer-aided design.*
CNN	*Convolutional Neural Network.*
CPU	Central Processing Unit.
CVPR	Conference on Computer Vision and Pattern Recognition.
DAG	Directed Acyclic Graph.
DFT	Discrete Fourier Transform.
ELU	Exponential Linear Unit.
FCN	Fully Convolutional Network.
FRRN	Full-Resolution Residual Network.
GAN	*Generative Adversarial Networks.*
GPU	Graphics Processing Unit.
HDD	Hard Drive Disk.
HOG	*Histogram of Oriented Gradients.*
ILSVRC	ImageNet Large Scale Visual Recognition Challenge.
IoU	*Intersection Over Union.*
k-NN	*k-Nearest Neighbors.*
LReLU	Leaky ReLU.
MLP	*MultiLayer Perceptron.*

NLL	Negative Log Likelihood.
OKS	*Object Keypoint Similarity*.
PAF	Part Affinity Field.
PReLU	Parametric ReLU.
ReLU	Rectified Linear Unit.
RGB	Red Green Blue.
RNN	*Recurrent Neural Network*.
RoI	Region of Interest.
RReLU	Randomized ReLU.
SGD	Stochastic Gradient Descent.
SIFT	*Scale Invariant Feature Transform*.
SPP	Spatial Pyramid Pooling.
SSD	Solid State Drive.
SURF	*Speeded Up Robust Features*.
SVM	*Support Vector Machine*.
TAC	tomografía computarizada.
YOLO	*You Only Look Once*.

Resumen

Durante los últimos años los algoritmos de *deep learning* han experimentado una evolución sin precedentes como consecuencia del desarrollo de CPUs y GPUs con cada vez una mayor capacidad computacional. Estos algoritmos, utilizados en CNNs, han propiciado la aparición de arquitecturas de redes neuronales como son AlexNet en 2012 o ResNet en 2015, capaces de resolver problemas de visión artificial de manera rápida y eficiente.

Este trabajo tiene por objetivo el estudio de estas arquitecturas y su implementación en un sistema de visión artificial capaz de detectar y mostrar los puntos representativos de la estructura tridimensional de distintos vehículos.

Palabras clave: Redes Neuronales Convolucionales, Visión artificial, Aprendizaje profundo, Detección de puntos clave

Abstract

During recent years the deep learning algorithms have experienced an unprecedented evolution as a result of the development of CPUs and GPUs with higher Computational capacity. These algorithms, used in CNNs, have propitiated the appearance of neural networks architectures such as AlexNet in 2012 or ResNet in 2015, which are able to resolve artificial vision issues on a quick and efficient manner.

This project aims to investigate these architectures and their implementation on an artificial vision system able to detect and show the representative points of the three-dimensional structure of the different vehicles.

Keywords: Convolutional Neuronal Networks, Computer vision, Deep learning, Keypoints detection.

Capítulo 1

Introducción

Este trabajo se centra en el estudio, prueba e implementación de Redes Neuronales Convolucionales (*Convolutional Neural Networks (CNNs)*),las cuales se emplean de forma eficiente en el procesado de imágenes para aplicaciones de detección, clasificación y segmentación. Las redes neuronales convolucionales entran dentro de las redes neuronales conocidas como multicapa, ya que existen capas de neuronas conectadas entre sí de manera que las salidas de una capa son entradas de la capa posterior. Estas redes están basadas en el artículo de K. Fukushima [1] y su funcionamiento se resume en hacer que las neuronas de una determinada capa solo aprendan de algunas neuronas de la capa anterior en lugar de tomar todas las salidas de dicha capa, consiguiendo así reducir el número de operaciones. Inicialmente las CNNs fueron diseñadas para reconocer patrones visuales, aunque también han sido utilizadas para reconocer, por ejemplo, patrones en oraciones o textos.

En el campo en el que se ha visto un gran avance de las CNNs, como solución para resolver problemas de visión artificial, ha sido en el de los sistemas inteligentes de transporte, donde se han ido implementando mejoras pensadas para reducir entre otras cosas el número de accidentes en carretera, aunque se espera que en un futuro, con la llegada de los vehículos autónomos, y el continuo desarrollo de estas técnicas, las ventajas se hagan mas notables, como pueden ser la reducción de atascos o de los niveles de contaminación al estar la conducción mucho más automatizada.

En este trabajo se va a implementar un sistema de visión artificial basado en CNNs que sea capaz de detectar los puntos más relevantes de la estructura de un vehículo, y de esta manera, al incorporarlo a un sistema inteligente de transporte, este será capaz de detectar distintos tipos de vehículos en la calzada. Este sistema podría ser utilizado en el desarrollo de nuevos vehículos autónomos consiguiendo dotar al sistema de una mayor seguridad, ya que este sería capaz de conocer la pose del resto de vehículos que circulan por la vía.

1.1 Visión artificial

.

La visión artificial o visión por ordenador, es una disciplina científica que tiene por objetivo la adquisición, procesado y análisis de imágenes para extraer información relevante de estas, de una manera similar a como lo hace la vista humana. Los sistemas de visión artificial son utilizados en

multitud de aplicaciones como detección o localización, clasificación o reconocimiento de objetos, reconocimiento de caracteres, seguimiento de trayectorias, etc...

La forma tradicional para resolver problemas de visión artificial, siendo capaces de extraer y analizar la información contenida en las imágenes, se realizaba obteniendo, en un primer paso, características (*features*) diseñadas de manera manual, como son los descriptores basados en *keypoints*, *Scale Invariant Feature Transform (SIFT)* [2], *Histogram of Oriented Gradients (HOG)* [3] o *Speeded Up Robust Features (SURF)* [4]. Tras esto, la salida de estos descriptores se utilizaba como entrada de un algoritmo de reconocimiento de patrones mediante el uso de métodos bayesianos, como podían ser, *k-Nearest Neighbors (k-NN)* ,*Support Vector Machines (SVMs)* o *MultiLayer Perceptron (MLP)*, siendo el principal problema de estos el hecho de que el rendimiento del sistema está altamente influenciado por como de bueno sea el extractor de características [5].

En los últimos años los algoritmos de *deep learning* han evolucionado de forma rápida, permitiendo integrar el proceso de obtención de características y su uso para comprender la imagen. Esto, junto con la aparición de grandes *datasets* de imágenes etiquetadas como *ImageNet* [6] ha hecho posible que se puedan entrenar modelos de CNNs cada vez más complejos y profundos, que han demostrado ser capaces de obtener los mejores resultados actuales.

Este tipo de redes neuronales tienen su origen en el año 1976 con el artículo "Neocognitrón"[1]. Posteriormente, entre los años 80 y 90, se implementaron nuevas mejoras [7] que facilitaban el entrenamiento de la red, como pueden ser el algoritmo de *backpropagation* [8] o las capas de *max pooling*. A partir del año 2006, con el desarrollo de *Graphics Processing Units (GPUs)* y *Central Processing Units (CPUs)* con una capacidad de computo mayor a las que existían hasta el momento, se aceleró de manera significativa el proceso de entrenamiento de las CNNs, consiguiendo primeros puestos en competiciones e incluso consiguiendo mejorar la capacidad humana. En el año 2012 se popularizó el uso de las CNNs de forma definitiva, cuando AlexNet consiguió la victoria en el *benchmark* de la base de datos de *ImageNet*, *ImageNet Large Scale Visual Recognition Challenge (ILSVRC)* [6], donde obtuvo un error mucho menor que el resto de competidores.

1.2 Objetivos del trabajo

El objetivo de este trabajo es el estudio, desarrollo e implementación de un sistema de detección de *keypoints* en la estructura tridimensional de distintos vehículos, siendo capaces de reconstruir dicha estructura a partir de estos *keypoints* y conseguir extraer información relevante, como por ejemplo, obtener la posición y orientación del vehículo en la vía haciendo uso de un sistema de visión artificial. Un ejemplo del objetivo a conseguir por el sistema diseñado se muestra en la figura 1.1, donde se puede ver el etiquetado de estos puntos sobre un coche.

Para conseguir este objetivo, es necesario comprender la base teórica que hay detrás de los sistemas de *deep learning* y su relación con las CNNs, y entender su funcionamiento. Se realizará un estudio de las distintas arquitecturas que existen hasta el momento y de los distintos problemas que resuelven. También se buscará una base de datos que se ajuste al problema a resolver en este trabajo y se expondrán los entornos y lenguajes de desarrollo de redes neuronales, entre los que se escogerá el más conveniente para la resolución del problema planteado.

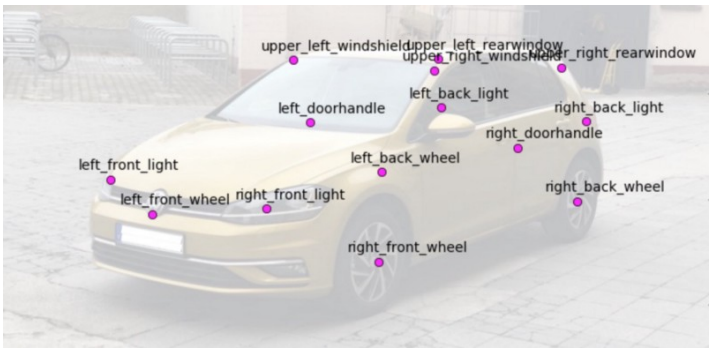


Figura 1.1: Ejemplo de etiquetado de keypoints. Figura obtenida de [9].

Para resolver el problema primero se localiza a los vehículos en una imagen, de manera similar a como lo hace el software de detección y clasificación *You Only Look Once (YOLO)* [10]. Tras esto, los vehículos detectados sirven de entrada a un segundo sistema que es el encargado de localizar y situar los puntos clave del vehículo en el lugar correspondiente de la imagen.

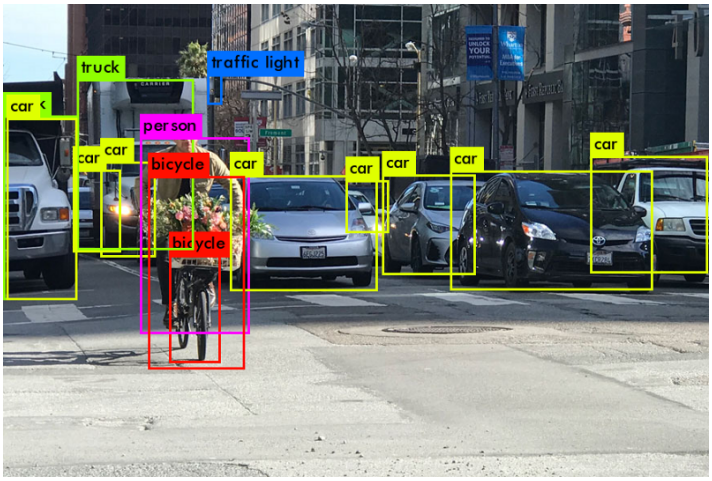


Figura 1.2: Ejemplo de detección de vehículos usando YOLO . Figura obtenida de [11].

Esta memoria esta estructurada de la manera que se detalla a continuación. En el apartado 2 se explica la base teórica y los conceptos necesarios para comprender el funcionamiento que hay detrás de las CNNs, presentando a su vez una serie de arquitecturas ya existentes, que han demostrado un buen funcionamiento en la detección y clasificación de objetos. En el apartado 3 se presenta el problema especifico a tratar, y se exponen los posibles *datasets* a utilizar. En el apartado 4, se describe la solución al problema planteado anteriormente, la arquitectura que se ha escogido para su resolución y las pruebas realizadas. Por ultimo, en los apartados 5 y 6 se muestran y analizan los resultados obtenidos y se exponen las conclusiones y las futuras lineas de trabajo y mejoras a conseguir.

Tras esto se incluyen los anexos, que cuentan con el pliego de condiciones necesarias para la realización de este proyecto (Apéndice A), y un presupuesto de los costes del mismo (Apéndice B).

Capítulo 2

Principios teóricos

En este capítulo se van a explicar los principios teóricos que hay detrás del funcionamiento de las Redes Neuronales Convolucionales así como algunas de las arquitecturas de redes existentes que abordan los problemas de detección, clasificación y segmentación semántica de imágenes. También se presentan una serie de trabajos previos y las mejoras que estos han aportado al estado del arte, con el objetivo de dar una imagen general del estado actual de estos sistemas.

2.1 Redes Neuronales Convolucionales (CNNs)

Las Redes Neuronales Convolucionales (CNNs) son un tipo de red neuronal profunda que surgió como evolución al perceptrón multicapa, estando compuestas por una serie de capas seguidas unas de otras, incluyendo las no linealidades. Este tipo de redes están pensadas para asemejarse a la corteza visual primaria de un cerebro biológico, en la que existen células especializadas en detectar distintas características, por ejemplo, bordes de objetos. Las CNNs son un sistema que recibe a su entrada un tensor[1] de dimensiones $W \times H \times C$, siendo W y H el ancho y alto del tensor y C el número de canales o profundidad.

Estos sistemas aplican una serie de transformaciones sobre dicho tensor, siendo capaces de extraer características más complejas según se avanza a través de las capas de la red neuronal. En la mayoría de arquitecturas planteadas las dimensiones de anchura y altura $W \times H$ se van reduciendo a la vez que el número de canales C aumenta. El hecho de que las características de la red sean más complejas según se avanza en profundidad implica que las primeras capas realizan la función de localizar donde están situados los objetos, ya que las características que detectan son bordes, esquinas, gradientes de colores, etc, mientras que las capas más profundas de la red detectan características más complejas como la rueda de un coche, ojos de personas, etc, siendo estas capas más profundas las encargadas de realizar la función de entender que es lo que hay en la imagen.

Las redes neuronales se han utilizado normalmente para la clasificación, siendo la salida un vector con un valor de probabilidad de pertenencia a cada una de las clases. En concreto, las CNNs se han utilizado para clasificación de imágenes. Como evolución a este tipo de redes surgen otras como las de segmentación semántica o las de detección, que además de especificar que objetos aparecen en

[1] Un tensor es una generalización del concepto de matriz para un número arbitrario de dimensiones

la imagen, también devuelven las coordenadas en las que el objeto se encuentra, de forma habitual mediante una *bounding box* que contiene dicho objeto.

Estas redes neuronales funcionan especialmente bien con imágenes bidimensionales multicanal, siendo las dos dimensiones anchura y altura y, por norma general, tres canales de color (*Red Green Blue (RGB)*), aunque el número de canales puede aumentar o disminuir sin perjudicar el rendimiento de la red. Las CNNs también pueden ser utilizadas en señales 1D o 3D, como puede ser una señal electrónica en la que solo se cuenta con una dimensión o imágenes médicas como una *tomografía computarizada (TAC)* en el que se superponen distintas imágenes en capas, obteniendo una imagen tridimensional del interior del organismo. En estos dos casos igualmente se puede contar con uno o más canales, quedando los filtros con 2 o 4 dimensiones respectivamente.

Las redes neuronales suelen estar compuestas casi siempre de la misma manera: una capa convolucional, seguida por una capa de pooling, tras la cual se aplica una función de activación que incorpora las no linealidades, repitiendo este esquema un número de capas N. La salida de la última capa de pooling serviría como entrada a las últimas capas de la red, que suelen ser capas *fully connected*, donde cada uno de los elementos del tensor de salida esta conectado a todas las neuronas de la capa siguiente. Estos elementos forman el sistema definido como *Directed Acyclic Graph (DAG)*, un grafo dirigido donde los datos (tensores) entran a una capa, son procesados, y salen hacia la capa siguiente, sin pasar dos veces por la misma.

En la figura 2.1 se muestra la arquitectura de la red de clasificación AlexNet [12], la cual surgió en 2012 y sigue siendo usada de ejemplo por su alta precisión a la hora de clasificar imágenes. Esta red parte de una imagen de entrada de 227x227x3 que va avanzando por las distintas capas hasta llegar a un volumen de 13x13x128, que contiene las características de más alto nivel, para ser convertido en un vector que pasará a las capas *fully connected* tras las que se obtendrá la salida de la red. Se puede observar que la red esta dividida en dos, debido a que se utilizaron dos GPUs para dividir el entrenamiento.

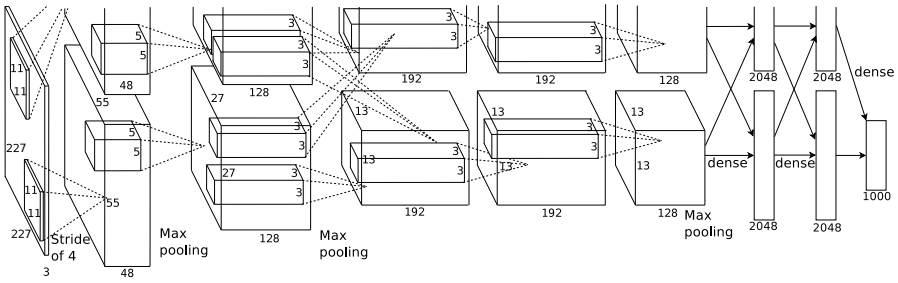


Figura 2.1: Estructura de la red AlexNet. Figura obtenida de [12].

A continuación se va a explicar como funcionan cada uno de los tipos de capas que forman la CNNs y como se realiza el proceso de aprendizaje.

2.1.1 Capas convolucionales

Las capas convolucionales constituyen la base de funcionamiento de las CNNs y es de ellas de las que toman su nombre. En una red neuronal clásica (*Artificial Neural Network (ANN)*), la salida de una capa se obtiene aplicando la función de activación al producto escalar del vector de salidas de la

capa anterior por un vector de pesos. En ocasiones a este producto escalar se le suele sumar un *bias* antes de aplicar la función de activación. Esta operación implica que cada una de las neuronas de la capa actual tiene que estar conectada mediante el vector de pesos a todas las neuronas de la capa previa. En el caso de las redes convolucionales, cada píxel de la capa estaría conectado a un conjunto reducido de píxeles de la capa anterior, mediante los filtros, que serían el equivalente en CNNs a los vectores de pesos de las ANNs. Estos filtros se aplican en ventana deslizante por la superficie del tensor de entrada y en toda su profundidad, siendo de sección normalmente cuadrada, y con la misma profundidad que la salida de la capa anterior. En la figura 2.2 se muestra un caso de aplicación de uno de estos filtros sobre una matriz de entrada en lugar de un tensor.

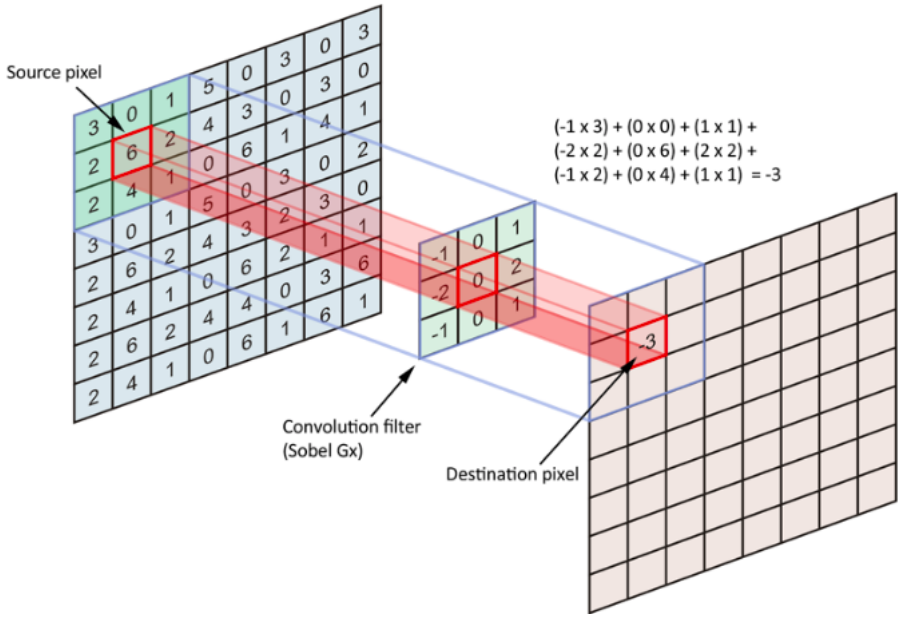


Figura 2.2: Aplicación de filtro convolucional en una CNNs. Figura obtenida de [13].

Al ser la profundidad del filtro igual a la del tensor de entrada, cada filtro genera a su salida un mapa de características bidimensional, es por esto que a cada tensor de entrada se le aplica un número determinado de filtros obteniendo de esta manera un tensor de salida de profundidad igual al número de filtros utilizados.

Los parámetros que se pueden encontrar dentro de una capa convolucional son los siguientes:

- La dimensión de los filtros de convolución. Estos, tal y como se ha dicho, son generalmente cuadrados, de tamaño $F \times F$. Estas dimensiones determinan el tamaño de la ventana deslizante, haciendo depender cada píxel del mapa de características de la salida únicamente de $F \times F$ píxeles de la capa anterior. Esto implica que la conexión de cada píxel con la capa anterior es local en superficie y total en profundidad.

- El número de filtros de convolución, C_{out}. Esto determina la profundidad del tensor de salida generando cada filtro un canal. Cada canal es lo que se denomina mapa de características y estos son independientes, pudiendo uno de los filtros especializarse en detectar bordes horizontales, otro gradientes de color, etc. De este modo, cada canal presentará valores altos en las ubicaciones en las que haya detectado el patrón para el que se ha especializado.

- El *stride*, S, que determina el desplazamiento en píxeles del filtro sobre el tensor de entrada en cada iteración, tanto vertical como horizontalmente.
- El *padding*, P, indica la cantidad de píxeles que se añaden en cada borde del tensor de entrada, es decir, el tensor pasaría de tener unas dimensiones de $W \times H \times C$ a $2P + W \times 2P + H \times C$. Por defecto, el relleno se realiza introduciendo ceros, técnica que se denomina *zero-padding*.

Con todos estos parámetros se puede determinar que, si se tiene un tensor de entrada $W_{in} \times H_{in} \times C_{in}$ procesado por una capa convolucional de C_{out} filtros y tamaño $F \times F \times C_{in}$, *stride* S y *padding* P, el tensor de salida tendrá unas dimensiones $W_{out} \times H_{out} \times C_{out}$, que vienen dadas por las siguientes ecuaciones:

$$\begin{aligned} W_{out} &= ((W_{in} - F + 2P)/S) + 1 \\ H_{out} &= ((H_{in} - F + 2P)/S) + 1 \end{aligned} \tag{2.1}$$

Tomando como ejemplo la figura 2.1 y teniendo en cuenta que AlexNet divide las capas en 2 GPUs, se puede observar que, con un tensor de entrada de dimensiones $227 \times 227 \times 3$, la primera capa tiene 96 filtros (divididos en 2, dando los 48 canales en cada GPUs) ($C_{out} = 96$) de dimensión $11 \times 11 \times 3$ ($F = 11$ y $C_{in} = 3$), *stride* $S = 4$ y *padding* $P = 0$. Esto produce como resultado W_{out} y $H_{out} = ((227 - 11 + 2 \times 0)/4) + 1 = 55$. Las dimensiones del tensor de salida serán por tanto $55 \times 55 \times 96$.

2.1.2 Capas de *pooling*

Las capas de *pooling* tienen por objetivo reducir la altura y anchura de los tensores de entrada, haciendo que el sistema sea más robusto ante cambios de posición de objetos. Además, esto ayuda a reducir los parámetros y la computación necesaria dentro de la red neuronal, lo cual ayuda a su vez a reducir la posibilidad de *overfitting*, uno de los principales problemas de las redes neuronales, y que provoca que la red sea incapaz de dar una salida correcta ante nuevos datos de entrada.

Estas capas funcionan de manera similar a las capas convolucionales, a modo de ventana deslizante de tamaño $F \times F$, con *stride* S, salvo que en este caso, el procesado de cada canal es independiente, dando como resultado un tensor de salida en el que la profundidad, o número de canales, es igual al del tensor de entrada ($C_{in} = C_{out}$), y no se suele hacer uso de padding. La ventana puede aplicar dos tipos de operación, siendo *average pooling* si se toma la media de los píxeles dentro de la ventana, o *max pooling* si se toma el valor más alto.

En el caso de las capas de *pooling*, los parámetros son F y S, y funcionan de la misma manera que en el caso de las capas convolucionales, siendo F el tamaño de ventana y S el desplazamiento de estas ventanas en píxeles sobre el tensor de entrada. Por norma general, se trabaja con $F = S$, para evitar que las ventanas se solapen, siendo S el factor por el que se dividen las dimensiones W_{in} y H_{in}. En la figura 2.3 se puede ver un ejemplo de funcionamiento de una capa de *max pooling*.

Aunque lo habitual es que las CNNs cuenten con capas de *pooling*, existen otras propuestas como la que se ve en [15], en la que se prescinde de este tipo de capas.

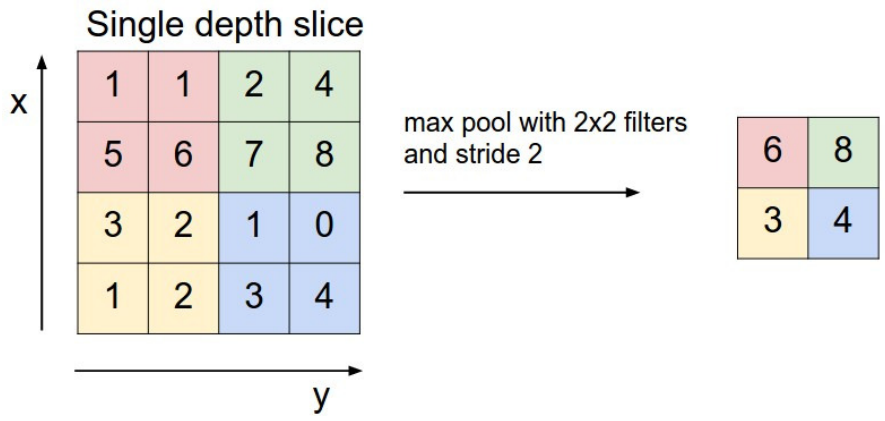


Figura 2.3: Ejemplo de *max pooling*. Figura obtenida de [14].

2.1.3 Función de activación

En las redes neuronales clásicas, la salida de una neurona se puede obtener como se muestra en la ecuación 2.2, donde a la suma de los valores de la capa anterior, se le aplica una función de activación, en este caso una sigmoide, para obtener la salida de la capa actual.

$$\begin{aligned} Z &= \Sigma(W \cdot a_1 + b) \\ a_2 &= \sigma(Z) \end{aligned} \tag{2.2}$$

De forma análoga, esto es lo que ocurre en las CNNs, donde al tensor de salida de la capa convolucional sustituiría a $W \cdot a_1$ en la ecuación anterior. En el caso de estas redes, la función de activación cuyo uso esta más extendido es la función de activación ReLU, definida como el máximo entre 0 y cada elemento del tensor. Estas funciones son aplicadas a la salida de cada una de las capas convolucionales, y en las capas *fully connected*, exceptuando la última capa, cuya función de activación suele ser una *softmax*, función que es utilizada en problemas de clasificación.

2.1.4 Capas de salida

En función del tipo de red que se esté entrenando, se pueden tener distintos tipos de capas a la salida.

En el caso de las redes de clasificación, donde se busca obtener la probabilidad p(c|x) de que la imagen de entrada x, muestre un objeto que pertenece a una determinada clase c, las capas de salida están formadas por la transformación a vector de los tensores más profundos de la red. Cada uno de los elementos de estos vectores formaría una neurona que estaría conectada, como en las redes neuronales clásicas, a todos los elementos de la siguiente capa de la red, siendo estas capas denominadas *fully connected*. En la figura 2.1 se pueden observar estas capas, representadas como vectores de 2048, 2048 y 1000 respectivamente. este ultimo vector con 1000 elementos es el que pasaría por la función *softmax*, devolviendo la probabilidad de cada una de las 1000 clases entre 0 y 1, siendo la suma de todas las probabilidades el 100 %.

Las redes de segmentación semántica, en las que el objetivo es obtener una clasificación o mapa de características a nivel de píxel, utilizan otro tipo de planteamiento. En estas redes las últimas

capas no son *fully connected*, si no que son, al igual que las del resto de la red, capas convolucionales, denominándose así este tipo de redes *fully convolutional*. En este caso también se utiliza una función de salida *softmax* para comprimir las probabilidades de pertenencia entre 0 y 1.

2.1.5 Entrenamiento de la red

El aprendizaje de las redes neuronales se realiza de forma supervisada, minimizando el error entre la predicción hecha por la red y el *ground truth*, o valor real. Dicho error es medido mediante la función de perdidas de la red.

Comúnmente el entrenamiento de las CNNs se realiza mediante el método de descenso por el gradiente estocástico o *Stochastic Gradient Descent (SGD)*, una variante del algoritmo de optimización de descenso por el gradiente, que actualiza los pesos de la red en la dirección de máxima pendiente de la función de perdidas. Este método es muy sencillo de implementar, siendo perfecto para redes neuronales profundas, ya que estas cuentan con un gran número de parámetros lo cual implicaría que un algoritmo más complejo requiriese una carga computacional mucho mayor.

Los pesos de la red son actualizados empleando el algoritmo de *backpropagation* [8], el cual consiste en, a través de la regla de la cadena, ver la implicación individual que tiene cada uno de los pesos de la red en el error cometido a la salida, y actualizarlos en consecuencia. Cada ejecución del entrenamiento está compuesta por dos fases:

- ***Forward pass***, del inicio al final de la red. En esta etapa se calcula la salida de la red, aplicando los productos y funciones que se han explicado con anterioridad desde la entrada hasta conseguir una predicción a la salida.
- ***Backward pass***, del final al inicio de la red. Es la etapa en la que se realiza el aprendizaje. Se calcula la derivada de la función de pérdidas respecto a la predicción hecha por la red (salida) y se propaga hacia las capas iniciales para actualizar los pesos de cada filtro de las capas convolucionales y otros parámetros de la red que deban ser entrenados. Este proceso se realiza como se explica a continuación [16]:
  - Para calcular el gradiente asociado a una capa d, se derivan sus salidas respecto a cada uno de los pesos que participan en su cálculo. Este valor se multiplica por el gradiente propagado desde la capa superior $d+1$ para cumplir la regla de la cadena.
  - Para realizar la propagación del gradiente de la capa d a la capa inferior $d-1$, se derivan las salidas de d respecto a cada salida de $d-1$ que haya participado en el calculo de la capa d. Una vez que se han obtenido las derivadas se multiplican por el gradiente de la capa $d+1$ para cumplir la regla de la cadena. De esta forma cada salida de $d-1$ acumula las aportaciones de todas las salidas de la capa d en las que ha influido y se obtiene el gradiente propagado.

En las CNNs, cuando se actualizan los pesos asociados a un píxel de salida se está actualizando todo el conjunto de pesos de un filtro. Cuando se ha calculado el gradiente para todos los píxeles de un tensor de salida, se actualizan los pesos del filtro mediante el algoritmo del SGD.

Cabe destacar que cada iteración o *epoch* del SGD comprende varias pasadas de entrenamiento, dependiendo del tamaño del *batch* de imágenes empleado, por ejemplo para un conjunto de 1000 imágenes con un batch de 250, se necesitarían 4 pasadas para completar una iteración. De este modo se realiza una estimación del gradiente global a partir del promediado de los gradientes obtenidos para cada *batch* del *dataset* de entrenamiento, en lugar de hacerlo con el conjunto entero de datos, como se hace en el descenso por el gradiente clásico. Una vez obtenida la estimación del gradiente para cada peso de la red, se puede proceder a su actualización empleando la ecuación 2.3 [17].

$$\begin{aligned} \nu_{t+1} &= \gamma\nu_t - \eta_t \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}_t; x(t), y(t)) \\ \mathbf{w}_{t+1} &= \mathbf{w}_t + \nu_{t+1} \end{aligned} \tag{2.3}$$

En la ecuación 2.3 se tiene ν, término de inercia cuyo objetivo es promediar de forma exponencial decreciente los gradientes anteriores, lo cual, añade estabilidad al proceso de aprendizaje. El término del *momentum* $\gamma \in [0, 1]$, tiene por objetivo controlar la velocidad de decaimiento del promedio de los gradientes previos. El término $\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}_t; x(t), y(t))$ es el gradiente de la función de pérdidas, calculada en función de los pesos actuales $\mathbf{w}_t$, y estimada a partir de la salida predicha obtenida a partir de x(t) y la salida real y(t) del *dataset*. Finalmente, el parámetro η_t es el denominado *learning rate* o tasa de aprendizaje, el cual se encarga de regular la velocidad del aprendizaje ponderando el gradiente. Todos estos parámetros se explican en profundidad más adelante, en la sección 2.2.4.3.

2.2 Estado del arte

Como se comentó en el apartado 1.1, las redes neuronales convolucionales han visto incrementada su popularidad debido a tres factores importantes: la aparición de GPUs con una alta capacidad de computo, la existencia de grandes bases de datos etiquetados, y la aparición de AlexNet [12] con unos resultados extraordinarios en la competición de detección y clasificación de objetos de ImageNet [6]. A continuación se exponen algunos de los tipos de variaciones existentes que se han realizado en la estructura de las CNNs.

2.2.1 Variaciones de la capa convolucionales

En la sección 2.1 se ha visto como las capas convolucionales constituyen la base de las CNNs, y la forma de comportarse de estas capas depende de muchos factores. El hecho de poder modificar los filtros de este tipo de capas de manera tan sencilla, facilita que surjan distintos tipos de arquitecturas. Las capas convolucionales permiten un funcionamiento rápido y eficiente de las CNNs, pudiendo mantener la estructura espacial de la imagen [18].

- **Tiled CNN [19].** Como se explicó anteriormente, la red pasa la entrada por una serie de filtros, y de cada filtro se obtiene un canal de salida. En el caso clásico, los pesos de cada filtro son fijos para toda la imagen de entrada, es decir, el filtro es el mismo para cada píxel de salida, y se va superponiendo sobre la entrada a modo de ventana deslizante. En el caso de estas redes,

se propone variar esos pesos cada N píxeles de la imagen de entrada, consiguiendo así que la siguiente capa de pooling pueda elegir entre distintos conjuntos de pesos locales y adquirir información sobre otros tipos de invarianza, como escala o rotación, que en las CNNs normales esta impuesta a toda la superficie al compartir pesos a nivel de canal.

- **Deconvolution [20–22].** En el caso de la convolución, una ventana de píxeles de la entrada se agrupan en un píxel de la salida mediante un producto escalar de estos con los pesos de un filtro. En la deconvolución, se generan píxeles para el tamaño de la ventana de salida partiendo de un único píxel de entrada. En este caso, el *stride* S se suele tomar con un valor fraccionario. El objetivo de la deconvolución es recuperar mapas de características de dimensiones mayores a partir de mapas menores que ya han experimentado varias etapas de *pooling*, para observar así como se comporta y que está aprendiendo la red, o para realizar segmentación por píxel.

- **Dilated convolution [23–25].** El objetivo de este filtro es ampliar el campo de visión sin aumentar el número de parámetros necesarios. Para ello el filtro es aplicado sobre píxeles no adyacentes de la imagen de entrada, rellenando los huecos con ceros, por ejemplo, un filtro 3×3 con orden 2 de dilatación quedaría como un filtro 5×5 como se puede ver en la figura 2.4. Esto es especialmente útil en redes de segmentación semántica donde es necesario tener información del contexto de cada píxel sin reducir la resolución del mapa de características.

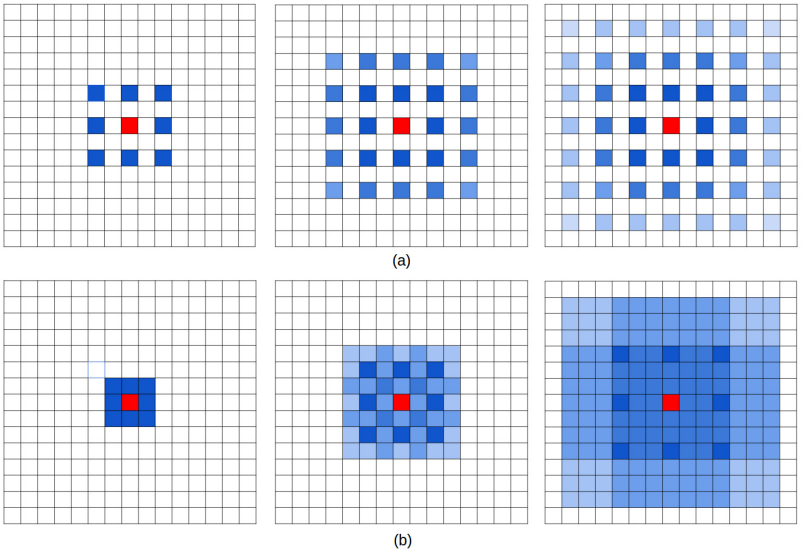
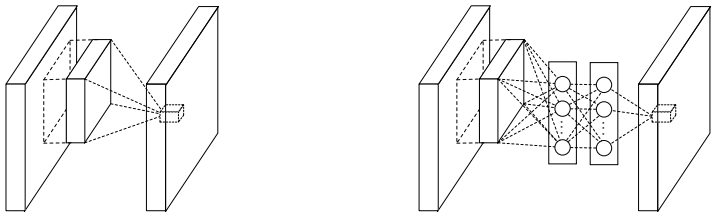


Figura 2.4: Ampliación del campo de visión mediante el uso de *dilated convolution* mediante el uso de filtros 3×3 y orden 2 de dilatación (a) con el consiguiente efecto de rejilla, y mediante el empleo de filtros 3×3 con orden de dilatación 1,2 y 3 respectivamente (b) para evitar ese efecto. Figura obtenida de [25].

El principal problema de estas capas es que su procesado en cascada con un orden de dilatación constante provoca que solo se seleccionen píxeles separados una distancia fija, lo que da lugar a la aparición de un patrón de rejilla. En [25] se propone una mejora que, modificando el orden de dilatación para que sea variable, y mediante secuencias en capas consecutivas, logra deshacerse del efecto de rejilla y amplía el campo de visión a todos los píxeles de la región sobre la que se aplica.

- **Network-In-Network [26].** La arquitectura *Network-In-Network* propone concatenar a la salida de una capa convolucional básica, un perceptrón multicapa en forma de varias capas *fully-connected*, logrando así una representación más compleja del campo de visión. En la figura 2.5 se puede ver un ejemplo de este tipo de redes, en comparación con las CNNs habituales.



(a) Linear convolution layer (b) Mlpconv layer

Figura 2.5: Capa convolucional clásica (izquierda) y capa convolucional *Network-In-Network* (derecha). Figura obtenida de [26].

2.2.2 Variaciones de la capa de *pooling*

Las capas de *pooling* son utilizadas principalmente para reducir la dimensión de los tensores de entrada. Estas capas son utilizadas debido a que, al desplazarse cada filtro a modo de ventana deslizante, la misma característica podría ser detectada en lugares próximos de la imagen, de modo que la capa de *pooling* serviría para reducir la dimensión, preservando únicamente la información útil. Las variaciones de estas capas afectan sobre todo a la manera en que se combina estos resultados adyacentes [5].

- ***Avarage pooling***. El píxel de salida toma el valor promedio de los píxeles de entrada dentro de la ventana. Esto puede suponer un problema si la información relevante se encuentra en los valores más altos, ya que se podría perder la detección de una característica.
- ***Max pooling***. El píxel de salida toma el valor del mayor de los píxeles de entrada que estén dentro de la ventana, preservando la detección más intensa.
- **Espaciado y solapado**. Como se explicó en la sección 2.1.2, estos son los parámetros usados en las capas de *pooling*. Dependiendo del *stride* S y las dimensiones de la ventana $F \times F$ elegidas, puede llegar a producirse solapamiento. Lo más común es usar un $F = 2$ y $S = 2$ para reducir las dimensiones del tensor a la mitad, o $F = 3$ y $S = 2$ (*pooling* con solapamiento). No se recomienda emplear capas de *pooling* con un campo de visión mayor, ya que esto provocaría una perdida de información. demasiado grande.
- ***Stochastic pooling* [27]**. Este método asigna una probabilidad a cada píxel dentro de la ventana en base a su nivel y elige el valor de salida con una distribución multinomial. De este modo no se preserva únicamente el máximo valor. Además, la introducción de la elección estocástica del valor de salida equivale a realizar modificaciones aleatorias en la imagen de entrada, lo que hace que funcione como un método de regularización, ayudando a evitar *overfitting*. Una vez entrenado el sistema no se debe mantener el comportamiento estocástico, ya que se podría introducir ruido en el sistema. En su lugar se aplica una variación del *average pooling* en el que los valores quedan ponderada por las probabilidades asignadas a cada píxel.

- ***Mixed pooling*** **[28]**. Se escoge de manera aleatoria entre la función *max pooling* y *avarage pooling*. Este método, al igual que el anterior, realiza una función de regularización y de la misma manera, cuando la red esta entrenada y para evitar introducir ruido, se fija cada capa de *pooling* a la función que más veces fue escogida durante el entrenamiento, es decir, o se fijan todas las capas como *max pooling* o como *avarage pooling*.

- ***Spectral pooling*** **[29]**. El *spectral pooling* transforma el mapa de características completo al dominio de la frecuencia empleando *Discrete Fourier Transform (DFT)* y toma únicamente los coeficientes centrales con una ventana del tamaño de salida deseado. Tras esto se emplea una DFT inversa para volver al dominio espacial. Esto equivale a aplicar un filtro paso bajo. Este proceso preserva una cantidad de información mayor que otros métodos más sencillos en el dominio espacial.

- **SPP [30]**. El SPP es una variación de la forma de aplicar el *pooling*. En lugar de emplear una ventana deslizante, el mapa de características de entrada es dividido en un número fijo de *bins* proporcionales al tamaño del mapa. Esta división se realiza en varios niveles empleando una cantidad de *bins* diferente para cada nivel quedando el último nivel como el mapa completo. Tras esto se aplica una operación de submuestreo clásica, como puede ser el *max pooling*, dentro de cada *bin*, y tras esto se almacenan los valores en un vector. Este método esta pensado para ser usado como última capa de *pooling* de la red, generando un mapa de características final de tamaño fijo para ser utilizado por las capas finales de clasificación *fully connected*. Un ejemplo se muestra en la figura 2.6.

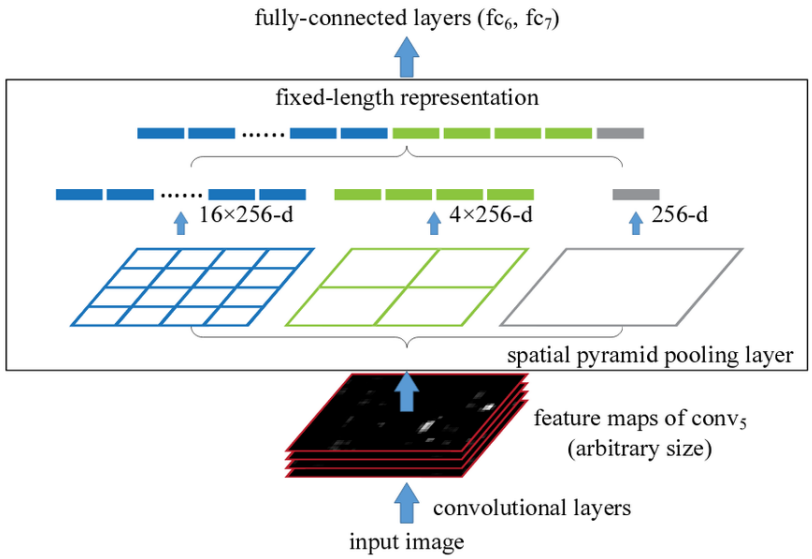


Figura 2.6: Ejemplo de SPP. Figura obtenida de [30].

2.2.3 Tipos de funciones de activación

Para poder obtener un modelo no lineal, las CNNs deben contar necesariamente a la salida de cada neurona con una función de activación no lineal. Estas funciones determinan si una neurona esta o no activada comparando su salida con un umbral de activación y, como se comentó en la sección

2.1.3, son aplicadas sobre el tensor de salida que se obtiene tras aplicar el filtro al tensor de entrada. De manera habitual se utilizaban como funciones de activación la función sigmoide (problemas de clasificación binaria) y la tangente hiperbólica (problemas de clasificación con datos de media cero), pero en redes profundas estas funciones hacen que surja el problema de desvanecimiento del gradiente. Esto sucede debido a que la forma característica de estas funciones, con una zona lineal central y asíntotas horizontales en sus extremos. Esta forma hace que, al utilizar *backpropagation*, se encadenen multiplicaciones por gradientes próximos a 0, provocando el desvanecimiento del gradiente. Además, cuando la función satura (su gradiente tiende a 0), cualquier ajuste en los filtros deja de tener efecto en la salida, haciendo que la red deje de aprender.

Por estos motivos, en *deep learning* se utilizan otro tipo de funciones de activación [17]:

- **ReLU**. Se trata de una función lineal en dos partes. Es el valor máximo entre 0 y el valor que tome la neurona $max(0, a_{i,j,k})$, lo que anula las activaciones negativas y sirve como una función identidad para las activaciones positivas. Su implementación es muy eficiente tanto en *forward pass* como en *backward pass*, además, presenta un alto rendimiento por lo que se ha convertido en la función de activación de referencia en las CNNs. A pesar de esto, sigue presentando problemas, pues al anular los valores negativos, provoca que las neuronas desactivadas no puedan participar en el aprendizaje. Las distintas funciones de activación existentes no son más que modificaciones de esta, en las que se trata de resolver este problema.

- ***Leaky ReLU (LReLU)***. Esta función es igual que la ReLU con la excepción de que la parte negativa tiene una pequeña pendiente definida por el parámetro $\lambda \in [0, 1]$ predefinido y constante. Esto permite conseguir un pequeño gradiente cuando la neurona está desactivada.

- ***Parametric ReLU (PReLU)*** **[31]**. De nuevo se emplea un parámetro para configurar la pendiente de la zona negativa de la función, pero en este caso se emplea uno independiente para cada canal, λ_k, que será entrenado. El objetivo de esta aproximación es especializar las activaciones dado que la LReLU no supone una gran mejora en la precisión de la red.

- ***Randomized ReLU (RReLU)*** **[32]**. Al igual que la PReLU, emplea un parámetro λ_k para cada canal, pero en lugar de entrenarlo, el valor se escoge aleatóriamente de una distribución uniforme en cada pasada n del entrenamiento: $\lambda_k^{(n)} \sim U(l, u); l < u \in [0, 1)$. Una vez más la razón de añadir un componente estocástico es la de introducir un mecanismo de regularización. Como ya se ha dicho con otros elementos estocásticos, es necesario eliminar el componente aleatorio tras la fase de entrenamiento, para ello, se fijan los parámetros a su valor medio durante el entrenamiento $\lambda_t = \frac{l+u}{2}$. En [32] se compara ReLU con su variantes probando empíricamente las mejoras de cada método. PReLU obtiene la mayor minimización de error en entrenamiento pero con mayor error en test, sobre todo con conjuntos de entrenamiento pequeños, esto indica que los parámetros están provocando *overfitting*. La mejor relación entre error de entrenamiento y test se obtiene con LReLU cuando se usa un parámetro de compresión bajo, y con RReLU, especialmente con *datasets* pequeños, donde es más efectivo su poder de regularización.

- ***Exponential Linear Unit (ELU)*** **[33]**. En este caso se mantiene en la parte positiva la función entidad mientras que para la parte negativa se emplea una función de saturación ex-

ponencial, haciendo que el gradiente sea no nulo cuando la neurona está "poco" desactivada y nulo si está desactivada. Esto aporta robustez frente a ruido.

- ***Maxout*** **[34]**. Este método se puede entender como una ReLU generalizada en la que, en lugar de tomar el máximo entre el valor del punto y 0, se toma el valor máximo entre todos los píxeles que comparten posición a lo largo de todos los canales. La salida de esta función devuelve un único mapa de características en el que se tienen en cuenta todos los canales de la entrada. Este método suele ser utilizado en conjunto con la técnica de regularización *dropout* detallada en la sección 2.2.4.2.

- ***Probout*** **[35]**. Surge como modificación del método *maxout*, con el objetivo de mejorar su invarianza. En este caso el valor de salida se elige escogiendo el valor del píxel en uno de los canales de manera aleatoria mediante una función multinomial. Mediante *probout* todos los canales tienen la posibilidad de ser muestreados. Cuando se van a realizar pruebas en el conjunto de test, se fijan los puntos haciendo un promedio de aquellos que han tenido mayor probabilidad durante el entrenamiento, para eliminar la aleatoriedad.

En la figura 2.7 se muestra de manera gráfica las funciones pertenecientes a la familia de las ReLU.

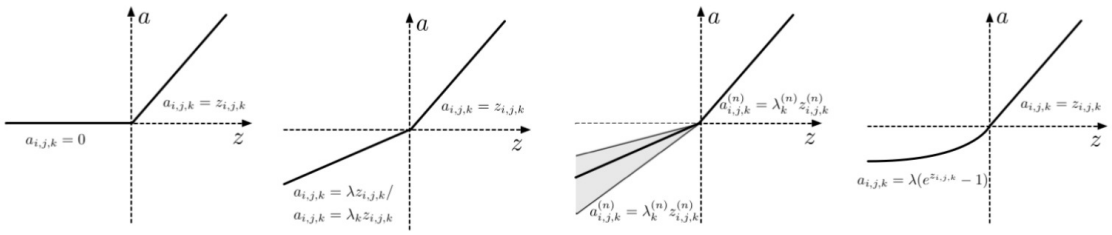


Figura 2.7: Funciones de activación tipo ReLU. De izquierda a derecha: ReLU, LReLU o PReLU (λ fijo o entrenable), RReLU y ELU. Figura obtenida de [17].

2.2.4 Mejoras de entrenamiento

El entrenamiento de una red neuronal consiste en encontrar el valor óptimo de pesos que minimicen una función de pérdidas. A continuación se exponen algunas de estas funciones y una serie de mecanismos de regularización, que tienen como objetivo prevenir el problema de *overfitting*, y de optimización, que buscan hacer el entrenamiento más eficiente.

2.2.4.1 Funciones de pérdidas

La función de pérdidas es la encargada de dirigir el proceso de entrenamiento de una red neuronal, ya que es esta función la que mide el error existente entre la salida real y la salida estimada por la red. Por ello, la elección de la función de pérdidas es de vital importancia, pues si no se escoge una función de pérdidas adecuada para el problema, los resultados no serán los deseados. A continuación se detallan algunas de las funciones de pérdidas más usadas:

- ***Hinge Loss* (L1 o L2) [17]**. Esta función tiene por objetivo no sólo estimar correctamente la clase de los datos, sino conseguir el máximo margen con la frontera de decisión, realizando la separación mediante la linea que maximice la distancia entre ambas clases. Inicialmente fue pensada para problemas de clasificación binarios, uno contra todos (*one vs all*), o uno contra uno (*one vs one*), pero se puede extender a problemas multiclase.

- ***Contrastive Loss* [17]**. Se encarga de medir la diferencia entre dos mapas de características. Se utilizan dos redes denominadas siamesas para codificar dos imágenes de entrada que están etiquetadas como çoincidentes.º "no coincidentes". La red es entrenada para reducir la distancia entre las codificaciones de las imágenes que sean coincidentes y alejar las que sean no coincidentes.

- ***Triplet Loss* [17]**. Mide la diferencia entre tres mapas de características, uno de referencia (*anchor*), uno negativo y uno positivo. El objetivo de esta función es alejar de la referencia los valores negativos y aproximar los valores positivos durante el entrenamiento. Este tipo de función de pérdidas es utilizada en sistemas de reconocimiento de rostros, por ejemplo, para controlar el acceso a una zona de trabajo donde se tendría como *anchor* y valor positivo dos fotografías de una misma persona, y como valor negativo una fotografía de una persona distinta.

- ***Cross Entropy Loss***. Es la función más utilizada en CNNs, empleada en problemas de clasificación multiclase. Esta formada por una combinación de *Negative Log Likelihood (NLL) Loss* y el logaritmo de la función de clasificación *softmax*, *LogSoftmax*. Mide la diferencia entre dos distribuciones de probabilidad, o como de cerca esta la distribución de clases predicha de la salida real.

- ***Euclidean error function***. Esta función calcula la distancia euclídea entre la salida estimada y la real. Se emplea en problemas de regresión [36].

2.2.4.2 Regularización

Cuando el número de parámetros entrenable de una red es considerablemente alto, esta red tiene una gran capacidad de aprendizaje. En estos casos es posible que si el *dataset* de entrenamiento es pequeño, se produzca *overfitting* o sobreaprendizaje, un problema que provoca que la red no sea capa de generalizar, y únicamente memorice las muestras del conjunto de entrenamiento. Esto es un problema porque una red en la que se ha producido sobreaprendizaje no funcionará correctamente cuando se le presenten nuevos datos. Para solucionar este problema hay dos opciones, o bien restringir la red, o ampliar la cantidad de datos de entrenamiento [37].

- ***Data Augmentation***. La técnica de *data augmentation* trata de conseguir ampliar el *dataset* de entrenamiento mediante modificaciones en dicho conjunto, sin alterar su naturaleza, generando de esta manera nuevos datos sintéticos. En el caso de imágenes, se suelen realizar transformaciones que pueden implicar invertir la imagen horizontal o verticalmente, realizar aumentos en una zona de la misma, cambiar el nivel de los canales RGB o variaciones a nivel de píxel. El uso de esta técnica hace que el sistema sea más robusto ante cambios de escala, color, etc. El uso de *Data Augmentation* sin embargo, puede suponer una sobrecarga y aumento de la

complejidad del entrenamiento dependiendo del tipo y cantidad de modificaciones realizadas, siendo aconsejable implementar transformaciones realistas en base al conocimiento previo que se tenga del problema a resolver.

- ***Lp-norm***. Este método de regularización añade términos que modifican la función objetivo, penalizando al modelo si este se vuelve demasiado complejo [37]. El método L2 es denominado también *weight decay* ya que el término que se añade provoca que al aplicar *backpropagation* y actualizar los pesos de la red, estos se vean reducidos de manera proporcional a su tamaño, limitando la libertad del sistema a la hora de ajustar los pesos.

- ***Dropout***. Este método retira, a medida que avanza por las capas y de manera aleatoria durante el entrenamiento, algunas neuronas, eliminando a su vez sus salidas y simplificando de este modo la red. Con esto se consigue que la red sea capaz de reaccionar cuando alguna de las entradas a una capa es retirada. En la fase de test se escalan las activaciones de las neuronas de manera inversa a la probabilidad de haber sido seleccionadas en el entrenamiento. Este método no puede aplicarse a las capas convolucionales, ya que se retirarían neuronas dispersas, quedando información de las neuronas vecinas que podría llegar a ser muy similar a las de los nodos retirados. Como solución a esto, en [38] se presenta una variante llamada *spatial dropout*, la cual anula todo el entorno de la neurona retirada, haciendo útil su aplicación en capas convolucionales.

- ***DropConnect*** **[39]**. Funciona de manera similar al *dropout*, con la diferencia de que en lugar de eliminar neuronas completas, elimina alguna de las entradas a dichas neurona, evitando así que la salida se anule por completo y al igual que en el caso anterior solo esta pensado para ser utilizado en salidas *fully connected*.

2.2.4.3 Optimización

Las estrategias de optimización están enfocadas en estructurar el proyecto de *deep learning* de manera que el aprendizaje sea más eficiente y eficaz. En la mayoría de los casos estas estrategias se centran en ajustar los parámetros de la red de manera adecuada. A continuación se explican algunos de los métodos más utilizados:

- **Inicialización de pesos**. La correcta inicialización de los pesos de la red juega un papel fundamental en el posterior entrenamiento de la misma, siendo especialmente importante inicializar de manera correcta los pesos de los filtros a utilizar. Por otra parte, el *bias* puede ser inicializado a cero o a una constante sin tener por ello que afectar al rendimiento de la red [12]. En el caso de los filtros, se busca una inicialización no nula y centrada en cero, evitando simetrías. En AlexNet [12] se propone una inicialización con una distribución gaussiana de media nula y desviación típica 0.01. Otro método de inicialización es el método "Xavier" [40] cuya propuesta es escalar la varianza utilizando la inversa del número de neuronas entrantes con el objetivo de mantener la escala.

- **SGD**. El gradiente por el descenso estocástico es una modificación del método del gradiente por el descenso, salvo que en lugar de utilizar todo el conjunto de entrenamiento, en este caso

se utilizan pequeños subconjuntos, estimando el gradiente a partir de estos *batch* de imágenes. Habitualmente se utiliza el término gradiente por el descenso estocástico para referirse al gradiente por el descenso estocástico por minilote (*minibatch*) aunque lo correcto sería referirse a este método como SGD solo en el caso de que el batch sea de una única imagen(también conocido como *heavy learning*. Con esto se consigue que el proceso de entrenamiento sea más eficiente consumiendo menos tiempo y recursos al procesarse menos imágenes en cada iteración. La manera más extendida de aplicación de este método incluye la aplicación de un término de inercia *momentum* (ecuación 2.3) para suavizar las actualizaciones de los pesos. La forma más sencilla de entender este término sería poner el clásico ejemplo de una persona descendiendo por una colina, de manera que avanzas un paso en la dirección de máxima pendiente (gradiente), a medida que das pasos en la misma dirección, aumentas tu inercia, de forma que si cambiase la dirección de la pendiente de manera brusca, la dirección a seguir por la persona cambiase de una manera más suave. Con esto se consiguen evitar cambios de dirección debidos a ruido en el sistema. Existe otra variante llamada Nesterov, que calcula el gradiente empleando los pesos ya desplazados con la inercia para anticipar la siguiente iteración.

La manera en la que sean seleccionados el *batch size* y el *momentum* afectarán al rendimiento del algoritmo. Típicamente se selecciona un *momentum* de 0.9, y el tamaño del *batch size* vendrá definido sobre todo por la cantidad de memoria con la que cuente la GPU a utilizar y el tamaño del *dataset* con el que se quiera realizar el entrenamiento.

- ***Learning rate***. El *learning rate* es el parámetro encargado de ponderar el gradiente, determinando de esta manera la velocidad del aprendizaje. La manera de escoger su valor es empírica, teniendo en cuenta que con un valor alto se conseguirá un entrenamiento rápido, pero que puede evitar mínimos, debido a que se podrían generar oscilaciones al rededor del mismo. Por otra parte un valor demasiado pequeño podría dejar al modelo atrapado en un mínimo local, además de hacer el proceso de entrenamiento demasiado lento. En ocasiones, se puede optar por tener varios valores distintos en un mismo modelo, por ejemplo cuando se quiere dejar una parte de un modelo preentrenado sin modificaciones, y solo se quieren entrenar las últimas capas de la red.

  Existen otras políticas o métodos de aprendizaje donde el *learning rate* se varía en función de la etapa del entrenamiento en la que se encuentre la red. Se puede realizar una primera etapa con un *learning rate* menor, hasta que el modelo converja y tras esto aumentar la tasa de aprendizaje [41]. También se puede reducir de manera polinómica a medida que el entrenamiento avanza, como se propone en [25].

- **Criterio de parada**. El criterio de parada viene dado de forma general por el número de iteraciones a partir de las cuales el error cometido en los conjuntos de entrenamiento y de validación se empieza a separar, aumentando de manera considerable en el conjunto de validación, lo cual es un signo de *overfitting*. Este número de iteraciones se suele obtener de manera experimental en base a pruebas previas u observando cuando el error alcanza un mínimo. La técnica de *early stopping* tiene un problema, y es que puede ser que el entrenamiento se pare demasiado pronto y por tanto no se esté consiguiendo alcanzar el valor óptimo que minimiza el error.

- ***Batch normalization***. Es común normalizar las imágenes de entrada para facilitar la con-

vergencia del modelo, por ejemplo, en [12] se resta la media de cada píxel al conjunto de entrenamiento. Sin embargo, según las características van siendo procesadas por la red y se va ganando profundidad, esta normalización se pierde y la distribución estadística de la entrada a una capa pasará a depender fuertemente de la inicialización y variaciones en los pesos de las capas previas. Esto se conoce como desplazamiento interno covariable. También existe el problema del desvanecimiento del gradiente. Mediante la normalización por *batches* se solucionan estos problemas al normalizar la entrada a cada capa por canales, calculando su media y varianza con todos los ejemplos del *batch*. Este operador se implementa como una capa más y pasa a formar parte del modelo y, aunque sirve como regularizador, su efecto más notable es la aceleración del entrenamiento [42]. Para su uso en test se acumulan estadísticas de todos los ejemplos vistos en el entrenamiento con una media móvil que será empleada para la normalización durante el test.

- ***Shortcut connections***. A medida que las redes neuronales se han ido haciendo cada vez más profundas, han surgido nuevos problemas debidos a este aumento significativo del número de capas. Este problema es debido a que en las redes muy profundas, no se puede conseguir una reducción significativa del error en un tiempo razonable, además de ser menos optimizables que las redes superficiales. A priori esto puede resultar contraintuitivo, pues una red neuronal profunda de N capas podría entenderse como la generalización una red superficial de M capas siendo $M < N$, donde el sistema hubiese aprendido esas primeras capas como una matriz identidad, quedando de esta forma .ªnuladas", de manera que el error de la red profunda debería ser como máximo el de la red superficial.

  Para solucionar esto, en [41] se propuso la introducción en la red de conectar capas no consecutivas, de manera que la salida de una capa previa podía pasar directamente a la salida de una capa posterior como se ve en la figura 2.8. Con estas conexiones se consigue facilitar al sistema el aprendizaje de las capas que han sido saltadas consiguiendo acelerar enormemente el proceso de entrenamiento de redes profundas.

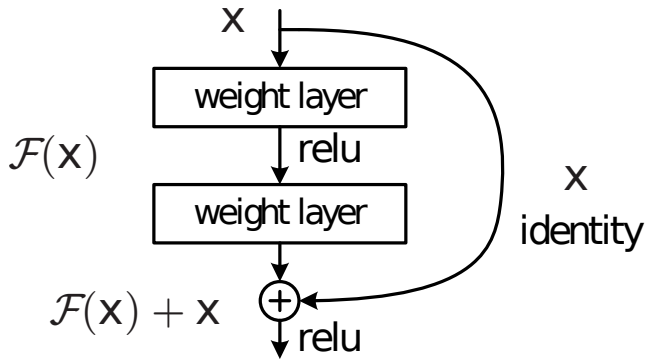


Figura 2.8: Estructura de una capa con *shortcut connection*. Figura obtenida de [41].

2.2.5 Arquitecturas existentes

En esta sección se muestran algunas de las arquitecturas de CNNs existentes para resolver los problemas actuales en visión artificial, como son clasificación, detección y segmentación semántica de la imagen:

- **AlexNet [12].** AlexNet fue la red que impulsó el estudio y por tanto la aparición de las CNNs actuales. Se utilizó en el reto de clasificación de imágenes de la base de datos de ImageNet [12] obteniendo un éxito aplastante. Esta red emplea únicamente 8 capas con pesos entrenables (5 convolucionales y 3 *fully connected*).

- **VGG [43].** Este trabajo inicia el estudio hacia redes cada vez más profundas al proponer el paso al uso de filtros con dimensiones espaciales reducidas pero en más capas. En este caso la variante más profunda de VGG alcanza las 19 capas entrenables. Este uso de filtros reducidos permite mantener el campo de visión al ser aplicados de manera recursiva, y aumentar el número de no linealidades entre ellos, mejorando así la capacidad semántica de la red a la vez que reduce el número de pesos a entrenar.

  En este trabajo también se propone el uso de una estructura compuesta por bloques de capas convolucionales colocadas en cascada y separados por capas de *pooling* que emplean un $F = 2$ y un *stride* $S = 2$. Esta configuración de parámetros del filtro hace que en el paso de un bloque al siguiente, queden reducidas las dimensiones de anchura W y altura H del tensor de entrada a la mitad. Además, en cada paso se duplican el número de filtros, duplicando de esta forma el número de canales o profundidad del tensor de salida.

- **GoogleNet o *inception* [44].** La arquitectura GoogleNet introduce la idea de utilizar filtros de distintos tamaños sobre un tensor de entrada, pero asegurando que todos den a su salida tensores en los que al menos las dimensiones de anchura W y altura H sean las mismas. De esta manera se pueden concatenar en profundidad todos esos tensores de salida obtenidos. Este tipo de módulo se denomina *inception* y nació con la intención de obtener redes cada vez más profundas. Esta concatenación de salidas obtenidas con distintos tipos de filtros hace que la entrada sea procesada con distintos campos de visión aportando invarianza de escala. Existen dos implementaciones del módulo *Inception*, en una aplicando los filtros directamente sobre el tensor de entrada, y otra con un paso previo en el que se aplica un filtro de 1×1 con el objetivo de reducir las dimensiones del tensor, y tras el que se aplica el filtro deseado para conseguir las dimensiones que se quieran a la salida. Esta estrategia de reducción previa de las dimensiones del tensor de entrada es conocida como *bottleneck* o cuello de botella, y reduce significativamente el número de cálculos a realizar por el sistema. En la figura 2.9 se muestran estas dos implementaciones.

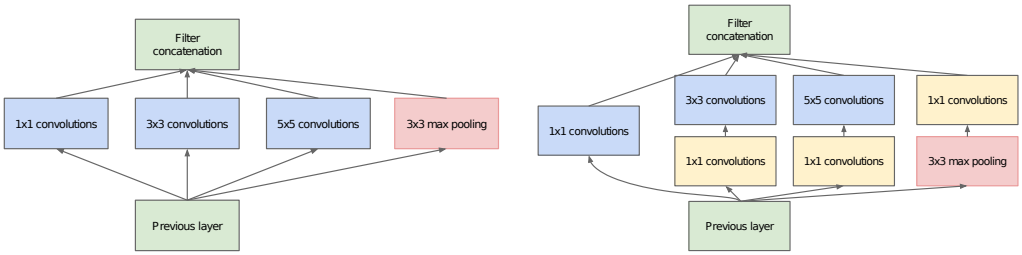


Figura 2.9: Módulo *Inception*. Implementación sin reducción de dimensiones (izquierda) y con reducción de dimensiones (derecha). Figura obtenida de [44].

- **ResNet [41].** ResNet es la red que introduce por primera vez las *shortcut connections* y que avanza hacia redes cada vez más profundas, siguiendo el camino de redes anteriores, empleando

filtros de tamaño reducido. Los bloques que se repiten de manera periódica en ResNet están compuestos por dos o tres capas convolucionales, donde la entrada a la primera capa esta puenteada a su vez mediante una *shortcut connection* a la salida del bloque. En ResNet, cuando se avanza de una etapa a otra, se realiza una convolución con stride $S = 2$ y se duplica el numero de filtros y por tanto de canales de la salida. Esto provoca que la información puenteada no tenga las mismas dimensiones que la información de salida por lo que es necesario realizar un escalado para poder sumar la salida y lo que viene por la *shortcut connection*. Este escalado se puede realizar o bien rellenando con un *zero padding* o utilizando filtros 1×1 para realizar una proyección. Para obtener la salida de la red se utiliza *average pooling* sobre la última capa convolucional, cuya salida es emplea como entrada de una única capa *fully connected* seguida por un clasificador *softmax*.

ResNet emplea dos tipos de bloques básicos que se muestran en la figura 2.10, el primero, con solo dos capas llega hasta las 34 capas de profundidad, tras lo que se pasa a aplicar el bloque de tres capas, que vuelve a hacer uso el concepto de *bottleneck* y reduce el número de cálculos a realizar durante el entrenamiento. ResNet es una de las arquitecturas más utilizadas actualmente, ya que es una red muy profunda y es mucho más eficiente en términos de capacidad computacional que sus predecesoras.

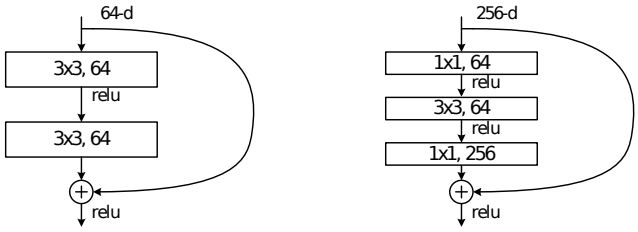


Figura 2.10: Módulos de ResNet. Figura obtenida de [41].

- **R-CNN [45].** Una de las primeras CNNs diseñadas para la detección de objetos. Esta arquitectura esta dividida en tres partes, un modelo encargado de decir si en una región o *bounding box* de la imagen hay o no objeto, un segundo modelo CNN que extrae las características de las *boudning box*, y un clasificador que determina que objeto hay en dicha *bounding box*. El método conocido como *Selective Search* es el empleado para obtener las *bounding box* donde podría haber un objeto. Estas regiones propuestas sirven de entrada a una versión modificada de AlexNet que extrae sus características y las envía a un SVM encargado de clasificar el objeto si lo hubiera. Tras esto se emplea un modelo de regresión para ajustar las coordenadas del objeto. En la figura 2.11 se muestra esta estructura.

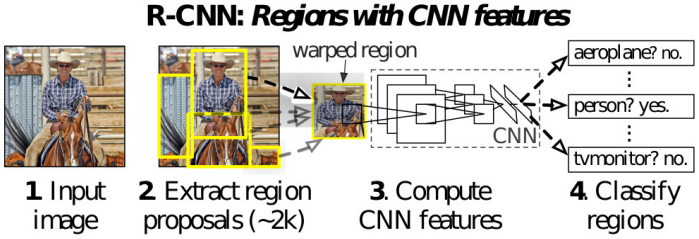


Figura 2.11: Ejemplo de aplicación de una R-CNN. Figura obtenida de [45].

- **Fast R-CNN [46].** Esta arquitectura es una modificación de R-CNN, ya que esta, al tener que pasar por AlexNet para cada *bouding box*, es muy lenta. Además, el entrenamiento se complica al tener que entrenar los tres modelos explicados en el punto anterior. Para hacer más rápida a la red, se obtiene directamente a partir de la imagen de entrada un mapa de características global. Con este mapa de características y las propuestas de objetos hechas por el *selective search*, se extraen vectores de características del mapa global mediante la técnica *Region of Interest (RoI) pooling*. Estos vectores de características de longitud fija pasan a dos capas *fully connected*, una que devuelve la probabilidad de cada clase, y otra que devuelve cuatro números reales que representan las coordenadas de la *bounding box*. Esto último también resuelve el problema del entrenamiento, ya que los tres modelos quedan combinados en uno.

- **Faster R-CNN [47].** Faster R-CNN es una modificación posterior de Fast R-CNN que busca sustituir el algoritmo de búsqueda *Selective Search*. En este caso las regiones son propuestas por un sistema que emplea una ventana deslizante sobre la imagen obteniendo n regiones potenciales de diferentes tamaños y con una probabilidad de contener un objeto asignada. En la figura 2.12 se muestra la estructura de este sistema.

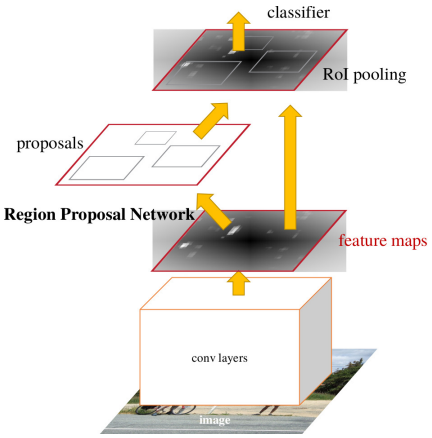


Figura 2.12: Ejemplo de Faster R-CNN. Figura obtenida de [47].

- **Mask R-CNN [48].** La última modificación realizada sobre R-CNN es la Mask R-CNN, en la que se trata de clasificar el objeto a nivel de píxel, es decir, realizar segmentación de la imagen. Para esto se añade otra rama a la red, en paralelo con las de clasificación y detección de *bounding box*, que genera una máscara binaria y dice si un píxel pertenece o no a un objeto. Para poder determinar esto, es necesario que esta nueva rama sea *fully convolutional* y, al igual que sus otras ramas paralelas, este alimentada con los mapas de características de cada región. También se añade una modificación al *RoI pooling* para alinear las regiones con los objetos que contienen, ya que en las Fast R-CNN estas se encuentran desalineadas. A continuación en la figura 2.13 se muestra un ejemplo del sistema Mask R-CNN.

- **YOLO[10].** YOLO es un sistema diseñado para la detectar y clasificar objetos en imágenes con solo un paso de *feed forward*. YOLO divide la imagen en una rejilla de tamaño $S \times S$ y, para cada celda, extrae haciendo uso de CNNs un vector de características que determine si hay objeto o no en esa venta, las*bounding boxes* B que haya definidas en la salida de la red, y la probabilidad de pertenencia a C clases de dicho objeto. El tensor de salida tiene un tamaño

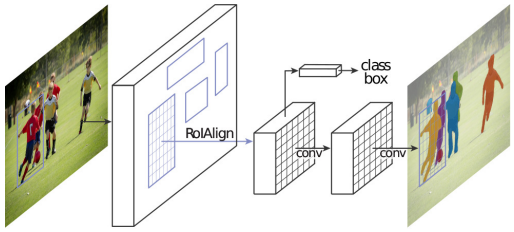


Figura 2.13: Ejemplo de Mask R-CNN. Figura obtenida de [48].

según estos parámetros de $S \times S \times (B \times 5 + C)$. La salida de esta red debe ser filtrada para quedarse solo con las detecciones que cuenten con una alta probabilidad, para ello se fija un *threshold* para eliminar las *bounding boxes* con baja probabilidad. Tras esto, para cada celda, entre las *bounding boxes* que hayan quedado, se queda únicamente con la que detecte la clase con una mayor probabilidad. La arquitectura YOLO es ligeramente más rápida que Faster R-CNN y comete menos errores de detección de fondos en los que no hay objetos. El principal inconveniente es que solo puede detectar un número limitado de regiones por celda, y que para cada celda solo puede detectar una única clase. En la figura 2.14 se muestra la CNN utilizada, donde el tamaño de la salida es $7 \times 7 \times 30$ ya que se detectaban 20 clases, 2 *bounding boxes* por celda y se dividía la imagen de entrada en 7×7 celdas.

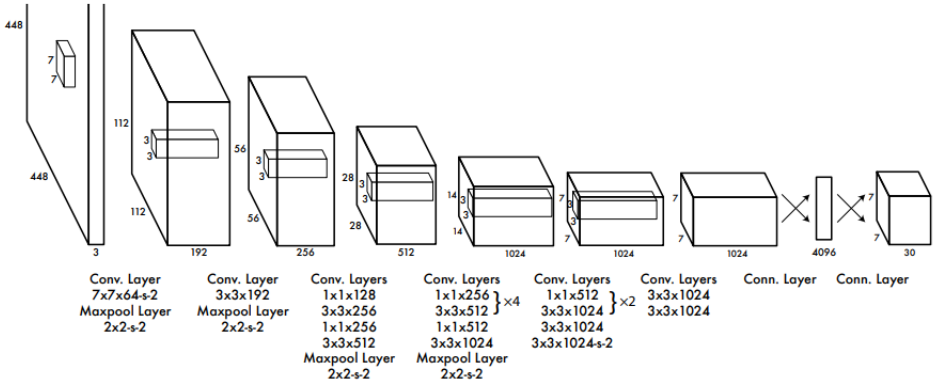


Figura 2.14: Arquitectura de la red YOLO. Figura obtenida de [10].

- ***Fully Convolutional Networks (FCNs)* [21].** La redes *fully convolutional* son utilizadas, como se mencionó en la sección 2.1.4, para realizar segmentación semántica de la imagen. En estas redes todas las capas, incluyendo las de salida, son capas convolucionales, y se ha comprobado que si se transforman las capas *fully connected* de las redes tradicionales en capas convolucionales, estas redes pueden ser utilizadas para segmentación semántica. Estas últimas capas convolucionales utilizan filtros que tienen el mismo tamaño que el tensor de entrada, y se emplean tantos filtros como neuronas tuviera la capa *fully connected* quedando a la salida un tensor de dimensiones espaciales 1×1, y con una profundidad igual a la que tuviera la capa *fully connected* de la que partía. En principio la imagen alimentada a estas capas es del mismo tamaño que el diseñado en la red original, sin embargo, se puede alimentar con cualquier tamaño de entrada, quedando la salida como un tensor con dimensiones de anchura y altura distintas de 1 en la que cada canal es un mapa de confianza con la probabilidad de pertenencia

a una clase de cada subbloque de la imagen. Como las sucesivas capas de la red submuestrean la entrada, para poder hacer segmentación a nivel de píxel es necesario realizar un procesado adicional mediante el cual se interpolen los mapas de características de la salida.

- ***Full-Resolution Residual Networks (FRRNs)*** **[49].** En este modelo no se parte de ninguna red anterior, si no que desarrolla una arquitectura completamente nueva en la que se tienen dos flujos de datos. En uno se mantiene la resolución completa de la imagen y en el otro se submuestrea, combinando así características locales para detectar fronteras de objetos, con características globales encargadas de identificar el objeto en sí. Además, durante el proceso de entrenamiento de la red, se almacenan los resultados intermedios del *forward pass* para la propagación posterior del gradiente.

Además de estas arquitecturas, utilizadas sobre todo para problemas de visión artificial, cabe destacar las dos que se muestran a continuación, que aunque se alejan del problema en cuestión, forman parte del estado del arte de estos sistemas y han conseguido resultados sorprendentes:

- ***Recurrent Neural Networks (RNNs)*** **[50].** Estas redes están principalmente pensadas para el procesamiento del lenguaje natural, aunque pueden ser utilizadas en cualquier tipo de datos que siga una secuencia temporal. La manera de alimentar estas redes consiste en introducir el primer dato de la secuencia en la primera neurona, y con la salida de esa neurona, alimentar la siguiente, además de añadir el siguiente dato de la secuencia. Con esto se consigue dar una percepción temporal a la red. De la misma manera se emplea el algoritmo de *back propagation* de manera secuencial desde la última neurona hasta la primera.
- ***Generative Adversarial Networks (GANs)*** **[51].** Las GANs son un tipo de sistema que consiste en enfrentar dos redes neuronales, una encargada de generar imágenes y otra encargada de identificar si la imagen es real o si por el contrario ha sido generada por la primera red. Para conseguir esto a la primera red, que consiste en una red deconvolucional, se le alimenta con una imagen de ruido blanco, generando a su salida una imagen con la que se alimentará a la red discriminadora, la cual es alimentada a la vez con un *dataset* de imágenes reales. De esta manera, la función de perdidas de la red discriminadora esta ligada a la de la red generadora, reduciendo las perdidas de la red generadora mientras aumentan las de la encargada de discriminar. Tras este entrenamiento, la red generativa es capaz de ofrecer imágenes que podrían parecer reales ante un espectador humano. En la figura 2.15 se puede ver un ejemplo de como funciona este tipo de red.

2.2.6 Herramientas

En esta sección se van a comentar algunas de las herramientas *software* disponibles para el desarrollo de sistemas de *deep learning*.

Dentro de la multitud de herramientas *software* existentes, se pueden encontrar desde librerías orientadas a *Machine Learning* que cuentan con funciones para generar CNNs en lenguajes de alto nivel, hasta *frameworks* completos con capas y modelos ya implementados de forma eficiente y preparados para ejecutarse en CPU o GPU.

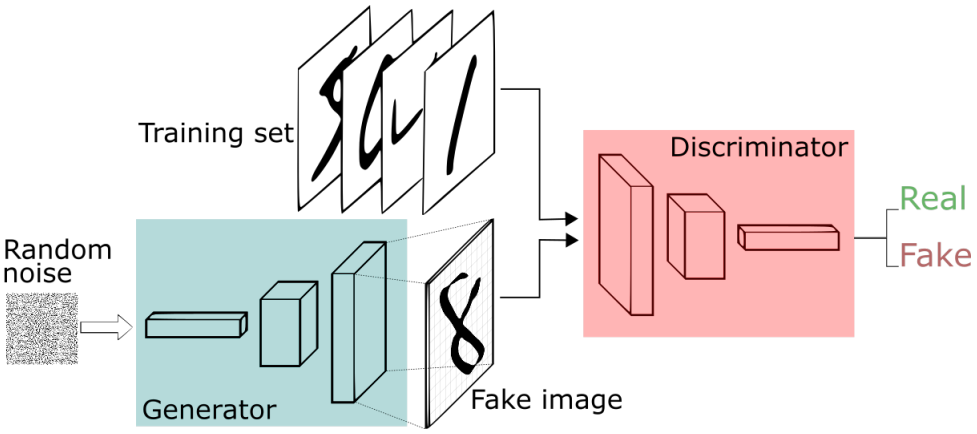


Figura 2.15: Ejemplo de arquitectura de una Red Generativa Antagónica o GAN. Figura obtenida de [52].

Entre estas opciones *software* se pueden destacar las siguientes:

- **TensorFlow [53].** Es una librería de código abierto desarrollada para *Machine Learning* y CNNs escrita en Python y desarrollada por Google. Está orientado al manejo y procesado de tensores. El flujo de trabajo típico es, definir una arquitectura, establecer las políticas de entrenamiento, y ejecutar la optimización.

- **Keras [54].** Al igual que TensorFlow, es una librería de código abierto y pensada para programar redes neuronales y entrenarlas en un periodo de tiempo razonable. Es capaz de ejecutarse sobre TensorFlow y, al estar estrechamente relacionado, el flujo de trabajo es muy similar en ambas librerías.

- **MATLAB [55].** MATLAB es una herramienta de *software* utilizada principalmente para cómputo matricial, que cuenta con su propio lenguaje de programación , siendo muy utilizado en entornos de ingeniería. Este *software* es además muy amigable con el usuario, lo que facilita su aprendizaje y hace que sea ampliamente usado para la docencia. MATLAB cuenta además con un gran número de *toolboxes* desarrolladas por la comunidad, habiendo ampliado en las últimas versiones el número de estas que se enfocan en la programación de redes neuronales, visión artificial y *deep learning*. El principal problema de MATLAB es que se trata de *software* privativo, lo que dificulta su interacción con otros sistemas.

- **PyTorch [56].** PyTorch es otro de los *Frameworks* existentes que facilitan el manejo de tensores, optimizado para ser utilizado tanto en CPU como GPU. Existe el paquete torch que proporciona una interfaz para ser utilizada en Python, el cual contiene el núcleo de la funcionalidad. PyTorch permite escribir arquitecturas mediante una programación orientada a objetos y facilita la depuración de los procesos de entrenamiento y testeo, contando con multitud de funciones preparadas para ello. Además cuenta con distintos modelos de CNNs preimplementados. Es esta cantidad de funciones y modelos, ademas de la facilidad de uso, lo que hacen que PyTorch sea ampliamente utilizado para la investigación y el prototipado.

2.3 Aplicación práctica: detección de *keypoints*

Como se ha explicado con anterioridad, en el presente trabajo se va a diseñar un sistema aplicando CNNs que sea capaz de detectar en distintas imágenes los puntos significativos de un vehículo. Para conseguir esto se han estudiado distintas arquitecturas de redes diseñadas concretamente para resolver el problema de detección de *keypoints*, estando estas principalmente enfocadas a la detección en seres humanos, y tratando de adaptar el modelo para que sea capaz de detectar estos puntos sobre la estructura de un coche. De manera habitual estas redes neuronales también se encargan de unir los puntos, aprendiendo cuales de ellos pertenecen al mismo cuerpo, y pudiendo así estimar la pose de cada persona que aparezca en la imagen.

Como ejemplo de arquitectura de una red neuronal de detección de *keypoints* se puede destacar OpenPose [57]. Esta red además de detectar las articulaciones del cuerpo humano, es capaz de asociar estos puntos mediante lo que denominan PAFs. Estos PAFs consisten en unos campos vectoriales que determinan la dirección en la que esta el miembro del cuerpo, permitiendo unir dos puntos que se encuentren sobre el mismo miembro, y evitando problemas de unión de *keypoints* que pertenezcan a distintas personas.

En la figura 2.16 se puede observar el PAFs de los antebrazos derechos de las personas presentes en la fotografía. Utilizando estos campos vectoriales, la red neuronal aprende que debe unir puntos que se encuentren sobre el mismo miembro.

Figura 2.16: Ejemplo de PAF. Figura obtenida de [57].

En la figura 2.17 se muestra la arquitectura de la red neuronal. La primera parte, resaltada en azul, es la encargada de detectar los PAFs, mientras que la capa sombreada en rosa detecta los *keypoints* de cada persona. La entrada a esta red es un mapa de características de la imagen de entrada, que se obtiene como la salida de la décima capa de una red VGG-19, vista en la sección 2.2.5 [43]. La principal ventaja de esta red frente a otras viene dada por el hecho de que OpenPose es capaz de realizar una estimación de pose multi-persona, sin necesidad de hacer una detección individual previa para cada persona en la imagen. En este caso, la red en primer lugar detecta los *keypoints* en la imagen, y mediante los campos vectoriales mencionados, puntúa cada pareja de puntos según puedan pertenecer a un mismo cuerpo o no. Con este enfoque se consiguen resultados con alta precisión sin hacer uso de un detector de personas previo.

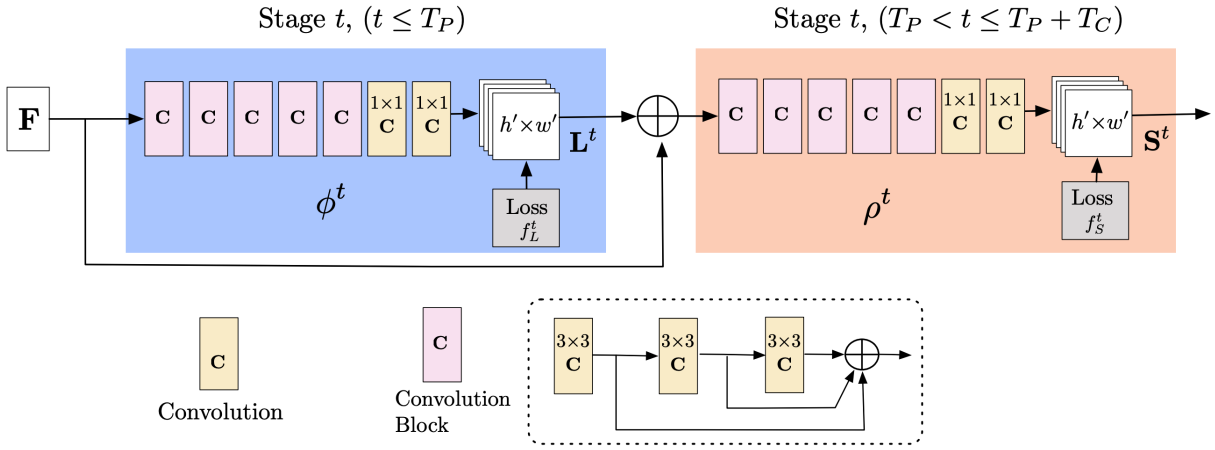


Figura 2.17: Arquitectura de la red neuronal de OpenPose. Figura obtenida de [57].

En [57], Zhe Cao et al. también prueban esta red modificada para trabajar sobre un *dataset* formado por imágenes de vehículos, en los que consiguen resultados como los que se pueden ver en la figura 2.18 con un *Average Precision (AP)* del 70.1 %.

Figura 2.18: Ejemplo de la red neuronal de OpenPose utilizada sobre imágenes de vehículos. Figura obtenida de [57].

La aplicación práctica a desarrollar en este proyecto es similar a lo que se observa en esta última figura, teniendo como primer objetivo la detección de los puntos y su representación en la imagen, prescindiendo del dibujo de la estructura que los une. En los siguientes capítulos se presentará el problema y la arquitectura empleada para su resolución, así como los *datasets* que se han diseñado y empleado, explicando las decisiones tomadas para el etiquetado del mismo.

Capítulo 3

Presentación del problema

A continuación se va a exponer de manera detallada el problema tratado en este proyecto, así como los *dataset* utilizados para el correcto entrenamiento de la red neuronal.

3.1 Diseño del sistema

En este trabajo se intenta conseguir un sistema capaz de detectar los puntos significativos de la estructura de un vehículo turismo. El sistema recibirá a su entrada una imagen y deberá devolver a su salida las coordenadas de la *bounding box* que contiene al vehículo, y las coordenadas de los *keypoints* de este, representándolos en la imagen de salida. Para conseguir esto se puede enfocar el problema de dos maneras, una con un enfoque *end-to-end* y otra con un enfoque modular:

- ***End-to-end***. El enfoque *end-to-end* consiste en un único sistema que se encargaría de resolver todo el problema, en este caso, para una imagen de entrada RGB, el sistema devolverá a la salida las coordenadas de cada una de las *bounding box* para cada vehículo presente en la imagen, y el etiquetado de los *keypoints* de cada vehículo, representando estos sobre la imagen.
- **Modular**. Un sistema con enfoque modular para este problema en concreto consistiría en contar con dos redes neuronales. La primera red sería la encargada de detectar una *bounding box* para cada vehículo, recortando la imagen según esa *bounding box* y devolviendo a su salida la imagen recortada. Esta salida sería utilizada como entrada para la segunda red, que devolvería para cada vehículo los puntos significativos encontrados, mediante el uso de mapas de calor.

El sistema quedaría por tanto como una red neuronal que recibe a su entrada una imagen con tres canales de color RGB, a partir de aquí y dependiendo del enfoque elegido se podrá trabajar de dos formas. Si se elige un enfoque *end-to-end*, el sistema directamente devolverá los *keypoints* para cada vehículo en la imagen, teniendo unicamente que entrenar la red para los puntos que se consideren significativos para la mayor parte de los vehículos existentes. Si en cambio se opta por un enfoque modular, la salida de la primera red tendrá que ser recortada y escalada para adecuarse a las dimensiones de entrada de la segunda red. Hay que tener en cuenta que la entrada de la segunda red estará formada por imágenes en las que solo aparece un vehículo que ocupa la mayor parte de

la imagen, puesto que la entrada de la segunda red son las *bounding box* que se han detectado en la primera. La segunda red será la encargada de detectar, para cada imagen de entrada, los *keypoints* del vehículo.

En concreto para el caso de este proyecto se va a optar por un enfoque modular, centrándose en la segunda red, encargada de la detección de *keypoints*. Para entrenar esta red se deberá contar por tanto con un *dataset* en el que para cada imagen de entrada solo estará etiquetado un único vehículo, debiendo duplicar la imagen si en ella hay más vehículos etiquetados.

3.2 *Datasets*

Para poder implementar el sistema en cuestión es necesario contar con una base de datos de vehículos lo suficientemente grande para poder entrenar la red. En esta base de datos tienen que estar las imágenes de los vehículos y su etiquetado correspondiente. La primera fuente de imágenes etiquetadas de coches ha sido el *dataset* de PASCAL3D+ [58] que cuenta con imágenes de ImageNet y de PASCAL VOC 2012 [59].

En [58] además de coches, trabajan con otro tipo de objetos y vehículos, desde pantallas de televisión hasta aviones o motocicletas, pero en el caso de este proyecto solo se ha hecho uso de la parte de turismos. Por otra parte se puede construir un *dataset* desde cero, tomando imágenes tanto de internet, como realizando capturas con cámaras propias, y etiquetando posteriormente mediante *software* dichas imágenes.

3.2.1 *Dataset* PASCAL3D+

Para este proyecto se han utilizado únicamente las imágenes de ImageNet. Esta base de datos cuenta con 5475 imágenes de coches, estando divididas en 2763 imágenes de entrenamiento y 2712 imágenes de validación. El etiquetado utilizado en [58] consiste en el *bounding box* y un total de 12 *keypoints*, siendo estos las cuatro ruedas, las esquinas superiores del parabrisas y la luna trasera, los dos faros, y las esquinas inferiores del maletero. Para cada *keypoint* cuenta también con una característica que determina si el punto es visible o no en la imagen.

Algunos ejemplos de las imágenes de este *dataset* se muestran en la figura 3.1, donde se puede ver que el conjunto de datos es muy variado, contando con vehículos antiguos, pickups, formula 1, etc...

También cuentan con una serie de archivos *Computer-aided design (CAD)* para distintos tipos de coches (familiar, sedán, coupé, etc...) los cuales son dibujados sobre el coche real. En este *dataset*, los autores proporcionan además su propia herramienta de etiquetado, programada en MatLab, de manera que los *keypoints* puedan ser etiquetados personalmente y se puedan añadir más imágenes al conjunto de manera sencilla. Esta herramienta se muestra en la figura 3.2.

Los principales inconvenientes de esta herramienta son dos, el primero es que solo permite etiquetar los 12 puntos mencionados anteriormente, sin permitirte añadir o eliminar puntos propios de

manera inmediata. El segundo problema se encuentra en que la manera de etiquetado se hace únicamente mediante el uso del ratón del ordenador, por lo que la precisión a la hora de fijar *keypoints* no es muy alta.

Figura 3.1: Ejemplos del conjunto de imágenes de ImageNet.

Figura 3.2: Herramienta de etiquetado de PASCAL3D+. Figura obtenida de [58].

3.2.2 *Dataset* propio

En el caso de este proyecto, se utiliza un *dataset* propio, haciendo uso de las imágenes de ImageNet, y de imágenes capturadas mediante las cámaras montadas en el vehículo autónomo del grupo INVETT de la Universidad de Alcalá, y mediante cámaras montadas en el puente situado en la calle Carretera Universidad Complutense que apuntan a la autopista A-2.

El etiquetado también será distinto al mencionado en la sección 3.2.1, añadiendo, eliminando y

reubicando *keypoints* según un criterio propio que se explicará en el capítulo 4. En las figuras 3.3 y 3.4 se pueden ver las cámaras utilizadas para tomar las imágenes que ayudarán a ampliar el *dataset* obtenido de PASCAL3D+.

Figura 3.3: Puente A2.

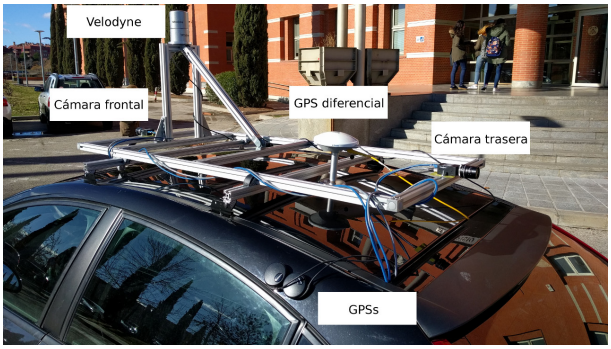


Figura 3.4: Plataforma móvil.

Además de las imágenes propias, también se ampliará el *dataset* haciendo uso de las imágenes de la base de datos pública de CompCars [60]. Esta base de datos se presentó en el *Conference on Computer Vision and Pattern Recognition (CVPR)* en 2015 y cuenta con 136726 imágenes de vehículos completos y obtenidas de dos fuentes distintas. Una parte de las imágenes de este *dataset* han sido obtenidas de internet, y otra parte de las distintas cámaras de tráfico que hay en las autopistas.

En este caso, en lugar de hacer uso del *dataset* completo, solo se tomará una parte de las imágenes que serán añadidas al conjunto de imágenes formado por las capturadas con las cámaras de la universidad y por las imágenes de ImageNet. En la figura 3.5 se pueden ver ejemplos de imágenes procedentes de esta base de datos.

Figura 3.5: Ejemplo de imágenes de CompCars.

Por otra parte, como se ha explicado anteriormente, hay que modificar los *keypoints* de las imágenes. Para realizar el etiquetado de este *dataset* se ha desarrollado una herramienta en python que hace uso de la librería de visión artificial OpenCV.

Esta herramienta accede a la carpeta donde se encuentran las imágenes y las va recorriendo, mostrando una interfaz gráfica que permite al usuario el etiquetado mediante teclado y ratón, modificando para cada imagen el archivo json asociado que sigue el formato utilizado en PASCAL3D+,

y que podrá ser utilizado para entrenar la red neuronal. Al poder utilizar tanto el teclado como el ratón, permite ajustar los *keypoints* de manera mucho más precisa que con la herramienta mostrada en la sección 3.2.1.

Capítulo 4

Descripción experimental

En este capítulo se van a presentar los pasos realizados para la realización de este trabajo, desde el razonamiento empleado para el etiquetado del *dataset*, hasta las arquitecturas de red implementadas para la detección de *keypoints*. En este caso el trabajo se ha realizado basándose en una red previa que ha sido desarrollada para la detección de *keypoints* en personas, y que se ha modificado para detectar puntos significativos en vehículos.

4.1 Etiquetado de las imágenes

La primera parte de este proyecto ha consistido en la creación de un *dataset* que contenga imágenes de turismos etiquetados de acuerdo a un razonamiento que permita que la red sea capaz de generalizar para la mayoría de turismos existentes.

Para construir este *dataset* se ha desarrollado una herramienta en python para realizar que se puede observar en las figuras 4.1, 4.2 y 4.3.

Figura 4.1: Herramienta de etiquetado. *Bounding Box*.

Figura 4.2: Herramienta de etiquetado. *Keypoints*.

En la primera figura (4.1) se puede ver el primer paso de la herramienta. En este paso, se muestra una ventana con la *bounding box* que hay actualmente etiquetada y mediante terminal se da la opción

Figura 4.3: Herramienta de etiquetado del grupo INVETT. Etiquetado individual.

de confirmar o elegir una nueva *bounding box*. Una vez seleccionada, la herramienta pasa a mostrar las figuras 4.2 y 4.3.

En la primera figura, se muestran los *keypoints* actualmente etiquetados y en rojo el que se esta modificando. Es sobre esta figura sobre la que puedes elegir mediante el ratón la posición aproximada del punto que se esté modificando. La figura 4.3 muestra un aumento de la primera imagen, centrándose en el *keypoint* que se esta modificando. Este punto puede ajustarse haciendo uso de las flechas de dirección del teclado del ordenador.

La herramienta permite desplazarse entre los distintos puntos mediante las teclas S y A del teclado. Una vez etiquetados todos los puntos, se presiona la tecla N (*next*) para que se guarde el etiquetado en el json correspondiente y se cargue la siguiente imagen.

El etiquetado consta de un total de 19 *keypoint* siendo estos los siguientes:

- Cuatro esquinas del parabrisas frontal.
- Cuatro esquinas de la luna trasera.
- Llanta delantera y llanta trasera.
- Faros antiniebla delanteros.
- Esquinas de la placa de matrícula.
- Espejos retrovisores.
- Logotipo de la marca del vehículo.

Estos puntos se han elegido porque son puntos comunes a la gran mayoría de vehículos. Se han escogido antes los faros antiniebla que los faros principales porque cada fabricante tiene un diseño distinto para sus faros tanto delanteros como traseros, sin embargo la forma de los faros antiniebla es similar en la gran mayoría de vehículos. En el caso de las llantas, solamente se ha escogido una delantera y una trasera porque nunca podrán aparecer más de dos llantas a la vez en una única imagen de un coche. Por último, el último punto que puede generar dudas es el logotipo del vehículo, aunque cada marca tiene el suyo, la red puede generalizar a partir de la situación relativa de este

punto con respecto a los demás, ya que el logotipo siempre suele estar situado encima de la placa de matrícula y entre los faros.

Por otra parte se han estudiado otros puntos que pueden ser comunes a la mayoría de vehículos como pueden ser las manetas de las puertas o el punto de separación de la ventana delantera y trasera, sin embargo, añadir más puntos repercutiría en la velocidad del entrenamiento de la red, que al tener que calcular un número mayor de *keypoints*, se vería ralentizada.

Tampoco se ha realizado el etiquetado de los puntos no visibles, como se hacia en el *dataset* de PASCAL3D+ debido a que etiquetar puntos no representados en la imagen podría llevar a la red a aprender de manera aleatoria, ya que la única información representativa sería la posición relativa de estos *keypoints* con respecto al resto.

El *dataset* creado esta repartido como se muestra en la tabla 4.1.

Dataset	**Entrenamiento**	**Validación**	**Total**
ImageNet	1961	840	2801
CompCars	629	269	898
Imágenes propias	267	114	381
Total	2857	1223	4080

Tabla 4.1: *Dataset* propio.

4.2 Arquitectura de la red

En este proyecto se han utilizado dos redes neuronales distintas, diseñadas en un principio para la detección de *keypoints* en humanos. Estas redes neuronales son presentadas en ambos casos por Bin Xao et al. en [61] y [62], en este último caso la red surge como una evolución de la primera y pretende extraer características de imágenes con alta resolución, sin tener que recuperarlas tras haber hecho un paso previo de la imagen a baja resolución, como ocurre en la mayoría de redes existentes.

4.2.1 *Simple Baseline*

La primera red neuronal utilizada en este trabajo sigue un enfoque modular, explicado en la sección 3.1, y se basa en la modificación de la red presentada por Bin Xiao et al. en [61]. Esta red toma como base una ResNet, y le añade tres capas deconvolucionales tras la última capa convolucional. Cada capa deconvolucional esta formada por 256 filtros de 4x4, con un *stride* de 2. Finalmente se utiliza una última capa convolucional de tamaño 1x1 que es la que genera los mapas de calor predichos para cada uno de los *keypoints*. La arquitectura completa de esta red se puede ver en la figura 4.4

La red neuronal esta preparada en primera instancia para detectar 17 puntos del cuerpo de una persona, entre ellos están 6 articulaciones de los brazos y 6 de las piernas más nariz, ojos y orejas.

La arquitectura de la red no va a tener modificaciones debido a que de esta manera ya se cuenta con modelos preentrenados que se podrán utilizar como punto de partida para el entrenamiento de la red, y que se pueden descargar desde la propia página de [61]. Entre ellos se pueden encontrar 6

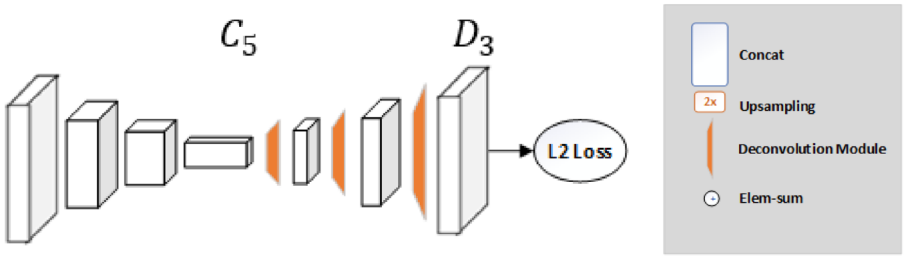


Figura 4.4: Arquietctura de la red *Simple Baseline* empleada para la estimación de pose. Figura obtenida de [61].

preentrenados con ImageNet, basados en Resnet-50, 101 y 152, y para imágenes de entrada de 256x192 y 384x288, reescalandose aquellas imágenes que tengan distintas resoluciones. En este proyecto se han entrenado las dos resoluciones para una ResNet-50. A parte de estos modelos, los autores del artículo también ofrecen unos archivos en formato yaml modificables, que son usados como argumentos de entrada del programa de entrenamiento.

A continuación se detallan las principales características que se pueden modificar en este archivo:

- **GPUs**. Define el número de GPUs que se van a utilizar.
- ***DATASET***. Determina el *dataset* con el que se va a entrenar la red. Está asociado a un programa en python con el mismo nombre en el que también se deben realizar una serie de modificaciones que se explicarán a continuación.
- ***IMAGE SIZE***. Este parámetro define el tamaño al que se va a redimensionar la imagen de entrada.
- ***NUM JOINTS***. Esta característica representa el número de puntos que se quieren detectar con la red, debiendo ser especificados por el usuario.
- ***HEATMAP SIZE***.Representa las dimensiones de los mapas de calor que devuelve la red.
- ***ROTATION FACTOR***. Este factor indica la mitad del número de grados entre los que puede rotar la imagen. Por ejemplo un factor de 40 indicaría que la imagen puede rotar con una probabilidad dada entre ± 80 grados.
- ***FLIP***. Este factor al igual que el anterior sirve para hacer *data augmentation* y produce un intercambio entre puntos simétricos con una probabilidad dada por programación.
- ***SCALE FACTOR***. El factor de escala determina el porcentaje que se reescalan las imágenes, también como herramienta de *data augmentation*. Para un factor de 0.3 las imágenes podrían ser reescaladas, siguiendo una distribución gaussiana, entre un $\pm 30\,\%$.
- ***END EPOCH***. Número de iteraciones que se harán sobre el conjunto total de imágenes.

- ***BATCH SIZE*** Determina el tamaño del *batch* de imágenes que se van a cargar en cada pasada del entrenamiento.

A parte de esto, también el archivo permite la modificación de algunos parámetros de la red, como el *learning rate* o el *momentum* vistos en la sección 2.1.5, o el *weight decay* visto en la sección 2.2.4.2.

4.2.2 HRnet

En segundo lugar, se tiene la arquitectura de [62], que como se dijo anteriormente, esta diseñada para no tener que recuperar las características desde un mapa de baja resolución. La mayoría de arquitecturas diseñadas para estimar la pose de personas tienen una estructura definida como *high-to-low*, es decir, de una imagen en alta resolución se obtiene un mapa de características en baja resolución y con un aumento del número de canales. Por ejemplo, en la red *Stacked Hourglass* [63] se hacen conversiones *high-to-low* y *low-to-high* simétricas. En la red vista en el apartado anterior, se introducen tres capas deconvolucionales a una ResNet tras la última capa convolucional, encargadas de aumentar la resolución de la imagen.

En el caso de HRnet, una subred se encarga de que la imagen de entrada en alta resolución se mantenga durante todo el proceso. A medida que se avanza por la red, se introducen de una en una más subredes *high-to-low* en paralelo con la subred principal, formando una red que comparte información entre distintos tipos de resolución. La arquitectura de esta red queda se puede observar de manera más clara en la figura .

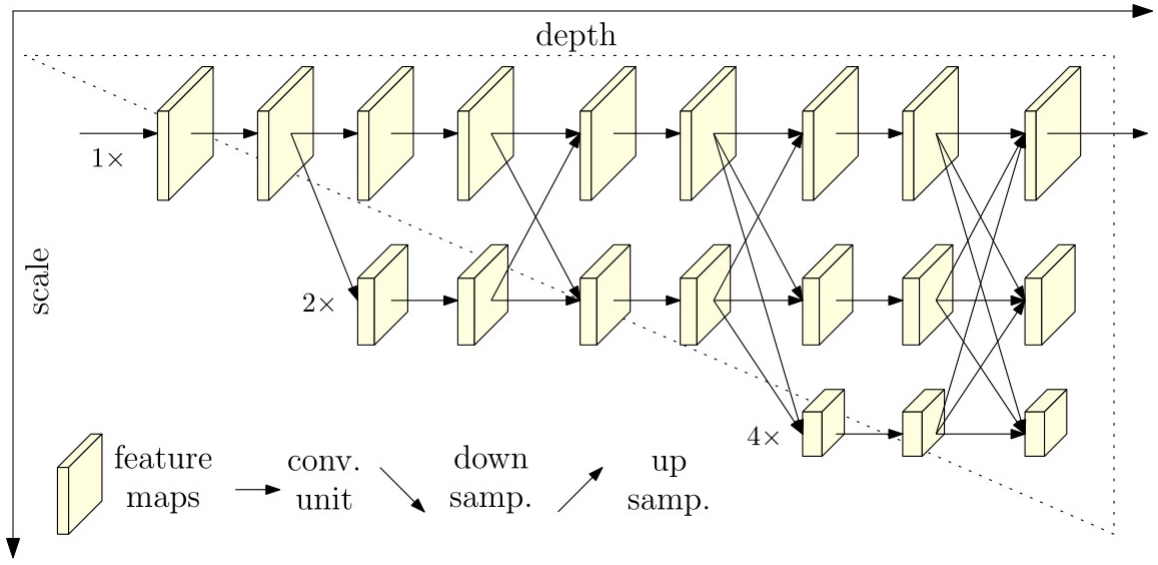


Figura 4.5: Arquitectura de la red HRnet empleada para la estimación de pose. Figura obtenida de [62].

Al igual que en el caso anterior, los autores ponen a disposición online los distintos modelos de la red y los archivos yaml correspondientes para realizar el entrenamiento. Los modelos utilizados serán de nuevo los que se han preentrenado con el *dataset* de ImageNet. En este caso se cuenta con dos modelos distintos, uno en el que la subred de alta resolución cuenta con un total de 32 canales, y otro en el que cuenta con 48 canales.

Para alcanzar estos 32 canales a partir de la imagen de entrada, la primera etapa de la red esta formada por 4 módulos residuales, la unidad básica de la red ResNet presentada en la sección 2.2.5 figura 2.9. La salida de estos módulos tiene un total de 64 canales, por lo que tras la salida del último módulo, se aplica un filtro que reduce el número de canales hasta 32 o 48, dependiendo de la red utilizada.

A partir de aquí se van generando distintas subredes, hasta un total de 4 en paralelo, y para cada subred se reduce la resolución a la mitad y se duplica el número de canales, al igual que se hace en ResNet. Estas reducciones son hechas mediante un filtro convolucional de tamaño 3x3. La red preentrenada cuenta con 4 etapas, añadiendose una subred nueva en cada una de ellas. En cada etapa, para compartir la información entre subredes, se cuenta con lo que ellos denominan *exchange blocks* teniendo la segunda, tercera y cuarta capa un total de 1, 4 y 3 bloques de intercambio respectivamente. Estos bloques de intercambio están formados a su vez por otras cuatro unidades residuales modificadas, a las que se les ha añadido a la salida de las dos capas convoluciones 3x3 que se hacían en ResNet, una *exchange unit* o unidad de intercambio, que es la base del intercambio de información entre capas.

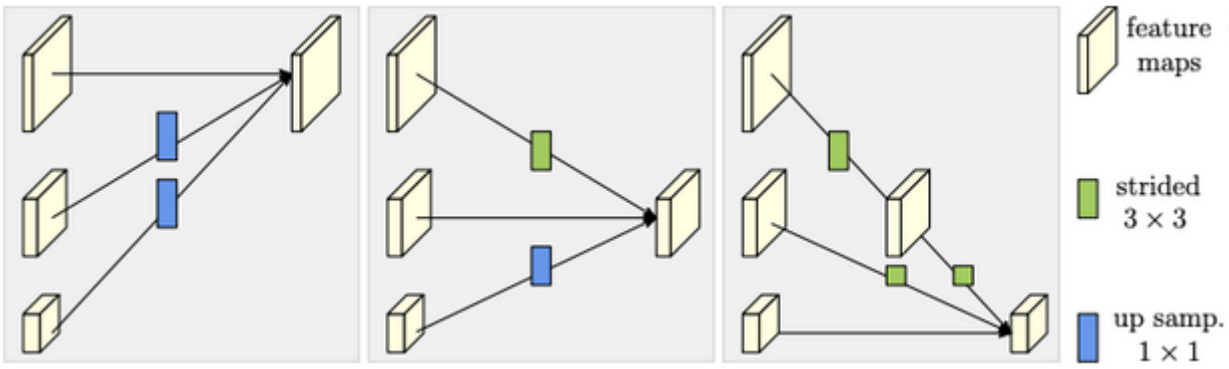


Figura 4.6: Unidad básica de intercambio de HRNet. De izquierda a derecha se muestra un intercambio de información para alta, media y baja resolución. Figura obtenida de [62].

La unidad de intercambio básica se puede ver en la figura 4.6. Aunque en la imagen se muestre separado, cada unidad de intercambio conecta completamente cada mapa de características de una capa con la siguiente, de manera similar a lo que se hace en una capa *fully connected* de una red clásica, solo que en este caso se trabaja con tensores, en lugar de con neuronas independientes. Para adaptar las resoluciones de cada subred se hace un *downsampling* empleando filtros convolucionales 3x3, representados en verde en la figura. En el caso del *upsampling* se emplean filtros deconvolucionales 1x1, mediante el método de *nearest neighbor up-sampling*. Este método se puede entender mejor a partir de la figura

Para esta red, a parte de los parámetros anteriormente configurados, se añade una nueva técnica de *data augmentation*. Esta técnica la denominan consiste en recortar la imagen para solo quedarse con la mitad del cuerpo, y se aplica aleatoriamente como las presentadas anteriormente. Configurando el ***HALF BODY FACTOR*** entre 0 y 1, se modifica la probabilidad de que se aplique esta técnica. La mitad con la que se queda la red se escoge de manera aleatoria, siempre y cuando la mitad elegida tenga más de 2 *keypoints*, de no ser así, se elegirá la otra, quedándose en última instancia con la mitad superior del cuerpo si ninguna cumple este requisito.

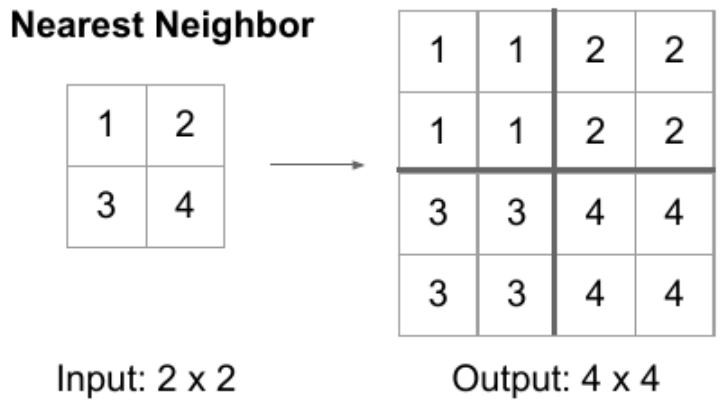


Figura 4.7: Método *nearest neighbor up-sampling*. Figura obtenida de [64].

Por último, tanto para la red de 32 canales como para la de 48, se tienen las mismas que tenia la red *Simple Baseline*, de 256x192 y de 384x288, y se ajustaran automáticamente las imágenes de entradas a estas resoluciones, pudiendo alimentar la red con imágenes de cualquier tamaño.

Habiendo entendido el funcionamiento de ambas redes, se procede a su modificación y adaptación al problema en cuestión.

4.3 Configuración del entrenamiento

Para la realización del trabajo, se han editado los parámetros anteriormente mencionados para cada red, como se indica en las tablas 4.2 y 4.3.

Red Neuronal	***Simple Baseline original 256x192***	***Simple Baseline modificada***
GPUs	'0'	'0'
Image size	256x192	192x256
Heatmap size	64x48	48x64
Num. Joints	17	19
Rotation factor	45	30
Flip	*true*	*true*
Scale factor	0.3	0.3
End Epoch	140	140
Batch size	32	64

Tabla 4.2: Configuración de las redes estudiadas. *Simple Baseline*.

Red Neuronal	**HRNet-32 256x192**	**HRNet-32 modificada**
GPUs	(0,1,2,3)	(0,)
Image size	256x192	192x256
Num. Joints	17	19
Rotation factor	45	30
Flip	*true*	*true*
Scale factor	0.35	0.3
Prob. half body	0.3	0.3
End Epoch	210	210
Batch size	32	32

Tabla 4.3: Configuración de las redes estudiadas. HRNet.

En las tablas 4.2 y 4.3, se puede observar que el parámetro que determina el tamaño del mapa de calor y el tamaño de la imagen de entrada esta invertido con respecto a la red original. Esto es debido a que se quiere encontrar vehículos en la imagen y estos por norma general suelen ocupar mayor espacio horizontal que vertical.

También han tenido que realizarse modificaciones en los archivos de Python correspondientes a los *datasets*, ya que se va a entrenar sobre un conjunto diferente a aquel para el que se ha diseñado la red. En este caso estas modificaciones se hacen sobre los *keypoints*, adaptándolos a los del *dataset* propio, y teniendo que numerarlos en Python siguiendo el mismo orden que el de las anotaciones asociadas a las imágenes. De igual manera se han tenido que que modificar las relaciones de simetría horizontal entre *keypoints* para que cuando se utilice como técnica de *data augmentation* el *flip* o reflexión horizontal, este sea ejecutado de manera correcta. Por último, para el caso de HRNet, se debe indicar que puntos pertenecen a la mitad izquierda del coche y cuales a la derecha, para que se pueda emplear la técnica de *half body*.

En las figuras 4.8 y 4.9 se pueden ver ejemplos de las técnicas de *data augmentation* explicadas.

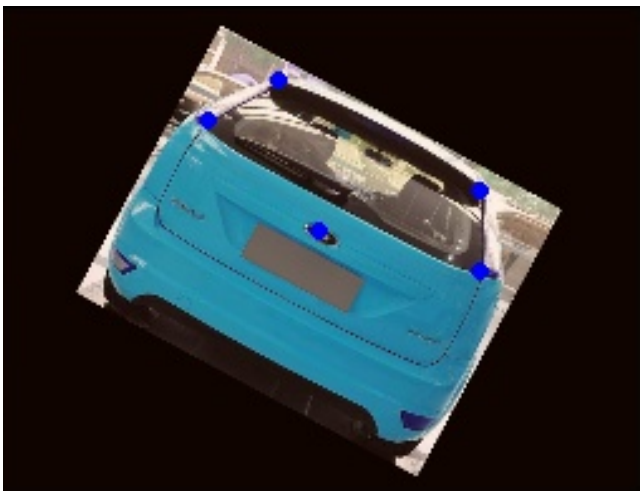


Figura 4.8: Ejemplo de rotación.

Figura 4.9: Ejemplo de *half body*.

A parte de estas modificaciones, se han realizado varios entrenamientos distintos de la red, variando los hiperparámetros como la tasa de aprendizaje y el *momentum* para ver su repercusión en la salida, con la intención de ajustarlos a aquellos que den mejores resultados. En el siguiente capítulo se exponen de manera detallada las variaciones realizadas.

Capítulo 5

Resultados

En este capítulo se van a presentar los resultados obtenidos en los experimentos descritos anteriormente. Además de esto, también se persigue realizar una comparativa entre las distintas configuraciones empleadas de cara a poder extraer las conclusiones apropiadas en el siguiente capítulo.

5.1 Evaluación cuantitativa

A continuación, se van a exponer las modificaciones realizadas sobre los distintos parámetros de la red, viendo el impacto individual que cada parámetro tiene sobre los resultados cuantitativos. Para poder evaluar el comportamiento de cada configuración, se van a utilizar métricas como las pérdidas de la red y la AP.

5.1.1 HRNet o *Simple Baseline*

En primer lugar, se muestra una comparación entre las dos redes estudiadas en este trabajo. En la figura 5.1 se pueden ver las diferencias entre las pérdidas de ambas redes a lo largo de varias *epochs*

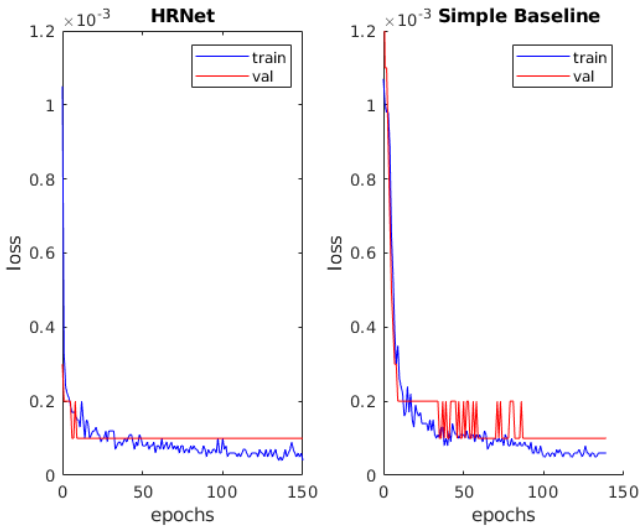


Figura 5.1: Pérdidas de la red HRNet frente a *Simple Baseline*.

En las gráficas se puede ver como HRNet ve reducidas sus pérdidas en un menor número de *epochs* frente a la red *Simple Baseline*. Por otro lado, la AP de HRNet es del 97 %, siendo la de *Simple Baseline* del 96 %. En una primera aproximación, no hay gran diferencia de forma cuantitativa entre ambas redes, sin embargo, para reducir la variabilidad de experimentos, se ha optado por elegir la red HRNet para estudiar las siguientes configuraciones. Igualmente en la sección 5.2 se puede ver las diferencias entre ambas redes a nivel cualitativo.

5.1.2 *Data Augmentation*

Para esta configuración, se ha modificado la red HRNet, prescindiendo de los tipos de *data augmentation* vistos en la sección 4.2, reduciéndose las probabilidades de *flip*, *half body*, *scale*, y *rotation* al 0 %.

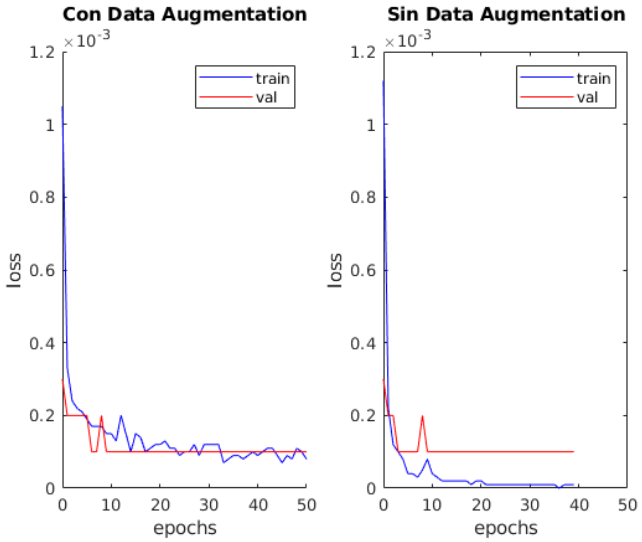


Figura 5.2: Efecto de *Data Augmentation*.

Como se puede observar en la figura 5.2, prescindir de *data augmentation* en la red lleva a que esta quede muy ligada al *dataset* y sobreaprenda, teniendo unas perdidas prácticamente nulas en entrenamiento. La precisión media de la red baja hasta el 93.4 %, lo que demuestra que funciona con menor eficiencia que en el caso en el que si se emplean estas técnicas.

5.1.3 Efecto del *Momentum*

En este caso se ha optado por modificar el *momentum*, aumentándolo de 0.9 a 0.95. Como se explicó en 2.1.5, el *momentum* tiene por objetivo controlar la velocidad de decaimiento de la red, teniendo en cuenta los gradientes previos de la red, y evitando así cambios bruscos. En la figura 5.3 se puede ver el efecto que ha tenido esta modificación en el entrenamiento de la red.

La modificación del *momentum* en este caso no aporta grandes diferencias a la hora de evaluar las pérdidas de la red, siendo la única diferencia una reducción del AP hasta el 95.4 %. En el caso de esta red, que ya funciona de manera adecuada con un *momentum* de 0.9, aumentarlo ha supuesto empeorar la precisión de la red.

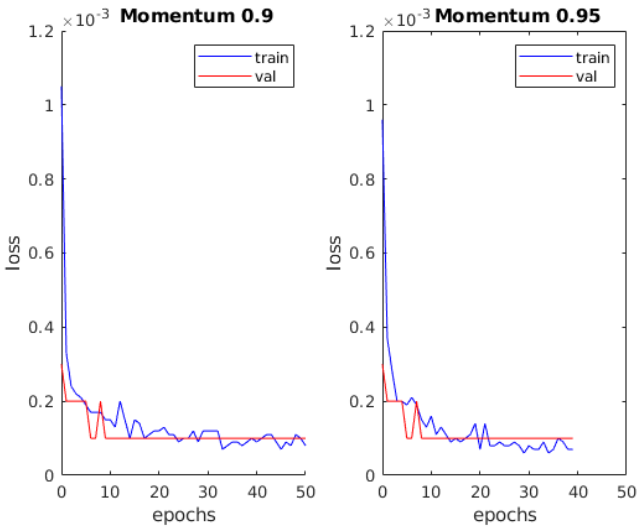


Figura 5.3: Efecto de aumentar el *momentum*.

5.1.4 Modificación de la resolución de entrada y número de canales

Lo siguiente que se ha hecho ha sido modificar por un lado la resolución de las imágenes de entrada de la red y por otro el número de canales de estas, utilizando HRNet 48. En ambos casos el efecto ha sido similar, viéndose reducidas las pérdidas de la red en un menor numero de *epochs* que en casos anteriores, como se puede ver en las figuras 5.4 y 5.5

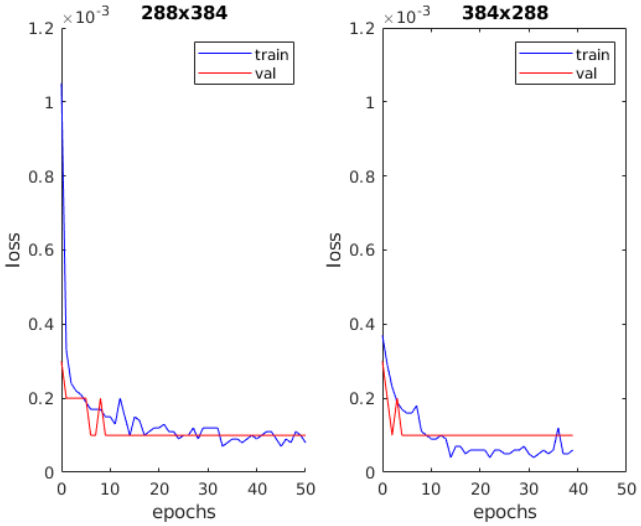


Figura 5.4: Aumento de la resolución de entrada.

En el caso de un aumento de la resolución el AP obtenido es del 97 % y en el caso del aumento del número de canales es del 96.4 %, siendo estos los mejores resultados obtenidos. La desventaja de esto es que al aumentar el tamaño del modelo y de la resolución de entrada de los datos, el *mini batch* utilizado debe ser más pequeño, lo que se traduce en un mayor tiempo de entrenamiento. En el caso de la red básica, el tiempo de entrenamiento aproximado ha sido de 50 minutos, siendo de 1 hora y 22 minutos para la resolución de 384x288 y de 1 hora 3 minutos para la red profunda.

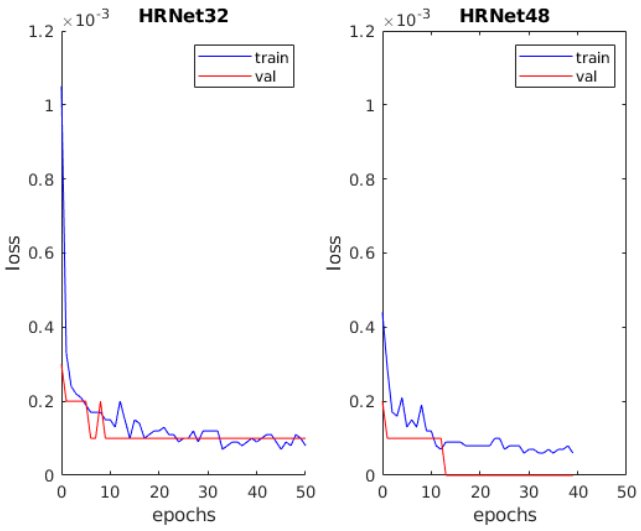


Figura 5.5: HRNet32 frente HRNet48.

5.1.5 Aumento de *learning rate*

Por último, se ha aumentado la tasa de aprendizaje de la red, desde 0.001 hasta 0.005. La tasa de aprendizaje es un parámetro que debe ser modificado con precaución, puesto que una tasa de aprendizaje muy pequeña llevaría a un entrenamiento muy lento, y una demasiado alta provocaría que la red evitase el mínimo de la función, no llegando a aprender nunca.

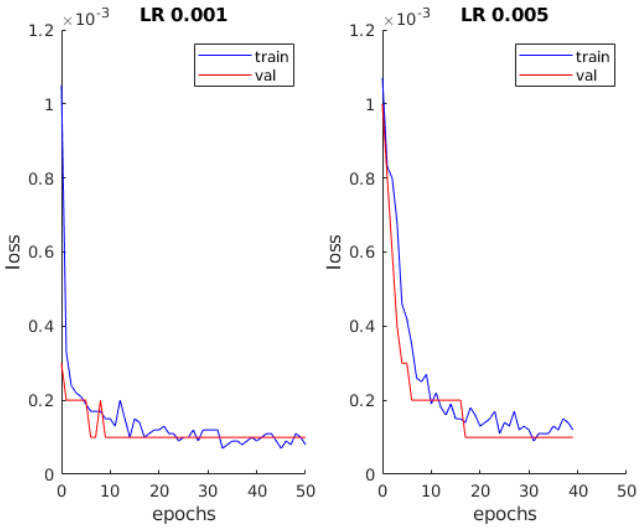


Figura 5.6: Efecto de aumentar el *learning rate*.

La precisión de la red conseguida en este caso ha sido peor a la conseguida con una tasa de aprendizaje más baja, siendo esta de 0.95 %. También se ha probado la red con una tasa de 0.01, no llegando a aprender nada en este caso, y devolviendo al final del entrenamiento una AP del 0 %.

5.2 Evaluación cualitativa

Para evaluar los resultados de la red de manera cualitativa se han comparado los diferentes etiquetados que han devuelto las configuraciones de la sección 5.1 para las imágenes del conjunto de validación.

En primer lugar, al igual que en el apartado anterior, se va a comparar las dos redes estudiadas. En las figuras 5.7 y 5.8 se pueden ver las diferencias entre ambas redes.

Figura 5.7: HRNet.

Figura 5.8: *SimpleBaseline*.

Ambas redes funcionan adecuadamente, sin embargo, hay ciertos puntos, como la rueda trasera o el logo en *Simple Baseline*, que no llegan a estar centrados.

A continuación se muestran resultados del resto de experimentos realizados, sobre una misma imagen de muestra, para poder estimar de manera cualitativa que efectos tiene cada una de las configuraciones descritas en el apartado anterior.

En las figuras 5.9-5.13 se pueden destacar varias cosas. La primera, es que como ya se predijo de a partir de los resultados cuantitativos, prescindir de técnicas de *data augmentation* provoca errores a la hora de etiquetar nuevos datos. Se puede observar en la figura 5.9, que el logo ha sido etiquetado de manera incorrecta.

En el caso de las figuras 5.11 y 5.12, al tener mayor resolución y mayor número de canales respectivamente, el etiquetado es más preciso, pero como ya se comentó, al aumentar el tamaño del modelo y los datos, el entrenamiento se ve ralentizado. Para este caso en una situación real, habría que elegir que es más prioritario, si los resultados obtenidos o el tiempo de entrenamiento.

Por último, en la figura 5.13, se ve un etiquetado incorrecto de la rueda delantera al aumentar la tasa de aprendizaje hasta 0.005. Para el caso de una tasa de aprendizaje de 0.01 no se llega a realizar ningún etiquetado, esto se puede ver en la figura 5.14.

Figura 5.9: Ejemplo de etiquetado sin *data augmentation*.

Figura 5.10: Ejemplo de etiquetado con *momentum* 0.95.

Figura 5.11: Ejemplo de etiquetado con resolución 384x288.

Figura 5.12: Ejemplo de etiquetado con red HRNet48.

Figura 5.13: Ejemplo de etiquetado con *learning rate* 0.005.

Figura 5.14: Ejemplo de etiquetado con *learning rate* 0.01.

Capítulo 6

Conclusiones y trabajo futuro

En este último capítulo se presenta la conclusión a este trabajo y se exponen posibles mejoras y adaptaciones que se podrían investigar en trabajos futuros.

6.1 Conclusiones

En este trabajo se han introducido las Redes Neuronales Convolucionales o CNNs y su aplicación en la resolución de problemas de visión artificial. Se ha mostrado como los sistemas de visión artificial han evolucionado a lo largo de los años, y como la aparición de las CNNs ha supuesto una revolución en esta área de conocimiento, consiguiendo hitos nunca vistos hasta la última década, ya sea en problemas de clasificación, segmentación semántica, detección, etc...

Como se ha podido comprobar durante el desarrollo de este proyecto, durante los últimos años se puede ver una evolución importante en las distintas arquitecturas de CNNs, yendo cada vez hacia modelos más eficientes, lo que las ha llevado a ser un punto de referencia en cualquier tarea presente en el campo de la visión. Estos sistemas se han visto mejorados, siendo cada vez más eficientes y eficaces a la hora de ser entrenados y de ser utilizados en la práctica.

Desde la aparición de AlexNet como red de clasificación, con una arquitectura que cuenta con un total de 8 capas, las Redes Neuronales Convolucionales han ido creciendo en complejidad, llegando a modelos como VGGNet o ResNet, esta última alcanzando en su versión más profunda un total de 152 capas. Además, ResNet ha demostrado ser la opción más utilizada, ya que aunque es una red muy profunda, ha probado ser también muy eficiente, debido a su estructura repetitiva basada en dos bloques constructivos básicos, y a su sistema de aprendizaje residual. Estos modelos de arquitecturas básicas han servido de punto de partida para el desarrollo de otros sistemas que se encargan de resolver otras tareas como detección de objetos o segmentación semántica, entre los que cabe destacar la familia de las R-CNN.

En cuanto a las herramientas software, se ha podido comprobar que existen actualmente múltiples opciones para realizar tareas de *deep learning* y *machine learning*. Como lenguajes de programación principales se pueden destacar C/C++ y Python, siendo este último más utilizado en el mundo de la investigación, y dejando C++ para implementación en sistemas finales. Además, en Python se puede contar con PyTorch, una librería *open source* diseñada para calculo tensorial y *machine*

learning. Además, también se dispone de MatLab, una herramienta perfecta para docencia y que puede permitir enseñar estas nuevas disciplinas en entornos académicos.

En este trabajo se ha planteado un sistema de detección de *keypoints* en objetos estructurados, en concreto se ha centrado en vehículos, y se ha implementado el módulo de detección de puntos significativos para un único objeto por imagen. A este sistema siempre se le podría adaptar un modulo previo como puede ser YOLO, encargado de detectar múltiples vehículos en una imagen, y adaptarlo para que la salida de este sirva como entrada al sistema planteado en este proyecto.

Al tratarse de un problema de *deep learning*, en el que se requiere de una gran cantidad de datos para entrenar la red, el primer obstáculo que se ha tenido que afrontar es la búsqueda de un *dataset* que se adapte a las necesidades del problema a resolver. En este caso, al ser un problema tan concreto y personalizado, esta tarea se vuelve aún más difícil, por lo que la mejor solución ha sido buscar un *dataset* con un pre-etiquetado que solo necesite pequeñas modificaciones.

La tarea de etiquetar un *dataset*, por pequeño que este sea, ha requerido de una gran parte del tiempo de este proyecto, por lo que el siguiente paso es hacer uso del *transfer learning* y encontrar una red neuronal pre-entrenada para un problema similar que se pueda adaptar. Para el caso de este proyecto se han encontrado dos redes neuronales entrenadas para la detección de *keypoints* en personas y para la estimación de pose de las mismas, además, estas redes cuentan con un código fácilmente adaptable al número de puntos que se quieran encontrar y ya viene preparada para aceptar cualquier tamaño de imagen de entrada. La modificación de los hiperparámetros de la red como pueden ser la tasa de aprendizaje y el *momentum* también se pueden modificar de manera sencilla, lo que permite realizar varios aprendizajes y comprobar el efecto de estas modificaciones.

Como se ha podido comprobar, la modificación de estos parámetros no es intuitiva, teniendo que realizar varias pruebas sobre ellos para conseguir mejorar los resultados de la red. En el caso del *momentum* no se han obtenido diferencias significativas en el rendimiento de la red. Un ligero aumento del *learning rate* ha hecho que la red se ralentice mínimamente, sin mejorar los resultados, y un aumento mayor ha provocado que la red no pueda aprender. Si se opta por reducir la tasa de aprendizaje, la red se verá igualmente ralentizada, y la mejora que puede aportar es muy leve, teniendo un 97 % de AP en el modelo básico, lo que hace que esta opción sea descartada.

Por otra parte, esta red cuenta con tres tipos de *data augmentation*, siendo estos la rotación, escalado e inversión de las imágenes, modificables de la misma manera que los demás parámetros, y que son ejecutados aleatóriamente y de manera individual para cada imagen. Estas técnicas como se ha estudiado sirven como mecanismo regularizador y permite que la red vea reducida su dependencia con el *dataset*, mejorando de esta forma su capacidad de generalización. En el caso de este proyecto, emplear estos métodos es altamente recomendable, debido a que el *dataset* es relativamente pequeño comparado con los volúmenes de datos que suelen manejar las CNNs. Al prescindir del *data augmentation* se ha visto como la red sobreaprende y da unos resultados mucho peores que el resto de configuraciones.

En el caso de esta red se ha utilizado el AP exclusivamente para comparar los resultados entre las distintas configuraciones, ya que su cálculo se realiza utilizando lo que en COCO llaman *Object Keypoint Similarity (OKS)*. Este OKS es un parámetro similar al *Intersection Over Union (IoU)*, pero en lugar de medir las similitudes entre *bounding box* real y predicha, mide la similitud entre los

puntos etiquetados y los devueltos por la red. Esto sirve para estimar si la red se ha confundido o no a la hora de realizar un etiquetado y por tanto para calcular la precisión media de la red.

El OKS de cada punto en este caso esta preparado para personas, no para vehículos, y las personas pueden adoptar mayor variabilidad de poses, pudiendo quedar la posición relativa entre *keypoints* de maneras muy distintas en cada imagen. En el caso de los vehículos, al ser una estructura rígida, hay menor variabilidad en estas poses, pudiendo cambiar ligeramente entre distintos modelos. Esto podría explicar las precisiones tan altas de la red, ya que toma como correctos etiquetados que no son tan precisos como deberían, lo cual lleva a que cualquier comparación con el estado del arte actual no pueda considerarse real. Igualmente a día de hoy no existen demasiados *datasets* enfocados en el marcado de *keypoints* en vehículos, pudiendo destacar el trabajo realizado por OpenPose, con una precisión del 70.1 % de AP o el *dataset* de la universidad *Carnegie Mellon*, aunque este último esta pensado para realizar, a parte de detección de *keypoints*, seguimiento de trayectorias.

Aun así, se ha podido comprobar de manera cualitativa que la red entrenada en este proyecto funciona de manera adecuada, detectando los *keypoints* de manera precisa.

6.2 Trabajo Futuro

Se plantean las siguientes mejoras sobre este detector de *keypoints* como posibles trabajos futuros.

- Implementación del sistema *end-to-end*, consiguiendo que detecte con una única red todos los vehículos en la imagen.
- Adaptar la red para que, mediante los puntos detectados, dibuje la estructura del vehículo, conociendo así la pose que este tiene en la carretera.
- Estudiar nuevas redes, como la presentada en [65], en la que se detectan todos los vehículos de la imagen con una sola red y se hace uso de los PAFs para unir los puntos que pertenezcan a la misma estructura.
- Implementar un sistema de tiempo real que pueda instalarse en un vehículo autónomo, capturando imágenes mediante la cámara y detectando la posición y orientación de todos los coches que hay a su alrededor.
- Ampliación del *dataset* propio, introduciendo un mayor número de imágenes con otro tipo de vehículos a parte de los turismos.
- Entrenar un sistema con el *dataset* ampliado de manera que pueda detectar la mayor parte de los vehículos presentes en las carreteras.

Apéndice A

Pliego de condiciones

Para la ejecución de los distintos programas desarrollados en el presente trabajo para el entrenamiento, validación y prueba del sistema, serán necesarios los siguientes elementos:

- Hardware:
  - Estación de trabajo PC Intel Core i7-920 @ 2.67 GHz x 8; 12 GB de RAM, o similar. Se necesita una CPU multihilo y con potencia suficiente para la carga de datos en las CNNs y la ejecución de todos los procesos necesarios. El uso de RAM también es intensivo pero se puede subsanar haciendo uso de memoria de *swap*. Es recomendable el uso de una unidad *Solid State Drive (SSD)*, dada la alta cantidad de imágenes que se tienen que mover de disco a GPU. De realizarse desde un *Hard Drive Disks (HDDs)* se vería considerablemente ralentizado el entrenamiento de la red.
  - GPU de alto rendimiento igual o superior a NVIDIA GTX 1080 Ti con 11 GB de memoria GDDR5X dedicada. Para poder cargar modelos de gran tamaño, como los tratados en este trabajo, es necesario contar con suficiente memoria dedicada en la GPU. En el caso de este trabajo se ha utilizado una GTX 2080 Ti con 11GB de memoria dedicada.
  - Cámaras para la obtención de imágenes con las que poder crear un *dataset* propio y con las que poder validar el modelo.
- Software:
  - Sistema Operativo GNU/Linux Kubuntu x64 o compatible.
  - Python 3.6 con las dependencias necesarias.
  - OpenCV compilado con el módulo Python cv2.
  - MATLAB
  - Drivers NVIDIA.
  - CUDA 8 y cuDNN. Será necesario CUDA10 para tarjetas NVIDIA de la serie 2000.
  - Framework de *deep learning* PyTorch[1].

[1] Instrucciones de instalación en http://pytorch.org/

Apéndice B

Presupuesto

En este apartado se especifica el presupuesto para el desarrollo del presente Trabajo de Fin de Máster desglosado en Presupuesto de Ejecución Material, Coste de Ejecución por Contrata y Presupuesto Total.

B.1 Presupuesto de Ejecución Material (PEM)

El presupuesto de ejecución material (PEM) se calcula teniendo en cuenta los costes del material informático y de personal.

B.1.1 Coste del material informático

En la tabla B.1 se pueden ver los costes del *hardware* necesario para el desarrollo del trabajo.

Concepto	Precio	Amortización	Meses de uso	Total
PC Intel Core i7-920 @ 2.67 GHz x 8; 12 GB RAM	1000,00 €	3 años	4 meses	111,00 €
NVIDIA GeForce GTX 2080 Ti	1200,00 €	3 años	4 meses	133,00 €
Total				244,00 €

Tabla B.1: Presupuesto de coste del material informático.

B.1.2 Coste de personal

En la tabla B.3 se pueden ver los costes del personal necesario para el desarrollo del trabajo desglosados en salario del personal y trabajo imputable, que viene expresada en horas de trabajo o páginas en el caso de los trabajos de documentación . En la tabla B.2 se pueden ver desglosadas las horas de trabajo.

Duración	Hito
75 horas	Revisión bibliográfica del estado del arte.
50 horas	Revisión de los distintos frameworks disponibles.
25 horas	Despliegue del sistema.
100 horas	Implementación del sistema de detección de *keypoints*, búsqueda de bases de datos, etiquetado y obtención de imágenes.
50 horas	Evaluación y análisis de los resultados obtenidos, conclusión del trabajo y redacción de la memoria.

Tabla B.2: Desglose de las horas de trabajo.

Concepto	Cuantía imputable	Salario	Total
Ingeniería	300 horas	30,00 €/hora	9000,00 €
Documentación	130 páginas	1,20 €/página	156,00 €
Total			9156,00 €

Tabla B.3: Presupuesto de coste de personal.

B.1.3 Presupuesto de Ejecución Material total

En la tabla B.4 se muestra el coste total del proyecto calculado mediante la suma del coste de material informático y el coste de personal.

Concepto	Total
Coste del material informático	244,00 €
Coste de personal	9156,00 €
Total	9400,00 €

Tabla B.4: Presupuesto de Ejecución Material total.

B.2 Coste de Ejecución por Contrata (CEC)

El presupuesto de Coste de Ejecución por Contrata (CEC) consta del coste de Ejecución Material anteriormente calculado, los gastos generales, el beneficio industrial y los honorarios de redacción y dirección que se aplican sobre el valor. El resultado se puede ver en la tabla B.5.

Concepto	Valor (% relativo al PEM)	Total
PEM	100 %	12395,00 €
Gastos generales y beneficio industrial	22 %	2726,90 €
Honorarios de redacción	7 %	867,65 €
Honorarios de dirección	7 %	867,65 €
Total	136 %	16857,20 €

Tabla B.5: Presupuesto de Coste de Ejecución por Contrata.

B.3 Presupuesto Total

El presupuesto total se calcula añadiéndole al Coste de Ejecución por Contrata (CEC) el Impuesto sobre el Valor Añadido (IVA) aplicable. El presupuesto total se puede ver en la tabla B.6

Concepto	Valor (% relativo al CEC)	Total
CEC	100 %	16857,20 €
IVA	21 %	3540,01 €
Total		20397,21 €

Tabla B.6: Presupuesto Total.

El presupuesto total del proyecto asciende a la cantidad de veinte mil trescientos noventa y siete euros con veintiún céntimos (20397,21 €).

En Alcalá de Henares, a 26 de septiembre de 2019.

Fdo. Antonio Hernández Martínez.

Bibliografía

[1] K. Fukushima y S. Miyake, "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition," *Biological Cybernetics*, vol. 36, no. 4, pp. 267–285, 1980. [Online]. Disponible en: https://doi.org/10.1007/BF00344251

[2] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, Nov 2004. [Online]. Disponible en: https://doi.org/10.1023/B:VISI.0000029664.99615.94

[3] N. Dalal y B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1. IEEE, 2005, pp. 886–893. [Online]. Disponible en: http://ieeexplore.ieee.org/abstract/document/1467360/

[4] H. Bay, T. Tuytelaars, y L. Van Gool, "Surf: Speeded up robust features," pp. 404–417, 2006.

[5] R. Bayot y T. Gonçalves, "A Survey on Object Classification using Convolutional Neural Networks." [Online]. Disponible en: https://dspace.uevora.pt/rdpc/handle/10174/17508

[6] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, y L. Fei-Fei, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, Dec 2015. [Online]. Disponible en: https://doi.org/10.1007/s11263-015-0816-y

[7] J. Schmidhuber, "Deep learning in neural networks: An overview," *CoRR*, vol. abs/1404.7828, 2014. [Online]. Disponible en: http://arxiv.org/abs/1404.7828

[8] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, y L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, Dec 1989.

[9] "Laanlabs," (Último acceso 17/09/2019). [Online]. Disponible en: https://medium.com/@laanlabs/real-time-3d-car-pose-estimation-trained-on-synthetic-data-5fa4a2c16634

[10] J. Redmon, S. Divvala, R. Girshick, y A. Farhadi, "You only look once: Unified, real-time object detection," June 2016.

[11] (Último acceso 17/09/2019). [Online]. Disponible en: https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088

[12] A. Krizhevsky, I. Sutskever, y G. E. Hinton, "Imagenet classification with deep convolutional neural networks," pp. 1097–1105, 2012. [Online]. Disponible en: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks

[13] (Último acceso 17/09/2019). [Online]. Disponible en: https://www.freecodecamp.org/news/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050/

[14] "CS231n Convolutional Neural Networks for Visual Recognition." [Online]. Disponible en: http://cs231n.github.io/convolutional-networks/#pool

[15] J. T. Springenberg, A. Dosovitskiy, T. Brox, y M. A. Riedmiller, "Striving for simplicity: The all convolutional net," *CoRR*, vol. abs/1412.6806, 2014. [Online]. Disponible en: http://arxiv.org/abs/1412.6806

[16] M. A. Nielsen, "Neural Networks and Deep Learning," 2015. [Online]. Disponible en: http://neuralnetworksanddeeplearning.com/chap2.html

[17] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, y G. Wang, "Recent advances in convolutional neural networks," *CoRR*, vol. abs/1512.07108, 2015. [Online]. Disponible en: http://arxiv.org/abs/1512.07108

[18] S. R. K. Sarvadevabhatla y R. Babu, "A taxonomy of deep convolutional neural nets for computer vision," *Frontiers in Robotics and AI*, vol. 2, 01 2016.

[19] J. Ngiam, Z. Chen, D. Chia, P. W. Koh, Q. V. Le, y A. Y. Ng, "Tiled convolutional neural networks," pp. 1279–1287, 2010. [Online]. Disponible en: http://papers.nips.cc/paper/4136-tiled-convolutional-neural-networks.pdf

[20] M. D. Zeiler y R. Fergus, "Visualizing and understanding convolutional networks," *CoRR*, vol. abs/1311.2901, 2013. [Online]. Disponible en: http://arxiv.org/abs/1311.2901

[21] J. Long, E. Shelhamer, y T. Darrell, "Fully convolutional networks for semantic segmentation," *CoRR*, vol. abs/1411.4038, 2014. [Online]. Disponible en: http://arxiv.org/abs/1411.4038

[22] H. Noh, S. Hong, y B. Han, "Learning deconvolution network for semantic segmentation," *CoRR*, vol. abs/1505.04366, 2015. [Online]. Disponible en: http://arxiv.org/abs/1505.04366

[23] F. Yu y V. Koltun, "Multi-scale context aggregation by dilated convolutions," *CoRR*, vol. abs/1511.07122, 2015. [Online]. Disponible en: http://arxiv.org/abs/1511.07122

[24] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, y A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *CoRR*, vol. abs/1606.00915, 2016. [Online]. Disponible en: http://arxiv.org/abs/1606.00915

[25] P. Wang, P. Chen, Y. Yuan, D. Liu, Z. Huang, X. Hou, y G. W. Cottrell, "Understanding convolution for semantic segmentation," *CoRR*, vol. abs/1702.08502, 2017. [Online]. Disponible en: http://arxiv.org/abs/1702.08502

[26] M. Lin, Q. Chen, y S. Yan, "Network in network," *CoRR*, vol. abs/1312.4400, 2013. [Online]. Disponible en: http://arxiv.org/abs/1312.4400

[27] M. D. Zeiler y R. Fergus, "Stochastic pooling for regularization of deep convolutional neural networks," *CoRR*, vol. abs/1301.3557, 2013. [Online]. Disponible en: http://arxiv.org/abs/1301.3557

[28] D. Yu, H. Wang, P. Chen, y Z. Wei, "Mixed pooling for convolutional neural networks," in *Rough Sets and Knowledge Technology*, D. Miao, W. Pedrycz, D. Ślzak, G. Peters, Q. Hu, y R. Wang, Eds. Cham: Springer International Publishing, 2014, pp. 364–375.

[29] O. Rippel, J. Snoek, y R. P. Adams, "Spectral Representations for Convolutional Neural Networks," jun 2015. [Online]. Disponible en: http://arxiv.org/abs/1506.03767

[30] K. He, X. Zhang, S. Ren, y J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *CoRR*, vol. abs/1406.4729, 2014. [Online]. Disponible en: http://arxiv.org/abs/1406.4729

[31] ——, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *CoRR*, vol. abs/1502.01852, 2015. [Online]. Disponible en: http://arxiv.org/abs/1502.01852

[32] B. Xu, N. Wang, T. Chen, y M. Li, "Empirical evaluation of rectified activations in convolutional network," *CoRR*, vol. abs/1505.00853, 2015. [Online]. Disponible en: http://arxiv.org/abs/1505.00853

[33] D. Clevert, T. Unterthiner, y S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *CoRR*, vol. abs/1511.07289, 2015. [Online]. Disponible en: http://arxiv.org/abs/1511.07289

[34] I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, y Y. Bengio, "Maxout networks," in *Proceedings of the 30th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, S. Dasgupta y D. McAllester, Eds., vol. 28, no. 3. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 1319–1327. [Online]. Disponible en: http://proceedings.mlr.press/v28/goodfellow13.html

[35] J. T. Springenberg y M. Riedmiller, "Improving Deep Neural Networks with Probabilistic Maxout Units," dec 2013. [Online]. Disponible en: http://arxiv.org/abs/1312.6116

[36] S. Srinivas, R. K. Sarvadevabhatla, K. R. Mopuri, N. Prabhu, S. S. S. Kruthiventi, y R. V. Babu, "A Taxonomy of Deep Convolutional Neural Nets for Computer Vision," *Frontiers in Robotics and AI*, vol. 2, p. 36, jan 2016. [Online]. Disponible en: http://journal.frontiersin.org/Article/10.3389/frobt.2015.00036/abstract

[37] K. Simonyan, A. Vedaldi, y A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," *CoRR*, vol. abs/1312.6034, 2013. [Online]. Disponible en: http://arxiv.org/abs/1312.6034

[38] J. Tompson, R. Goroshin, A. Jain, Y. LeCun, y C. Bregler, "Efficient object localization using convolutional networks," *CoRR*, vol. abs/1411.4280, 2014. [Online]. Disponible en: http://arxiv.org/abs/1411.4280

[39] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, y R. Fergus, "Regularization of neural networks using dropconnect," vol. 28, no. 3, pp. 1058–1066, 17–19 Jun 2013. [Online]. Disponible en: http://proceedings.mlr.press/v28/wan13.html

[40] X. Glorot y Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," vol. 9, pp. 249–256, 13–15 May 2010. [Online]. Disponible en: http://proceedings.mlr.press/v9/glorot10a.html

[41] K. He, X. Zhang, S. Ren, y J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Disponible en: http://arxiv.org/abs/1512.03385

[42] S. Ioffe y C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015. [Online]. Disponible en: http://arxiv.org/abs/1502.03167

[43] K. Simonyan y A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014. [Online]. Disponible en: http://arxiv.org/abs/1409.1556

[44] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, y A. Rabinovich, "Going deeper with convolutions," *CoRR*, vol. abs/1409.4842, 2014. [Online]. Disponible en: http://arxiv.org/abs/1409.4842

[45] R. B. Girshick, J. Donahue, T. Darrell, y J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *CoRR*, vol. abs/1311.2524, 2013. [Online]. Disponible en: http://arxiv.org/abs/1311.2524

[46] R. B. Girshick, "Fast R-CNN," *CoRR*, vol. abs/1504.08083, 2015. [Online]. Disponible en: http://arxiv.org/abs/1504.08083

[47] S. Ren, K. He, R. B. Girshick, y J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," *CoRR*, vol. abs/1506.01497, 2015. [Online]. Disponible en: http://arxiv.org/abs/1506.01497

[48] K. He, G. Gkioxari, P. Dollár, y R. B. Girshick, "Mask R-CNN," *CoRR*, vol. abs/1703.06870, 2017. [Online]. Disponible en: http://arxiv.org/abs/1703.06870

[49] T. Pohlen, A. Hermans, M. Mathias, y B. Leibe, "Full-resolution residual networks for semantic segmentation in street scenes," *CoRR*, vol. abs/1611.08323, 2016. [Online]. Disponible en: http://arxiv.org/abs/1611.08323

[50] T. Mikolov, M. Karafiát, L. Burget, J. Černocký, y S. Khudanpur, "Recurrent neural network based language model," *INTERSPEECH-2010*, 2010. [Online]. Disponible en: https://www.isca-speech.org/archive/archive_papers/interspeech_2010/i10_1045.pdf

[51] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, y Y. Bengio, "Generative adversarial nets," pp. 2672–2680, 2014. [Online]. Disponible en: http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

[52] (Último acceso 17/09/2019). [Online]. Disponible en: https://skymind.ai/wiki/generative-adversarial-network-gan.

[53] "TensorFlow." [Online]. Disponible en: http://www.tensorflow.org/

[54] "Keras." [Online]. Disponible en: https://keras.io

[55] "MATLAB." [Online]. Disponible en: https://es.mathworks.com/help/nnet/index.html

[56] "PyTorch." [Online]. Disponible en: https://pytorch.org/

[57] Z. Cao, G. Hidalgo, T. Simon, S. Wei, y Y. Sheikh, "Openpose: Realtime multi-person 2d pose estimation using part affinity fields," *CoRR*, vol. abs/1812.08008, 2018. [Online]. Disponible en: http://arxiv.org/abs/1812.08008

[58] Y. Xiang, R. Mottaghi, y S. Savarese, "Beyond pascal: A benchmark for 3d object detection in the wild," in *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2014.

[59] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, y A. Zisserman, "The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results," http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[60] L. Yang, P. Luo, C. C. Loy, y X. Tang, "A large-scale car dataset for fine-grained categorization and verification," *CoRR*, vol. abs/1506.08959, 2015. [Online]. Disponible en: http://arxiv.org/abs/1506.08959

[61] B. Xiao, H. Wu, y Y. Wei, "Simple baselines for human pose estimation and tracking," in *European Conference on Computer Vision (ECCV)*, 2018.

[62] K. Sun, B. Xiao, D. Liu, y J. Wang, "Deep high-resolution representation learning for human pose estimation," *CoRR*, vol. abs/1902.09212, 2019. [Online]. Disponible en: http://arxiv.org/abs/1902.09212

[63] A. Newell, K. Yang, y J. Deng, "Stacked hourglass networks for human pose estimation," *CoRR*, vol. abs/1603.06937, 2016. [Online]. Disponible en: http://arxiv.org/abs/1603.06937

[64] (Último acceso 17/09/2019). [Online]. Disponible en: http://www.programmersought.com/article/709640128/

[65] S. Kreiss, L. Bertoni, y A. Alahi, "Pifpaf: Composite fields for human pose estimation," *CoRR*, vol. abs/1903.06593, 2019. [Online]. Disponible en: http://arxiv.org/abs/1903.06593

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá