Trabajo Fin de Grado
Grado en Ingeniería de Tecnologías Industriales

Detección Automática de Puntos Característicos en el Sistema de Alcantarillado basado en Redes Neuronales Convolucionales

Autor: José Gómez Dugo

Tutor: Fernando Caballero Benítez

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Trabajo Fin de Grado
Grado en Ingeniería de Tecnologías Industriales

Detección Automática de Puntos Característicos en el Sistema de Alcantarillado basado en Redes Neuronales Convolucionales

Autor:
José Gómez Dugo

Tutor:
Fernando Caballero Benítez
Profesor Titular

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Trabajo Fin de Grado: Detección Automática de Puntos Característicos en el Sistema de Alcantarillado basado en Redes Neuronales Convolucionales

Autor: José Gómez Dugo
Tutor: Fernando Caballero Benítez

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

En primer lugar agradecer a Fernando Caballero, que sin ninguna duda ha sido el gran responsable de que pueda estar entregando este trabajo. Sin su confianza, ánimos y conocimientos pongo en duda que este trabajo se hubiera finalizado. Muchas gracias por estar ahí estos años.

Por supuesto, agradecer a mis padres, porque sin su apoyo y cariño yo no podría haber tenido las posibilidades que tengo. Mis éxitos están basados en ellos.

También quiero mencionar a mi hermana, muchas gracias por apoyarme cuando me ha hecho falta.

Y por último, a mi futura esposa Laura Isabel, muchas gracias por todo. Es imposible describir aquí lo importante que eres y lo mucho que me has ayudado para que esto sea posible.

José Gómez Dugo
Sevilla, 2019

Resumen

Desarrollo de un sistema de inteligencia artificial basado en redes neuronales convolucionales (CNNs) para detectar puntos característicos en redes de saneamiento publica. El desarrollo incluye el procesamiento de las bases de datos necesarias, el diseño de las CNNs y el ajuste de los parámetros. Como resultado final se desea obtener un clasificador robusto y con alta precisión que permita su uso sin necesidad de recurrir a imágenes tridimensionales.

Abstract

Development of an artificial intelligent system based on convolutional neural networks (CNNs) in order to detect key points in the sewer net. The development includes processing data bases, design of CNNs and tuninig the parameters. Is expected as final result a robust and high accuracy classifier, which operates independently from depth images.

Índice

Notación

SIAR	Sewer Inspection Autonomous Robot
ROS	Robot Operative System
OpenCV	Open Computer Vision
W	Matriz de pesos
x	Entradas del sistema
b	Ordenada en el origen
CNN	Convolutional Neural Network
$tanh$	Tangente Hiperbólica
SGD	Stochastic Gradient Descent

1 Introducción

El campo de la robótica lleva en continuo crecimiento desde la creación de los primeros robots industriales en la década de los 60. El desarrollo de las tecnologías ha posibilitado el crecimiento de esta industria, ampliando cada vez más el área de aplicación de los robots, lo que hace que cada vez encontremos más soluciones basadas en la robótica para problemas en múltiples campos. Esto ha conllevado que se reduzcan los costes de producción, aumente el rendimiento de los procesos industriales y se eviten muchos trabajos monótonos, repetitivos o peligrosos. El trabajo aquí descrito se desarrolla en el marco de un proyecto cuyo objeto es precisamente mejorar las condiciones en el proceso de inspección de los sistemas de alcantarillado.

1.1 Marco del proyecto

La Escuela de Ingeniería de la Universidad Pablo de Olavide y el Departamento de Ingeniería de Sistemas y Automática de la Universidad de Sevilla han trabajado en el proyecto SIAR. Este proyecto consiste en el desarrollo de un robot que es capaz de inspeccionar el sistema de alcantarillado de una ciudad, así como la monitorización de agua, aire y sedimentos del mismo. Para ello, se emplea un robot de seis ruedas específicamente diseñado para este cometido, equipado con cámaras RGBD como sensor principal que, junto a otros sensores y sistemas, permiten al robot desarrollar el trabajo de forma autónoma, así como ser operado desde la superficie por un trabajador.

Para conseguir que el robot realice la tarea de forma autónoma es necesario que tenga un correcto sistema de posicionamiento, que le permita saber en todo momento dónde se encuentra y hacia donde tiene que dirigirse. Una de las piezas que forma este sistema de posicionamiento es de lo que trata este trabajo.

1.2 Motivación del proyecto

Los sistemas de localización de los robots son una parte muy interesante y actual en el desarrollo de los mismos. Es clave para los robots autónomos que tengan una correcta percepción de dónde se encuentran para poder tomar las decisiones correctas. Es un campo complejo donde confluyen distintas tecnologías, siendo el uso de la visión artificial una de las más extendidos por su versatilidad y resultados. La explosión en los últimos años en el uso de redes neuronales convolucionales para la clasificación de imágenes ha favorecido al uso de la visión artificial, ya que con el número suficiente de imágenes para el proceso de entrenamiento podemos obtener clasificadores muy potentes con tiempos de clasificación muy breves.

El robot SIAR expuesto en la sección 3.1 ayudará a mejorar el proceso de inspección de los sistemas de alcantarillado, y a mejorar las condiciones de trabajo de los operarios que hacen actualmente este trabajo. El poder participar en la elaboración de un proyecto que incide positivamente en

la sociedad dota a este trabajo de una motivación extra. Nos encontramos, por tanto, ante un reto tecnológico de actualidad cuya resolución satisfactoria puede incidir directamente en mejorar de las condiciones de trabajo la calidad de estos trabajadores.

1.3 Objetivos

En este Trabajo Fin de Grado se desarrolla un clasificador binario que se integra dentro del sistema de localización del robot y que permite determinar si se encuentra bajo una alcantarilla o no, gracias a la información obtenida por la cámara instalada sobre el robot. La utilidad de este detector reside en que permite al robot conocer su posición dentro del mapa de alcantarillado que previamente conoce. El robot va realizando una odometría durante la inspección, que conlleva la acumulación de errores a lo largo del tiempo. Cuando localiza que se encuentra bajo una tapa de alcantarilla, es capaz de volver a posicionarse de forma absoluta, eliminando el error acumulado hasta ese punto.

Para este clasificador se empleará una Red Neuronal Convolucional que recibirá como entrada imágenes del techo del sistema de alcantarillado, obteniendo como salida si es o no una tapa de alcantarilla lo que se encuentra sobre el robot. El grado de confianza en esta clasificación debe ser muy alto, ya que es una parte clave en el sistema de posicionamiento del robot.

2 Clasificación de imágenes

En este capítulo vamos a hablar del problema de la clasificación, centrándonos en la clasificación de imágenes, y explicando cual es el enfoque realizado por el aprendizaje automático para este propósito.

2.1 Introducción al problema de la clasificación

El problema de la clasificación se encuentra dentro del campo del reconocimiento de patrones. Digamos que la clasificación es una aplicación directa de dicho campo. ¿En qué consiste, entonces, reconocer patrones en un conjunto de datos? Lo podemos definir como la búsqueda automática, realizada por máquinas, por medio de algoritmos para encontrar características comunes en un conjunto de datos. La clasificación se realiza de manera que dado un conjunto de datos, se asigna a cada muestra de dicho conjunto una categoría. El motivo de que estos dos conceptos estén tan relacionados es que, de forma habitual, se aplica el reconocimiento de patrones a un conjunto de datos para ser capaces de clasificar estos datos en distintas categorías.

Dar una solución para el problema de la clasificación es de aplicación en múltiples campos del conocimiento y de la industria, siendo algunos ejemplos reales de esto:

- Reconocimiento de dígitos: cada imagen es un dígito del 0 al 9, y queremos que el algoritmo de clasificación asigne a cada imagen una de las 10 categorías distintas. Los usos de esto son incontables: identificación de los portales de las casas en las calles, lectura de contadores de agua, luz gas por medio de fotografías, digitalización de escritos a mano, etc.
- Análisis de sentimiento: es el análisis de textos para determinar información subjetiva contenida en ellos. Un ejemplo clásico es el de analizar las reseñas de hoteles, restaurantes, comercios... y clasificarlas en buenas, malas o neutrales.
- Reconocimiento automático de voz: consiste en ser capaz de reconocer y pasar a texto el lenguaje hablado.

Existen muchísimos más ejemplos, pero con lo expuesto se obtiene una idea de la utilidad y vigencia de este campo.

2.2 Clasificación de imágenes

La clasificación de imágenes es una parte clave dentro de los problemas de clasificación, así como del campo de la visión artificial. Consiste en asignar a cada imagen de un conjunto de datos concreto una categoría, por lo que a la entrada de nuestro clasificador tendremos una imagen, y a la salida un valor que corresponde a una clase.

Figura 2.1 Proceso de clasificación.

La identificación de elementos en una imagen, que es algo trivial para el ser humano, se convierte en una tarea que puede ser muy complicada para una máquina. Las imágenes en los ordenadores se guardan como matrices numéricas. Las imágenes a color necesitan de 3 matrices para formar el color de cada píxel de la imagen. Pongamos por ejemplo el caso de una imagen de 100 píxeles de ancho por 100 píxeles de alto y a color:

$$N_{pixel} = 100 \times 100 \times 3 = 30000$$

Tendremos por tanto 30000 valores numéricos distintos para una imagen relativamente pequeña, para acabar asignándole una sola categoría.

Otra dificultad añadida a la clasificación de imágenes es que, para una misma categoría de clasificación, las imágenes a nivel de píxel pueden ser muy distintas entre sí, debido a:

- La variación del punto de vista: un mismo objeto puede ser fotografiados desde múltiples orientaciones y distancias, pudiendo dar lugar a imágenes muy distintas entre sí.
- Las condiciones de iluminación: una escena puede estar iluminada de múltiples maneras, obteniendo valores de brillo y color muy distintos.
- Las deformaciones: hay elementos o seres vivos que pueden tomar formas distintas.
- La variación de tamaño: mismos elementos pueden tener tamaños muy distintos y pertenecer a la misma categoría.
- La variación dentro de una clase: podemos definir categorías formadas por elementos muy distintos entre sí, pero que semánticamente sean lo mismo.

Esta es una lista no exhaustiva, si bien, nos ayuda a entender la dimensión del problema que encaramos, teniendo que desarrollar técnicas de clasificación complejas para adaptarse a este problema.

2.3 Aproximación por aprendizaje automático

El aprendizaje automático consiste en el uso de algoritmos que puedan aprender de los datos para realizar predicciones. Es un campo muy amplio, en uso desde hace muchos años, y muy vigente en la actualidad por el uso masivo de las redes neuronales, contenidas dentro de este campo. Esto no significa que el aprendizaje automático no se usara antes de las redes neuronales, aunque es indudable que han relanzado el uso de estas técnicas.

Para encarar el problema de la clasificación de imágenes, resulta muy útil el enfoque del aprendizaje automático, cuya idea principal es usar los datos que tenemos, junto a los que puedan ser generados posteriormente, para entrenar algoritmos que sean capaces de realizar predicciones. Esto se adapta perfectamente a nuestro problema de clasificación, en el que vamos a tener imágenes como conjunto de datos, y las predicciones serán las categorías que asignemos a estas imágenes.

Nuestro objetivo es desarrollar un sistema que permita clasificar las imágenes en categorías. Para ello, se utilizará un modelo, cuya entrada consistirá en imágenes y cuya salida serán etiquetas que se asignarán a cada imagen. Estas etiquetas pertenecen a un conjunto previamente conocido. Nuestro trabajo consiste en encontrar el modelo que mejor prediga estas etiquetas. Esta búsqueda del modelo consiste en un proceso iterativo, en el cual usaremos datos para entrenar a este modelo y datos para evaluar al modelo. El modelo resultante dependerá, por tanto, de la calidad de los datos y del correcto método de entrenamiento.

Pongamos un caso simple en el que nuestro problema lo podemos modelar de la siguiente forma:

$$f(x) = Wx_i + b$$

Tenemos un modelo lineal, en el que hemos almacenado los píxeles de la imagen i en una columna x_i, siendo la matriz W y el vector b parámetros de la función. Nuestro objetivo es encontrar los valores de W y b para nuestro problema concreto.

Como hemos dicho, xi son los píxeles de la imagen, por lo que el vector x_i tamaño $1 \times p$. W pondera los valores de x_i, y tendrá tamaño $k \times p$, siendo k el número de categorías en los que se podrá clasificar la imagen. b, por tanto, es tamaño $k \times 1$ al igual que el vector resultante de la función. Cada elemento del vector de salida se interpreta como la puntuación que ha obtenido cada categoría para esa imagen en concreto.

Función de pérdida

La pérdida cuantifica el desempeño del modelo en el proceso de clasificación. De manera que, cuanto peor es la clasificación, y si la pérdida es nula, implica que el modelo se adapta perfectamente a las entradas que se han evaluado. Para cuantificar la pérdida, necesitamos una función que evalúe las predicciones del modelo, comparándolas con la etiquetas de las entradas que ha predicho. Hay muchos algoritmos para calcular la pérdida, y dependerá del problema concreto el uso de uno u otro.

Optimización de la función de pérdida

Con la función de pérdida sabemos si nuestro modelo se comporta bien con nuestro problema, pero el objetivo no es sólo medir el desempeño, si no modificar nuestro modelo para que mejore, y para ello tenemos que modificar nuestro parámetros en W, es decir, queremos encontrar W que minimice la función de pérdida. En el ejemplo del uso de las support-vector machines (SVMs), tenemos que la función de pérdida es convexa, lo que hace que sea posible su optimización, pudiendo llegar a dar con el mínimo de la función. Esto no será tan inmediato para el caso de las redes neuronales,

cuestión que se abordará en detalle durante el presente trabajo.

El enfoque que realiza el aprendizaje automático para el proceso de entrenamiento de nuestro modelo es iterativo. El motivo fundamental de esto es porque es la mejor manera para aprovechar el potencial computacional de las máquinas, y porque con ejemplos sencillos es posible dar con soluciones de forma analítica, pero para ejemplos más complejos se convierte en una tarea complicada. Por ello, la técnica que se utilizará es el descenso por gradiente. Básicamente, en cada iteración calcularemos el gradiente, y actualizaremos los pesos siguiendo la dirección negativa del gradiente, que es la que nos llevará al mínimo de la función.

Cómputo del gradiente

Queremos calcular el gradiente de nuestra función. El gradiente nos da la información de la dirección hacia la que la función decrece. Si seguimos dicha dirección, daremos con los pesos del modelo que harán que la función sea mínima.

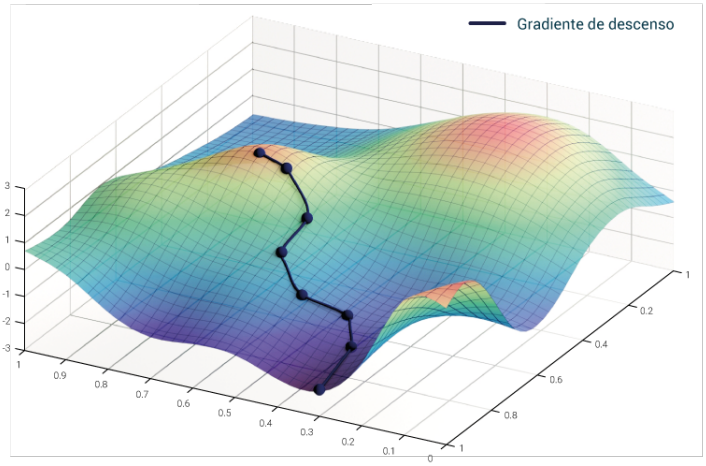

Figura 2.2 Minimización por descenso de gradiente.

Para computar dicho gradiente usamos las diferencias finitas, que es un método numérico que permite convertir un problema de ecuaciones diferenciales en un problema de ecuaciones en diferencias. Este es el enfoque más usado para la resolución numérica de ecuaciones diferenciales.

Para computar el gradiente, tomemos la definición de derivada para un función f(x)

$$\frac{df(x)}{dx} = lim_{h\to 0}\frac{f(x+h)-f(x)}{h}$$

Esto nos da la pendiente en un punto de la función. En nuestro caso, x es un vector de magnitudes, y el gradiente es el vector de derivadas parciales para cada dimensión. Para computar la fórmula anterior para cada valor de x, necesitamos dar un valor a h. En la formulación matemática, la derivada está definida en el límite cuando h tiende a 0, pero para computar dicho límite necesitamos un valor de h suficientemente pequeño.

El resultado de este cálculo nos da a dirección de actualización de los parámetros, pero no sabemos a priori el módulo del vector. Este parámetro, que en aprendizaje automático se denomina tasa de aprendizaje, es clave para el entrenamiento de algoritmo. Tasas de aprendizaje demasiado pequeñas

dan lugar a modelos que aprenden muy lento, precisando de un tiempo y coste computacional elevados. Por otra parte, seleccionar una tasa de aprendizaje elevada puede desembocar en la no convergencia del modelo, por lo que la pérdida será mayor. La tasa de aprendizaje es uno de los parámetros de calibración de nuestro modelo.

Computar el gradiente sobre todos los datos en cada iteración puede ser una operación muy costosa, ya que podemos tener millones de ejemplos por iteración. La técnica más usada para afrontar este problema es usar el descenso de gradiente estocástico de minilote. Un minilote es un subconjunto de nuestros datos, típicamente de entre 100 y 1000. Lo que se hace es calcular el gradiente sobre este minilote en cada iteración, reduciendo de forma drástica el costo computacional, y obteniendo buenos resultados.

Una vez obtenido el gradiente de la función, actualizamos los parámetros de nuestro modelo multiplicándolos por el gradiente y por la tasa de aprendizaje, y eso ocurre en cada iteración. Esto hará que, en un modelo correcto, la pérdida vaya disminuyendo a lo largo de las iteraciones.

Generalización del modelo

Hemos visto que conforme avanza el entrenamiento del modelo, el resultado de la función de pérdida se reduce. Este proceso es iterativo y, a priori, no tiene un final establecido, por lo que parece que podríamos iterar indefinidamente, hasta que el valor de dicha función de pérdida se estabilizase o se hiciese cero.

Pongamos un ejemplo de clasificación binario. Los datos representados abajo son los datos que poseemos para entrenar nuestro modelo. Tras un número suficiente de iteraciones, conseguimos un modelo que separa las dos clases perfectamente sobre nuestros datos de entrenamiento. Obtenemos, por tanto, una pérdida de 0 y una precisión en la clasificación perfecta.Con nuestro modelo entrenado, obtenemos posteriormente nuevos datos, y los representamos junto a los anteriores y junto al modelo.

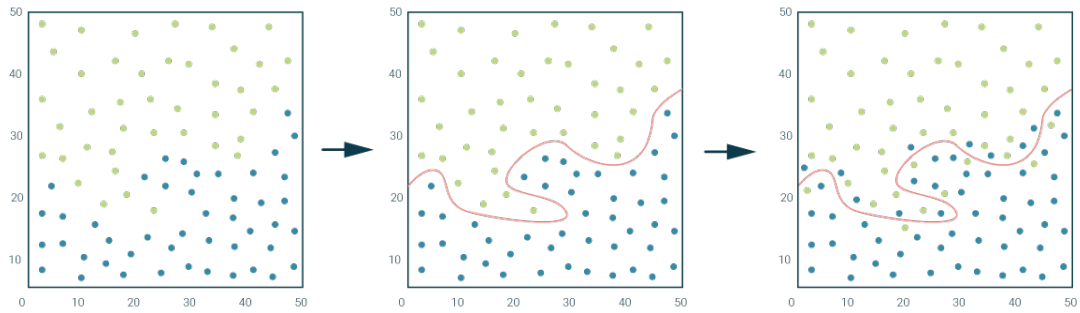

Figura 2.3 Sobreajuste del modelo a los datos.

Podemos apreciar que el modelo se ajusta mal a los nuevos datos obtenidos. A este problema se le conoce como sobreajueste, y se interpreta como que nuestro modelo se ha ajustado a los datos de entrenamiento demasiado, produciendo como efecto que no se adapte bien a los datos reales del problema. Esto puede deberse a varios motivos:

- Una base de datos demasiado pequeña: si los datos de entrenamiento no son suficientes, es normal que el modelo se adapte bien a dichos datos, pero la poca cantidad de estos hace que sea bastante probable que falle estrepitosamente cuando se añadan más datos.
- Una variedad de datos insuficiente: similar motivo en el fondo al anterior. Aunque tengamos muchos datos, si no son lo suficientemente variados no conseguirán un modelo robusto.

- Modelo muy complejo: si el modelo tiene un grado de complejidad superior al necesario, es capaz de conseguir resultados muy ajustados a los datos pero poco representativos de la realidad global. Este concepto es el que se maneja cuando usamos polinomios de muchos grados para realizar la regresión en un conjunto de puntos, consiguiendo que nuestro polinomio pase por todos los puntos, pero al añadir datos nuevos falla de manera ostensible. En cambio, con una simple recta obtenemos un resultado mucho más fiel al problema.
- Sobreentrenar el modelo: siguiendo el concepto de la sección anterior, si iteramos demasiado sobre los datos de entrenamiento, podemos tener como resultado un modelo demasiado ajustado a estos datos.

Si el sobreajuste es debido a la poca cantidad o calidad de los datos es difícil poder obtener buenos resultados. Hay técnicas para solventar la escasez de datos, pero siempre será una limitación a la hora de conseguir modelos robustos.

Si tenemos una cantidad de datos suficiente, existen técnicas para evitar el sobreajuste, denominadas generalización del modelo. Generalizar el modelo consiste en aplicar técnicas que evitan en el proceso de entrenamiento que el modelo se ajuste demasiado bien a los datos de entrenamiento, para posibilitar de esta forma que se ajuste mejor a los nuevos datos con los que no entrenamos. Son técnicas que por lo general reducen la precisión de la fase de entrenamiento del modelo, consiguiendo modelos más simples que abstraen mejor las características clave de los datos, evitando ajustarse demasiado a características menos generales y más concretas, o al ruido existente en los propios datos.

Gestión de los datos

Es esencial poder probar el modelo en datos distintos a los usados para entrenar el modelo. En la sección anterior hemos visto los problemas del sobreajuste, y de cómo para percatarse de ello es necesario probar en nuevos datos. Estos datos nuevos son simplemente datos no usados para entrenar, por lo que una técnica básica en los algoritmos de aprendizaje automático es dividir nuestro conjunto de datos en datos de entrenamiento y datos de prueba. Con esta división generamos 2 bases de datos:

- Datos de entrenamiento: deben suponer la mayoría de los datos que tenemos a disposición, en general en torno al 90 % de los datos disponibles. Esto puede variar si disponemos de una gran cantidad de datos. Es necesario que esta porción sea lo suficientemente grande, ya que de su variedad y cantidad ajustaremos los parámetros de nuestro modelo.
- Datos de prueba: usaremos esta porción de los datos para probar con las métricas pertinentes el desempeño del modelo, y así determinar si se avanza en la dirección correcta en el proceso de entrenamiento.

Es importante evitar ciertos errores típicos en la gestión de los datos. El primero consiste en ajustar demasiado al conjunto de prueba, es decir, se realizan los ajustes necesarios para que nuestro modelo se comporte muy bien en el conjunto de prueba, pudiendo en un extremo sobreajustar el modelo a estos datos. Una forma de evitar este problema es dividir los datos de prueba en dos subconjuntos, uno lo usaremos para validar durante el entrenamiento, y el otro lo usaremos al final del proceso, para comprobar que efectivamente el modelo se comporta de forma correcta. El segundo error, muy frecuente y grave, es usar datos de entrenamiento para la validación. Esto tiene como efecto que nos encontramos ante modelos con gran precisión desde tempranas fases del entrenamiento, dando lugar a equívocos en la validación. Hay que evitar a toda costa tener este error, por lo que si en fases tempranas del desarrollo nos encontramos con unos resultados demasiado buenos, hay que revisar de nuevo los datos de los subconjuntos.

Extracción de features

En cualquier problema de clasificación, el primer pilar sobre el que se sustenta el proceso de elección y diseño del algoritmo de clasificación es definir el problema que estamos intentando resolver, es decir, qué es lo que queremos clasificar. Para responder dicha pregunta es importante conocer los datos que alimentarán nuestro algoritmo y que tendrán que ser clasificados en la categoría correspondiente. Es tarea de quién diseña el clasificador la elección del formato de los datos que alimentarán dicho clasificador. Ejemplifiquemos el proceso primero con un caso que no sea de clasificación de imágenes.

Pongamos que queremos realizar un clasificador que decida si los comentarios realizados sobre restaurantes por los clientes son positivos, negativos, o neutrales. La base de datos consiste en los textos completos escritos por dichos clientes. ¿Cuál debe de ser la entrada del algoritmo? ¿El texto completo? ¿Quitamos ciertas palabras, como preposiciones, artículos, etc.? La respuesta siempre será casuística, pero con un poco de análisis podemos filtrar estos datos para que el clasificador pueda procesarlos mejor. Por ejemplo, Los adjetivos parecen palabras más interesantes para decidir si una opinión es positiva o negativa, en cambio, un artículo no parece que aporte mucha información. Este proceso de análisis, filtrado y procesado de los datos es lo que se denomina ingeniería de atributos. Los motivos por los que se realiza son:

- Imposibilidad de usar lo datos: hay ocasiones en que las bases de datos que se tienen no son procesables por el algoritmo si no se realiza la ingeniería de atributos. Por ejemplo, en general los algoritmo necesitan de números, ya sea en vectores o matrices, para poder procesar los datos. En el ejemplo anterior se podría asignar a los adjetivos valores numéricos.
- Aumentar la calidad de los datos de entrada: si adaptamos los datos en bruto a datos cuya información es más representativa del problema, posibilitamos que el algoritmos de clasificador sea capaz de extraer la información mejor. Si no, es posible que no pueda llegar a abstraer la información.
- Mejorar el tiempo de procesamiento: si los datos de entrada del algoritmo vienen filtrados, la cantidad de datos procesar se disminuye, y por lo tanto disminuimos los tiempos de computación.

En el caso de las imágenes, la ingeniería de datos es uno de los campos clásicos de investigación. En gran parte de los algoritmos de clasificación de imágenes no es viable introducir directamente los píxeles de la imagen como valores de entrada, siendo necesario hacer una extracción de características de la imagen para que estos algoritmos pudieran tener un buen desempeño. Hay multitud de filtros y técnicas para para extraer información útil de las imágenes, y su aplicación depende de las imágenes que se quieren clasificar.

Un caso clásico es el del reconocimiento de dígitos o letras. Supongamos que tenemos una base de datos de dígitos extraídos de fotografías. Para este problema, se podría hacer la imagen binaria, es decir, que sus píxeles solo puedan tomar dos valores distintos, suponiendo un valor para el traza del dígito, y otro para el fondo. Con la imagen binarizada podemos: calcular el porcentaje de píxeles de trazo respecto a píxeles de fondo, el perímetro, el momento, la relación alto/ancho, etc. Todos estos valores numéricos serían la entrada del clasificador.

Salida como probabilidad

En la sección anterior se ha tratado el formato de la entrada de datos, y en esta sección trataremos la salida del algoritmo. Conceptualmente, lo que queremos es que nuestro clasificador categorice cada elemento de entrada. Si es un clasificador de animales, quiero que la salida sea que animal es; si es un reconocedor de dígitos, la salida quiero que sea una un valor entre 0-9. Pero los clasificadores no

van a ofrecer exactamente este formato de salida.

Al igual que para las entradas del clasificador, para las salidas vamos a tener valores numéricos. La más sencillo sería asignar a cada posible categoría un valor, y que el resultado de salida fuera uno de dichos valores. Pero en dicho caso estaríamos perdiendo una información muy útil que es intrínseca al proceso de clasificación, y es la incertidumbre de la clasificación. Cada entrada del algoritmo de clasificación va a tener una salida en una categoría, y también un grado de confianza respecto a esa predicción.

Los valores de salida de los clasificadores pueden ser muy diversos, y para su correcta interpretación es interesante adaptarlo a un valor entre 0 y 1, que sea interpretable como la probabilidad de que dicha predicción es correcta.

3 Redes neuronales convolucionales

En este capitulo se explicará en qué consiste una red neuronal convolucional, su estructura y como configurarla.

3.1 Redes neuronales: el concepto

Como su propio nombre indica, las redes neuronales artificiales deben su nombre a la analogía existente con las redes neuronales cerebrales. Esta analogía parte del concepto inicial con en que se desarrollaron dichas redes, y tiene su origen en la neurona.

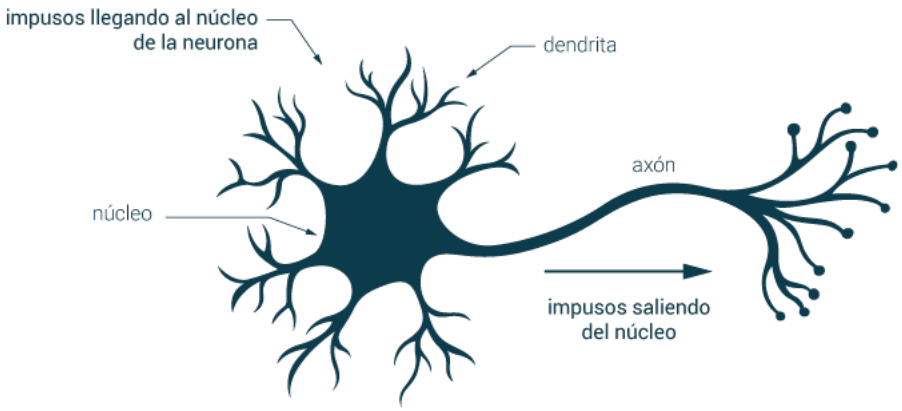

Figura 3.1 Representación de una neurona.

De una forma muy simple, podemos definir una neurona como una célula principal del sistema nervioso encargada de recibir, procesar y emitir impulsos químicos y eléctricos. Las neuronas están conectadas entre sí y todas reciben, procesan y emiten información de unas a otras. En la figura 3.1 vemos la representación de una neurona, que está formada esencialmente de dendritas, que es por donde se reciben los impulsos de otras neuronas, el núcleo, que es el encargado de integrar los impulsos que recibe y generar otros nuevos, que los emite a través del axón, que es la parte que se conecta a otras neuronas.

De la misma forma, podemos ahora definir un nodo computacional como el de la figura 3.2.

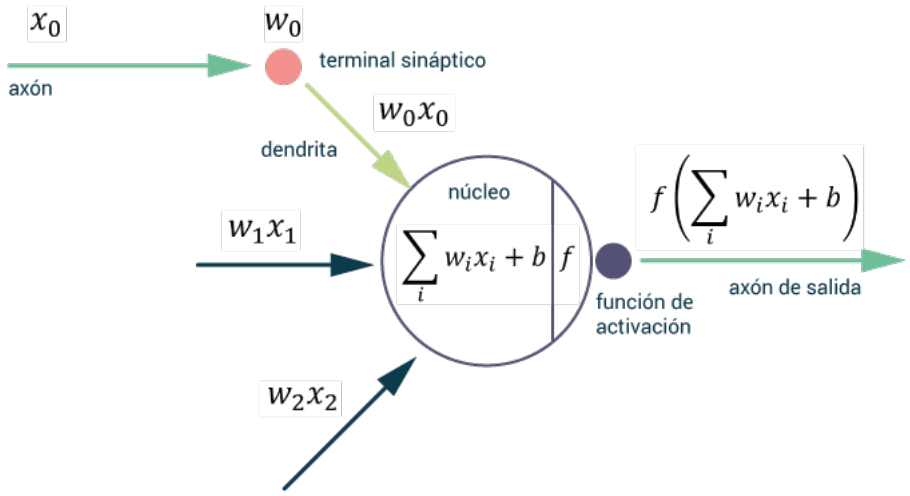

Figura 3.2 Nodo computacional.

En este caso las señales de entrada serían las distintas x_i, que se integran multiplicándolas por un determinado w_i y sumándoles un parámetro b. La salida de esta computación se para por una determinada función de activación, que determina dependiendo del resultado de la operación si este nodo se ha activado con la entrada recibida o no. Esta salida es emitida y será la entrada de otro nodo conectado a la red. Este nodo computacional tiene el nombre de perceptrón, y podríamos decir que es la unidad básica de una red neuronal, que básicamente se encarga de recibir una gran cantidad de entradas, procesarlos, y por medio de una determinada función de activación, propagar ese impulso al resto de la red.

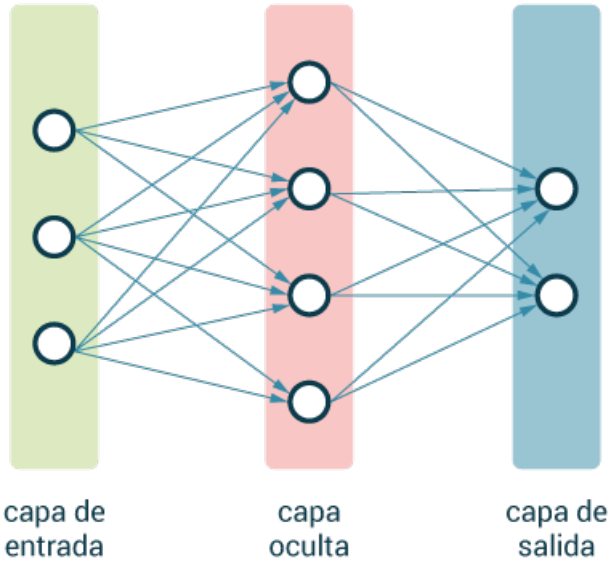

Figura 3.3 Esquema de red neuronal.

La red se compone por tanto de distintas capas formadas por estos perceptrones. Cada capa puede tener conectados todos los perceptrones de la capa anterior con los de la capa siguiente, formando

así lo que se conoce como capas totalmente conectadas. En este tipo de redes tendremos una serie de entradas, que se conectan a una o varias capas ocultas, y que acaban conectándose a una última capa, la capa de salida. La capa de salida debe de contener información sobre la entrada, dependiendo cual sea el propósito de clasificación de la red.

Hay que señalar que la analogía con las neuronas biológicas hay que tratarla con cuidado, ya que el proceso realizado por los algoritmos que se detallarán más adelante no corresponden directamente con lo que ocurre realmente a nivel biológico. Más bien hay que tomarlo simplemente como una inspiración, ya que recientes investigaciones sugieren que hay muchos tipos de neuronas, que realizan operaciones complejas no lineales y que no es por tanto directamente asimilable a los algoritmos que enmarcamos bajo el nombre de redes neuronales.

3.1.1 Estructura y funcionamiento de las redes neuronales

Una red neuronal artificial es un algoritmo encajado dentro del campo del aprendizaje automático que consiste en un conjunto de neuronal artificiales, o perceptrones, organizados en distintas capas y conectadas entre sí, que a una entrada dada, van transmitiendo la información a través de la estructura de la red para dar una salida. Estas capas se van modificando durante el proceso iterativo de entrenamiento para poder mejorar su capacidad de predicción.

Una red neuronal simple de dos capas puede definirse de la siguiente forma:

$$f = W_2 max(0, W_1 x)$$

Esta sería una red de 2 capas que es suficiente para explicar los conceptos básicos de las redes neuronales.

Como entrada tendremos un vector de datos x de tamaño $1 \times n$. Este vector está compuesto de valores numéricos que suponen una muestra de lo que sea que queramos clasificar con nuestra red. En el caso concreto de clasificación de imágenes, en este vector agruparíamos todos los píxeles de la imagen. El vector de entrada es multiplicado por la matriz W de tamaño $n \times m$, que es una matriz de pesos. Estos pesos son parámetros configurables que irán cambiando conforme la red neuronal vaya entrenándose. La salida de esta operación es pasada por una función de activación, que es este caso consiste en el máximo entre 0 y el valor resultante de la multiplicación con la primera capa, es decir, si el valor es negativo se pone a 0, y si es positivo lo conservamos. El siguiente paso es multiplicar el vector resultante y de tamaño $1 \times m$ por la matriz W_2 de tamaño $m \times q$. Obtendremos un vector de salida que tendrá q filas, cada una de ellas debería consistir en una categoría de clasificación distinta.

A priori parece fácil de entender el funcionamiento, simplemente definimos una estructura de matrices que acaban en un vector con el número de categorías. Cada uno de los valores del vector se interpreta como la puntuación de esa muestra para cada clase. Ahora bien, debemos plantearnos varias cuestiones: ¿Qué valores tienen las matrices W_1 y W_2? ¿Cómo se consigue ir mejorando el desempeño de la red con los datos de entrenamiento? Como expusimos en la sección dedicada al aprendizaje automático, tendremos que realizar un proceso iterativo de entrenamiento para poder mejorar las predicciones de la red. Para ello suponemos disponer de un conjunto de datos previamente categorizados y así podremos evaluar cuál es el desempeño actual de la red, con el objetivo de mejorar los resultados.

Para evaluar si ha predicho correctamente la red la categoría de una muestra, necesitamos una función de pérdida. Esta función toma las predicciones realizadas sobre la muestra en concreto, y sabiendo cual es la categoría real de dicha muestra, puntúa la predicción. A mayor valor tenga

se obtenga de la función de pérdida, peor será el desempeño de la red. Por lo tanto, el objetivo principal del entrenamiento es minimizar la función de pérdida. Teóricamente, conforme iteremos usando las muestras de nuestra base de datos de entrenamiento, la función de pérdida debería ir disminuyendo, y si esto no ocurre, implica que la red no está siendo capaz de aprender, es decir, de adaptar sus parámetros para que la pérdida (el error) disminuya.

El objetivo por tanto es obtener el gradiente de la función de pérdida, para así poder modificar nuestros parámetros en el sentido decreciente del gradiente y minimizar nuestra función. Esta tarea para redes neuronales complejas de varias capas es matemáticamente complicada, por lo que se usa la llamada técnica de la retropropagación, que consiste en la aplicación recursiva de la regla de la cadena a lo largo de la red para computar los gradientes de cada uno de las funciones locales de la red. Esto permite que, una vez obtenido el resultado de la red para una muestra y evaluar su pérdida, podamos usar dicho valor para ir actualizando los pesos de la red. A mayor pérdida, más cambiará el valor de los pesos. Esto se repite en cada iteración, minimizando de esta forma la función de pérdida.

3.2 Redes convolucionales: la solución para la clasificación de imágenes

Hemos visto como formar una red neuronal en la que los elementos de entrada consistían en un vector con todos los datos. En un problema de clasificación de imágenes, los datos de entrada consisten en imágenes, que pueden ser almacenadas y representadas como matrices del tamaño que sea la imagen, con una determinada profundidad, que son los canales de la imagen; por ejemplo, para las imágenes a color convencionales constan de tres canales para poder representar el color.

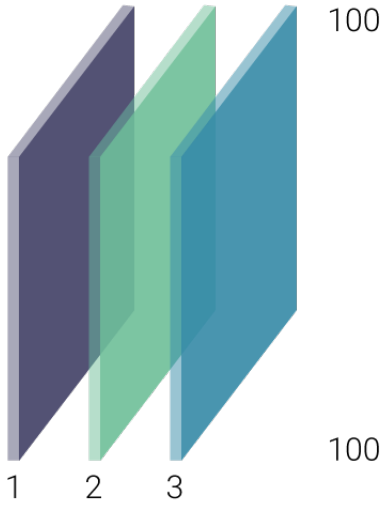

Figura 3.4 Imagen de 3 canales.

Para poder adaptar dicha imagen a la entrada de una red como la de la sección anterior, deberíamos estrechar todos los píxeles a un vector. Por ejemplo, para la anterior imagen de $100 \times 100 \times 3$ tendríamos un vector resultante de 30000 filas. El problema de esta transformación es que perdemos la información espacial de la imagen, y a priori parece razonable pensar que esta información puede ser muy importante para poder identificar características en la imagen. La solución a esto consiste en usar una red con capas convolucionales.

3.2.1 Las capas convolucionales

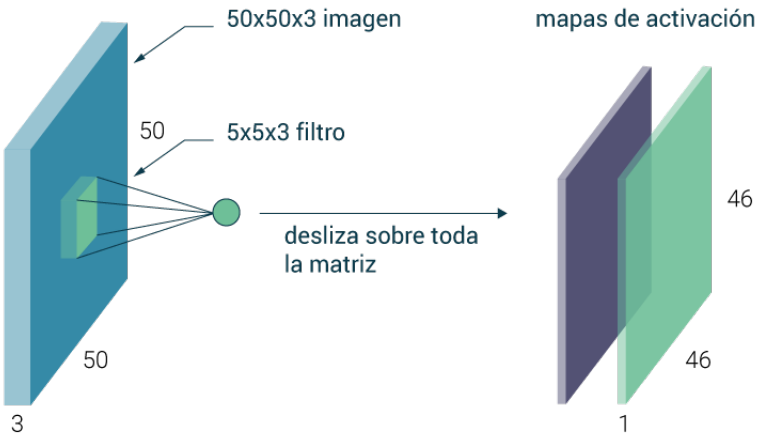

Figura 3.5 Capas convolucionales.

El funcionamiento de estas capas es el siguiente:

- Usamos un filtro de un tamaño determinado. Por ejemplo, en nuestro caso un filtro de $5 \times 5 \times 3$. Este filtro se "desliza" por toda la imagen realizando el producto escalar para cada posición de la imagen. Matemáticamente es como si redujéramos el filtro a un vector para realizar dicho producto escalar.
- Si deslizamos por toda la imagen haciendo el producto escalar con un filtro, obtenemos un mapa de activación que conserva la información espacial de la imagen.
- Podemos usar varios filtros sobre la imagen, obteniendo por cada uno de ellos un mapa de activación.
- Al igual que en las redes neuronales no convolucionales, tras cada conjunto de mapas de activación usaremos una función de activación. A esta secuencia de capas convolucionales con funciones de activación entre ella es lo que denominamos una red convolucional.
- En este tipo de arquitectura de red, los pesos que se entrenan son los que se encuentran en los filtros.

Como se habrá podido apreciar en la imagen, los mapas de activación reducen su tamaño respecto al anterior. Para el ejemplo de antes, hemos supuesto además que deslizamos el filtro de píxel en píxel, lo que se conoce como un paso de 1. Pero también podríamos ir pasando en filtro de 2 en 2 píxeles, como dando un salto un píxel por cada movimiento. Dependiendo de cómo se realice esta operación, tenemos que el tamaño de salida del mapa de activación se corresponde con la siguiente fórmula:

$$Size = (N - F)/paso + 1$$

Siendo N el tamaño de la capa sobre la que aplicamos el filtro, F el tamaño del filtro y paso el paso que usamos. Tenemos, lógicamente, que elegir una combinación de paso y tamaño de filtro que resulte un número entero. Una técnica habitual para no perder tamaño consiste en añadir un borde a la imagen de ceros. De esta forma aumentamos N artificialmente. Se ha comprobado que es

una técnica efectiva en muchos casos.

3.2.2 Las capas de agrupamiento

Una práctica habitual en las redes convolucionales es ir buscando conforme se avanza en la red menor tamaño y complejidad, para conseguir una mayor abstracción de las características de la imagen. Una posible solución a esto es reducir el tamaño de la capa. Para ello podríamos, por ejemplo, no añadir el borde de ceros en las capas de convolución, y dejar que la imagen se vaya reduciendo. Pero esto general el siguiente problema que conforme dejamos reducir la imagen sin añadir borde, vamos perdiendo información de la imagen que se encuentra junto a los bordes, haciendo nuestra red completamente dependiente de la información que se encuentra centrada. Para resolver este problema, existen las llamadas capas de agrupamiento.

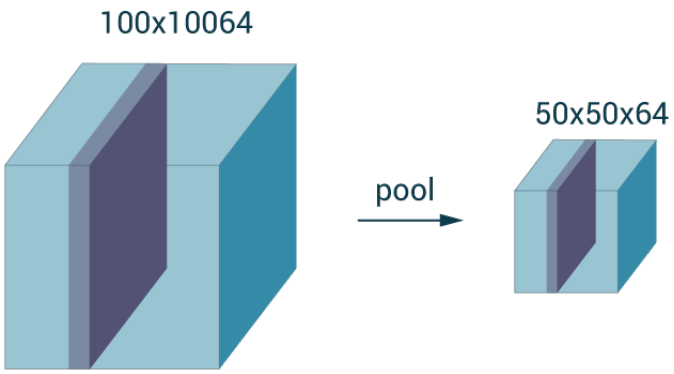

Figura 3.6 Capa de agrupamiento.

Estas capas reducen de forma controlada la dimensión de la capa a la que se aplique. Además, opera sobre cada mapa de activación de forma independiente, por lo que la reducción de la dimensionalidad sólo afecta al tamaño de ancho y alto, pero no a la profundidad proporcionada por los filtros. La forma más habitual para reducir el tamaño suele ser usando filtros de un tamaño determinado, y quedándonos con el valor máximo contenido en dicho filtro.

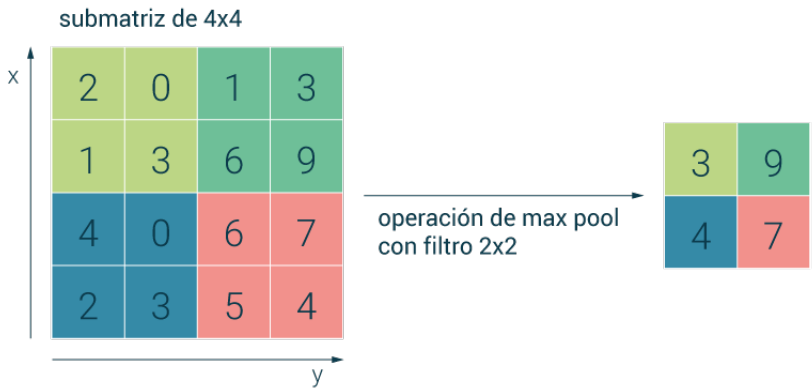

Figura 3.7 Filtro de agrupamiento.

Esta técnica es usada habitualmente en el campo y sus resultados se han demostrado de forma más empírica que porque se haya elaborado una fundamentación teórica de fondo. Esto, de hecho,

es una constante bastante habitual en el campo de las redes neuronales convolucionales.

3.2.3 Capa completamente conectada

Una vez tez tenemos la secuencia de capas convolucionales con sus correspondientes capas de activación y de agrupamiento, es común que la última etapa de la red consista en una capa completamente conectada. Esta capa es como las que se usan en las redes neuronales ordinarias, y consiste en conectar al volumen total de la última capa con un determinado número de neuronas, servirán para discernir la categoría de la imagen.

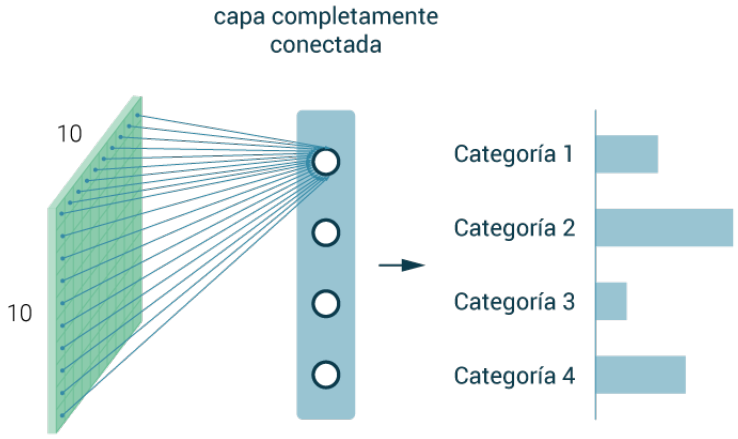

Figura 3.8 Capa completamente conectada.

3.2.4 Funciones de activación

La función de activación es una parte esencial de la red neuronal convolucional, ya que es la que "activa" a la neurona. Esto significa que una vez realizada la operación de convolución (o, mas bien, el producto escalar) es la función que evalúa si ese valor de entrada es el que activa o no una determinada característica. Es cierto que debido a la complejidad que puede tomar una red neuronal por su multitud de capas, a veces no será tan evidente que tipo de característica activa una determinada neurona. Lo que sí es evidente es que esta función es clave para el correcto desempeño de la red. A continuación listamos una serie de funciones de activación, desde clásicas, en desuso, hasta las más comunes actualmente.

Sigmoide

La función sigmoide es históricamente muy popular debido a que se asemeja al comportamiento atribuido a las neuronas biológicas.

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

La sigmoide encaja cualquier valor de entrada en el intervalo $[0,1]$, siendo esto en principio un comportamiento adecuado para una función de activación. Pero realizando una pequeño análisis de la función detectamos varios problemas. El primero es que las neuronas saturadas acaban con los gradientes. ¿Qué significa esto? Que para valores muy positivos o muy negativos, el gradiente de la

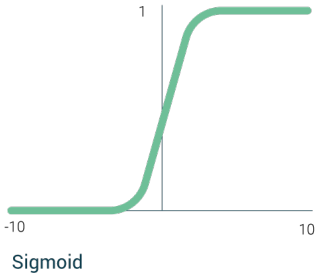

Figura 3.9 Función sigmoide.

función es cero, por lo que en la fase de retropropagación no pasan los valores por esas neuronas y las neuronas conectadas en capas anteriores a esta dejan de aprender.

Un segundo problema que tiene esta función de activación es que, si la entrada a la neurona es siempre positiva, cosa que por ejemplo ocurre en imágenes si no están preprocesadas, la dirección de actualización del gradiente tendrá siempre el signo del valor que venga en la retropropagación. Esto significa que, o todos los parámetros se incrementan o todos se decrementan en cada iteración, dando lugar a una actualización de gradiente ineficiente. Este problema se puede solventar centrando nuestros datos en cero, planteamiento que se discutirá más adelante.

Un tercer problema de la sigmoide es que la computación de una exponencial es cara, y teniendo en cuenta la cantidad de veces que tiene que computarse esta operación, conviene intentar probar con una función menos costosa, ya que se reducirá notablemente el tiempo de computación.

Tangente hiperbólica

La tangente hiperbólica se asemeja a la sigmoide, pero con una clara ventaja sobre ésta, y es que la tangente hiperbólica está centrada en cero, por lo que nos soluciona el problema de la ineficiencia en la actualización por gradiente. Si bien, si se dará el problema de saturación de neuronas que presentaba la sigmoide.

$$\tanh(x)$$

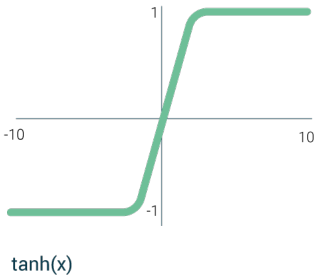

Figura 3.10 Función tanh.

Unidad Lineal Rectificada (ReLU)

La función ReLU es de las más populares actualmente, ya que incorpora mejoras ostensibles respecto a las anteriores mencionadas. En primer lugar, no satura en la región positiva, lo que hace que sea muy conveniente para evitar el problema de las neuronas muertas. Además es computacionalmente

muy eficiente.

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

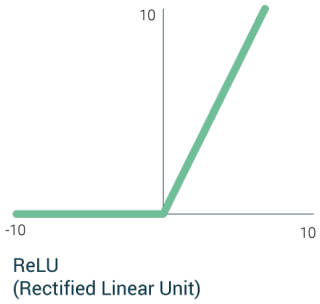

Figura 3.11 Función ReLU.

Aun así, no está completamente libre de ciertos problemas, que consisten en que no está centrada en cero, y que, aunque no satura en regiones positivas, sí satura en regiones negativas. De todas formas se ha demostrado que para la mayoría de casos funciona mejor que la sigmoide o la tangente hiperbólica, además de converger mucho más rápido.

Unidad Lineal Exponencial (ELU)

Lo bueno de esta función de activación es que tiene los beneficios de ReLU, pero mejorando sus puntos débiles, ya que no satura para valores negativos pequeños, pero sí para valores negativos grandes, de forma que la hace robusta para el ruido. Además, es más cercana a estar centrada en cero que ReLU. Como punto negativo debemos destacar que computacionalmente es más costosa que ReLU

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

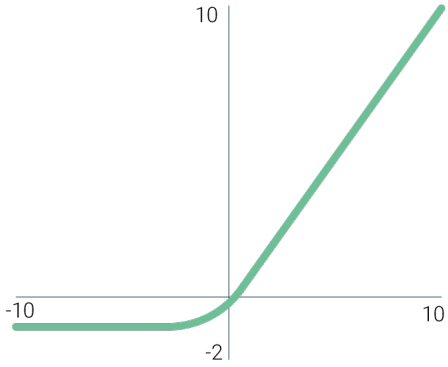

Figura 3.12 Función ELU.

3.2.5 Preprocesado de los datos

Como hemos señalado anteriormente en el presente trabajo, el procesado de los datos es una parte muy importante para poder tener éxito en el proceso de entrenamiento de una red neuronal, y de cualquier algoritmo de aprendizaje automático en general. Por una parte, hay que conocer muy bien cuáles son los datos de los que disponemos, ya que, en general, si no tienes el conocimiento de los datos sobre los que estás desarrollando cualquier algoritmo de clasificación, las posibilidades de realizar un algoritmo exitoso descienden drásticamente. El tratamiento de datos será una materia que analizaremos con más profundidad cuando abordemos el desarrollo de la red, en su sección correspondiente.

Existen multitud de métodos de procesado de datos, pero los más usuales y básicos consisten en centrar los datos en cero y en normalizar los datos.

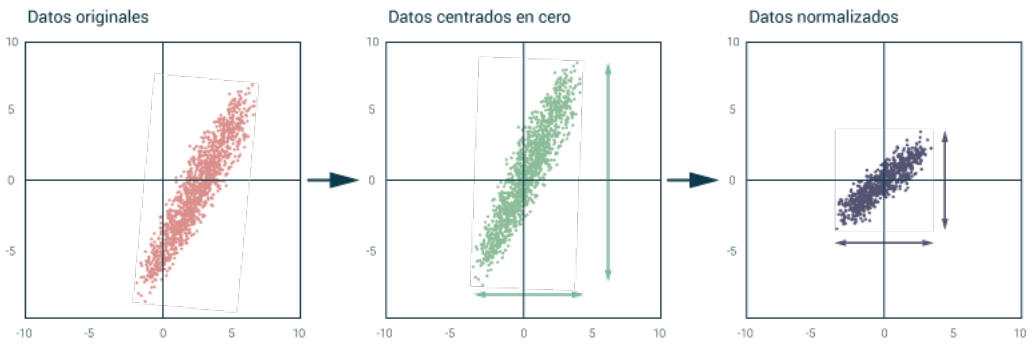

Figura 3.13 Preprocesado de datos.

- El centrado de datos en cero consiste en sustraer la media a nuestros datos, para que se encuentren centrados en 0. Esta técnica es muy interesante, por ejemplo, para paliar los problemas vistos con la función de activación sigmoide, ya que al no estar la propia función centrada en 0, es una ayuda si los datos que introducimos como entrada no son todos positivos. Digamos que puede agilizar el proceso de actualización del gradiente.
- La normalización dividiendo entre la varianza de los datos también es una técnica muy común en los algoritmos de aprendizaje automático.

Para aplicar estas técnicas sobre nuestros datos, es importante tener en cuenta que hay que aplicarlas tanto a los datos de entrenamiento como a los datos de testeo. Lo más razonable sería calcular la media y la varianza de la base de datos al completo, y usar estos valores en cada muestra. Hay que destacar que aunque existe más técnicas de preprocesado, para el caso de imágenes con redes convolucionales lo más común es aplicar solo en centrado de los datos.

3.2.6 Inicialización de los pesos

Un tema por el que se ha pasado por encima hasta ahora es la inicialización de los pesos. En todo momento hemos hablado a lo largo del presente estudio sobre cómo el proceso de entrenamiento ajusta los pesos de los filtros de nuestra red para hacer que la función de pérdida sea cada vez más pequeña. Pero, para modificar estos pesos, en la primera iteración han tenido que tener un valor. Por ejemplo, pongamos que inicializamos todos los pesos a cero: en este caso, en la primera iteración todas las neuronas se actualizarán siguiendo la misma operación, por lo que tendrán el mismo

gradiente, y esto no es un comportamiento deseado, ya que queremos que cada neurona aprenda algo distinto. Por ello hay varios métodos de inicialización de los pesos:

- Inicialización con números aleatorios pequeños. De esta forma conseguimos romper la simetría anteriormente expuesta, y funciona razonablemente bien para redes no muy grandes, pero da problemas para redes más profundas. Puede pasar en estas redes que las activaciones se vayan a 0.
- Inicialización de Xavier. Especificamos que la varianza de la entrada sea igual a la varianza de la salida.

$$W = \frac{N(n_{inputs}, n_{outputs})}{\sqrt{n_{inputs}}}$$

  Lo que significa esto es que si las entradas son números pequeños, dividiremos entre un número pequeño y obtendremos pesos mayores, y esto es deseable porque si las entradas son pequeñas necesitaremos pesos mayores para poder tener la misma varianza a la salida, y lo mismo ocurre al contrario. Es una manera de obtener una distribución gaussiana en los pesos de cada capa. Para que esto funcione, se está asumiendo que hay activaciones lineales en las funciones de activación. En el caso de usar ReLU como función de activación, tenemos que tener en cuenta que aproximadamente la mitad de las neuronas no se activan al inicializarse debido a que entran en la parte de gradiente 0 de la función de activación. Para solucionar esto podemos tomar esta suposición y añadirla a la fórmula anterior.

$$W = \frac{N(n_{inputs}, n_{outputs})}{\sqrt{n_{inputs}/2}}$$

Hay que destacar que el campo de la inicialización de las redes neuronales es un tema de investigación actual, y existen muchos estudios sobre las maneras adecuadas de inicializar un red neuronal convolucional, por lo que aún no hay una respuesta a priori clara que resuelva el problema de la inicialización.

3.2.7 Algoritmos de optimización

El algoritmo de optimización del que se ha hablado anteriormente para el cálculo del gradiente ha sido el SGD, o descenso de gradiente estocástico. Recordamos que este algoritmo calcula el gradiente sobre los llamados minilotes. Una vez tenemos el valor del gradiente se multiplica por la tasa de aprendizaje y se sustrae a cada peso. Este método es sencillo y además parece efectivo, ya en las iteraciones iremos actualizando los pesos en la dirección del gradiente. Si bien, este algoritmo no tampoco está exento de problemas, concretamente los siguientes:

- Cuando la función de pérdida varía más en una dirección que en otra, el progreso en la dimensión poco acentuada es muy lento, y, en cambio, muy rápido en la dimensión donde el cambio es mayor.
- Tiene problemas en los mínimos locales y en los puntos de pendiente cero. Tiende a quedarse estancado, y no consigue actualizar los pesos.
- Además, al realizar los cálculos del gradiente usando minilotes, es muy probable que las actualizaciones sean bastante ruidosas.

SGD + Momentum

Una primera solución es el uso de SGD + Momentum. Este algoritmo añade un término de velocidad sumado al gradiente.

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$

$$x_{t+1} = x_t v_t - \alpha v_{t+1}$$

La velocidad v va acumulando la media de los gradientes, y la fricción ρ sirve como freno de la velocidad. Esta velocidad sirve esencialmente para evitar que se quede el algoritmo "atascado" en un punto llano de gradiente cero. Lo podríamos asimilar a tirar una pelota por una ladera. El caso del SGD sólo es como si la pelota no fuera acelerándose conforme baja. Es cómo si supiera que si hay pendiente hacia abajo tiene que seguir la pendiente, pero no se va acelerando, simplemente avanza. Esto haría que al llegar a un llano ya no siguiera avanzando. En cambio, añadiendo la velocidad conseguimos que aún llegando a un punto donde la pendiente se estabiliza, pudiéramos usar esa inercia que teníamos de antes para seguir avanzando.

Figura 3.14 Punto llano.

Adagrad

El optimizador Adagrad tiene como característica que modifica la tasa de aprendizaje por cada iteración, basándose en los gradientes pasados.

$$G_{t+1} = G_t + \nabla f(x_t) * \nabla f(x_t)$$

$$x_{t+1} = x_t - \frac{\alpha}{\sqrt{G_{t+1} + \varepsilon}} \nabla f(x_t)$$

El principal beneficio de Adagrad es que elimina la necesidad de ajustar manualmente la tasa de aprendizaje. En general se selecciona el valor por defecto de 0.01 y se deja actuar. Su principal problema es la acumulación del cuadrado de los gradientes en el denominador, ya que el término siempre es positivo y va creciendo durante el entrenamiento, pudiendo ocasionar que la tasa de aprendizaje se reduzca demasiado.

RMSProp

RMSProp intenta solucionar el problema de Adagrad de la reducción del gradiente añadiendo una tasa de deterioro que se adapta a la frecuencia a la que un parámetros se actualiza durante el entrenamiento. A más veces se actualiza, más se reduce el ratio de aprendizaje.

$$G_{t+1} = drG_t + (1 - dr)\nabla f(x_t)\nabla f(x_t)$$

$$x_{t+1} = x_t - \frac{\alpha}{\sqrt{G_{t+1} + \varepsilon}} \nabla f(x_t)$$

Adam

Adam es un algoritmo de optimización que coge elementos de los anteriores algoritmos. Por una parte, como RMSProp, almacena de forma exponencial el cuadrado de los gradientes, y por otra parte almacena también exponencialmente una tasa de deterioro media de los gradientes pasados, similar al momentum.

$$m_{1t} = \beta_1 m_{1t-1} + (1 - \beta_1) \nabla f(x_t)$$
$$m_{2t} = \beta_2 m_{2t-1} + (1 - \beta_2) \nabla f(x_t)^2$$

$$\hat{m}_{1t} = \frac{m_{1t}}{1 - \beta_1^t}$$
$$\hat{m}_{2t} = \frac{m_{2t}}{1 - \beta_2^t}$$

$$x_{t+1} = x_t - \frac{\alpha}{\sqrt{\hat{m}_{2t}} + \varepsilon} \hat{m}_{1t}$$

Adam ha demostrado ser una primera buena opción para los entrenamientos iniciales de cualquier red neuronal convolucional que se empiece a desarrollar.

3.2.8 Generalización

En la sección de aprendizaje automático ya se destacó la importancia de generalizar los modelos. Es muy importante en todo proceso de entrenamiento mantener el modelo siempre los más genérico posible, de forma que pueda adaptarse bien a los nuevos datos que se le introduzcan, y no solo a los datos usados para entrenar. Los modelos que tienen una gran precisión en los datos de entrenamiento pero que, por el contrario se adaptan mal a los datos de validación se considera que son modelos sobreajustados.

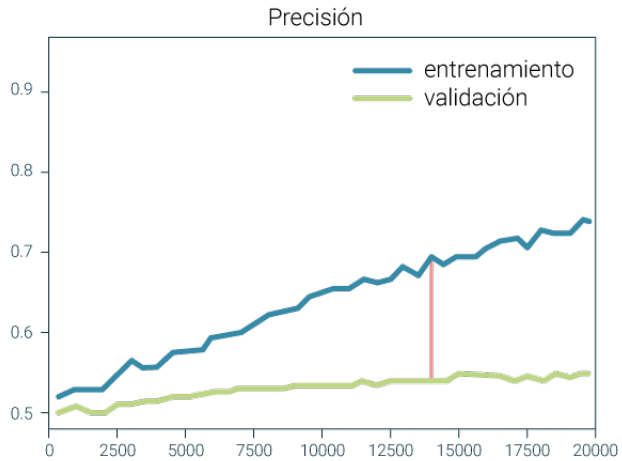

Figura 3.15 Sobreajuste de un modelo.

Es indicativo de este fenómeno que cuando la precisión sobre los datos de entrenamiento mejora, no lo hace en los datos de validación. Esto puede suceder si los datos de validación y de entrenamiento no han sido tomados en las mismas condiciones y que contienen información similar, pero también sucede con bases de datos perfectamente equilibradas entre validación y testeo.

La forma de evitar el sobreajuste es intentando que los pesos no se adapten tan bien a los datos de entrenamiento. Un método para conseguir esto es añadir un término de regularización en el cálculo de la función de pérdida. Este término penaliza dicha función, haciendo que cuando el modelo tiene pérdidas muy pequeñas, se le sume esta regularización que hace que el modelo siga aprendiendo y no se ajuste demasiado a los datos de entrenamiento. Una regularización muy común es la $L2$

$$R(W) = \sum W^2$$

Esta regularización es la suma de los cuadrados de los pesos. Es una forma de penalizar pesos grandes, ya que se entiende que estos pesos se han vuelto grandes porque ajustan mucho la red a los datos de entrenamiento.

Dropout

El dropout consiste en que, por cada iteración, en el proceso de predicción, es decir, cuando la entrada está siendo procesada por la red para obtener la salida, se fijan los pesos neuronas aleatorias a 0. El porcentaje de neurona a las que se le aplica en un hiperparametro configurable de la red.

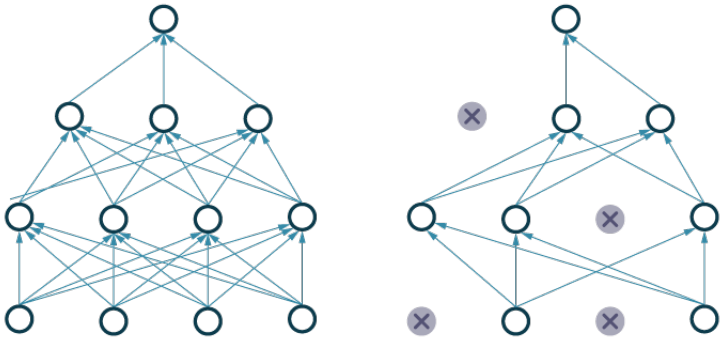

Figura 3.16 Efecto del dropout en una red.

Esta técnica puede resultar confusa, ya que no parece que tenga mucho sentido eliminar la información aprendida por la red de forma aleatoria. Una posible interpretación de por qué el dropout funciona es que fuerza a la red a ser redundante y no depender de una sola neurona para identificar una característica determinada. También evita que unas características se adapten dependiendo de otras. Todo esto ayuda a mejorar la robustez de la red.

4 Herramientas usadas

Expondremos en este capitulo las herramientas que se han usado para desarrollar el proyecto.

4.1 ROS

ROS es una framework creado para el desarrollo de software relacionado con el ámbito de la robótica. Este framework integra una colección de herramientas, librerías, y convenciones cuya principal intención es simplificar la tarea de crear un funcionamiento robusto y complejo en la amplia variedad de plataformas robóticas existentes. La magnitud de ROS es enorme, permite el control de dispositivos a bajo nivel, abstracción del hardware concreto, manejo de paquetes, comunicación entre procesos, etc., todo sin ser realmente un sistema operativo, siendo lo que se conoce como middleware robótico.

ROS es usado en el robot SIAR en su distribución ros-indigo. Su arquitectura software está diseñada como una serie de módulos que hacen uso de la plataforma robótica, sensores y comunicaciones para cumplir los objetivos deseados; este diseño está implementado con ROS. Para nuestra parcela concreta, los datos obtenidos por las diferentes cámaras del robot se encuentran empaquetados en bags, que es un tipo de formato de archivo de ROS para almacenar mensajes.

La estructura utilizada en ROS para comunicar los distintos módulos consiste en el uso de nodos. Los nodos son los procesos básicos en ROS. Estos procesos pueden publicar mensajes en los topics, y/o leer la información de dichos topics. Con el uso del paquete rosbag podemos grabar en bags los mensajes publicados en un topic, para poder posteriormente reproducirlos o analizarlos.

En nuestro caso, tenemos acceso a datos que se han generado en alcantarillas reales en pruebas realizadas en distintas localizaciones y distintos días. Estos datos consisten en un fichero bag con la información de todas las cámaras del robot obtenidas durante las pruebas. También está la información del momento en el que el robot se encuentra bajo una boca de alcantarilla.

4.2 OpenCV

OpenCV es una librería de visión por computador y aprendizaje de máquina. Es software libre y está escrita en C++, teniendo interfaces de uso en C++, Python, Java y MATLAB. Proporciona una multitud de algoritmos optimizados, tanto clásicos como del estado del arte en los campos antes mencionados. Estos algoritmos permiten realizar multitud de acciones sobre imágenes y videos, tales como: detección, identificación y clasificación de objetos, acciones humanas, rostros; seguimiento

de objetos en movimiento; generación de nubes de puntos en espacios en tres dimensiones; relación de imágenes similares, etc.

Se trata de una librería con gran bagaje, y actualmente soportada y en desarrollo. Tiene una gran potencia, y permite el tratamiento de imágenes a muy bajo nivel, ofreciendo de esta forma una gran versatilidad.

En este proyecto concretamente es usada para tratar y modificar las imágenes originales, para generar imágenes nuevas, y, así, aumentar las bases de datos y para la representación las imágenes.

4.3 Keras

Keras es una librería de código abierto de alto nivel para el uso de redes neuronales. Está escrita en python, y está preparada para funcionar como una capa de abstracción por encima otras librerías, como TensorFlow, CNTK o Theano. Las características de Keras para su uso como librería de deep learning son:

- Permite un prototipado rápido y fácil.
- Soporta redes convolucionales y recurrentes.
- Funciona de igual forma en CPU o GPU
- Es compatible con Python 2.7-3.6

Para nuestro problema hemos usado TensorFlow junto a Keras, es decir, Keras ofrece esa capa superior más amigable con la que el programador interacciona, y a un nivel inferior se encuentra la librería TensoFlow, que es con la que Keras trabaja internamente, haciéndolo de forma transparente al usuario.

La estructura básica de Keras es el modelo, que define la manera de organizar las capas. Con un modelo definimos la estructura de capas que tendrá la red, siendo cada capa configurable entre las amplias opciones que hay en la librería. El modelo básico es el modelo secuencial. En él, las capas se almacenan unas tras otras de forma linear. Esta organización se adecua perfectamente a las redes neuronales, ya que podemos ir añadiendo y modificando las capas de la red de forma independiente, dando como resultado el conjunto de todas estas nuestra red neuronal. Tras configurar y añadir todas las capas, se compila el modelo, y si todo es correcto ya se puede entrenar. Tras el entrenamiento, Keras ofrece también herramientas para generar las métricas que sean necesarias para evaluar la red obtenida.

De este modo, se ajusta a las necesidades de nuestro proyecto, ya que:

- Tiene la profundidad perfecta para nuestro propósito, permitiendo cambiar de forma rápida la configuración del modelo para las distintas pruebas con la red.
- Nos ofrece las métricas más comunes y útiles para el tipo de problema que encaramos.
- Es una librería de código abierto y está en continuo desarrollo.
- El uso de Python como lenguaje de programación, siendo éste un lenguaje fácil y ampliamente utilizado por la comunidad para este tipo de investigaciones.
- Nos ofrece la posibilidad de usar GPU o CPU, lo que hace que los entrenamientos y el desarrollo sea independiente de la máquina que usemos, abstrayendo nuestro problema del hardware concreto.

4.4 TensorFlow

TensorFlow es una librería de código abierto para la computación numérica. Es una librería simbólica matemática, ampliamente usada en el campo del aprendizaje automático, y en concreto para redes neuronales. Fue desarrollada por Google Brains para uso interno en un principio, pero tras detectar su potencia y mejorarla, fue lanzada al mundo en 2015. Está desarrollado en C++, y sus APIs más populares son las de C++ y Python.

Una de las características más identificables de TensorFlow es su flexibilidad, que casa con la idea de poder acceder al mayor número de usuarios posibles. Para ello, ofrece múltiples niveles de abstracción que permiten trabajar con la librería dependiendo de las necesidades de cada desarrollador y de cada proyecto. También permite su ejecución en la nube o en local, en unas máquinas o en grandes clústeres.

TensorFlow es usado en este proyecto como el backend que utiliza Keras. Su uso es interesante debido a que tenemos la posibilidad de acceder a funciones de más nivel para la configuración de nuestro algoritmo cuando fuera necesario.

4.5 Python

Python es un lenguaje de programación interpretado de alto nivel y de propósito general. Algunas de las características de python son:

- Es un lenguaje dinámico.
- Tiene una gestión automática de la memoria.
- Soporta múltiples paradigmas de programación.
- Tiene una sintaxis simple, y como característica usa espacios en blanco para la separación semántica de bloques.
- Está diseñado para ser extensible con módulos

Es uno de los lenguajes más conocidos y usados en la actualidad, y en concreto en el ámbito de la ciencia de datos y en el aprendizaje automático. Todas las librerías y módulos usados en el proyecto tienen APIs de Python para permitir su uso, lo que hace que junto a su facilidad de aprendizaje sea la elección más lógica para el desarrollo del presente proyecto.

5 Desarrollo del clasificador

En este capítulo se explica todo el proceso de desarrollo para obtener el sistema de clasificación que permite discernir si el robot se encuentra bajo la tapa de una alcantarilla o no, siendo esta cuestión el objeto principal del proyecto que estamos presentando. Este proceso nos lleva desde la obtención de los datos, en este caso imágenes, el tratamiento de las mismas para poder usarlas como entrada en el entrenamiento de la red convolucional, la elección de parámetros y arquitectura de la red, el entrenamiento, y el análisis y post-procesado de los datos. Como se comentó en la introducción, hasta ahora la plataforma robótica SIAR ha usado un clasificador que obtiene la información de la imagen de profundidad, obteniendo según [3] unos resultados de en torno al 97 % en validación. Es nuestra intención alcanzar a unos resultados similares, lo que permitiría el uso de imágenes de cámara normal para el reconocimiento de puntos característicos en los sistemas de alcantarillado.

5.1 Obtención de los datos para el clasificador

El proyecto que da lugar a la plataforma robótica SIAR tiene como objetivo la creación de una solución robótica para la inspección de la red de alcantarillado de los núcleos urbanos. Este trabajo ha sido tradicionalmente realizado por trabajadores, que se introducen en estos sistemas de alcantarillado para realizar tareas de inspección y mantenimiento. Como podemos imaginar, esta tarea es difícil de realizar, debido a que las personas que la realizan tienen que introducirse en espacios confinados, algunas veces con accesos difíciles, y en condiciones de humedad y calidad del aire que no son las más adecuadas para la salud. Parece por tanto lógico que, en la época actual en la que cada vez más procesos son automatizados, ya sea por ahorro de costes o por eficiencia y precisión en la fabricación, también se usen estas tecnologías para la sustitución de trabajos peligrosos y rutinarios.

Con este objetivo, se pone en marcha el desarrollo de la plataforma robótica SIAR, que es un robot diseñado específicamente para realizar esta tarea. La solución dada consiste en un robot terrestre de seis ruedas equipado con múltiples sensores que puede introducirse por los sistemas de alcantarillados y recorrerlos de forma autónoma, almacenando la información recogida por los sensores, para poder realizar la tarea de inspección sin necesidad de que un trabajador acceda al interior de las alcantarillas. El trabajador se situará fuera de las alcantarillas, en la superficie, y puede tanto dejar que el robot realice las acciones de forma autónoma como teleoperarlo.

En cuanto al sistema de navegación autónoma del robot, está condicionado por la imposibilidad de poder usar un sistema de geolocalización, ya que el robot va a operar bajo tierra. Por ello, se

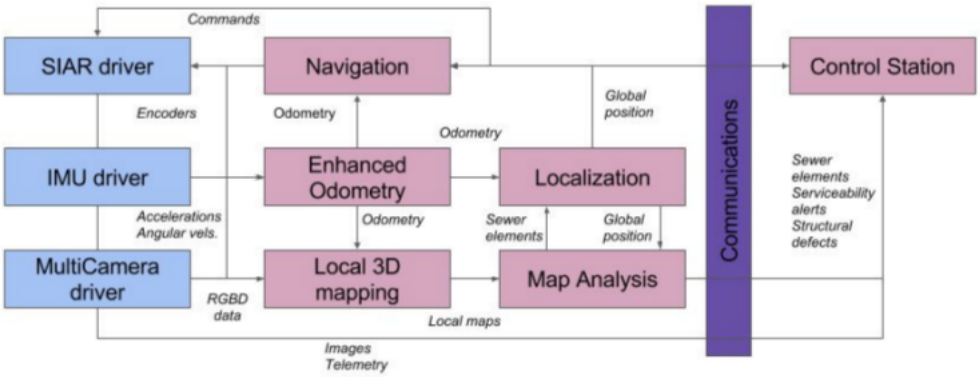

Figura 5.1 Arquitectura software de la plataforma robótica SIAR..

recurre a una solución que utiliza la información existentes de los mapas de alcantarillado, y por medio de odometría localiza al robot en el mapa. La plataforma tiene la posibilidad de relizar la odometría clásica con las ruedes del robot, pero no es lo suficientemente robusta. Por ello, se realiza una adomoetría vusial con las cámaras que posee para este porpósito.

A priori esto sería suficiente para poder realizar la localización del robot en todo momento, ya que disponemos del mapa, y con la odometría, teóricamente, situamos al robot en dicho mapa en cada instante de tiempo. Sin embargo encontramos otro problema ya que toda odometría produce derivas, que a lo largo del tiempo se acumulan y puede producir errores grandes. Para ello se recurre al concepto de las balizas, con esos puntos de referencia que permiten “limpiar” el error acumulado en la odometría. Este sistema se podría asimilar al utilizado en las técnicas de gps diferencial, en las que tenemos puntos que conocemos su localización exacta, y, por tanto, podemos inferir con mucha más precisión los puntos que se encuentra cerca de ese punto de referencia.

En el caso de este proyecto, las balizas son las bocas de alcantarilla, vistas desde dentro de la red de alcantarillado. En los mapas de los sistemas de alcantarillas están localizadas estas bocas, por lo que una vez que el robot se encuentra dentro de los túneles, conforme va moviéndose, si es capaz de reconocer cuando se encuentra bajo una boca, es posible inferir a partir de la localización actual hecha por la odometría en qué punto exacto de la red se encuentra. Calculamos aproximadamente que las bocas de alcantarillas se encuentran en torno a unos 10 metros de una a otra, por lo que la deriva de la odometría en ese espacio no llegará a ser tan grande como para que no sea posible determinar cuál es la boca en la que el robot se encuentra.

En el proyecto SIAR el cálculo de puntos característicos se realizó con los datos de imágenes tridimensionales. El propósito de nuestro trabajo es demostrar si es posible realizar dicha localización con imágenes normales, ya que la información que pueden aportar una y otra es distinta. Puede ser interesante que sea pueda localizar estas bocas de alcantarillas con ambos tipos de imágenes, en caso de no poder utilizar una de las dos cámaras, o porque en determinados casos una funciones mejor que la otra.

5.1.1 Bases de datos: rosbags

Como hemos mencionado, la detección de las bocas de alcantarilla se realiza con la información de una cámara situada en el robot que apunta hacia arriba. También, como se explicó en la sección de1.3, para realizar una red convolucional con este propósito es necesario una base de datos lo suficientemente amplia para poder entrenar y verificar los resultados.

En el marco del proyecto del robot SIAR, se han realizado una serie de pruebas en sistemas de alcantarillado reales con el robot. Estas pruebas han consistido en recorridos programados y supervisados en los que se ha puesto en funcionamiento el robot para simular un recorrido real por un sistema de alcantarillado de una ciudad. Durante las pruebas se han activado los sensores del robot y se han recogido y almacenado los datos. Por tanto, tenemos acceso sin necesidad de trabajar directamente con la plataforma a datos muy valiosos para el desarrollo de nuestro clasificador. Los datos se han almacenado en un tipo de formato de ROS llamado bags dentro de la nomenclatura de ROS. Estos bags son paquetes de datos que, usando el ROS, pueden simular las conexiones y formatos de datos que se usan en ROS cuando un robot está funcionando. Esto quiere decir que, podemos usar un robot para hacer una prueba en un entorno real, capturar todos los datos generados en este proceso y almacenarlos en bags. Luego, estos bags se reproducen y forman las mismas estructuras de datos, y emiten los mismo datos que si el robot estuviera realmente emitiendo. También permite acceder a información parcial de un sensor en concreto, y almacenar los datos del sensor en el formato que más nos interese. Este funcionamiento es muy útil, ya que permite recoger los datos que nos interesen de las pruebas que se almacenen, darles el formato que nos interese, y a la vez reproducir la prueba para ver datos complementarios que quizás no queramos almacenar pero sí visualizar.

Los bags de los que hemos hecho uso para la realización de este trabajo son:

- Bag 1: más de 18000 imágenes en Mercat del Born, el 21 de septiembre de 2017.
- Bag 2: más de 21000 imágenes en Mercat del Born, el 11 de octubre de 2017.
- Bag 3: más de 24000 imágenes en Mercat del Born, el 17 de octubre de 2017.
- Bag 4: más de 36000 imágenes en Creu de Pedralbes, el 18 de mayo de 2018.
- Bag 5: más de 11000 imágenes en Creu de Pedralbes, el 12 de junio de 2018.

Los datos que se encuentran disponibles en cada bag se pueden consultar en el apéndice A. En nuestro caso nos interesan los datos captados por la cámara del robot que se encuentra apuntando hacia arriba. En concreto, nos interesa las imágenes que se publican en el topic */up/rgb/image_raw/compressed*. Este topic publica mensajes de tipo *sensor_msgs/CompressedImage*, que las podemos extraer cómo imágenes RGB de 320×240 píxeles, y guardarlas en formato .jpeg.

5.1.2 Etiquetado de los datos

Para poder realizar el entrenamiento de la red neuronal convolucional y su posterior evaluación, es necesario tener los datos etiquetados. Este etiquetado se realiza en general de forma manual; de manera que, necesitaríamos de una persona a la que se le definan claramente las reglas para decidir cual es la categoría de cada imagen. Este criterio tiene que ser claro, y este proceso suele ser bastante tedioso pero imprescindible. Es nuestro caso, en el topic */ground_truth* tenemos información de en que puntos concretos de la prueba hay una boca de alcantarilla. Estos datos fueron generados de forma supervisada durante los experimentos, y van a ser muy útil para poder etiquetar los datos.

Para este proceso, vamos a usar esta información contenida en el bag de las localizaciones de las bocas, y también revisaremos en las imágenes guardadas para ajustar los datos etiquetados. Las bases de datos se corresponden a experimentos en los que las imágenes han sido tomadas de forma continua a lo largo del tiempo. Esto implica que nuestras imágenes son como frames de un vídeo, por lo que cuando el robot se aproxime a una boca de alcantarilla, habrá imágenes donde la alcantarilla se ve de forma parcial, que corresponde a cuando el robot se aproxima o se aleja de una alcantarilla, e imágenes donde la alcantarilla se ve de forma completa. El criterio utilizado ha sido el siguiente: se guardan como positivas las imágenes donde la boca de alcantarilla aparece al

completo, y en otra categoría las imágenes en las que la boca de alcantarilla aparece de manera parcial. Estas ultimas no se usan para entrenamiento.

La decisión de sacar las imágenes parciales del entrenamiento se toma porque queremos evitarnos problemas con los falsos positivos. Este clasificador debe ser muy robusto a la hora de dar un positivo, porque en caso de dar como positivo algo que no es, podemos introducir errores graves en el sistema de localización. Creemos que dejando estas imágenes fuera, conseguiremos reducir este problema.

Figura 5.2 Los 3 grupos de imágenes.

El resto de los datos serán, por tanto, imágenes del techo del sistema de alcantarillado. Aquí nos encontramos con casos más diversos, ya que el resto del techo no es una categoría tan definida como la de bocas de alcantarilla. Hay desde partes con muchas formas, que pueden ser suciedad, musgo, a partes más planas. También hay veces que nos encontramos con tuberías en el techo y otros elementos no identificables. La categoría de no boca de alcantarilla es por tanto más heterogénea que la de boca de alcantarilla, algo que es normal en los problemas de clasificación binaria.

5.1.3 Aumento de datos de forma artificial

Un problema habitual de las bases de datos para la clasificación es la diferencia en la cantidad de datos en cada clase. Lo ideal para cualquier algoritmo de clasificación que se entrene con datos de entrada es tener una cantidad de muestras representativas y similares de cada categoría, pero esto no es siempre posible. De hecho, es es lo que ocurre en nuestro caso de estudio: la toma de datos es realiza como una grabación del robot en un recorrido por el sistema de alcantarillado, por lo que la mayoría de los datos recogidos en esta prueba son en momentos en los que no se encuentra el robot debajo de una alcantarilla. Esto implica que tendremos muchas más imágenes categorizadas en el grupo de no bocas de alcantarilla que en el grupo de bocas de alcantarilla. Lo que puede ser un problema en el proceso de entrenamiento de la red neuronal, ya que la descompensación de las categorías puede provocar que los pesos de las neuronas no aprendan bien los casos de la categoría de boca de alcantarilla.

Para paliar este problema recurrimos a las técnicas de aumento de datos de forma artificial. Esto consiste en la generación de datos nuevos a partir de los datos existentes aplicando operaciones sobre las imágenes que las modifiquen pero conserven su significado y su valor para la clasificación. En nuestro caso hemos aumentado los números de muestras de la categoría de positivos de la siguiente forma:

- Volteo de la imagen en horizontal.
- Volteo de la imagen en vertical.
- Añadiendo ruido gaussiano.

5.1.4 Preprocesado de los datos

Los datos deben ser preprocesados antes de introducirlos como entrada en la red neuronal convolucional. Como se explica en la sección 3.2.5, para los casos de imágenes y redes neuronales, lo más habitual es realizar un centrado de los datos en cero, y es lo que se ha realizado en nuestro caso. Para realizar el centrado de los datos, se toma la imagen y se le substrae la media a cada píxel. Para ello, la matriz de la imagen se carga como valores de punto flotante, se calcula la media de todos sus píxeles, y se le resta a cada píxel dicha media. Luego los valores de los píxeles son reconvertidos a enteros nuevamente.

Otra transformación realizada a las imágenes ha consistido en pasar de una imagen RGB con tres canales a una en escala de grises de un canal. Esta decisión se toma porque se considera que en el entorno donde se toman las imágenes el color de dichas imágenes no aporta información relevante para abstraer las características de nuestro puntos característicos, y lo que podría resultar de introducir la información de color en los datos es un efecto contraproducente de que "aprendiera" ciertas características no generalizables a todos los sistemas de alcantarillas.

5.2 Proceso para obtener el clasificador

Con los datos preparados para ser alimentados en nuestro clasificador, nos disponemos a desarrollar la red neuronal convolucional, seleccionando su arquitectura y ajustando los parámetros. Este proceso es iterativo y consiste en ir probando poco a poco dentro de todas las posibilidades que existen para, conforme se vayan obteniendo unos resultados, ir ajustando lo necesario aplicando el conocimientos que se tiene sobre la materia y la intuición.

5.2.1 Implementación de la red neuronal convolucional

Para la implementación de la red neuronal convolucional usamos Keras. En Keras vamos formando nuestra arquitectura de la red añadiendo las distintas capas de forma secuencial, y tras esto tan solo tenemos que compilar el modelo y entrenar. Detallamos los puntos más importantes para la codificación de la red, explicitando sobre que parámetros actuaremos.

Código 5.1 Modelo secuencial.

```
model = Sequential()
```

Definimos que nuestro modelo es un modelo secuencial, que consiste en apilar capas de forma lineal.

Código 5.2 Añadir capas.

```
model.add()
```

Con esta función podemos añadir cualquiera de las capas disponibles al modelo.

Código 5.3 Compilar el modelo.

```
    compile(optimizer, loss=None, metrics=None, loss_weights=None,
        sample_weight_mode=None, weighted_metrics=None, target_tensors=
        None)
```

Con compile se configura el modelo. las opciones más importantes son:

- optimizer: permite elegir el optimizador deseado dentro de los implementados en la librería.
- loss: permite seleccionar la función de pérdida.
- metrics: las métricas que evalúan el modelo durante el proceso de entrenamiento y validación.

Código 5.4 Entrenar el modelo.

```
    fit(x=None, y=None, batch_size=None, epochs=1, verbose=1, callbacks=
        None, validation_split=0.0, validation_data=None, shuffle=True,
        class_weight=None, sample_weight=None, initial_epoch=0,
        steps_per_epoch=None, validation_steps=None, validation_freq=1)
```

Entrena el modelo un número determinado de iteraciones sobre la base de datos (epoch). Las opciones más relevantes son:

- x: Numpy array con los datos de entrenamiento.
- y: Numpy array con las etiquetas de los datos.
- batch size: número de muestras por actualización del gradiente.
- epochs: número de iteraciones sobre la base de datos al completo.
- validation data: datos de validación, no usados en el entrenamiento.

Las capas utilizadas en la red son:

Código 5.5 Capa convolucional.

```
keras.layers.Conv2D(filters, kernel_size, strides=(1, 1), padding='valid
    ', data_format=None, dilation_rate=(1, 1), activation=None, use_bias
    =True, kernel_initializer='glorot_uniform', bias_initializer='zeros',
     kernel_regularizer=None, bias_regularizer=None,
    activity_regularizer=None, kernel_constraint=None, bias_constraint=
    None)
```

Este es el filtro de convolución de 2 dimensiones para imágenes. Las opciones principales son:

- filters: número de filtro usados, que darán lugar al número de mapas de activación generados.
- kernel size: especifica la altura y la anchura del filtro.
- padding: especificamos si queremos mantener añadir borde para poder mantener el tamaño en los mapas de activación.
- strides: el paso de aplicación del filtro
- activation: función de activación elegida.

Código 5.6 Capa pooling.

```
keras.layers.MaxPooling2D(pool_size=(2, 2), strides=None, padding='valid
    ', data_format=None)
```

Esta es la capa de pooling para imágenes, que sirve para reducir el alto y ancho de los mapas de activación.

- pool size: factor de reducción de la imagen.
- strides: el paso de aplicación del filtro

Código 5.7 Capa DropOut.

```
keras.layers.Dropout(rate, noise_shape=None, seed=None)
```

La capa para realizar el dropout. En esta capa el parámetro esencial es *rate*, que es la fracción de unidades de entrada para desactivar.

Código 5.8 Capa Dense.

```
keras.layers.Dense(units, activation=None, use_bias=True,
    kernel_initializer='glorot_uniform', bias_initializer='zeros',
    kernel_regularizer=None, bias_regularizer=None, activity_regularizer
    =None, kernel_constraint=None, bias_constraint=None)
```

Es la capa básica de conexión completa. Necesita que la capa previa no sea una matriz de dos dimensiones, por lo que se tendrá que usar una capa de "aplanado" para su uso.

- units: dimensionalidad de salia.
- activation: función de activación.
- kernel initiazlizer: algoritmo usado para la inicialización de los pesos.

5.2.2 En búsqueda de la red

Comenzamos en esta fase a realizar entrenamientos sobre los datos, y – en base a los resultados que vayamos obteniendo- decidiremos qué alternaciones debemos ir realizando para que nuestra red vaya aprendiendo. Para empezar a probar se usa una red sencilla, incluida en el apéndice A. Esta es una que tiene tan solo dos capas convolucionales, se elige esta para empezar para que, acorde a los resultados que se obtengan, ir aumentando el número de capas y cambiando los filtros e hiperparámetros correspondientes.

Cambios en las bases de datos

La técnica usada para realizar los entrenamientos consiste en realizar pruebas cortas, es decir, de pocas iteraciones sobre las bases de datos, y dependiendo de los resultados obtenidos hemos ido seleccionando los Para estas primeras pruebas se indaga cambiando entre varios optimizadores y funciones de activación. Los resultados que se van analizando para determinar cómo va funcionando el entrenamiento son los valores de las funciones de pérdida sobre los datos de entrenamiento y validación, y la precisión sobre ambas bases de datos.

Los primero que realizamos es dividir las imágenes que tenemos en dos porciones:

- Porción de entrenamiento: decidimos usar un 85 % de los datos para entrenamiento.
- Porción de validación: el 15 % restante lo dejamos para la validación.

La división de estos datos se realiza de forma aleatoria, es decir, se agrupan todas las imágenes disponibles y se van repartiendo usando una función aleatoria entre datos de entrenamiento y datos de prueba, de forma que se cumpla que la proporción entre ambos sea la anterior.

El resultado de estos entrenamientos tiene un característica común: el valor de la función de pérdida de la validación es menor que el del entrenamiento. Esta circunstancia no es normal ni, en principio, deseable, ya que no es un buen indicador que se ajuste mejor el modelo los datos con los que no se está entrenando que los datos sobre los que se están realmente realizando los ajustes de los pesos de la red. Una posible explicación a este fenómeno se puede encontrar en los datos. La primera acción a realizar es cerciorarse que no se han introducido datos del entrenamiento en la validación. Una vez comprobado que, efectivamente, no es nuestro caso, pasamos a hacer un análisis más en profundidad de los datos. Nos encontramos entonces con los siguientes problemas:

- El principal problema detectado es causa directa de la forma en la que se tomaron los datos y se dividieron luego en validación y entrenamiento: el robot realiza recorridos de aproximadamente 1h en los sistemas de alcantarillado tomando imágenes, pero durante la prueba ocurre que se queda parado a veces varios minutos, tomando durante este tiempo imágenes muy similares. Luego, reanuda la marcha y vuelve a pararse durante un tiempo. Esto genera dos problemas: el primero es que las bases de datos una descompensación en cuanto a su variedad, ya que tenemos muchas imágenes de exactamente la misma zona, dotando a dichas zonas de un peso muy relevante a la hora de entrenar la red. El segundo es que, al dividir los datos entre entrenamiento y validación de forma aleatoria, es muy probable que gran parte de estas imágenes de las mismas zonas se distribuyeran en ambos, alterando los resultados en validación.

Figura 5.3 Imágenes distintas del mismo lugar.

- Otro problema es que hay momentos donde la cámara graba a los operarios que supervisan la prueba. Sobre todo sucede al principio y al final de las pruebas sobre todo. Esto implica que estamos metiendo en las bases de datos de categoría negativa imágenes que no entran en dicha categoría. Que si bien, no quiere decir que no sean imágenes donde, efectivamente, no hay una boca de alcantarilla, si no más bien que nuestro problema está definido para un entorno con una condiciones concretas, y todo lo que sean imágenes fuera de ese entorno pueden el proceso de entrenamiento.

Las acciones que se toman para solucionar estos problemas son las siguientes:

- Se limpian las bases de datos: eliminamos los grandes espacios de tiempo en los que se toman imágenes mientras el robot está parado. También se eliminan las imágenes de los momentos en que la cámara no está captando imágenes de la prueba en la red de alcantarillado.

Figura 5.4 Imágenes con operario.

- Se procede a dividir los datos de entrenamiento y de validación de la siguiente forma: se deja un bag completa para validar, en concreto el bag 3. Así nos aseguramos que los datos se validan sobre datos que carecen de toda sospecha. Añadimos robustez a la precisión obtenida en la validación.

Tras este filtrado de los datos nos quedamos con el siguiente número de imágenes:

- Unas 60000 imágenes para entrenamiento.
- Una 23000 imágenes para validación.

Elección de parámetros y arquitectura de red

Realizada la limpieza de los datos, volvemos a poner en marcha las pruebas para dar con qué tipo de red nos ofrece los resultados deseados. El proceso para dar con esta red lo podríamos resumir de la siguiente forma:

- Realizamos combinaciones de varios parámetros sobre una arquitectura de red concreta, y lanzamos entrenamientos cortos, cogemos los resultados que tengan más sentido y descartamos las combinaciones que no han funcionado. En nuestro caso empezamos iterando entre varios optimizadores y funciones de activación.
- Entrenamos un poco más sobre esta selección de parámetros.
- Seleccionamos los que van teniendo buen comportamiento, y vemos que falla en ellos.
- Probamos a cambiar otros parámetros con la intuición obtenida del paso anterior
- Cambiamos la arquitectura de la red añadiendo capas o quitando.
- De vez en cuando probamos algo muy distinto con alguno de los parámetros a ver si hay una mejora no esperada, ya que se puede caer en el error de ajustar muy fino sobre un mínimo local, y hace falta salir de alguna forma de ese mínimo para avanzar.

Los parámetros sobre los que hemos actuado son:

- Optimizadores: se prueban varios optimizadores, RMSProp, Adagrad, Adam.
- Funciones de activación: relu, elu, tanh.
- Ratio del dropout.
- Tamaño de los kernels en las capas de convolución.
- Número de capas de convolución.
- Número de capas de pooling.

- Ratio de aprendizaje.
- Orden de las capas

Como se detalla, se combina entre muchos grados de libertad, que además no son todos los posibles, pero sí quizás los más relevantes. El principal problema en las frases más tempranas de la búsqueda consistió en evitar el sobreajuste de la red.

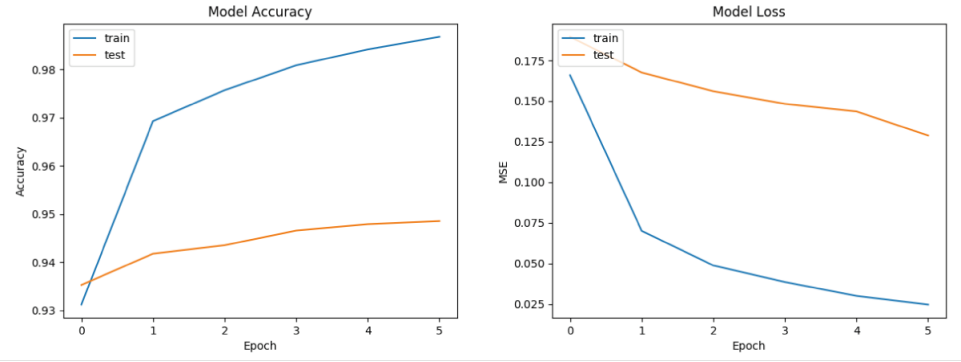

Figura 5.5 Ejemplo de entrenamiento con sobreajuste.

Esto puede ocurrir cuando le damos a la red un grado de complejidad superior al necesario, provocando que la red rediga muy bien sobre los datos de entrenamiento, pero provocando en un determinado momento que dejara de aprender y empeorara con los datos de validación. Para solucionar este problema es interesante centrarse en controlar bien el ratio de aprendizaje, y aumentar el ratio de las capas de dropout. El dropout es la manera elegida para regularizar la red y evitar el sobreajuste. Controlando dichos parámetros se prueba a añadir varias capas y tamaños de filtro, siendo evidente que con nuestros datos no es necesario añadir excesiva complejidad a la red.

En general, en la mayoría de las pruebas, tras realizar ajuste dependiendo del tipo de red, era sencillo obtener un 99 % de precisión sobre el entrenamiento y en torno al 95 % en validación. En otras redes se han llegado a picos de precisión de 98 %, pero se han descartado porque observando las curvas de la función de perdida, nos percatamos que se ha decido más a una consecuencia de un sobreajuste en la red que provoca picos durante el entrenamiento ,pero cuya validez debe quedar en entredicho.

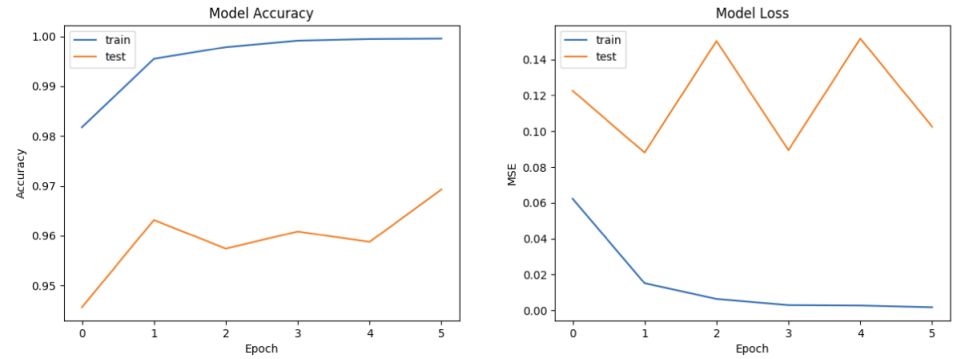

Figura 5.6 Entrenamiento ruidoso.

En el caso de la red elegida, que es la mostrada en el apéndice B obtenemos una precisión en

validación del 97 % y unas gráficas de la función de pérdida con un comportamiento adecuado. Se observa un ligero sobreajuste en la red, pero dentro de los límites normales.

5.3 Análisis de la CNN

Vamos a analizar con algunos resultados más cuál es el desempeño de la red obtenida. Cómo se ha detallado en la anterior sección, nuestra red tiene un 97 % de precisión sobre la validación. También se ha detallado que dichas imágenes de validación nos ofrecen confianza suficiente, ya que son correspondiente a una prueba distinta de las usadas para entrenar.

El dato de la precisión por sí solo puede ser engañoso, porque este dato no diferencia entre predicciones de una u otra clase. Es interesante evaluar como funciona la red neuronal de forma separada para los datos positivos y negativos. Para ello evaluamos la red de forma separada con estos datos. Tenemos que tener en cuenta que la salida de nuestra red es un valor entre 0 y 1, indicando 0 que pertenece a la categoría de no boca de alcantarilla, y 1 a la categoría de boca de alcantarilla. Esto es interesante porque nos indica para cada imagen la seguridad de la clasificación.

Para el caso de no tapa de alcantarilla obtenemos estos resultados

Tabla 5.1 Resultados de clasificación sobre las imágenes dónde no hay tapa de alcantarilla.

0 a 10 %	10 a 50 %	50 a 90 %	90 a 100 %
97,2 %	1,8 %	0,8 %	0,2 %

Para el caso de tapa de alcantarilla obtenemos estos resultados

Tabla 5.2 Resultados de clasificación sobre las imágenes donde hay tapa de alcantarilla.

0 a 10 %	10 a 50 %	50 a 90 %	90 a 100 %
4,8 %	23,3 %	5,6 %	66,2 %

En las tablas mostradas podemos ver para los casos de datos positivos y negativos cuales son las clasificaciones. Se han elegido estos umbrales para diferenciar las imágenes que claramente obtienen una clasificación de las que tienen menos confianza en la clasificación. Lo primero que destaca quizás es ese 23 % de imágenes positivas con una confianza del 10 % al 50 %. Para ver que ha pasado nos vamos a las datos y vemos lo siguiente:

Gran parte de las imágenes clasificadas con este porcentaje son de esta alcantarilla, que pertenecen a un momento de la toma de datos en las que las condiciones de luminosidad son muy bajas, y el borde de la alcantarilla no se diferencia muy bien de alrededor. Luego hay otras imágenes en este grupo de otras bocas, pero son imágenes dispersas, que tienen imágenes de la boca que sí tienen porcentajes altos de confianza.

Aún así, tenemos un clasificador en el que se cumple dos cosas muy beneficiosas: Los parámetros sobre los que hemos actuado son:

Figura 5.7 Alcantarilla clasificada con un 30 % de confianza.

- Hay un porcetaje bajo de falsos positivos, esto es un punto importante buscado desde el principio.
- Si trazamos nuestro umbral de clasificación en torno al 10 % tenemos un 97,2 % de imágenes bien clasificadas para los negativos, y un 95,2 % de imágenes bien clasificadas para positivas. El establecer un valor para este umbral define a qué le damos más valor, si a tener más confianza que no dar un falso positivo, o preferir asumir un poco más de riesgo para no perder la localización de ninguna boca de alcantarilla.

Tabla 5.3 Resultados de clasificación sobre las imágenes parciales.

0 a 10 %	10 a 50 %	50 a 90 %	90 a 100 %
47,6 %	10,6 %	12,7 %	29,2 %

Esta tabla muestra los resultados en el caso de las imágenes parciales. Cómo era de esperar, están mucho más distribuidos. Analizando los datos tenemos que las bajas probabilidades, en general, se corresponden a imágenes donde las alcantarillas ocupan una parte muy reducida de la imagen, subiendo el porcentaje conforme más alcantarilla aparece.

Resulta también interesante ver ejemplos donde el clasificador ha dado una categoría incorrecta con un alto grado de confianza. En el caso de fallo de imágenes positivas, nos encontramos que el principal problema reside en momentos donde la iluminación es escasa, y esto provoca que nuestro clasificador interprete que no hay alcantarilla.

Figura 5.8 Ejemplo de falsos negativos.

En el otro extremo, tenemos imágenes clasificadas con una alta confianza cómo imágenes que tienen alcantarilla en ellas, cuando no la tienen. En este caso nos encontramos con imágenes que

tienen una forma en medio bastante oscura, rodeadas de zonas más iluminadas. Parece hasta cierto punto lógico este resultado para estos casos.

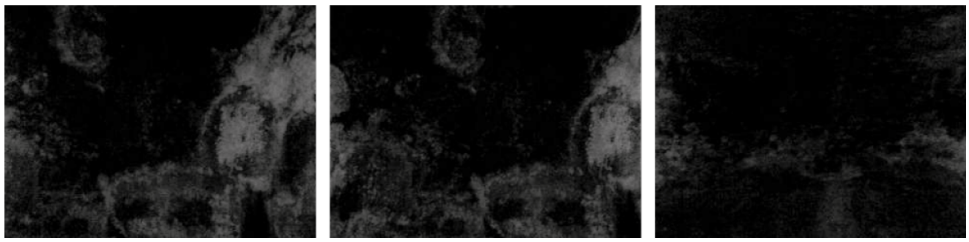

Figura 5.9 Ejemplos de falsos positivos.

Por último, se va a realizar una muestra de una gráfica en la que en el eje horizontal corresponde a las imágenes ordenadas tal y como se tomaron, y en el eje vertical se señalará la probabilidad de pertenecer a una clase u otra obtenida. Así podremos apreciar de forma clara si los puntos donde hay alcantarilla se diferencian del resto. Además, se ha incluido en el algoritmo de generación de la gráfica un filtro, que sería implementable en la realidad, que computa que varias imágenes seguidas tengan una probabilidad suficiente, de forma que nos evitamos que si aleatoriamente una imagen da una probabilidad elevada, la clasifiquemos positivamente.

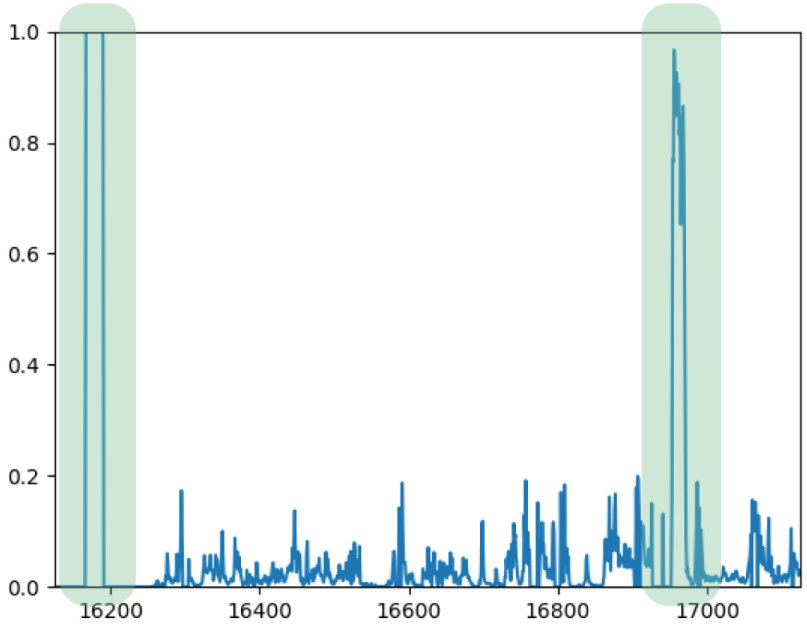

Figura 5.10 1000 primeras imágenes de la base de datos.

Apreciamos en estas imágenes que los dos picos de probabilidad corresponden efectivamente a las dos primeras bocas de alcantarilla encontradas. Luego observamos como las probabilidades del resto de la zona sin alcantarilla se mantienen bajas.

En este segundo ejemplo observamos la alcantarilla antes mencionada, primero en una gráfica donde se representan más imágenes, y luego más de cerca. Observamos que a pesar de tener parte de probabilidad baja, la boca de alcantarilla sigue siendo muy identificable.

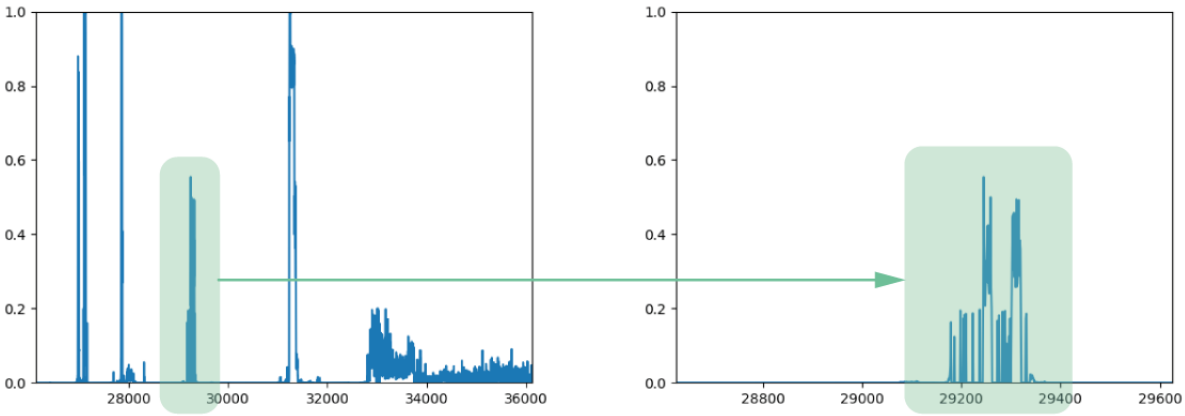

Figura 5.11 Figura de alcantarilla con menos probabilidad.

6 Conclusiones

Consecución de los objetivos

Podemos concluir en primer lugar que el clasificador obtenido cumple con el cometido para el que se desarrolló. El estudio realizado ha consistido en usar las imágenes RGB de una cámara de la plataforma robótica SIAR, y comprobar si con estas imágenes era factible realizar un clasificador que detectase si el robot se encontraba bajo un punto característico en el mapa del sistema de alcantarillado, en este caso concretos, una boca de alcantarilla. Los datos que arrojan la validación permiten concluir que sí, es posible este clasificador, y eso suficientemente robusto para usarlo con dicho fin.

El principal problema en el desarrollo ha consistido en el sobreajuste de la red neuronal, indicando esto que la complejidad del mismo no era muy elevado. Estamos acostumbrados a escuchar resultados de grandes investigaciones que usan este tipo de tecnología sobre problemas muy complejos, siendo éste un campo muy pujante en la actualidad. Esto posibilita que se hayan desarrollado herramientas con una comunidad muy grade detrás para el uso de estas tecnologías, permitiendo su aplicación en problemas no tan complejos, pero arrojando resultado muy válidos.

Desarrollos futuros

Sería interesante el haber comparado resultados del clasificador sobre imágenes tridimensionales, y ver si los errores se producen en las mismas situaciones, o si por el contrario cada uno tiene un comportamiento distinto. En dicho caso Sería igualmente interesante la posibilidad de investigar un uso conjunto de ambos.

También queda pendiente la implementación de este clasificador en la plataforma real para comprobar su uso en tiempo real.

Apéndice A
Lista de contenidos en los BAGS

Estos son los datos que contiene uno de los bags, la estructura y el tipo de los mensajes es igual en todos los casos, cambiando únicamente la cantidad de los datos.

Código A.1 Escritura de una ecuación.

```
path:       siar_2017-10-11-11-05-03.bag
version:    2.0
duration:   1hr 1:23s (3683s)
start:      Oct 11 2017 11:07:35.21 (1507712855.21)
end:        Oct 11 2017 12:08:59.20 (1507716539.20)
size:       17.0 GB
messages:   4218327
compression: none [22255/22255 chunks]
types:      geometry_msgs/PoseStamped [d3812c3cbc69362b77dc0b19b345f8f5]
            manhole_detector/Manhole [d72a9da8bfd11237ff1a1c706ed15351]
            nav_msgs/Odometry        [cd5e73d190d741a2f92e81eda573aca7]
            rssi_get/Nvip_status     [46e264d82a2ec13f9fceaaa5150831e9]
            sensor_msgs/CameraInfo   [c9a58c1b0b154e0e6da7578cb991d214]
            sensor_msgs/CompressedImage [8
               f7a12909da2c9d3332d540a0977563f]
            sensor_msgs/Imu          [6a62c6daae103f4ff57a132d6f95cec2]
            siar_driver/SiarStatus   [ca917eeaa438739625b131195aca95e5]
            tf2_msgs/TFMessage       [94810edda583a504dfda3829e70d7eec]
topics:     /arduimu_v3/imu                                      709806
   msgs   : sensor_msgs/Imu
            /back/depth_registered/camera_info                    21146
               msgs   : sensor_msgs/CameraInfo
            /back/depth_registered/image_raw/compressedDepth      21146
               msgs   : sensor_msgs/CompressedImage
            /back/rgb/camera_info                                 34487
               msgs   : sensor_msgs/CameraInfo
            /back/rgb/image_raw/compressed                        34487
               msgs   : sensor_msgs/CompressedImage
```

```
/back_left/depth_registered/camera_info                35035
    msgs   : sensor_msgs/CameraInfo
/back_left/depth_registered/image_raw/compressedDepth 35035
    msgs   : sensor_msgs/CompressedImage
/back_left/rgb/camera_info                             35568
    msgs   : sensor_msgs/CameraInfo
/back_left/rgb/image_raw/compressed                    35568
    msgs   : sensor_msgs/CompressedImage
/back_right/depth_registered/camera_info               35873
    msgs   : sensor_msgs/CameraInfo
/back_right/depth_registered/image_raw/compressedDepth 35873
     msgs  : sensor_msgs/CompressedImage
/back_right/rgb/camera_info                            35626
    msgs   : sensor_msgs/CameraInfo
/back_right/rgb/image_raw/compressed                   35626
    msgs   : sensor_msgs/CompressedImage
/baseline                                              35065
    msgs   : geometry_msgs/PoseStamped
/front/depth_registered/camera_info                    21508
    msgs   : sensor_msgs/CameraInfo
/front/depth_registered/image_raw/compressedDepth      21508
    msgs   : sensor_msgs/CompressedImage
/front/rgb/camera_info                                 34403
    msgs   : sensor_msgs/CameraInfo
/front/rgb/image_raw/compressed                        34403
    msgs   : sensor_msgs/CompressedImage
/front_left/depth_registered/camera_info               35277
    msgs   : sensor_msgs/CameraInfo
/front_left/depth_registered/image_raw/compressedDepth 35277
     msgs  : sensor_msgs/CompressedImage
/front_left/rgb/camera_info                            35564
    msgs   : sensor_msgs/CameraInfo
/front_left/rgb/image_raw/compressed                   35564
    msgs   : sensor_msgs/CompressedImage
/front_right/depth_registered/camera_info              35406
    msgs   : sensor_msgs/CameraInfo
/front_right/depth_registered/image_raw/compressedDepth
    35406 msgs : sensor_msgs/CompressedImage
/front_right/rgb/camera_info                           35564
    msgs   : sensor_msgs/CameraInfo
/front_right/rgb/image_raw/compressed                  35564
    msgs   : sensor_msgs/CompressedImage
/ground_truth                                             60
    msgs   : manhole_detector/Manhole
/odom                                                 128269
    msgs   : nav_msgs/Odometry
/rssi_nvip_2400                                         4788
    msgs   : rssi_get/Nvip_status
/siar_status                                          128269
    msgs   : siar_driver/SiarStatus
```

```
            /tf                                                    2277418
               msgs   : tf2_msgs/TFMessage       (3 connections)
            /up/depth_registered/camera_info                         36242
               msgs   : sensor_msgs/CameraInfo
            /up/depth_registered/image_raw/compressedDepth           36242
               msgs   : sensor_msgs/CompressedImage
            /up/rgb/camera_info                                      35627
               msgs   : sensor_msgs/CameraInfo
            /up/rgb/image_raw/compressed                             35627
               msgs   : sensor_msgs/CompressedImage
```

Apéndice B

Redes neuronales convolucionales

Código B.1 Primera red usada para clasificación.

```
model = Sequential()
model.add(Convolution2D(x, 5, 5, border_mode='valid', input_shape
    =(60, 80,1), activation='relu'))
model.add(Dropout(0.2))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Convolution2D(y, 3, 3, border_mode='valid', activation='
    relu'))
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(150, init='uniform', activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(50, init='uniform', activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1, init='uniform', activation='sigmoid'))
# Compile model
model.compile(optimizer='Adam', loss='binary_crossentropy', metrics
    =['accuracy'])

history = model.fit(X_train, Y_train, validation_data=(X_test,
    Y_test), nb_epoch=5, batch_size=32)
```

Código B.2 Red Final.

```
model = Sequential()
model.add(Convolution2D(5, 5, 5, border_mode='valid', input_shape
    =(60, 80,1), activation='relu'))
model.add(Dropout(0.4))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Convolution2D(5, 5, 5, border_mode='valid', activation='
    relu'))
model.add(Dropout(0.4))
```

```
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Convolution2D(5, 3, 3, border_mode='valid', activation='
        relu'))
    model.add(Dropout(0.4))
    model.add(Flatten())
    model.add(Dense(150, init='uniform', activation='relu'))
    model.add(Dropout(0.4))
    model.add(Dense(50, init='uniform', activation='relu'))
    model.add(Dropout(0.4))
    model.add(Dense(1, init='uniform', activation='sigmoid'))

    model.compile(optimizer='adagrad', loss='binary_crossentropy',
        metrics=['accuracy'])

    history = model.fit(X_train, Y_train, validation_data=(X_test,
        Y_test), nb_epoch=15, batch_size=32)
```

Índice de Figuras

Índice de Tablas

Índice de Códigos

Bibliografía

[1] I. Goodfellow, Y. Bengio y A. Courville, *Deep Learning*, Massachusetts, EEUU, 2016.

[2] C. M. Bishop, *Pattern Recongnition and Machine Learning*, Berkeley, EEUU, 2009.

[3] D. Alejo, F. Caballero y L. Merino *RGBD-based Robot Localization in Sewer Networks*, 2017.

[4] Stanford University, *CS231n Convolutional Neural Networks for Visual Recognition*, [En línea]. Available: http://cs231n.github.io/

[5] Y. LeCun, L. Bottou, G.B. Orr y Klaus-Robert Müller, *Efficient BackProp*, 1998. [En línea]. Available: http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf

[6] Y.Bengio, *Practical Recommendations for Gradient-Based Training of Deep Architectures*, 2012. [En línea]. Available: https://arxiv.org/pdf/1206.5533.pdf

[7] Y. LeCun, L. Bottou, Y. Bengio y P. Haffner, *Gradient-Based Learning Applied to Document Recognition*, 1998. [En línea]. Available: http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf

[8] A. Krizhevsky, I. Sutskever y G. E. Hinton, *ImageNet Classification with Deep ConvolutionalNeural Networks*. [En línea]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf