Aprendizaje profundo en imágenes para identificación de características de vehículos en movimiento en IoT
Deep learning in images for identification of features of moving vehicles in IoT

Trabajo de Fin de Máster
Curso 2023–2024

Autor
Juan Manuel Núñez Sánchez-Elvira

Director
Gonzalo Pajares Martinsanz

Máster en Internet de las Cosas
Facultad de Informática
Universidad Complutense de Madrid

Convocatoria: Junio 2024
Calificación: 9,5

Dedicatoria

Me gustaría dedicar este trabajo a mi familia por ser un gran apoyo durante todos estos años y por estar siempre ahí

Agradecimientos

Quiero expresar mi más sincero agradecimiento a Gonzalo, mi director del Trabajo de Fin de Master, por su invaluable orientación y apoyo a lo largo de este proyecto final del Máster en Internet de las Cosas. Su entusiasmo por el Aprendizaje Profundo no solo ha enriquecido mi perspectiva académica, sino que también ha subrayado la relevancia del IoT en el progreso de nuestra sociedad.

Agradezco profundamente su dedicación y esfuerzo durante todos estos años.

Resumen

En este trabajo se propone una solución conceptual dentro del paradigma IoT (Internet of Things). Más específicamente se trata de identificar características de vehículos en movimiento en vías urbanas o interurbanas, destacando como esencial la región de la matrícula y los caracteres que la conforman. Para ello se utiliza la cámara de un dispositivo móvil que captura imágenes. Para el procesamiento de estas imágenes se utilizan técnicas avanzadas de tratamiento en el ámbito de la Visión por Computador y por ende dentro de la Inteligencia Artificial.

El proceso consta de varias fases, de forma que en primer lugar se detectan los vehículos en movimiento mediante análisis del flujo óptico, a continuación, se utilizan modelos de redes neuronales convolucionales (CNN, Convolutional Neural Networks) para identificar tanto el propio vehículo como la región de la matrícula dentro de él.

Una vez reconocida la región de la matrícula, el siguiente paso consiste en identificar los caracteres que la conforman, para ello se utiliza un OCR (Optical Character Recognition) de carácter genérico.

Los resultados del procesamiento de las imágenes se almacenan en la nube, para ello se utiliza la plataforma ThingSpeak específica de IoT

Las diferentes funcionalidades que implementan los procesos mencionados, se integran para conformar la solución IoT de concepto integrada. Esta solución tiene como finalidad sentar las bases para un futuro despliegue hacia un entorno real, dentro de los futuros desarrollos de ciudades inteligentes (Smart Cities). Para cada uno de los distintos procesos se obtienen los resultados más relevantes, siendo el objetivo el proporcionar dicha solución conceptual, antes que la evaluación de las diferentes técnicas, que constituyen un análisis fuera del contexto IoT del presente trabajo.

En base a estas consideraciones se plantean los objetivos concretos y se evalúan las soluciones proporcionadas a través de los resultados obtenidos.

Palabras clave

Internet of Things (IoT), Flujo óptico, Convolutional Neural Networks (CNN), Optical Character Recognition (OCR), Procesamiento en la nube, Visión por Computador, Inteligencia Artificial.

Abstract

This paper proposes a conceptual solution within the IoT (Internet of Things) paradigm. More specifically, the aim is to identify characteristics of vehicles in motion on urban or interurban roads, with the region of the plate and the characters that make up the plate number plate being essential. For this purpose, the camera of a mobile device that captures images is used. For the processing of these images, advanced processing techniques are used in the field of Computer Vision and therefore within Artificial Intelligence.

The process consists of several phases, so that first the moving vehicles are detected by optical flow analysis, then Convolutional Neural Network (CNN) models are used to identify both the vehicle itself and the license plate region within it.

Once the license plate region is recognized, the next step is to identify the characters that make it up, using a generic OCR (Optical Character Recognition).

The results of the image processing are stored in the cloud using the IoT-specific ThingSpeak platform.

The different functionalities that implement the aforementioned processes are integrated to form the integrated IoT concept solution. The purpose of this solution is to lay the foundations for a future deployment towards a real environment, within the future developments of Smart Cities. For each of the different processes, the most relevant results are provided, the objective being to provide such conceptual solution, rather than the evaluation of the different techniques, which constitute an analysis outside the IoT context of this work.

Based on these considerations, the specific objectives are set and the solutions provided are evaluated through the results obtained.

Keywords

Internet of Things (IoT), Optical Flow, Convolutional Neural Networks (CNN), Optical Character Recognition (OCR), Cloud Computing, Computer Vision, Artificial Intelligence.

Índice

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

El propósito de este trabajo fin de máster es el estudio, implementación e integración de técnicas de visión artificial y aprendizaje automático para la detección de características generales en vehículos en movimiento, incluyendo el propio vehículo dentro de la imagen y el reconocimiento de la región que define la matrícula y los caracteres que la conforman, todo ello bajo el paradigma de Internet de las Cosas (Internet of Things, IoT).

No cabe la menor duda que la explosión tecnológica en la que estamos inmersos facilita enormemente el desarrollo de nuevas tecnologías cada vez más potentes. En este sentido, IoT se presenta como un paradigma de especial interés en distintos campos tecnológicos, entre los que cabe mencionar el control de vehículos tanto desde el punto de vista de regulación del tráfico como de los accesos a recursos disponibles de estos, por ejemplo zonas de aparcamiento. Estos aspectos cobran especial relevancia hoy en día, y están llamados a ser clave en los futuros desarrollos de ciudades inteligentes (*Smart Cities*).

En este sentido, son diversas las empresas tecnológicas orientadas hacia la consecución de estos objetivos. Tal es el caso de Lector Vision (2024) cuya línea de trabajo fundamental se ubica en el desarrollo de "soluciones integrales de visión artificial, tanto software como hardware, aplicadas al control de vehículos", dentro de las cuales se utilizan técnicas inteligentes, basadas en Visión Artificial, para el reconocimiento de matrículas en imágenes de vehículos. Dentro de dichas técnicas cabe mencionar las basadas en *Deep Learning* (DL) y obviamente en los denominados OCR (*Optical Character Recognition*).

Bajo estos mismos planteamientos y enfocados directamente en el ámbito de IoT se encuentran estrategias diversas tanto para el control y vigilancia del tráfico de vehículos como para su monitorización, mediante la identificación de matrículas, como parte de la identificación de los distintos tipos de vehículos. Tal es el caso de Gigabyte (2024), empresa tecnológica puntera que propone soluciones en este sentido con fines de monitorización del tráfico, control de velocidad y semáforos o de acceso a la ciudad y los estacionamientos habilitados. Se proponen soluciones tecnológicas incluyendo procesamiento de vídeo avanzado, aprendizaje profundo y envío de datos de la manera más eficiente posible dentro del sistema IoT. En este sentido la conectividad de los diferentes sistemas sensoriales y de procesamiento deviene en una característica esencial de las *Smart Cities* (AVUTEC, 2024).

Existen soluciones en la literatura que abordan el problema del reconocimiento de matrículas desde distintas perspectivas, en gran medida mediante aprendizaje automático. En

Wang et al. (2019) se realiza una revisión de distintas técnicas en tal sentido, incluyendo métodos de aprendizaje tradicionales como Scale-Invariant Feature Transform (SIFT) (Lowe, 1999) o Local Binary Patterns (LBP) (Ojala et al., 2002); así como técnicas basadas en Aprendizaje Profundo (He et al., 2019; Peng et al., 2019). Conviene reseñar que en estos dos últimos trabajos se realiza un procesamiento de las imágenes en dos etapas, de forma que en una primera fase se reconocen regiones candidatas a ser matrículas y en una segunda se llega al reconocimiento de los caracteres que componen la matrícula. El presente trabajo se inspira precisamente en esta estrategia, de forma que, como se verá en el planteamiento del mismo, existen también dos fases que progresan desde un análisis global a un refinamiento para procesar la matrícula misma. Por otra parte, dado el avance tecnológico tanto en el campo del Aprendizaje Profundo como en IoT, son varias las soluciones avanzadas incluyendo ambos paradigmas, tal es el caso de las estrategias mencionadas previamente como otros desarrollos específicos donde se prioriza el procesamiento de datos en el borde (edge) (Ammar et al., 2023).

En el presente trabajo se plantea una solución de concepto dentro del paradigma IoT, donde se aplican técnicas inteligentes, basadas en Deep Learning, sobre imágenes captadas con una cámara a color (sensor) y con tratamiento de la información en la nube.

1.1. Motivación

Dentro de las diferentes soluciones y estrategias presentadas por las compañías tecnológicas y las referencias mencionadas previamente destacan como esenciales las siguientes características:

1. Procesamiento de imágenes y/o vídeos

2. Utilización de métodos de aprendizaje profundo con fines de reconocimiento, en este caso de matrículas

3. Integración de tecnologías sensoriales (cámaras para captura de imágenes) bajo un ecosistema IoT

El punto de partida que suscita la motivación de este trabajo hay que situarlo en el desarrollo previo presentado en el trabajo Fin de Grado correspondiente al curso 2022-23 (Núñez et al., 2023). El autor de la presente memoria es, a su vez, co-autor del mencionado trabajo Núñez et al. (2023). En este último trabajo se plantea el reconocimiento de matrículas de vehículos de una forma muy preliminar utilizando técnicas de Visión por Computador y Aprendizaje Profundo, de tal suerte que tras su planteamiento se detectaron necesidades y carencias específicas de cara a su implantación en forma de un sistema verdaderamente operacional. Es aquí donde surge la idea de conformar y configurar un sistema con ciertas garantías de aplicabilidad en entornos reales donde la conectividad entre sensores, plataformas y procesamiento en la nube sea el hilo conductor hacia los desarrollos futuros dentro de las Smart Cities. Esta ampliación y mejora se propone considerando el paradigma IoT, que da cobertura al planteamiento, a la vez que soluciona las carencias detectadas. No obstante, el desarrollo de un sistema de esta naturaleza constituye un reto de especial relevancia, máxime teniendo en cuenta que los sistemas integrales inteligentes dentro de las Smart Cities se encuentran todavía en una fase de desarrollo operativa y especialmente en lo que respecta a la conectividad entre ellos.

Es bajo este planteamiento y teniendo en cuenta las perspectivas de futuro expresadas previamente, donde se encuadra este trabajo y donde surge la motivación para su desarrollo. En este sentido, y desde el punto de vista motivacional, se propone la integración de técnicas específicas para cada una de las características técnicas indicadas previamente y sobre todo dando cobertura a la conectividad indicada bajo el paradigma IoT. Cada una de las técnicas a integrar requiere de un proceso de adaptación propio para abordar el problema conjunto de identificación de la matrícula de un vehículo de forma integrada y especialmente si este se encuentra en movimiento. Desde el punto de vista de la reutilización de técnicas y métodos, parte del trabajo desarrollado en Núñez et al. (2023) se aprovecha en el desarrollo de la aplicación que se presenta, tanto desde el punto de vista teórico-conceptual como de implementación a nivel funcional y modular. Este planteamiento se justifica sobre la base de aprovechar las sinergias existentes y ya desarrolladas, a la vez que se concentra el esfuerzo en la integración y la conectividad necesarias en un planteamiento de estas características. Esta es la razón por la que determinados conceptos y desarrollos, suficientemente maduros, propuestos en Núñez et al. (2023) se aprovechan y reutilizan en el presente trabajo.

1.2. Objetivos

Como se ha mencionado previamente, el objetivo fundamental de este trabajo consiste en plantear una solución conceptual inteligente dentro del paradigma IoT para reconocimiento de matrículas de vehículos en movimiento y/o estáticos.

Bajo tal consideración surgen los siguientes objetivos específicos:

1. Analizar modelos IoT que permitan la captura de imágenes, su procesamiento inteligente mediante técnicas de DL, incluyendo OCRs y gestionar la información mediante una plataforma específica IoT.

2. Estudiar técnicas basadas en Visión por computador para análisis del movimiento en secuencias de imágenes.

3. Estudiar técnicas especificas de DL para el reconocimiento de imágenes con el fin de detectar tanto los vehículos en movimiento como las regiones donde se ubican las matrículas dentro de éstos.

4. Estudiar técnicas OCR para reconocimiento de los caracteres de la matrícula.

5. Verificar los distintos desarrollos e integrar las técnicas consideradas como factibles.

6. Extraer las conclusiones y opciones posibles tanto para el presente como en el futuro.

1.3. Plan de trabajo

El plan de trabajo describe el conjunto de actividades básicas tendentes a la consecución de los objetivos previos. En cualquier caso, en relación con el planteamiento de dichas actividades, se contempla la necesidad de revisión de las mismas según se avanza en su desarrollo, con el fin de detectar deficiencias y subsanarlas en un proceso iterativo por naturaleza.

En este sentido, las actividades planteadas y realizadas son las siguientes:

1. Estructuración de la solución tecnológica a plantear en el trabajo, definición y alcance de los objetivos, así como la determinación de los plazos temporales o hitos durante el desarrollo.

2. Estudio de las tecnologías existentes desde los siguientes puntos de vista: métodos de procesamiento de imágenes, técnicas de reconocimiento mediante DL y manejo de datos a través de una plataforma IoT.

3. Desarrollo de un procedimiento basado en detección de movimiento en secuencias de imágenes para identificar zonas de interés relativas a vehículos en circulación.

4. Desarrollo de métodos basados en Redes Neuronales Convolucionales, dentro del paradigma DL para reconocimiento de regiones de interés en las imágenes, incluyendo tanto la zona donde se encuentra la matrícula como posteriormente el reconocimiento de los caracteres que la conforman.

5. Configuración de una plataforma IoT para la gestión de la información extraída a partir de las secuencias de imágenes.

6. Integración de los desarrollos y configuraciones indicadas previamente.

7. Redacción y elaboración de la memoria.

1.4. Organización de la memoria

La memoria se estructura, atendiendo a las consideraciones previas, de forma que en el capítulo dos se describen los conceptos teóricos relativos al procesamiento de imágenes, distinguiendo entre técnicas básicas y avanzadas. En el capítulo tres se introducen los conceptos asociados al DL, incluyendo varios modelos de redes neuronales convolucionales. En el capítulo cuatro se describe el diseño modular del sistema IoT completo con sus distintas partes integradas. En el capítulo cinco se muestran los resultados obtenidos mediante la aplicación desarrollada. Finalmente, en el capítulo seis se exponen las conclusiones y trabajo futuro.

Capítulo 2

Técnicas de procesamiento de imágenes

2.1. Introducción

Dentro de la Visión Artificial cabe distinguir dos enfoques característicos. El primero de ellos orientado a preparar la imagen como paso previo al segundo, que se orienta al análisis de aquella en función de su contenido. Es lo que se denominan técnicas básicas y avanzadas, que se abordan en este capítulo.

Las técnicas básicas comprenden el estudio de los modelos de color como parte esencial de las imágenes analizadas, la binarización como paso previo a la identificación de regiones de interés, seguido de la aplicación de operaciones morfológicas para depurar tales regiones. Las técnicas avanzadas se enfocan directamente hacia el análisis de las imágenes, para detectar, por un lado regiones donde se produce el movimiento (flujo óptico) y por otro, reconocimiento de caracteres (OCR).

Los fundamentos teóricos, tanto de las técnicas básicas como avanzadas, se basan en Pajares and de la Cruz (2007) que se utiliza como texto de referencia del que se extraen y reproducen las partes esenciales de los mismos aplicables a este trabajo.

2.2. Técnicas básicas

2.2.1. Modelos de color

2.2.1.1. El modelo RGB

En el modelo RGB, cada color se representa a través de sus componentes espectrales primarias: rojo, verde y azul. Este modelo se fundamenta en el sistema de coordenadas cartesianas. El subespacio de color relevante es un tetraedro, como se muestra en la figura 2.1. En este tetraedro, los valores RGB se encuentran en tres vértices, mientras que cian, magenta y amarillo ocupan los otros tres vértices. El negro está en el origen y el blanco en el vértice más distante del origen. En este contexto, la escala de grises se extiende del negro al blanco a lo largo de la diagonal que conecta estos dos puntos, y los colores son puntos dentro del tetraedro, definidos por vectores que parten del origen. Por simplicidad, se considera que todos los vectores están normalizados, lo que convierte al tetraedro de la figura 2.1 en un tetraedro unitario, con todos los valores de R, G y B dentro del rango [0,1]. Las imágenes en este modelo se generan combinando en diferentes proporciones cada

uno de los colores primarios RGB. Esto es lo que sucede en los fotosensores de monitores y cámaras de vídeo, por ejemplo.

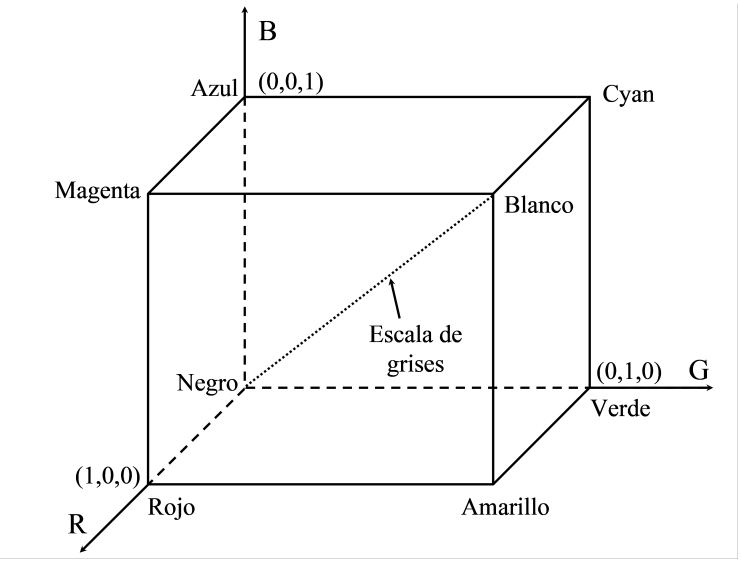


Figura 2.1: Tetraedro de color RGB. Los puntos a lo largo de la diagonal principal tienen valores de gris, desde el negro en el origen al punto (1,1,1)

2.2.1.2. El modelo HSI

El matiz es una característica del color que describe su pureza (por ejemplo, amarillo puro, rojo, naranja), mientras que la saturación indica el grado en que el color puro está mezclado con luz blanca. El modelo de color HSI es útil por dos razones principales. Primero, la componente de intensidad (I) puede separarse de la información del color en la imagen. Segundo, las componentes de matiz y saturación están estrechamente relacionadas con la forma en que los humanos perciben el color. Estas características hacen del modelo HSI una herramienta ideal para desarrollar algoritmos de procesamiento de imágenes basados en la percepción del color del sistema visual humano. La representación gráfica de este modelo se muestra en la figura 2.2.

2.2.1.3. Conversión de RGB a HSI

Como se mencionó anteriormente, el modelo RGB se define respecto a un tetraedro unitario. Sin embargo, las componentes de color del modelo HSI (matiz y saturación) se definen respecto al triángulo de color mostrado en la figura 2.2(a). Este triángulo está en un plano, por lo que no hay perspectiva y todos los puntos en él son coplanares. El punto W es el punto de intersección de las medianas del triángulo. Es importante recordar que en el diagrama de cromaticidad, todos los colores que se obtienen mediante la combinación de tres colores específicos se encuentran dentro de un triángulo cuyos vértices están definidos por esos tres colores. En la figura 2.2(a), el matiz (H) de un punto de color P es el ángulo del vector mostrado respecto al eje rojo. Así, cuando H = 0^{o}, el color es rojo; cuando H = 60^{o}, el color es amarillo; cuando H = 120^{o}, el color es verde, y así sucesivamente. La saturación S del punto de color P es el grado de dilución del color con blanco y es proporcional a la distancia de P al centro del triángulo. A medida que P se aleja de W y se acerca a cualquiera de los lados del triángulo, la saturación del color aumenta.

La intensidad en el modelo HSI se mide con respecto a una línea perpendicular al triángulo que pasa por su centro. Las intensidades a lo largo de esta línea disminuyen hacia el negro por debajo del triángulo y aumentan hacia el blanco por encima del triángulo.

Si se combina matiz, saturación e intensidad en un espacio de color tridimensional se obtiene la representación piramidal de la figura 2.2(b). En esta representación, cada triángulo marcado con líneas más gruesas corresponde a un nivel específico de intensidad. Una vez establecido el nivel de intensidad, se aplican los conceptos explicados en la figura 2.2(a) para determinar el matiz y la saturación. Cualquier punto en la superficie de la estructura piramidal representa un color saturado puro, ya que son puntos situados en los lados de los triángulos. El matiz de dicho color se determina por su ángulo respecto al eje rojo y su intensidad por su distancia perpendicular desde el punto correspondiente al negro (es decir, cuanto mayor es la distancia desde el negro, mayor es la intensidad del color). Todo lo anterior es aplicable a puntos dentro de la estructura, con la única diferencia de que los colores se vuelven menos saturados a medida que se acercan al eje vertical que une el blanco con el negro.

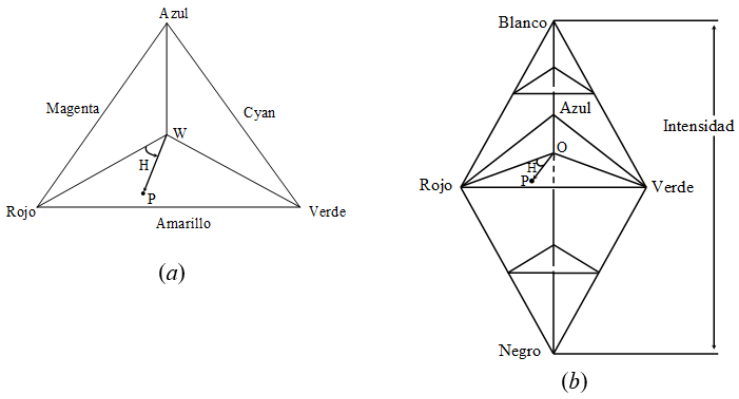


Figura 2.2: (a) Triángulo de color HSI; (b) Color e intensidad en el modelo HSI

En el modelo HSI, los colores se definen en relación con los valores normalizados de rojo, verde y azul, expresados en términos de los colores primarios RGB mediante

$$r = \frac{R}{R+G+B}; \qquad g = \frac{G}{R+G+B} \qquad b = \frac{B}{R+G+B} \tag{2.1}$$

donde

$$r + g + b = 1 \tag{2.2}$$

Mientras que R, G y B pueden tomar el valor de 1 simultáneamente, las variables normalizadas deben satisfacer la ecuación 2.2. Esta ecuación representa el plano que contiene el triángulo HSI.

Para cualquier combinación de tres componentes de color R, G y B, cada una en el rango [0,1], la componente de intensidad en el modelo HSI se define como:

$$I = \frac{1}{3}(R+G+B) \tag{2.3}$$

cuyos valores están en [0,1].

2.2.2. Binarización

2.2.2.1. Introducción

Las imágenes están compuestas por píxeles, y las técnicas de filtrado de imágenes se encargan de transformar estos píxeles. Las únicas características de un píxel son su posición y su nivel de gris o intensidad.

En las imágenes, ciertas áreas o zonas se caracterizan por ser agrupaciones de píxeles conectados entre sí. Además de la conexión, estos píxeles comparten propiedades o características comunes; estas áreas se denominan regiones. A continuación, se presentan diferentes técnicas para extraer las regiones de interés de una imagen. Posteriormente, mediante la aplicación de técnicas de flujo óptico o reconocimiento de caracteres, se extraen propiedades de estas regiones que permiten diferenciarlas entre sí.

Se estudian técnicas de binarización, considerando primero que los píxeles de una región determinada presentan una distribución de intensidad similar. Por tanto, a partir del histograma de niveles de gris se puede determinar la zona correspondiente a una región específica. Entre las técnicas de binarización destacan el método de Otsu y la técnica conocida como *Maximal Stable Extremal Regions* (MSER), ambas descritas más adelante.

2.2.2.2. Binarización mediante detección de umbral

El uso de umbrales en el procesamiento de imágenes es una de las técnicas principales en los sistemas de visión artificial para la detección de objetos, especialmente en aplicaciones que requieren procesar grandes cantidades de datos.

Supongamos que el histograma de intensidad mostrado en la figura 2.3 (izquierda) corresponde a una imagen $f(x, y)$ compuesta por objetos claros sobre un fondo oscuro, donde los píxeles del objeto y del fondo tienen intensidades agrupadas en dos tonos dominantes. Una forma evidente de extraer los objetos del fondo es seleccionar un nivel T que separe los dos tonos de intensidad. De esta manera, un píxel (x, y) para el cual $f(x, y) > T$ será considerado un píxel del objeto; en caso contrario, el píxel pertenecerá al fondo. En la figura 2.3 (derecha) se muestra un caso ligeramente más complejo del uso de esta técnica. Aquí, el histograma tiene tres modos dominantes (por ejemplo, dos tipos de objetos sobre un fondo oscuro). El proceso es similar: se clasifica un punto como perteneciente a una de las clases de objetos si $T_1 < f(x, y) \leq T_2$, a la otra clase de objeto si $f(x, y) > T_2$ y al fondo si $f(x, y) \leq T_1$.

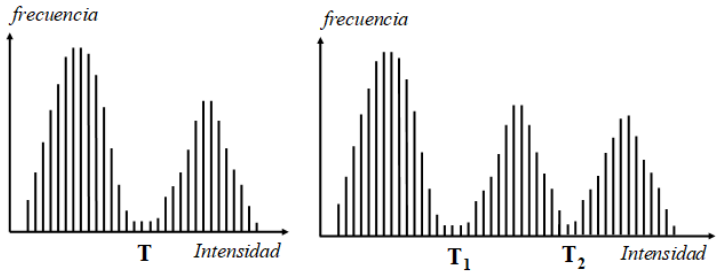


Figura 2.3: Representación de histogramas bimodal y trimodal

Los procedimientos que utilizan umbrales multinivel suelen ser menos confiables que

aquellos que emplean un umbral simple debido a la dificultad de establecer múltiples umbrales que aíslen efectivamente las regiones de interés.

Basándonos en los conceptos anteriores, podemos considerar la fijación de umbral como una operación que implica pruebas con respecto a una función T de la forma:

$$T = T[x, y, p(x, y), f(x, y)] \tag{2.4}$$

donde $f(x, y)$ es la intensidad en el punto (x, y) y $p(x, y)$ es alguna propiedad local del punto; por ejemplo, la intensidad media de un entorno de vecindad centrado en (x, y). Se creará una imagen binaria $g(x, y)$ definiendo:

$$g(x, y) = \begin{cases} 0 & \text{si } f(x, y) > T \\ 1 & \text{si } f(x, y) \leq T \end{cases} \tag{2.5}$$

Examinando $g(x, y)$, se observa que los píxeles con valor 0 corresponden a los objetos, mientras que los píxeles con valor 1 corresponden al fondo.

Cuando T depende únicamente de $f(x, y)$, el umbral se llama global. Si T depende tanto de $f(x, y)$ como de $p(x, y)$, entonces el umbral se llama local. Si T depende de las coordenadas espaciales x e y, se llama umbral dinámico.

En la figura 2.4 se muestra un ejemplo de umbral global. En (a) se presenta la imagen original con una serie de objetos en tonos de gris más oscuros que el fondo. En (b) se muestra el correspondiente histograma de niveles de gris en el rango de 0 a 255, donde aparece el valor de umbral global $T = 198$ que separa los dos modos del histograma. Finalmente, en (c) se muestra la imagen binarizada donde los objetos aparecen en negro (nivel de gris 0) y el fondo en blanco (nivel de gris 255).

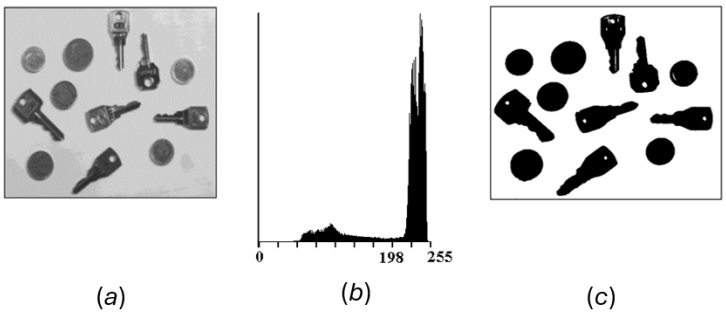


Figura 2.4: (a) Imagen original; (b) Histograma de intensidades en el rango de 0 a 255; (c) Imagen resultante binarizada obtenida utilizando la ecuación 2.5 con un umbral global de 198

2.2.2.3. Método de Otsu

Para el histograma bimodal, se asume que la imagen presenta un histograma compuesto por la suma de dos distribuciones de probabilidad gaussianas, cada una aproximando uno de los lóbulos. Esto implica que, conforme las gaussianas se asemejan al histograma real, las desviaciones estándar deberían disminuir. En consecuencia, se debe seleccionar el umbral que minimice la suma de las varianzas de los dos lóbulos del histograma (Otsu, 1979).

Considerando una imagen con L niveles de intensidad y asumiendo que el umbral buscado es T, las probabilidades acumuladas hasta T y desde T hasta L son,

$$w_1(t) = \sum_{z=1}^{T} P(z) \text{y} w_2(t) = \sum_{z=T+1}^{L} P(z) \tag{2.6}$$

A continuación se calculan las medias y varianzas respectivas,

$$\mu_1(t) = \sum_{z=1}^{T} zP(z) \text{y} \mu_2(t) = \sum_{z=T+1}^{L} zP(z) \tag{2.7}$$

$$\sigma_1^2(t) = \sum_{z=1}^{T} (z - \mu_1(t))^2 \frac{P(z)}{w_1(t)} \text{y} \sigma_2^2(t) = \sum_{z=T+1}^{L} (z - \mu_2(t))^2 \frac{P(z)}{w_2(t)} \tag{2.8}$$

Finalmente, se determina la varianza ponderada

$$\sigma_w^2(t) = w_1(t)\sigma_1^2(t) + w_2(t)\sigma_2^2(t) \tag{2.9}$$

Se selecciona el umbral T que corresponde al nivel de intensidad que minimiza la varianza ponderada definida en (2.9).

2.2.2.4. MSER: Regiones Extremas Máximamente Estables

El método conocido como Maximally Stable Extremal Regions (MSER) (Matas et al. (2002); Chen et al. (2011)) tiene como objetivo encontrar un umbral óptimo para obtener un conjunto de regiones que sean lo más estables y óptimas posible.

Para ilustrar de manera intuitiva el algoritmo, consideremos un conjunto de imágenes que muestran el carácter G umbralizado, como se muestra en la figura 2.5. La región que define el carácter G en este caso se conoce como una región conectada (R). En el contexto del procesamiento de imágenes, una región conectada es un área de la imagen donde cada píxel puede ser accedido desde cualquier otro píxel dentro de esa área. En cada caso, se indica el valor de umbral aplicado (U), que varía en incrementos de 40, según la secuencia U = 40, 80, 120, 160, 200, 240 y 255 (el máximo valor de intensidad en imágenes de 8 bits). Para cada uno de estos valores de umbral, se registra el área del carácter G, que representa el número de píxeles que pertenecen a la región conectada que define el carácter G.

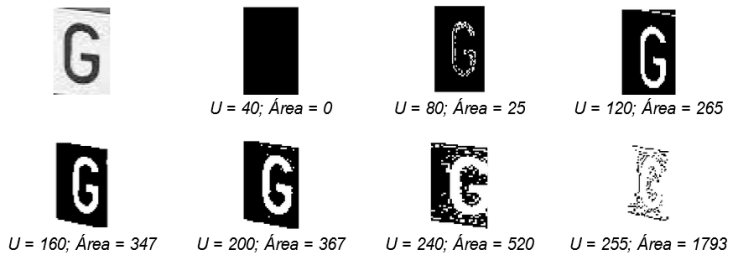


Figura 2.5: Imágenes con diferentes valores de umbral (U) y las áreas obtenidas del carácter G

Bajo estas consideraciones, la región conectada que define el carácter G se identifica como una región MSER debido a que el área de la región conectada no varía significativamente en el rango de umbral de 160 a 200; específicamente, el área es de 347 para U = 160

y de 377 para U = 200. Por lo tanto, la región definida con U = 160 representa la región deseada para este carácter.

2.2.3. Operaciones morfológicas

2.2.3.1. Principios y transformaciones básicas

La morfología matemática se basa en las propiedades de los conjuntos de puntos y en resultados de la geometría integral y la topología. Parte del supuesto de que las imágenes reales se pueden modelar mediante conjuntos de puntos en cualquier dimensión, como el espacio euclídeo N-dimensional. En particular, el espacio euclídeo bidimensional (E^2) y sus subconjuntos son apropiados para describir formas planas. Desde esta perspectiva, se utilizan operaciones habituales en conjuntos, tales como inclusión ($\subset$ o $\supset$), unión ($\cup$), intersección ($\cap$), complemento (^{c}) y conjunto vacío ($\emptyset$). La diferencia de conjuntos se expresa como $X - Y = X \cap Y^c$.

En visión artificial, se usa el equivalente digital del espacio euclídeo, donde los conjuntos están formados por pares de enteros en imágenes binarias o por tripletes en imágenes en escala de grises.

Cada punto se representa como un par de enteros correspondientes a las coordenadas de la imagen digital. Por tanto, una imagen binaria puede ser vista como un conjunto de puntos 2D. Los puntos que pertenecen a los objetos de la imagen forman un conjunto X, donde esos puntos son píxeles con valor binario uno. Los puntos del conjunto complementario X^c representan el fondo y tienen un valor binario de cero. Observando la figura 2.6, el origen, marcado con un punto, tiene coordenadas (0,0). Las coordenadas de cualquier punto se interpretan como (posición en la fila, posición en la columna) respecto al origen. En esta figura, los puntos pertenecientes al objeto están marcados con "1". Cualquier punto x en una imagen discreta X puede considerarse un vector desde el origen (0,0).

$$\begin{bmatrix} \bullet\,0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figura 2.6: Ejemplo de conjunto de puntos

En este caso, el conjunto de puntos X, considerando solamente los valores lógicos 1, es: X = (0,1), (0,2), (0,3), (1,1), (1,2), (1,3), (2,2), (2,3), (3,2), (3,3).

Una transformación morfológica Φ se define por la relación de la imagen (conjunto de puntos X) con otro pequeño conjunto de puntos B, conocido como elemento estructural. B se representa respecto a un origen local O (denominado punto representativo). Algunos elementos estructurales típicos (también desde el punto de vista binario) se muestran en la figura 2.7. En esta figura, el punto representativo se marca con un punto. El elemento estructural en la figura 2.7(a) tiene la siguiente expresión en forma de conjunto de puntos: B = (-1, -1), (-1,0), (-1,1), (0, -1), (0,0), (0,1), (1, -1), (1,0), (1,1), mientras que el conjunto en la figura 2.7(b) es: B = (-1,0), (0,-1), (0,0), (0,1), (1,0). La figura 2.7(c) presenta un

elemento estructural donde el punto O no figura como miembro de B, y estaría representado como B = (0,-1), (0,1), donde el punto (0,0) está ausente debido a su valor binario cero.

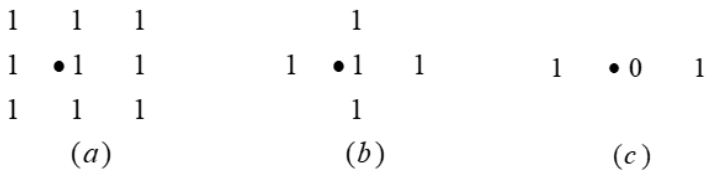


Figura 2.7: Elementos estructurales típicos

La transformación morfológica $\Phi(X)$ aplicada a la imagen X implica que el elemento estructural B se desplaza por toda la imagen. Al posicionar B sobre un punto de la imagen, el píxel correspondiente al punto representativo O se denomina píxel actual. El resultado de la relación (cero o uno) entre la imagen X y el elemento estructural B en la posición actual se almacena en dicho píxel. Las transformaciones morfológicas básicas incluyen dilatación, erosión, apertura y cierre, que se analizan a continuación.

2.2.3.2. Dilatación

La transformación morfológica de dilatación $\oplus$ agrega dos conjuntos mediante la adición de vectores (o adición de conjuntos de Minkowski). La dilatación $X \oplus B$ es el conjunto de puntos resultante de todas las posibles sumas vectoriales entre pares de elementos de los conjuntos X y B.

$$X \oplus B = \left\{ d \in E^2 : d = x + b \;\; para \;\; cada \;\; x \in X \; y \; b \in B \right\} \tag{2.10}$$

La figura 2.8 muestra un ejemplo de dilatación:

$X = \{(0,1),(1,2),(2,0),(2,1),(3,0),(3,1)\}$,
$B = \{(0,0),(0,1)\}$,
$X \oplus B = \{(0,1),(1,2),(2,0),(2,1),(3,0),(3,1),(0,2),(1,3),(2,2),(3,2)\}$

En la figura 2.9(a) se aprecia una imagen binaria original (el emblema de la Universidad Complutense de Madrid) y en (b) se observa el resultado tras su dilatación usando el elemento estructural de la figura 2.7(a). Esta dilatación es isotrópica (actúa igual en todas las direcciones) y expande el objeto un píxel, a menudo denominado rellenado o crecimiento.

$$\begin{bmatrix} \bullet 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \oplus \begin{bmatrix} \bullet 1 & 1 \end{bmatrix} = \begin{bmatrix} \bullet 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figura 2.8: Ejemplo de dilatación

La dilatación con un elemento estructural isotrópico de 3x3 puede interpretarse como una transformación que altera todos los píxeles de fondo vecinos al objeto.

(a) (b)

Figura 2.9: (a) Imagen binaria original; (b) Dilatación como expansión isotrópica con el elemento estructural de la figura 2.7(a)

La dilatación posee propiedades interesantes que facilitan su implementación tanto en software como en hardware. A continuación se presentan algunas de estas propiedades sin demostración. Para un análisis más profundo, véase Serra (1982) o Haralick et al. (1987). La operación de dilatación es conmutativa,

$$X \oplus B = B \oplus X \tag{2.11}$$

y también asociativa

$$X \oplus (B \oplus D) = (X \oplus B) \oplus D \tag{2.12}$$

y además es invariante a la traslación

$$X_h \oplus B = (X \oplus B)_h \tag{2.13}$$

La dilatación es una transformación creciente,

$$Si \quad X \subseteq Y \quad entonces \quad X \oplus B \subseteq Y \oplus B \tag{2.14}$$

La figura 2.10 ilustra el resultado de la dilatación si el punto representativo no es un miembro del elemento estructural B, en cuyo caso el resultado de la dilatación difiere del conjunto de entrada,

$X = \{(0,1),(1,2),(2,1),(2,2),(3,1),(3,2),(3,4),(4,4)\}$,
$B = \{(0,-1),(0,1)\}$,
$X \oplus B = \{(0,0),(1,1),(2,0),(2,1),(3,0),(3,1),(3,3),(4,3),$
$(0,2),(1,3),(2,3),(3,3),(3,5),(4,5)\}$

$$\begin{bmatrix} \bullet 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \oplus \begin{bmatrix} 1 & \bullet 0 & 1 \end{bmatrix} = \begin{bmatrix} \bullet 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$

Figura 2.10: Dilatación donde el punto representativo no es miembro del elemento estructural y donde la imagen resultante aumenta el número de filas con respecto a la imagen original

Obsérvese que la imagen resultante ha aumentado de tamaño pasando de 5 columnas a 6, ello es debido a la presencia de los unos en la $5^{\underline{a}}$ columna de la imagen original. Esto mismo puede suceder con las filas si la imagen original tiene unos en la fila cero o última, veamos con la misma imagen el aumento de las filas en la imagen resultante para la misma imagen de entrada por el siguiente elemento estructural

$X = \{(0,1),(1,2),(2,1),(2,2),(3,1),(3,2),(3,4),(4,4)\}$,
$B = \{(0,-1),(0,0),(0,1)\}$,
$X \oplus B = \{(-1,1),(0,2),(1,1),(1,2),(2,1),(2,2),(2,4),(3,4),$
$(0,1),(3,1),(3,2),(4,4),(1,3),(2,3),(3,3),(3,5),(4,5)\}$

$$\begin{bmatrix} \bullet 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \oplus \begin{bmatrix} 1 & 0 \\ \bullet 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ \bullet 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

Figura 2.11: Dilatación donde el punto representativo es miembro del elemento estructural y donde la imagen resultante aumenta el número de filas y columnas con respecto a la imagen original

Gráficamente, el proceso de dilatación se realiza de la siguiente manera: se recorre la imagen de izquierda a derecha y de arriba abajo. Cuando se encuentra un píxel con el valor 1, se coloca el origen del elemento estructural sobre ese píxel. En esta posición, se lleva a cabo la unión del elemento estructural con la parte de la imagen con la que se solapa. Si todos los píxeles de valor uno en el elemento estructural coinciden con píxeles de valor uno en la imagen, se marca el píxel de la imagen correspondiente al origen del elemento estructural con el valor 1. Este método gráfico es el propuesto por Low et al. (1991).

Aplicando a 2.9(a) el elemento estructural de la figura 2.10 se obtiene la imagen dilatada de la figura 2.12.

Figura 2.12: Dilatación de la figura 2.9(a) con el elemento estructural de la figura 2.10

2.2.3.3. Erosión

La transformación morfológica de la erosión $\otimes$ combina dos conjuntos mediante la substracción de vectores. Es dual de la dilatación. Ni la erosión ni la dilatación son transformaciones invertibles.

$$X \otimes B = \left\{ d \in E^2 : d + b \ \in X \ para \ cada \ b \in B \right\} \tag{2.15}$$

Este enunciado implica que cada punto d del conjunto X, que en nuestro caso será la imagen, es evaluado. El resultado de la erosión se obtiene para los puntos d donde todos los posibles $d + b$ están en X. La figura 2.13 muestra un ejemplo del conjunto de puntos X erosionados por el elemento estructural B.

$X = \{(0,2),(1,2),(2,0),(2,1),(2,2),(2,3),(3,2),(4,2)\}$,
$B = \{(0,0),(0,1)\}$,
$X \otimes B = \{(2,0),(2,1),(2,2)\}$

$$\begin{bmatrix} \bullet 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \otimes \begin{bmatrix} \bullet 1 & 1 \end{bmatrix} = \begin{bmatrix} \bullet 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figura 2.13: Erosión

Gráficamente, la erosión se lleva a cabo de la siguiente manera: se recorre la imagen de izquierda a derecha y de arriba abajo. Cuando se encuentra un píxel con valor 1, se coloca el origen del elemento estructural sobre este. Si todos los píxeles con valor 1 en el elemento estructural coinciden con píxeles de valor 1 en la imagen, se marca el píxel de la imagen en el que está el origen del elemento estructural con el valor 1. Este método gráfico es el que utiliza Low et al. (1991).

2.2.3.4. Apertura y cierre

La erosión y la dilatación son transformaciones no invertibles. Si se aplica primero una erosión y luego una dilatación a una imagen, no se recupera la imagen original; el resultado es una imagen más simplificada y menos detallada.

La erosión seguida de una dilatación da lugar a una transformación morfológica importante llamada apertura. La apertura de una imagen X por un elemento estructural B se denota como $X \circ B$ y se define como:

$$X \circ B = (X \otimes B) \oplus B \tag{2.16}$$

Por otro lado, la dilatación seguida de una erosión da lugar a la transformación morfológica llamada cierre. El cierre de una imagen X por un elemento estructural B se denota como $X \bullet B$ y se define como:

$$X \bullet B = (X \oplus B) \otimes B \tag{2.17}$$

Si una imagen X no cambia después de ser sometida a apertura respecto al elemento estructural B, se dice que es abierta respecto a B. Análogamente, si una imagen X no cambia tras el cierre con respecto al elemento estructural B, se dice que es cerrada respecto a B.

La apertura y el cierre con un elemento estructural isótropo se utilizan para eliminar detalles específicos de la imagen que sean menores que el elemento estructural.

Estas operaciones no distorsionan la forma global de los objetos. El cierre conecta objetos cercanos, rellena pequeños huecos y suaviza los contornos del objeto, mientras que la apertura produce el efecto contrario (Low et al., 1991). Los términos pequeño y cercano están relacionados con el tamaño y la forma del elemento estructural.

La apertura y el cierre se ilustran en la figura 2.14(a) y (b), respectivamente. Ambas transformaciones se aplican sobre la imagen binaria de la figura 2.9(a) utilizando el elemento estructural mostrado en la figura 2.7(a).

Tanto la apertura como el cierre son invariantes a la traslación del elemento estructural.

Dado que la dilatación y la abertura son transformaciones incrementales, tanto la apertura como el cierre también lo son.

La apertura es anti-extensiva ($X \circ B \subseteq X$) y el cierre es extensivo ($X \subseteq X \bullet B$).

La apertura y el cierre, al igual que la dilatación y la erosión, son transformaciones duales:

$$(X \bullet B)^c = X^c \circ \breve{B} \tag{2.18}$$

(a) (b)

Figura 2.14: (a) Apertura; (b) Cierre. Ambas aplicadas a la imagen de la figura 2.9(a) con el elemento estructural isótropo 3x3 de la figura 2.7(a)

2.3. Técnicas avanzadas

2.3.1. Flujo óptico

La detección del movimiento en secuencias de imágenes está basada en el concepto de flujo óptico, cuya descripción teórica se describe a continuación. Para ello, en primer lugar, se analiza el concepto de flujo en función del movimiento de los objetos en la escena que se obtiene por la diferencia que aparece entre dos imágenes consecutivas de una misma secuencia. En segundo lugar, se describen tres métodos para el cómputo del flujo óptico, de entre los existentes y que son los que se utilizan en el presente trabajo, a saber: Lucas-Kanade, Horn-Schunck y Farneback

Como se ha mencionado previamente, la detección del movimiento en secuencias de imágenes está basado en el concepto de flujo óptico, cuya descripción teórica se encuentra en Davies (2018) y mayormente en Pajares and de la Cruz (2007). Referencia esta última de donde se toman los conceptos principales relativos al mencionado concepto de flujo óptico.

Una representación bidimensional de un movimiento tridimensional se denomina *campo de movimiento*, donde cada punto tiene asignado un *vector velocidad* correspondiente a la dirección del movimiento, velocidad y distancia a partir de un observador en una localización apropiada de la imagen. Un enfoque diferente analiza el movimiento considerando el concepto de *flujo óptico*, que requiere un pequeño intervalo temporal entre imágenes consecutivas sin que los cambios sean excesivamente grandes entre esas imágenes. La obtención del flujo óptico desemboca en la determinación de la dirección del movimiento y de la velocidad del movimiento en todos los puntos de la imagen. El objetivo inmediato del análisis de imágenes basado en el flujo óptico es determinar el campo de movimiento. Como se verá posteriormente, el flujo óptico no siempre se corresponde con el verdadero campo de movimiento porque los cambios de iluminación de la escena también se reflejan en el flujo óptico. Los parámetros del movimiento de objetos pueden derivarse de los vectores obtenidos del flujo óptico. En realidad, las estimas del flujo óptico o correspondencias de puntos son ruidosas. Además, la interpretación tridimensional del movimiento está mal condicionada y requiere una alta precisión del flujo óptico o correspondencias de puntos.

El campo de movimiento o *campo de velocidad* representa una técnica de compromiso; se determina una información similar al flujo óptico pero basada en las imágenes adquiridas en distintos instantes de tiempo, que son demasiado cortos con el fin de asegurar pequeños cambios debido al movimiento. En la figura 2.15 se puede apreciar el campo de movimiento, que corresponde al caso de traslación pura. La dirección de traslación de la cámara es a lo largo del rayo de proyección que pasa por el punto donde todos los vectores de campo divergen. En el supuesto de que la cámara se aleje de la escena dicho punto será de convergencia. El punto de convergencia o divergencia de todos los vectores de movimiento de campo se denomina *foco de expansión* (FOE) o *foco de contracción* (FOC) respectivamente (Nalwa, 1993).

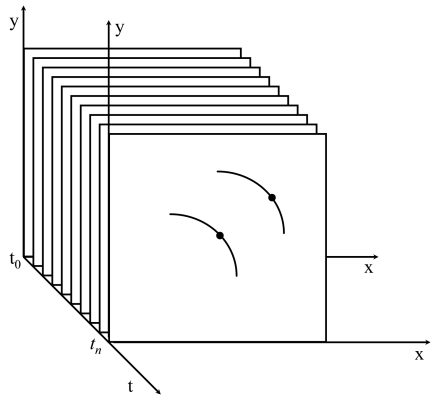


Figura 2.15: Representación espacio-temporal del movimiento

La captura de imágenes con una cámara en movimiento introduce una dimensión extra: la dimensión de un punto de vista variable. Suponiendo que la cámara tiene una velocidad constante y que captura imágenes a intervalos de tiempo iguales, entonces la dimensión que el movimiento de la cámara proporciona a los datos de la imagen puede ser descrita como una *dimensión temporal*. Por consiguiente, lo que se obtiene es la variación de la intensidad sobre el plano de la imagen y sobre el tiempo, que se conoce como variación *espacio-temporal* de la intensidad de la imagen, donde la variación espacial se refiere a las dos dimensiones de la imagen y la *variación temporal* al eje de tiempo sobre el cual evoluciona la imagen, figura 2.15.

El análisis del movimiento a partir de una secuencia de imágenes se encamina hacia la estimación del movimiento relativo entre los objetos en la escena y las imágenes. Uno de los métodos más importantes para la estimación del movimiento está basado en el gradiente. Esto es, en la observación del cambio de los niveles de intensidad en la imagen. El flujo óptico refleja los cambios de la imagen debido al movimiento durante un intervalo de tiempo dt y el campo de flujo óptico es el campo de velocidad que representa el movimiento tridimensional de puntos de los objetos a través del movimiento bidimensional de la imagen.

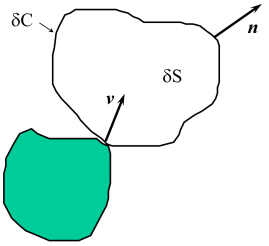


Figura 2.16: Definición de símbolos para la ecuación de restricción del flujo óptico

Considerando una analogía con la ecuación de continuidad de fluidos, se puede obtener la correspondiente ecuación de restricciones (Nomura et al., 1994) observando un cambio temporal de la intensidad de la imagen en un área local ∂S,

$$\frac{\partial}{\partial t}\int_{\partial S} f ds = -\oint f\ v.ndc + \int_{\partial S} \varphi ds \tag{2.19}$$

donde $f(x, y, t)$ es una distribución espacio temporal de intensidad de la secuencia de imágenes, ∂C es el contorno envolviendo $\partial S, v = (u, v)$ es un vector velocidad a determinar, n es un vector unitario normal apuntando hacia fuera del contorno ∂C y φ es la razón de generación de intensidad sobre un píxel en ∂S, figura 2.16. La generación de intensidad significa disminución o aumento de la intensidad en una imagen bajo iluminación no uniforme. En la ecuación (2.19) el cálculo de la integral a lo largo del contorno ∂C se transforma en el cálculo integral sobre ∂S por el teorema de la divergencia de Gauss, por tanto, la expresión diferencial de la ecuación (2.19) queda como sigue:

$$\frac{\partial f}{\partial t} = -f\ div(v) - \mathrm{v}\ .grad(f) + \varphi \tag{2.20}$$

Para resolver la ecuación anterior es habitual formular dos restricciones:

1. La divergencia de v es nula ($div(v) = 0$) (Nomura et al., 1994).

2. La intensidad observada para cualquier punto de un objeto es constante en el tiempo bajo las condiciones específicas de movimiento de las cámaras o movimiento de los objetos en la escena, lo que significa que la razón de generación de intensidad de un píxel φ es nula (Nalwa, 1993; Sonka et al., 1995). Esta restricción es quizás un tanto exigente como sostiene Nalwa y sólo sería válida por ejemplo para superficies Lambertianas, que se trasladan bajo iluminación homogénea invariante en el tiempo, por ejemplo, no hay fuentes de luz moviéndose en el entorno.

Con estas dos restricciones la ecuación (2.20) se transforma en

$$\frac{\partial f}{\partial t} + \mathrm{v}\ .\mathrm{G} = 0 \tag{2.21}$$

donde por simplicidad de notación $grad(f)$ es G. Este gradiente G es el gradiente espacial de una imagen bidimensional en la localización (x, y).

La ecuación (2.21) es fundamental para los cálculos del flujo óptico. Es importante reseñar que aunque esta ecuación es exacta bajo las anteriores suposiciones, cuando las derivadas en esta ecuación se implementan como diferencias finitas entre las intensidades de píxeles adyacentes, la ecuación es realmente un desarrollo en serie de Taylor de primer orden para la variación espacio temporal de la intensidad de la imagen (Sonka et al., 1995; Ballard and Brown, 1982). Veamos con un poco más detalle esta afirmación. Supongamos que tenemos una imagen continua; $f(x, y, t)$ se refiere al nivel de intensidad del píxel (x, y) en el instante t. Representando una imagen dinámica como una función de la posición y el tiempo es posible expresarla como un desarrollo en serie de Taylor;

$$\begin{aligned} f(x+dx, y+dy, t+dt) &= f(x,y,t) + \frac{\partial f}{\partial x}dx + \frac{\partial f}{\partial y}dy + \frac{\partial f}{\partial t}dt + O(\partial^2) \\ &= f(x,y,t) + f_x dx + f_y dy + f_t dt + O(\partial^2) \end{aligned} \tag{2.22}$$

donde $O(\partial^2)$ son los términos de orden superior en el desarrollo de Taylor que son despreciados.

Se puede asumir que la vecindad inmediata de (x, y) se traslada alguna pequeña distancia (dx, dy) durante el intervalo dt, esto es, se puede encontrar dx, dy, dt tal que

$$f(x+dx, y+dy, t+dt) = f(x,y,t) \tag{2.23}$$

Si dx, dy, dt son muy pequeños, los términos de orden superior en la ecuación (2.22) se pueden despreciar y por tanto

$$-f_t = f_x \frac{dx}{dt} + f_y \frac{dy}{dt} \tag{2.24}$$

Ahora el objetivo es obtener la velocidad $v = \left(\frac{dx}{dt}, \frac{dy}{dt}\right) = (u, v)$ sabiendo que fx, fy y ft son todas cantidades medibles o al menos puede obtenerse una medida aproximada a partir de $f(x, y, t)$.

La ecuación (2.24) es idéntica a la ecuación (2.21). Se puede observar a partir de cualquiera de estas dos ecuaciones que la diferencia de intensidad f_t en la misma localización (x, y) de la imagen en los instantes t y $t + dt$ es un producto de la diferencia del nivel de intensidad espacial y la velocidad en dicha localización de acuerdo al observador.

La ecuación anterior asume que durante el movimiento la intensidad de la imagen permanece constante a lo largo de la trayectoria s del movimiento, resultando $\frac{df}{ds} = 0$. Esto implica entre otras las siguientes consideraciones: cualquier cambio en la intensidad de la imagen se debe al movimiento, la iluminación de la escena permanece constante o las superficies de los objetos son opacas (superficies Lambertianas). En ocasiones la suposición de que la intensidad permanece constante no es cierta, particularmente si la iluminación cambia, para evitar esta posibilidad se aplica la restricción de gradiente constante a lo largo de la trayectoria del movimiento por tanto, se recurre a la consideración de que a pesar de que pueda violarse la restricción de intensidad constante lo que si va a permanecer constante a lo largo de dicha trayectoria es el gradiente, esto es $\frac{d\nabla f}{ds} = 0$.

2.3.1.1. Lucas-Kanade

Teniendo en cuenta las consideraciones previas, la ecuación (2.24) se expresa como sigue,

$$\begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = -\frac{\partial \left(\vec{\nabla} f \right)}{\partial t} \tag{2.25}$$

considerando un entorno de vecindad Ω alrededor de cada punto sobre el que se extiende el sumatorio, la ecuación (2.25) se puede expresar como sigue,

$$\begin{bmatrix} \sum_\Omega f_x^2 & \sum_\Omega f_x f_y \\ \sum_\Omega f_x f_y & \sum_\Omega f_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum_\Omega f_t f_x \\ \sum_\Omega f_t f_y \end{bmatrix} \equiv Av = b \tag{2.26}$$

La solución de la ecuación (2.26) puede obtenerse por mínimos cuadrados,

$$v(u, v) = \left(A^T A \right)^{-1} A^T b \tag{2.27}$$

El filtrado espacial se define mediante los núcleos: $H_x = [-1\ 0\ 1]$ y $H_y = [-1\ 0\ 1]^t$, mientras que el temporal se calcula como la diferencia entre las dos imágenes de la secuencia.

2.3.1.2. Horn-Schunck

Se trata de un método desarrollado por Barron et al. (1994), y asume que no existen movimientos bruscos en la secuencia de imágenes, de esta forma la estimación del vector velocidad v se obtiene minimizando la función de energía definida por la ecuación (2.28),

$$E = \iint \left(f_x u + f_y v + f_t \right)^2 dxdy + \alpha \iint \left\{ \left(\frac{\partial u}{\partial x} \right)^2 + \left(\frac{\partial u}{\partial y} \right)^2 + \left(\frac{\partial v}{\partial x} \right)^2 + \left(\frac{\partial v}{\partial y} \right)^2 \right\} dxdy \tag{2.28}$$

El segundo término de la ecuación anterior representa la restricción de suavidad global. En él se incluyen las variaciones de las componentes de velocidad u y v en función de las derivadas espaciales. El coeficiente α se utiliza para determinar la contribución de esta restricción de suavidad en el cómputo de la energía.

La solución a la ecuación anterior se obtiene normalmente de forma iterativa como sigue (Matlab, 2024a):

1. Se calculan f_x y f_y mediante los operadores de Sobel (Pajares and de la Cruz, 2007), como extractor de bordes horizontales y verticales.
2. Se calcula f_t como la diferencia entre dos imágenes consecutivas en la secuencia.
3. Se calculan las componentes de las velocidades en un píxel (x, y) dado en la iteración k según las siguientes expresiones, teniendo en cuenta que las velocidades que aparecen en la parte derecha de la igualdad son velocidades promedio de las velocidades de todos los píxeles en la vecindad del píxel dado.

$$u^{k+1} = u^k - \frac{f_x \left[f_x u^k + f_y v^k + f_t \right]}{\alpha^2 + f_x + f_y}; \qquad \mathrm{v}^{k+1} = v^k - \frac{f_y \left[f_x u^k + f_y v^k + f_t \right]}{\alpha^2 + f_x + f_y} \tag{2.29}$$

2.3.1.3. Farneback

El método de Farneback (2003) se fundamenta en el concepto de expansión polinomial alrededor de un pixel, y en cómo la caracterización de los coeficientes de los polinomios permite calcular la estimación del flujo óptico, al aplicar traslaciones, como el desplazamiento que se produce entre los píxeles de dos imágenes consecutivas. Aunque este método parece muy costoso computacionalmente, lo habitual es implementarlo eficientemente usando un esquema jerárquico de convoluciones separables (Matlab, 2024a).

El seguimiento comienza en el nivel de resolución más bajo y continúa hasta la convergencia. Las ubicaciones de los puntos detectados en un nivel se propagan como puntos clave para el nivel siguiente. De este modo, el algoritmo refina el seguimiento con cada nivel. La descomposición piramidal permite al algoritmo manejar grandes movimientos de píxeles, que pueden ser distancias superiores al tamaño de la vecindad. La figura siguiente, tomada de Matlab (2024a) representa la estructura piramidal del algoritmo del procedimiento.

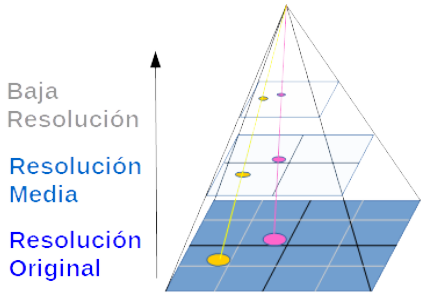


Figura 2.17: Estructura piramidal del método de Farneback en diferentes niveles (Matlab (2024a))

La idea consiste en asumir que una pequeña región de la imagen, puede representarse por una forma polinomial cuadrática tal como se indica en la ecuación (2.30) con $x = (x, y)$ siendo las coordenadas de un píxel,

$$f(x) = x^T Ax + b^T x + c \tag{2.30}$$

Donde A es una matriz simétrica, b un vector y c un escalar. Los coeficientes se estiman con un ajuste ponderado por mínimos cuadrados en los valores de la vecindad.

Suponiendo un desplazamiento ideal, d, entre dos imágenes consecutivas f_1 y f_2 se puede llegar a la siguiente expresión donde d es el desplazamiento que se quiere obtener, siendo equivalente a la determinación del flujo óptico, como en los casos anteriores,

$$\begin{aligned} f_2(x) &= f_1(x - d) \\ &= (x - d)^T A_1 (x - d) + b_1^T (x - d) + c_1 \\ &= x^T A_1 x + (b_1 - 2A_1 d)^T x + d^T A_1 d - b_1^T d + c_1 \\ &= x^T A_2 x + b_2^T x + c_2 \end{aligned} \tag{2.31}$$

Igualando los coeficientes se tiene,

$$\begin{aligned} A_2 &= A_1, \\ b_2 &= b_1 - 2A_1 d, \\ c_2 &= d^T A_1 d - b_1^T d + c_1 \end{aligned} \tag{2.32}$$

En las expresiones anteriores existe solución si A es no singular, y así se obtiene el valor del desplazamiento buscado d,

$$\begin{aligned} 2A_1 d &= -\left(b_2 - b_1\right), \\ d &= -\frac{1}{2} A_1^{-1} \left(b_2 - b_1\right) \end{aligned} \tag{2.33}$$

2.3.2. OCR

El reconocimiento de caracteres en matrículas de vehículos mediante imágenes digitales es un proceso que permite identificar los caracteres presentes en dichas matrículas. Varias técnicas y procedimientos se emplean en el reconocimiento de texto en imágenes, incluyendo documentos, rótulos comerciales y, como en este caso, matrículas de vehículos. El objetivo de este proyecto es automatizar el reconocimiento de caracteres en matrículas españolas para identificar el vehículo.El método descrito aquí se ha tomado y transcrito en su totalidad de Núñez et al. (2023) por ser totalmente conforme con el planteamiento que se presenta.

Para identificar matrículas de vehículos en movimiento, se siguen diversos procedimientos, generalmente estructurados en una serie de pasos hasta llegar a la identificación de los caracteres mediante OCR.

1. Preprocesamiento de la imagen
2. Extracción de regiones correspondientes a caracteres
3. Descripción de las regiones que representan los caracteres
4. Identificación de los caracteres
5. Posprocesamiento para corregir discrepancias y asegurar coherencia

2.3.2.1. Descripción de regiones

Después de identificar regiones potencialmente correspondientes a caracteres, se extraen propiedades que deben coincidir con las de los caracteres buscados.

La aplicación del método MSER genera varias regiones candidatas, algunas de las cuales no coinciden con los caracteres deseados. Por ejemplo, la figura 2.18 muestra varias regiones coloreadas identificadas por el método. Algunas de estas regiones no corresponden a caracteres, como las áreas alargadas. El objetivo es eliminar estas regiones no textuales.

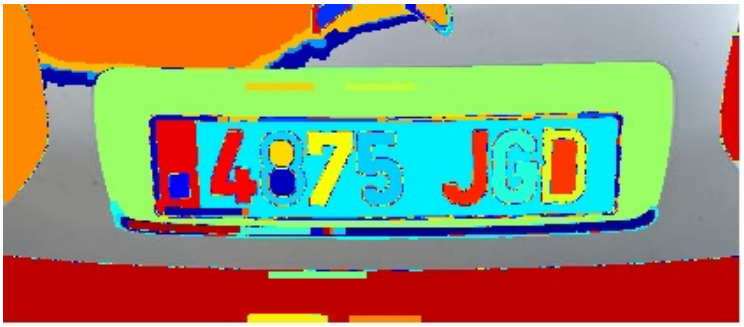


Figura 2.18: Regiones identificadas por MSER

Un método común para lograr esta eliminación consiste en aplicar un conjunto de reglas basadas en propiedades geométricas (Neumann and Matas (2012), Matas et al. (2002)) similares a las empleadas en Matlab (2024a). La descripción de estas propiedades y sus valores específicos se detallan a continuación (Li and Lu, 2012).

Primero, se definen conceptos claves: cada región tiene un rectángulo delimitador conocido como bounding box con un ancho (W) y alto (H), y su área Ab = WxH. En la figura 2.19(a), el rectángulo rojo delimita el carácter G. Además, cada región tiene ejes mayor (M) y menor (m), representando las mayores distancias entre puntos de borde. La figura 2.19(b) muestra la envolvente conexa (Ec) que encierra la región. El número de píxeles blancos define el Área (A) de la región.

Figura 2.19: (a) Bounding Box de ancho W y alto H; (b) Envolvente conexa

1. Razón de aspecto (Ra): cociente entre W y H, Ra = W/H. Un cuadrado tiene una razón de aspecto de uno.
2. Excentricidad (Ex): cociente entre M y m, Ex = m/M.
3. Número de Euler (Eu): diferencia entre el número de componentes conexas y el número de huecos, Eu = C-B. Por ejemplo, los caracteres G, A y B tienen números de Euler de 1, 0 y -1, respectivamente.
4. Solidez (So): cociente entre el Área y la envolvente conexa, So = A/Ec.
5. Extensión (Ex): cociente entre el Área y el área del Bounding Box, Ex = A/Ab.

Siguiendo la metodología de Matlab (2024a), las reglas aplicadas para descartar regiones no candidatas son las siguientes:

1. Ra >3

2. Ex >0.995
3. Eu <-4
4. So <0.3
5. Ex >0.9

Aplicando estas reglas, la figura 2.20 muestra las regiones candidatas restantes, coincidiendo con los caracteres a identificar.

Figura 2.20: Regiones no candidatas filtradas

Además, con técnicas adicionales como la evaluación del grosor del trazo, se puede discernir mejor entre caracteres y otras regiones. Li and Lu (2012) sugiere extraer un esqueleto de la imagen, como se describe en Pajares and de la Cruz (2007). Este esqueleto ayuda a medir la constancia del grosor del trazo, como se observa en la figura 2.21, donde el carácter G mantiene un grosor constante, confirmando así su autenticidad.

Figura 2.21: Trazos de grosor constante

El siguiente paso es agrupar los caracteres que han pasado los filtros previos para formar un panel de entrada al OCR. La figura 2.22 muestra un ejemplo, donde los Bounding Boxes en amarillo indican los caracteres candidatos y el rectángulo verde los agrupa para

la identificación OCR. Este paso es crucial ya que determina las zonas de carácter pero no los identifica aún, tarea que se lleva a cabo por el OCR.

Figura 2.22: Bounding Boxes individuales y global

Cabe mencionar que existen otros métodos para localizar la región de interés de la matrícula a partir de una imagen global. Enfoques alternativos pueden basarse en la concentración de bordes o zonas de alto contraste, características típicas de las matrículas españolas.

2.3.2.2. Identificación de los caracteres

Esta fase es la esencia del OCR. Tras delimitar los caracteres, se extraen propiedades que los distinguen unívocamente. Entre estas, se encuentran los momentos invariantes de Hu (1962), que generan siete propiedades formando un vector de siete dimensiones. Estos vectores se emplean en modelos de aprendizaje automático como K-Medias. Los momentos de Hu son invariantes a rotaciones, traslaciones y cambios de escala, lo cual es crucial para el reconocimiento de matrículas, ya que las perspectivas y distancias desde la cámara pueden variar.

La aplicación desarrollada emplea Matlab (2024a) y el motor Tesseract mejorado de Smith (2007) basado en una red neuronal convolucional entrenada con caracteres alfanuméricos, como se ilustra en la figura 2.23.

Figura 2.23: Base de datos de caracteres de entrenamiento de la red neuronal del OCR

Matlab permite crear bases de datos y definir métodos de aprendizaje automático, proporcionando una metodología abierta y flexible.

2.3.2.3. Posprocesamiento

En la etapa de posprocesamiento, se mejoran los resultados del OCR mediante análisis adicionales para verificar la coherencia de los caracteres reconocidos. Por ejemplo, si se detecta una vocal en una matrícula española moderna, esto se corrige ya que no es posible. Asimismo, se pueden corregir confusiones entre la letra J y el número 1.

En este trabajo, se aplican criterios lógicos para manejar ambigüedades, asegurando que las matrículas nuevas contienen cuatro dígitos y tres letras, en secuencias específicas para formar la matrícula completa.

Capítulo 3

Redes Neuronales Convolucionales

Las CNN (Convolutional Neural Networks) constituyen un método específico en el desarrollo de este trabajo. Como bien es sabido, son métodos basados en modelos cuyas capas son de naturaleza convolucional, donde además se realizan distintas operaciones básicas, de forma que en el presente capítulo se exponen en primer lugar las operaciones más relevantes y en segundo lugar los tres modelos utilizados en el presente trabajo, a saber: AlexNet, GoogleNet y SqueezeNet. Tanto para la descripción de las operaciones mencionadas como para los modelos de redes indicados se ha seguido la referencia Pajares et al. (2021) de la que se extraen y se describen con un alto grado de fidelidad los conceptos expuestos en este capítulo.

3.1. Operaciones en redes neuronales convolucionales

3.1.1. Métodos de optimización

La optimización se trata de ajustar la variable x para minimizar o maximizar una función $f(x)$. Usualmente, se enfoca en la minimización, dado que maximizar es lo mismo que minimizar $-f(x)$.

La función a minimizar o maximizar se denomina función objetivo o criterio y se usa para evaluar una solución potencial, como los pesos en una red neuronal. Al minimizar esta función, también se le conoce como función de coste, pérdida o error (Goodfellow et al., 2016). La función de pérdida es crucial para calcular el error que se propagará hacia atrás durante el proceso de entrenamiento. Así, las funciones que se optimizan se llaman funciones de pérdida. El valor mínimo o máximo alcanzado se denota con el símbolo $*$, representado como $x* = argmin\ f(x)$.

Es fundamental que la función de pérdida sea diferenciable. Durante el entrenamiento, esta función compara la salida de la red neuronal (predicción) con la etiqueta verdadera (*ground-truth*). Existen distintas funciones de pérdida según los modelos de red. Cada capa en la red neuronal realiza cálculos que, si son diferenciables, harán que la función de pérdida también sea diferenciable. Sin embargo, a veces se usan funciones de activación como ReLU que no son diferenciables; en estos casos, se necesita una aproximación derivativa para aplicarla en la retropropagación mediante gradiente descendente. La función de pérdida mide qué tan bien el modelo predice el resultado esperado. Entre los métodos más utilizados para encontrar el mínimo de una función está el "gradiente descendente".

Este método es como deslizarse por una montaña ondulante para alcanzar el punto más bajo, conceptualizando así la optimización basada en gradientes. Al calcular la pérdida, se mejora el modelo propagando el error hacia atrás, ajustando los pesos. Esto cierra el ciclo de aprendizaje entre la entrada de los datos, la generación de predicciones y la mejora en la retropropagación. Al ajustar los pesos, el modelo mejora y aprende.

La derivada de una función en un punto x da la pendiente en ese punto, permitiendo escalar un pequeño cambio en la entrada para obtener un cambio en la salida: $f(x+\varepsilon) \approx f(x) + \varepsilon f'(x)$. La derivada es útil para minimizar una función, indicando cómo ajustar x para mejorar la salida. Por ejemplo, $f(x - \varepsilon \text{sign}(f'(x)))$ es menor que $f(x)$ para un ε pequeño. Así, se puede reducir $f(x)$ cambiando x en pasos pequeños en la dirección opuesta a la derivada, conocido como gradiente descendente (*gradient descent*). La Figura 3.1 ilustra cómo el gradiente descendente usa derivadas para descender hacia un mínimo. Por ejemplo, para $f(x) = \frac{1}{2}x^2$, la derivada $f'(x) = x$ da lugar a las siguientes observaciones:

1. Para $x < 0$, $f'(x) < 0$ y f disminuye hacia la derecha.
2. Para $x > 0$, $f'(x) > 0$ y f disminuye hacia la izquierda.
3. Para $x = 0$, el gradiente se detiene ya que $f'(x) = 0$.

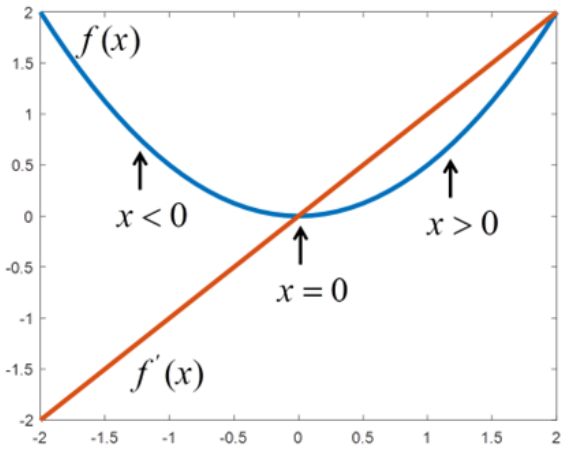


Figura 3.1: Gradiente descendente

3.1.1.1. Gradiente Descendente Estocástico

En el contexto del aprendizaje automático, el problema del Gradiente Descendente Estocástico (SGD, Stochastic Gradient Descent) es minimizar una función objetivo con la siguiente forma (Bishop, 2006; Ruder, 2017):

$$J(w) = \frac{1}{n} \sum_{i=1}^{n} J_i(w) \tag{3.1}$$

Aquí, el parámetro w, que minimiza $J(w)$, debe ser estimado. J_i está relacionado con la i-ésima observación en el conjunto de datos de entrenamiento. El término estocástico se refiere a la selección aleatoria de las muestras para ajustar el modelo, pudiendo hacerlo por lotes en cada iteración. $J_i(w)$ es el valor de la función de pérdida en el i-ésimo ejemplo y $J(w)$ representa el riesgo empírico.

El gradiente descendente se emplea para minimizar la siguiente función de manera iterativa, a través de las iteraciones t.

$$w(t+1) = w(t) - \varepsilon \frac{1}{n} \sum_{i=1}^{n} \nabla J_i(w) \tag{3.2}$$

Este método recorre el conjunto de entrenamiento y actualiza el modelo según la ecuación anterior para cada muestra. Aquí, gradientes negativos aumentan el peso y viceversa, siempre buscando minimizar la función objetivo. Se pueden realizar múltiples iteraciones sobre el conjunto de entrenamiento hasta que el algoritmo converge. En cada paso, se pueden seleccionar aleatoriamente los datos para evitar ciclos, esto es el criterio estocástico. Estas muestras seleccionadas forman un lote (*batch*). Implementaciones típicas pueden usar una tasa de aprendizaje adaptativa para ayudar a la convergencia del algoritmo. El pseudocódigo del gradiente descendente estocástico es:

1. Elegir un vector inicial de parámetros w (puede ser aleatorio) y una tasa de aprendizaje, ε.
2. Repetir hasta alcanzar un mínimo aproximado:
   a) Seleccionar aleatoriamente ejemplos en el conjunto de entrenamiento.
   b) Para $i = 1, 2, .., n$, ajustar $w(t+1) = w(t) - \varepsilon \nabla J_i(w)$.

3.1.1.2. Adaptive Moment Estimation (Adam)

Adam, que viene de *Adaptive Moment Estimation* (Kingma and Ba, 2014), actualiza los parámetros usando un mecanismo basado en momentos. Mantiene una actualización de la media móvil, componente a componente, tanto de los gradientes como de sus valores al cuadrado.

$$\begin{aligned} m(t) &= \beta_1 m(t-1) + (1-\beta_1)\, \nabla J_i(w(t-1)) \\ v(t) &= \beta_2 v(t-1) + (1-\beta_2)\, (\nabla J_i(w(t-1))^2 \\ w(t) &= w(t-1) - \frac{\alpha m(t-1)}{\sqrt{v(t-1)} + \varepsilon} \end{aligned} \tag{3.3}$$

Una variante de Adam es AdaMax, donde la norma L_2 se generaliza a L_p y β_2 a β_2^p. Las normas con valores altos de p tienden a ser inestables, por lo que en la práctica se usan frecuentemente las normas L_1 y L_2. Por esta razón Kingma and Ba (2014) proponen AdaMax y demuestran que $v(t)$ con L_∞ converge a un valor más estable. Para esta norma, se define $u(t)$:

$$u(t) = \beta_2^\infty v(t-1) + (1-\beta_2^\infty)\, (\nabla J_i(w(t-1))^\infty = max((\beta_2 \cdot v(t-1))\,, \nabla J_i(w(t-1)) \tag{3.4}$$

Con esta modificación, se reemplaza el término $\sqrt{v(t-1)} + \varepsilon$ en la regla de actualización:

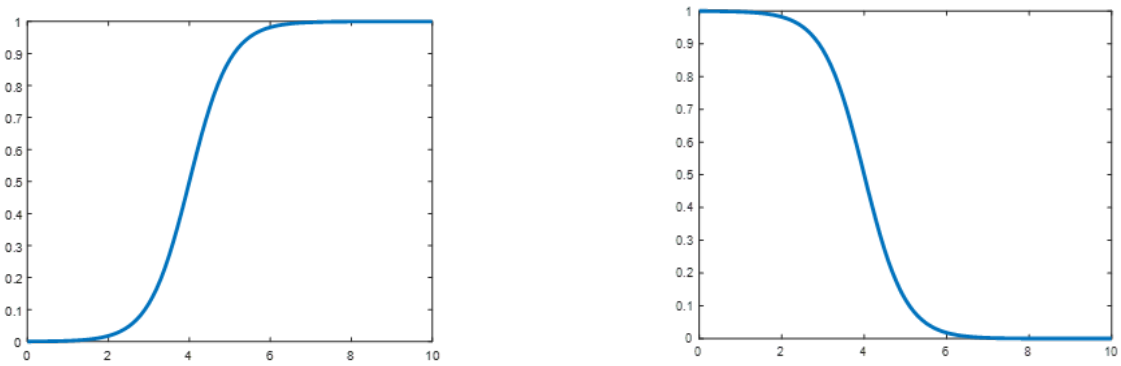
$$w(t) = w(t-1) - \frac{\alpha m(t-1)}{u(t-1)} \tag{3.5}$$

3.1.2. Funciones de activación no lineal

Una función de activación común es la sigmoide, definida como:

$$f(a, x, c) = \frac{1}{1 + e^{-a(x-c)}} \tag{3.6}$$

Dependiendo del signo del parámetro a, la función sigmoide se abre hacia la izquierda o hacia la derecha, siendo útil para conceptualizar valores "muy grandes" o "muy negativos". La Figura 3.2 muestra ejemplos de funciones sigmoides: (a) con $a = 2$ y $c = 4$; y (b) con $a = -2$ y $c = 4$.



(a) Con $a = 2$ y $c = 4$ (b) Con $a = -2$ y $c = 4$

Figura 3.2: Funciones sigmoide

La función sigmoide, que asigna salidas en el intervalo [0, 1], enfrenta dos problemas principales:

1. Saturación del gradiente: Cuando la activación se aproxima a 0 o 1, el gradiente tiende a 0, afectando el ajuste de los pesos de la red.

2. Pesos positivos continuos: La activación media no es cero, lo que hace que los pesos tiendan a ser positivos.

Estos problemas causan una convergencia lenta en el entrenamiento.

En Courbariaux et al. (2015), se define una variante llamada función sigmoide dura (*hard-sigmoid*):

$$f(x) = \text{máx}\,(0, \text{mín}\,(1, \frac{1}{2}(x + 1))) \tag{3.7}$$

La función tangente hiperbólica (*tanh*) proporciona salidas reales en un rango diferente, siendo una variante de la función sigmoide: tanh(x) = 2sigmoid(2x)-1. Sin embargo, también enfrenta el problema de la saturación del gradiente. La Figura 3.3 muestra la representación de la función *tanh*.

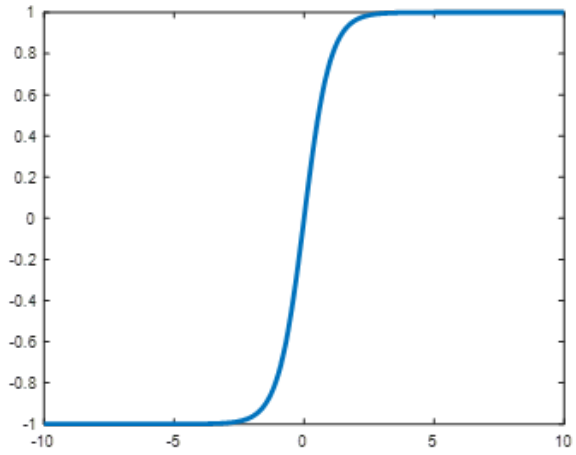
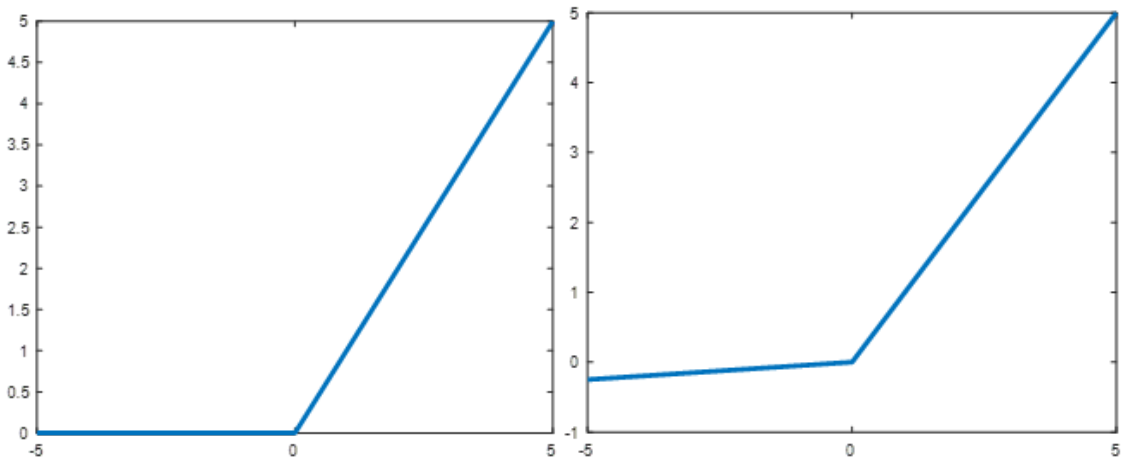


Figura 3.3: Función tanh

La función Unidad Lineal Rectificada (ReLU, Rectified Linear Unit), f(x) = max(0, x), representada en la Figura 3.4(a) tiene las siguientes ventajas:

1. Gradiente no saturado: Para $x > 0$, el problema de la dispersión del gradiente se mitiga, permitiendo la rápida actualización de parámetros en la primera capa.

2. Baja complejidad computacional: Dada su definición, ReLU es computacionalmente eficiente. Sin embargo, tiene la desventaja de que la neurona ReLU puede "morir" ante gradientes negativos altos durante la retropropagación. Esto se mitiga con una cuidadosa inicialización de los pesos o usando ReLU con fugas (LReLU), donde la salida es lineal multiplicada por un pequeño valor (aproximadamente 0.001) cuando la entrada es negativa, f(x) = max(0.01x, x), mostrado en la Figura 3.4(b).



(a) Función ReLu (b) Función LReLu

Figura 3.4: Funciones de Activación No Lineales

3.1.3. Normalización

Como se mencionó previamente, ReLU no necesita normalización para evitar la saturación (Krizhevsky et al., 2012). Si al menos algunos ejemplos de entrenamiento producen

una entrada positiva en una unidad ReLU, se producirá aprendizaje en esa neurona. Sin embargo, una normalización local aún puede ser útil para mejorar la generalización. Si se denota por $a^i_{x,y}$ la actividad de una neurona obtenida después de aplicar el núcleo i en la posición (x, y) y luego la no linealidad ReLU, la actividad normalizada $b^i_{x,y}$ está dada por:

$$b^i_{x,y} = \frac{a^i_{x,y}}{\left(k + \alpha \sum_{j=max(0,i-n/2)}^{min(N-1,i+n/2)} \left(a^i_{x,y}\right)^2\right)^{\beta}} \tag{3.8}$$

donde la suma se extiende sobre n mapas adyacentes generados por los núcleos en la misma posición (x, y) y N es el número total de núcleos en la capa. El orden de los mapas es arbitrario y se determina antes de comenzar el aprendizaje. Esta normalización implementa una forma de inhibición lateral inspirada en las neuronas reales, creando una competencia para altas actividades frente a salidas de neuronas obtenidas con distintos núcleos. Los hiperparámetros k, n, α y β se determinan usando un conjunto de validación. Krizhevsky et al. (2012) proponen los valores $k = 2$, $n = 5$, $\alpha = 10^{-4}$ y $\beta = 0{,}75$. En el mismo estudio, se indica que esta normalización se aplica en ciertas capas después de la aplicación de ReLU.

3.1.4. Convolución

En términos simples, la convolución implica una operación entre dos funciones con valores reales. Para ilustrar este concepto, imagina una fuente de luz cuya intensidad varía y se mide con un sensor. Este sensor proporciona una salida en una determinada posición x en el tiempo t, o $x(t)$. Tanto x como t son valores reales, lo que significa que, debido a la variabilidad de la fuente de luz, se pueden registrar diferentes valores en distintos momentos. Si además consideramos que la señal medida por el sensor puede estar afectada por algún ruido, una forma de obtener una señal más nítida es realizar un promedio de múltiples mediciones. Dado que las medidas más recientes son más relevantes que las más antiguas, se puede ponderar este promedio dando más importancia a las mediciones recientes mediante una función de ponderación $w(a)$, donde a representa la separación temporal de la medición. Realizando este promedio ponderado en cada instante t, se obtiene una nueva función promedio de esta manera:

$$s(t) = \int x(a)w(t-a)da \tag{3.9}$$

Esta operación se conoce como convolución y se representa así:

$$s(t) = \int (x * w)(t) \tag{3.10}$$

Es importante mencionar que w debe ser una función de densidad de probabilidad para asegurar que la salida esté promediada adecuadamente. Además, w debe ser cero para todos los valores negativos, evitando así la inclusión de valores futuros, lo cual no es factible en la práctica.

En el contexto de las redes neuronales convolucionales (CNN), el primer argumento x se conoce como la entrada (*input*) y el segundo w como el núcleo (*kernel*) de convolución. La salida s se denomina comúnmente como mapa de características (*feature map*). Desde una perspectiva computacional, y siguiendo con el ejemplo de la fuente de luz, las mediciones son discretas, realizadas en intervalos de tiempo determinados, por ejemplo, cada segundo.

Así, el tiempo t toma solo valores enteros, lo que permite definir la convolución discreta de la siguiente manera:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \tag{3.11}$$

En el aprendizaje automático, la entrada es un vector o matriz multidimensional de datos, y el núcleo es generalmente un vector o matriz multidimensional de parámetros ajustados durante el proceso de aprendizaje. Estas estructuras multidimensionales se denominan tensores. Dado que cada elemento de la entrada y del núcleo debe almacenarse por separado, los valores que caen fuera de estos elementos se consideran nulos, lo que, en la práctica, resulta en una suma finita de valores en un número finito de elementos. Además, las convoluciones se realizan sobre múltiples ejes simultáneamente; por ejemplo, en imágenes bidimensionales (I), que actúan como entrada, se necesita un núcleo bidimensional (K).

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{3.12}$$

La convolución es una operación conmutativa, lo que significa que la expresión anterior puede reescribirse de la siguiente manera, lo que resulta en una menor variación en los valores de m y n y una mayor eficiencia en la implementación:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n) \tag{3.13}$$

La propiedad conmutativa de la convolución se entiende como un reflejo del núcleo respecto a la entrada, de manera que, a medida que el índice m aumenta, el índice en la entrada también aumenta mientras que disminuye en el núcleo. Sin embargo, esta propiedad no es especialmente relevante en las CNN. Lo que sí es pertinente es la correlación cruzada, definida como:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n) \tag{3.14}$$

Esta misma expresión en dos dimensiones (2D) se puede extender fácilmente a tres dimensiones (3D). Así, en la posición (i,j,k) con un núcleo tridimensional, queda de la siguiente manera:

$$S(i,j,k) = (K * I)(i,j,k) = \sum_m \sum_n \sum_p I(i+m, j+n, k+p)K(m,n,p) \tag{3.15}$$

Del mismo modo, es posible ampliar la operación de convolución usando tensores y núcleos de más de tres dimensiones.

A veces, en la terminología especializada, correlación cruzada y convolución se usan indistintamente. Sin embargo, en el aprendizaje automático, el algoritmo de aprendizaje determina los valores apropiados del núcleo en ambos casos. La Figura 3.5 muestra un ejemplo de convolución usando el núcleo K, sin reflejar el núcleo, aplicado a un tensor 2D, como podría ser una imagen I. En este caso, la convolución se representa con solapamiento total del núcleo, resultando en una imagen de menor dimensión que la original (de 6×7 a 4×5). Para conservar el tamaño de la imagen original, se puede ampliar la imagen con

ceros (zero-padding) antes de procesarla con el núcleo. Este procedimiento se conoce como "same". Si no se añaden ceros, se denomina "valid". La operación sin zero-padding se detallará más adelante.

En convoluciones, el campo receptivo se define como la región de entrada que contribuye a la salida generada por el filtro. La Figura 3.5 muestra campos receptivos de la imagen I que contribuyen a las salidas P y Q generadas por el filtro K.

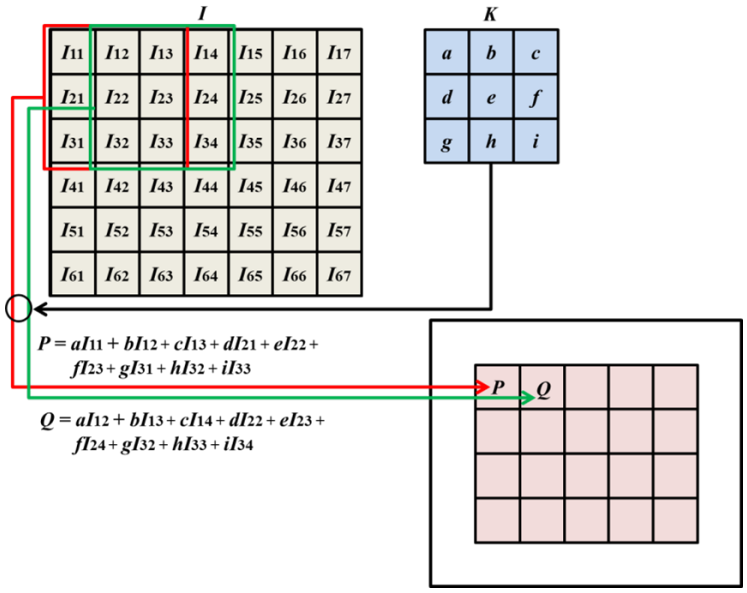


Figura 3.5: Ejemplo de convolución 2D

La convolución mostrada en la Figura 3.5 es de tipo 2D, pero también podría ser de tipo 3D, donde el núcleo sería un cuboide que se desplaza a través de las dimensiones de alto, ancho y profundidad del mapa de características. Por ejemplo, en una imagen RGB con tres canales, los núcleos serían cuboides con tres dimensiones. Esto puede generalizarse a convoluciones N-D para dimensiones N.

En la figura anterior, los desplazamientos del núcleo son de una posición a la siguiente, pero en lugar de desplazarse una posición de izquierda a derecha y de arriba abajo, el desplazamiento podría ser de más de una unidad, es lo que se conoce como desplazamiento o *stride*.

La colección de núcleos que define una convolución discreta tiene una estructura que se corresponde con una permutación del tipo (n, m, k1,. . . , kN), donde:

1. n es el número de mapas de características de salida.
2. m es el número de mapas de características de entrada.
3. k_j es la dimensión del núcleo a lo largo del eje j.

La dimensión oj de una capa de convolución a lo largo del eje j tiene las siguientes propiedades:

1. i_j es la dimensión de entrada a lo largo del eje j.

2. k_j es la dimensión del núcleo a lo largo del eje j.

3. s_j es la distancia entre dos posiciones consecutivas del núcleo (stride) a lo largo del eje j.

4. p_j es el número de ceros añadidos al inicio y al final del eje j (zero-padding).

La Figura 3.6 ilustra una convolución bidimensional (N=2), con $i_1 = i_2 = 5$ como dimensiones de la imagen de entrada (I) de tamaño 5×5, y dimensiones $k_1 = k_2 = 3$ para el núcleo de convolución (K), con desplazamiento (stride) de uno en ambas direcciones ($s_1 = s_2 = 1$) y zero-padding ($p_1 = p_2 = 0$). Generalmente, los valores de i, k, s y p suelen ser iguales en todos los ejes.

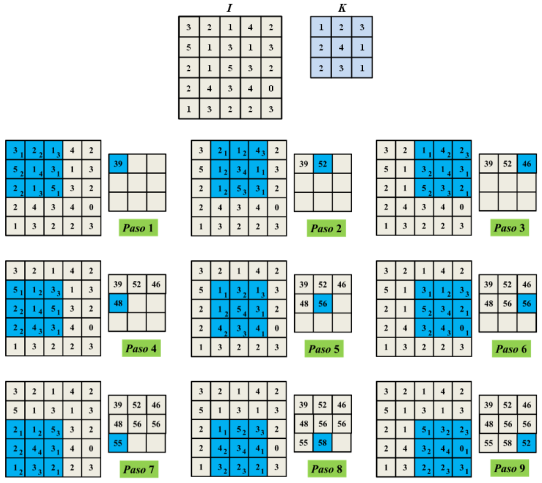


Figura 3.6: Ejemplo de una convolución discreta con $N = 2$, $i_1 = i_2 = 5$, $k_1 = k_2 = 3$, $s_1 = s_2 = 1$, $p_1 = p_2 = 0$

3.1.5. Softmax

La función softmax, o función exponencial normalizada, suele usarse en las últimas capas ocultas. Está definida en la ecuación (3.16). Su objetivo es transformar un vector n-dimensional x de valores reales en un vector n-dimensional softmax(x) de valores reales en el intervalo [0, 1], de tal modo que cada componente i del vector softmax(x) se obtiene considerando las n componentes del vector x:

$$\text{softmax}(x)_i = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)} \quad \text{para } i = 1, \ldots, n \quad \text{y} \quad x = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n \tag{3.16}$$

Imaginemos una red neuronal con cuatro neuronas ($n = 4$) en la capa de salida. En un momento dado, los valores de x son (1.2, 2.0, 5.6, 3.1). Aplicando la función softmax, se obtienen los valores $q = \text{softmax}(x) = (0{,}011, 0{,}024, 0{,}892, 0{,}073)$. La suma de todos los elementos de q es 1.

En resumen, la función exponencial se aplica a cada elemento x_j del vector de entrada x, y estos valores se normalizan mediante la suma de las exponenciales, asegurando que la suma de las componentes del vector de salida sea 1. Es posible modificar los exponentes de las exponenciales para crear distribuciones de probabilidad más concentradas en las

posiciones de los valores de entrada más altos, esto se consigue como sigue, con $\alpha x_i > 0$ o $-\alpha x_i > 0$. Para $i = 1, .., n$,

$$softmax(x)_i = \frac{\exp(\alpha x_i)}{\sum_{j=1}^{n} \exp(\alpha x_j)} \tag{3.17}$$

3.2. Modelos de Redes Neuronales Convolucionales

3.2.1. AlexNet

AlexNet es una red (Russakovsky et al., 2015; Krizhevsky et al., 2012; Model, 2023) destacada en el ámbito de las CNN, que obtuvo el primer lugar en la competición de LSVRC (2012) en 2012.

La figura 3.7 presenta la estructura de la red AlexNet, ilustrando gráficamente las operaciones realizadas, detallando las dimensiones de los filtros en las capas convolucionales y totalmente conectadas para las operaciones de convolución (conv), ReLU (relu), Normalización (norm), Pooling (pool), dropout (drop) o softmax. Se incluyen las dimensiones de los filtros, el stride (s), el padding (p) y el número de filtros (K) en cada capa.

La tercera columna de la tabla 3.1 expone los parámetros (pesos) que se aprenden en este modelo, asociándose con los pesos de las capas (primera columna) de convolución (conv1 a conv5) y las completamente conectadas (Fully connected, fc6, fc7 y fc8), independientemente de si se aplica o no dropout en ellas. Cabe notar que en cada capa de convolución se incluye un valor de bias por cada filtro, actuando como un parámetro de ajuste adicional. Por ejemplo, similar al término independiente b en una ecuación lineal (y = ax + b). Asimismo, la misma tabla 3.1 incluye en la segunda columna las dimensiones de salida correspondientes a cada capa.

Nombre de la capa	Dimensión de salida	Pesos
conv1	55x55x96	W=11x11x3x96 b=1x1x96
conv2	27x27x256	W=5x5x48x256 b=1x1x256
conv3	13x13x384	W=3x3x256x384 b=1x1x384
conv4	13x13x384	W=3x3x192x384 b=1x1x384
conv5	13x13x256	W=3x3x192x256 b=1x1x256
fc6	1x1x4096	W=4096x9216 b=4096x1
fc7	1x1x4096	W=4096x4096 b=4096x1
fc8	1x1x4096	W=1000x4096 b=1000x1

Tabla 3.1: Representación Red AlexNet

También, en la estructura mostrada en la figura 3.7 se destacan las proporciones, que establecen la relación entre las dimensiones de salida de cada capa (o) basándose en los parámetros de entrada (i, k, p, s). La figura 3.8 define las diversas proporciones posibles, junto con su número correspondiente.

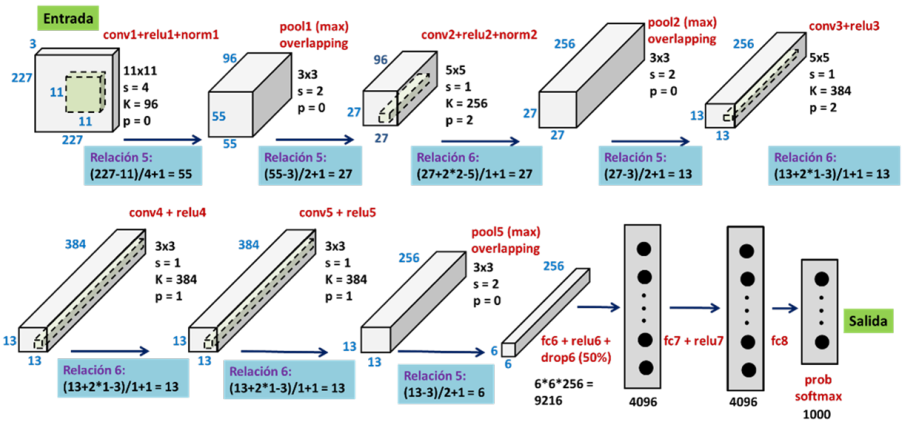


Figura 3.7: Red AlexNet

En algunas implementaciones, Matlab (2024a) después de la función relu7 se lleva a cabo una operación de dropout al 50 %, modificando las dimensiones de la salida de esta capa.

Este diseño de red también se puede visualizar en la figura 3.8, mostrando dos redes paralelas que alteran las dimensiones de los diferentes volúmenes, en contraste con el modelo de la figura 3.7. Aquí se aplican las mismas operaciones de padding, stride, pooling y dimensiones de los filtros, junto con métodos de normalización y activación similares. Así, la tercera dimensión de los volúmenes pasa de 96, 256, 384, 384 y 256 a volúmenes con dimensiones que son la mitad de estos valores, es decir, 48, 128, 192, 192 y 128, respectivamente. Este diseño incluye 3 conexiones cruzadas, marcadas con flechas en la segunda capa y en las capas finales (antepenúltima y penúltima) totalmente conectadas. Termina en una concatenación en la salida, alcanzando finalmente las mismas 1000 categorías de clasificación que en la estructura mostrada en la figura 3.7. Por lo tanto, la figura 3.7 representa una çonvolución de grupo"(grouped convolutions) en comparación con la figura 3.8. Este modelo puede ser considerado para su uso e inclusión en los diseños de arquitecturas de redes siamesas.

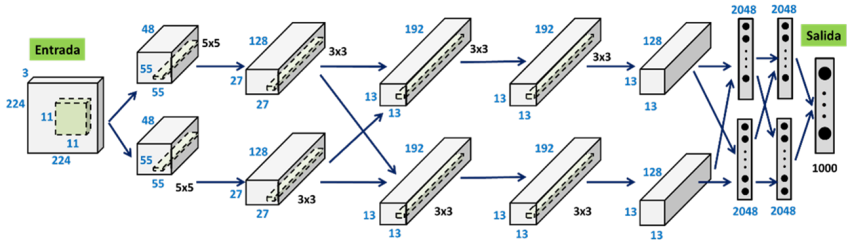


Figura 3.8: Representación alternativa de la Red AlexNet

3.2.2. GoogLeNet

La red GoogLeNet obtuvo el primer lugar en el concurso LSVRC (2014), conocido también como Inception V1 de Google Model (2020), basada en el trabajo de Szegedy et al. (2014). Alcanzó una tasa de error del 6.67 %, un resultado muy cercano al nivel humano que requirió la intervención de expertos. La evaluación por los expertos fue complicada, requiriendo un entrenamiento adicional para precisar la exactitud de GoogLeNet. Tras unos días de entrenamiento, el experto humano (Andrej Karpathy) obtuvo una tasa de error del 5.1 % (modelo único) y 3.6 % (en conjunto).

La estructura de la red se basa en una CNN inspirada en LeNet, y añade un componente innovador denominado módulo *inception*. Se emplearon métodos como la normalización por lotes (*batch*), distorsiones de imagen y RMSProp (Root Mean Square Propagation) como técnica de optimización. El módulo inception (Inc) utiliza múltiples convoluciones pequeñas para disminuir significativamente el número de parámetros. La red GoogLeNet cuenta con una CNN de 22 capas de profundidad, reduciendo el número de parámetros de 60 millones (AlexNet) a 4 millones. Nueve de estas 22 capas son del tipo inception de distintas categorías. En total, la red consta de 144 capas, con cada una de las 9 capas inception presentando la arquitectura mostrada en la figura 3.9, incluyendo las unidades ReLU en el módulo. El modelo completo propuesto por Szegedy et al. (2014) se visualiza en la figura 3.10, mostrando las diferentes capas: convolución (Conv) con dimensiones de los filtros; *Pooling*, ya sea promedio (AvgPool) o máximo (MaxPool) indicando las dimensiones de la ventana; Normalización (LRN, Local Response Normalization); capas totalmente conectadas (FC) y, por supuesto, los módulos Inception (Inc), en este caso V1, que forman el núcleo de este tipo de redes. Las capas de convolución y pooling también indican el desplazamiento (*stride*), expresado entre paréntesis con el símbolo s seguido del valor correspondiente de desplazamiento en el caso same y V también con el valor de desplazamiento en el caso *valid*.

Como se puede observar, este esquema incluye dos redes adicionales, que son clasificadores auxiliares compuestos por un pooling promedio de dimensión 5x5 y s = 3 de tipo V, resultando en salidas de tamaño 4x4x512 y 4x4x528 respectivamente. Estas capas son seguidas por una de convolución 1x1 con 128 filtros para reducción de la dimensionalidad y una unidad ReLU. Continúan con una unidad dropout y capas totalmente conectadas que finalizan con unidades softmax (Softmax0 y Softmax1). La salida final de la red es también una unidad softmax (Softmax2). Estas redes son clasificadores auxiliares cuyo objetivo es mejorar la discriminación en las capas más bajas del clasificador, intensificar la señal del gradiente hacia atrás, y proporcionar regularización adicional. Según Szegedy y col. (2015b), el clasificador principal se beneficia de la normalización por lotes (*batch normalization*) o dropout en las ramas laterales. Lee y col. (2014) sostienen que los clasificadores auxiliares facilitan un aprendizaje más estable y una mejor convergencia.

Al final del entrenamiento, una red con ramas auxiliares tiende a superar en precisión comparada con una que no tiene estas ramas, alcanzando una meseta de rendimiento óptimo. En la figura 3.10 se detallan los componentes del módulo troncal (stem). La figura 3.11 ofrece un esquema simplificado de esta red inception V1.

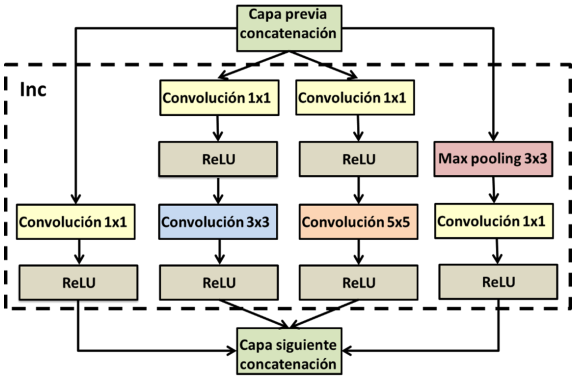


Figura 3.9: Módulo Inception con incorporación de unidades ReLU

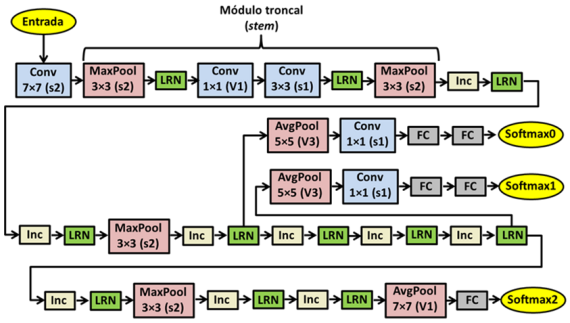


Figura 3.10: Modelo completo GoogLeNet (inception V1)

GoogLeNet proporciona una notable mejora en la reducción de la dimensión, que puede interpretarse desde una perspectiva computacional como un caso especial de factorización convolucional. Por ejemplo, considerando una capa de convolución de dimensiones 1x1 seguida de una de 3x3. Desde la perspectiva de las redes, se espera que las salidas de activaciones de capas cercanas estén altamente correlacionadas. Por lo tanto, se podría anticipar que sus activaciones puedan reducirse antes de la agregación para optimizar la eficiencia computacional. Esta es la premisa inicial formulada por Szegedy y col. (2015b) en sus propuestas para los módulos inception que evolucionan en las denominadas versiones V2, V3, y V4, completando la versión original V1. Para ello, se plantea una factorización en convoluciones más pequeñas.

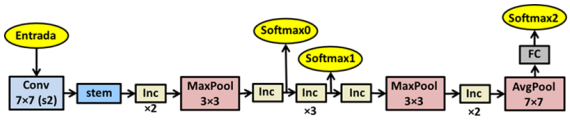


Figura 3.11: Modelo simplificado GoogLeNet (inception V1)

3.2.3. SqueezeNet

Este tipo de redes surge con la idea de conseguir modelos arquitectónicos con menos parámetros que otras redes con arquitecturas equivalentes, logrando resultados de precisión y desempeño similares. Están fundamentadas en lo que se conoce como *fire modules*, que constituyen los elementos base de este tipo de redes. Iandola et al. (2016) proponen tres estrategias para conseguir los objetivos de estas redes en base a su diseño modular:

1. Estrategia 1: reemplazar los filtros de dimensión 3×3 con filtros de dimensión 1×1, sobre la suposición de que estos tienen 9 veces menos parámetros que los primeros.

2. Estrategia 2: disminuir el número de canales de entrada a los filtros 3×3. Si se considera una capa de convolución formada íntegramente por filtros 3×3, el número de parámetros en esta capa es: número de canales de entrada × número de filtros × 3 × 3. Por ello, disminuyendo el número de canales de entrada se consigue una reducción en este sentido. Esto se consigue mediante el uso de lo que se conoce como capas squeeze, que se describen posteriormente.

3. Estrategia 3: submuestreo posterior en la red de modo que las capas de convolución poseen mapas de activación con una elevada dimensión. En una CNN cada capa de convolución produce un mapa de activación con una resolución espacial que es al menos de dimensión 1×1, y en la mayoría de los casos de dimensión superior. El ancho y alto de esos mapas de activación están controlados tanto por la dimensión de los datos de entrada como por la elección de capas en las que realizar el submuestreo en la arquitectura de la CNN. Generalmente el submuestreo se lleva a cabo fijando el desplazamiento (*stride*) a valores superiores a la unidad (s >1). De esta forma, si sobre las capas previas a una dada se aplican desplazamientos elevados, los mapas de activación resultantes poseen dimensiones reducidas. Por el contrario, si muchas capas de la red tienen desplazamientos la unidad, y los desplazamientos mayores que 1 se concentran en las capas finales, entonces aparecen muchas capas con mapas de activación de dimensiones grandes.

4. Según la intuición expresada por Iandola et al. (2016), la estructura de red con mapas de activación grandes debido al retraso del submuestreo conduce a mayores niveles de precisión en la clasificación. En He and Sun (2015) se proporcionan detalles relativos a este comportamiento de las CNN, en función del número de filtros, dimensiones de estos y otros parámetros asociados.

Teniendo en cuenta las estrategias 1 y 2, la propuesta consiste en disminuir el número de parámetros mientras se preserva la precisión. Mediante la estrategia 3 se trata de incrementar la precisión dado un número limitado de parámetros, para lo cual Iandola et al. (2016) proponen el módulo *fire* como sigue. Este consta de una capa de convolución comprimida (*squeeze*) que tiene solo filtros de dimensión 1×1, alimentando una capa de expansión (*expand*) con una mezcla de filtros de convolución de dimensiones 1×1 y 3×3, tal y como se muestra en la figura 3.12. En un módulo de esta naturaleza existen tres parámetros sintonizables, tal y como indican Iandola et al. (2016): s1 (número de filtros en la capa *squeeze*, todos de dimensión 1×1), e1 (número de filtros en la capa expandida, de dimensión 1×1) y e3 (número de filtros en la capa expandida, de dimensión 3×3). En aras del cumplimiento de la estrategia 2, se establece la siguiente restricción: s1 <e1 + e3. Estos módulos, aunque diferentes en su diseño, tienen una filosofía similar que las capas inception descritas previamente. A la salida de las respectivas capas (comprimida y expandida) se realiza una concatenación, cuyo resultado se proporciona a las correspondientes unidades de activación de tipo ReLU.

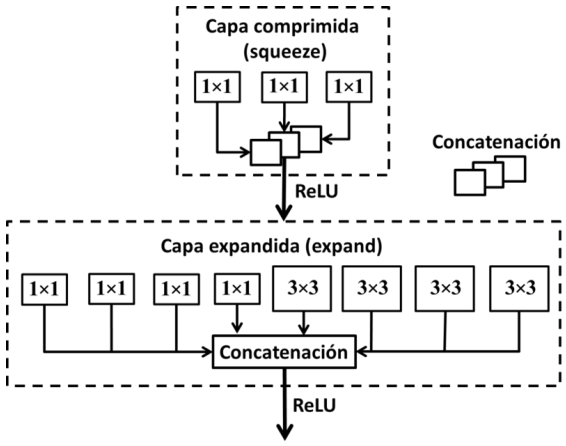


Figura 3.12: Módulo*fire*

Con los módulos*fire* se obtienen arquitecturas más complejas, o mejor dicho, con un número mayor de capas, como es el caso de la arquitectura propuesta en Iandola et al. (2016) que posee 10 capas, de las cuales 8 son módulos*fire*. En esta misma referencia se proporcionan mecanismos para el diseño de redes con este tipo de módulos. Para ello se define un conjunto de meta-parámetros de alto nivel de todos los módulos en la CNN. Se define b_e (base) como el número de filtros en la capa expandida del primer módulo*fire* en una CNN. Después de cada nf módulos*fire* se incrementa el número de filtros en la capa expandida por de; esto es, para el módulo*fire* i, el número de filtros en la capa expandida es: $e_i = b_e + d_e \left(\frac{i}{n_f}\right)$, dado que en esta capa existen filtros de dimensiones 1×1 y 3×3 se determina el número de ambos en función de un porcentaje, p_e, que se define al efecto para cada una de tales dimensiones. El número de filtros de la capa comprimida, s_i, se establece en función del parámetro p_s establecido en un valor fijo en el rango [0,1] para todos los módulos, de forma que $s_i = p_s e_i$. Iandola et al. (2016) realizan un estudio comparativo con diversos valores de tales parámetros para las imágenes procedentes de la base de datos ImageNet (2020), determinando que los mejores resultados se obtienen con $p_s = 0.75$ y $p_e = 50\,\%$.

Capítulo 4

Diseño e implementación de la aplicación

4.1. Introducción

Teniendo en cuenta los planteamientos teóricos expuestos previamente, en este capítulo se describe el diseño e implementación de la aplicación con base en ellos. Como se ha mencionado previamente, la aplicación ha sido desarrollada con el objetivo de detectar y clasificar vehículos, extraer las matrículas y almacenar la información en la nube para su posterior análisis, todo ello considerando técnicas de Visión por Computador, DL y un ecosistema IoT. Se describen los aspectos clave del diseño a alto y micro nivel, junto con los métodos y tecnologías utilizados para lograr un sistema eficiente y preciso.

4.2. Diseño a alto nivel

El diseño a alto nivel de la aplicación se enfoca en la estructura general del sistema, incluyendo los componentes principales y su interacción. En la figura 4.1 se presenta un diagrama que ilustra esta arquitectura, que consta de cuatro componentes integrados (Entrada, Proceso, Nube, Explotación de Datos), que se describen a continuación.

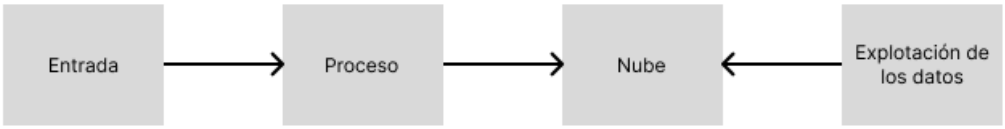


Figura 4.1: Diagrama con el diseño a alto nivel

4.2.1. Entrada

El sistema acepta dos tipos de entrada: videos pregrabados y capturas en tiempo real desde una cámara de dispositivo móvil. Estas opciones permiten flexibilidad en la fuente de datos y facilitan el procesamiento tanto de videos existentes como de nuevas grabaciones.

4.2.2. Proceso

El núcleo del sistema es el procesamiento de imagen, que incluye la detección de movimiento, la identificación de vehículos y la extracción de matrículas. Se emplean técnicas avanzadas de visión por computadora y aprendizaje profundo para asegurar una alta precisión en cada etapa del proceso. Las tecnologías involucradas en este módulo son las descritas en sendos capítulos dos y tres, respectivamente.

4.2.3. Nube

Los datos extraídos mediante el módulo de procesamiento previo se suben a una plataforma en la nube para su almacenamiento y análisis. En este caso, se utiliza ThingSpeak (Matlab, 2024b), una plataforma de IoT que permite la gestión y visualización de datos en tiempo real. Esto facilita el acceso remoto a la información y el análisis de patrones a largo plazo.

4.2.4. Explotación de los datos

La información almacenada en la nube se procesa para obtener información sobre el tráfico y otros parámetros relevantes. Se emplean modelos de aprendizaje automático para predecir la situación del tráfico y analizar tendencias, proporcionando un valor añadido a los datos recolectados. Conviene reseñar en este punto que la solución propuesta es de carácter conceptual, buscando como objetivo el estudio de las posibilidades que ofrece dicha solución de cara a su posible implantación en la realidad en el futuro.

4.3. Ampliación del Diseño a nivel micro

En este punto, nos adentraremos de lleno en el desglose arquitectónico del proceso de ejecución del proyecto, explicando más detalladamente cada elemento y su función dentro del sistema. La figura 4.2 muestra el diseño a nivel detallado.

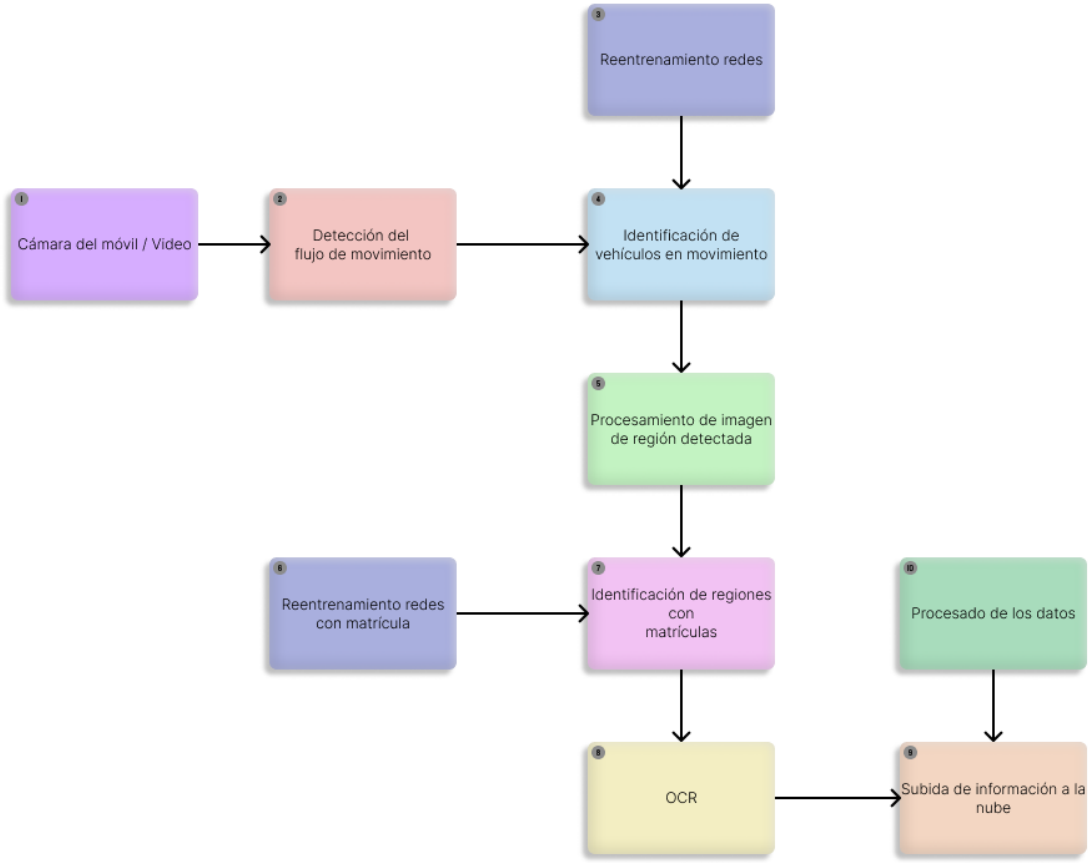


Figura 4.2: Diagrama con el diseño a nivel micro

4.3.1. Fase de Entrada

El sistema desarrollado para este estudio conceptual ofrece un espectro de dos opciones como entrada principal, siendo éstas la alimentación a través de un video pre-capturado o bien el uso directo de la cámara de un dispositivo móvil.

Primero se examina la opción de una entrada de video, la cual facilita la primera etapa de análisis en el sistema. Este método espera que se seleccione un archivo de vídeo alojado en memoria, una vez seleccionado lo abre y consigue el tamaño del frame en el video además del número de frames. Una vez que se dispone de la imagen se selecciona un frame de comienzo preconfigurado en el código y sus posteriores N frames también configurados en el código, así se puede procesar una parte menor del video para su uso de prueba sin necesidad de tener que ejecutarlo en su totalidad.

La segunda opción de entrada es la cámara de un teléfono móvil. Para implementar esta opción recurrimos a la aplicación móvil de Matlab que proporciona la funcionalidad de poder ejecutar código desde la nube utilizando el MATLAB Drive. Este código ha sido adaptado para poder activar la cámara al inicio de la ejecución y poder capturar así N número de fotos.

4.3.2. Detección del flujo de movimiento

Ahora se continúa con el segundo paso, el cual es donde el objetivo principal del programa se manifiesta: detectar el movimiento entre los fotogramas. De esta forma, se permite enfocar el esfuerzo en aquellos vehículos exclusivamente en movimiento, ignorando los objetos estacionarios. Para asegurar la precisión y efectividad de este análisis, un requisito previo indispensable es que el video se grabe desde una posición fija e inamovible.

El flujo de movimiento entre fotogramas se puede detectar mediante varios métodos de flujo óptico. Tal y como se explica en el capítulo dos, los métodos de flujo óptico son técnicas que miden el movimiento de objetos entre dos fotogramas consecutivos de un video. En este contexto, se utilizan tres métodos principales cuya síntesis se reproduce a continuación, estos son: Farneback, Lukas-Kanade y Horn-Schunck.

1. Método de Farneback: Utiliza una técnica densa para calcular el flujo óptico, proporcionando una estimación de movimiento en cada punto de la imagen. Es particularmente útil cuando se requiere un análisis detallado del movimiento en toda la escena.
2. Método de Lukas-Kanade: Utiliza una técnica basada en el seguimiento de características, que es más eficiente que el anterior computacionalmente y es adecuado para detectar movimientos en áreas específicas con alta precisión.
3. Método de Horn-Schunck: Asume que el flujo óptico es suave en toda la imagen y proporciona una estimación global del movimiento, siendo útil para análisis de movimientos generales en escenas más uniformes.

Una vez seleccionado el método de flujo óptico, el proceso de detección de movimiento comienza con la captura de dos fotogramas consecutivos del video. Se convierte cada fotograma a escala de grises, lo que simplifica el análisis al eliminar la información de color que no es relevante para la detección de movimiento.

El flujo óptico se estima entre estos dos fotogramas, generando un campo de vectores de movimiento que describe la dirección y magnitud del movimiento en cada punto de la imagen. Visualmente, esto se representa mediante vectores de flujo superpuestos sobre la imagen, mostrando claramente las áreas donde aparecen los objetos en movimiento en la imagen.

Para identificar las regiones que representan movimiento significativo, se calcula la magnitud del flujo óptico. Se establece un umbral basado en la media y desviación estándar de la magnitud del flujo para segmentar las áreas con movimiento considerable. Este umbral asegura que solo los movimientos significativos se consideren en el análisis, descartando el ruido o pequeños movimientos irrelevantes.

Las regiones de movimiento se etiquetan mediante la técnica de componentes conexas explicadas en el capítulo dos, y se analizan sus propiedades geométricas, como el área. Solo se consideran aquellas regiones que superan un umbral mínimo de área, por ejemplo, 500 píxeles. Este filtrado adicional asegura que se enfoquen solo las regiones suficientemente grandes para ser potencialmente vehículos en movimiento.

Finalmente, las regiones seleccionadas se pueden recortar de la imagen original para su posterior clasificación utilizando técnicas de DL. Este paso asegura que solo las áreas de

interés, aquellas con movimiento significativo y tamaño adecuado, sean procesadas en las siguientes etapas del análisis.

4.3.3. Identificación de vehículos en movimiento

El siguiente paso permite discernir con claridad cuáles de las regiones identificadas corresponden potencialmente a un vehículo en movimiento, para su posterior procesamiento.

Para ello primeramente es necesario tener entrenada la primera red neuronal convolucional. La cual tendrá como clases las siguientes: asfalto, bus, camión, coche, moto, lineas y muro. Se trata de regiones que aparecen en el entorno donde se capturan las imágenes, esto es, vehículos circulando en vías urbanas o interurbanas.

Para entrenar un modelo de aprendizaje profundo, primero se deben cargar y preprocesar los datos de imagen. Esto implica organizar las imágenes en clases de acuerdo con sus etiquetas, lo cual facilita el proceso de clasificación. Las imágenes se dividen en subconjuntos de entrenamiento y validación. El subconjunto de entrenamiento se utiliza para ajustar los pesos del modelo, mientras que el subconjunto de validación permite evaluar el rendimiento del modelo y ajustar hiperparámetros para evitar el sobreajuste. Esta separación es crucial para obtener una evaluación precisa del rendimiento del modelo en datos no vistos hasta el momento.

Es útil visualizar algunas imágenes del conjunto de datos antes de proceder con el entrenamiento. La exploración visual ayuda a verificar la calidad de las imágenes y su diversidad. Asegurarse de que las imágenes estén correctamente etiquetadas y que el conjunto de datos sea representativo de las diferentes clases, lo cual es fundamental para el éxito del modelo. Esto puede incluir la identificación de imágenes mal etiquetadas o de baja calidad que podrían afectar negativamente el entrenamiento.

La transferencia de aprendizaje implica reutilizar un modelo preentrenado en una tarea nueva. Modelos como AlexNet, GoogleNet o SqueezeNet, que han sido entrenados en grandes conjuntos de datos como ImageNet (ImageNet, 2020), pueden ser ajustados para clasificar un nuevo conjunto de imágenes específicas. Este enfoque es eficiente porque las primeras capas de estos modelos ya han aprendido características generales útiles, como bordes y texturas, que pueden ser aplicadas a la nueva tarea. Las últimas capas del modelo se modifican para adaptarse al número de clases específicas del nuevo conjunto de datos.

La aumentación (Image Augmentation) de datos es una técnica que aplica transformaciones aleatorias a las imágenes de entrenamiento para mejorar la robustez del modelo y evitar el sobreajuste. Estas transformaciones incluyen operaciones tales como rotaciones, traslaciones, escalados y reflejos. La aumentación de datos incrementa la variabilidad del conjunto de entrenamiento sin necesidad de recolectar más datos, lo cual es especialmente útil en conjuntos de datos pequeños. Este proceso ayuda al modelo a generalizar mejor en relación a nuevas imágenes, ya que aprende a reconocer las clases independientemente de las variaciones en la apariencia de las imágenes.

El modelo ajustado se entrena utilizando los datos de entrenamiento, donde ajusta sus pesos para minimizar el error en la clasificación. Este proceso se realiza iterativamente a través de épocas, durante las cuales el modelo optimiza sus parámetros utilizando algoritmos de optimización tales como el gradiente descendente. El objetivo es minimizar

una función de pérdida que mide la discrepancia entre las predicciones del modelo y las etiquetas reales de las imágenes.

Una vez entrenado el modelo se evalúa utilizando el conjunto de datos de validación. Este paso es crucial para medir el desempeño del modelo en datos no vistos y asegurarse de que no está sobreajustado a los datos de entrenamiento. Las métricas de evaluación, como la precisión, la sensibilidad y la especificidad, permiten cuantificar el rendimiento del modelo. Si el rendimiento en el conjunto de validación no es satisfactorio, puede ser necesario ajustar los hiperparámetros del modelo, tales como la tasa de aprendizaje o el número de épocas, y volver a entrenar el modelo.

Visualizar los pesos de las capas convolucionales iniciales del modelo puede proporcionar una idea sobre qué características está aprendiendo el modelo de las imágenes. Los pesos en estas capas representan los valores de los filtros que el modelo utiliza para detectar características básicas, como bordes, texturas y patrones. Al visualizar estos pesos, se puede obtener una comprensión más profunda de cómo el modelo procesa las imágenes y qué tipo de características son más importantes para la tarea de clasificación. Esto puede ayudar a diagnosticar problemas en el entrenamiento y a interpretar el comportamiento del modelo.

Ahora, una vez que se tiene el modelo preparado podemos seleccionar cada una de estas regiones y recortarla. Este recorte necesita un redimensionamiento al tamaño de entrada aceptado por cada una de las tres redes utilizadas en el proyecto, siendo las dimensiones de GoogleNet de 224x224x3, AlexNet 227x227x3 y SqueezeNet 227x227x3.

Una vez que se tiene el recorte preparado y redimensionado podemos pasarlo por el modelo para realizar su clasificación. Una vez clasificado se nos devuelve una etiqueta y un porcentaje de acierto, si la etiqueta es asfalto, muro o lineas se descarta, al igual que si el porcentaje es menor que 0.5.

Con esto se consigue poder identificar en qué región están los vehículos en movimiento y poder discernir a qué clase pertenecen, es decir, coche, moto, bus y camión. A continuación se pasa a la siguiente fase con cada uno de los vehículos identificados para conseguir detectar su matrícula y extraerla.

4.3.4. Procesamiento de imagen de la región detectada

Habiendo identificado la región que contiene el vehículo, los esfuerzos se centran ahora en procesar la imagen para detectar la región que contiene la matrícula. Este proceso implica varios pasos de preprocesamiento y segmentación para aislar posibles regiones de interés (ROI, por sus siglas en inglés) que podrían contener la matrícula.

Tal y como se explica en la capítulo dos, primero se convierte la imagen de la región detectada a un espacio de color adecuado para la segmentación, como el espacio de color HSV, donde la componente de valor (V) se puede utilizar para crear una imagen binaria. Este paso de binarización convierte la imagen en una representación de blanco y negro basada en un umbral calculado automáticamente, facilitando la identificación de regiones contrastantes.

A continuación, se aplican operaciones morfológicas, como la apertura y el cierre, para

eliminar el ruido y mejorar la conectividad de las regiones en la imagen binarizada. La apertura elimina pequeños objetos ruidosos mientras que el cierre conecta áreas cercanas, lo que resulta en regiones más limpias y coherentes.

Luego, se etiquetan las regiones conectadas en la imagen binaria y se calculan sus propiedades, tales como el área y la caja delimitadora. En este punto, se descartan las regiones que no superan un determinado umbral de tamaño (por ejemplo, 500 píxeles) para reducir el número de regiones candidatas y enfocarse en aquellas que son más probables de contener la matrícula.

4.3.5. Identificación de regiones con matrícula

En esta etapa, el objetivo es identificar la región exacta que contiene la matrícula del vehículo dentro de la región previamente procesada. Los modelos de DL, previamente entrenados para reconocer matrículas, son utilizados para este fin.

Una vez filtradas las regiones por tamaño, se procede a la extracción de características de cada región candidata. Para este fin, se utilizan redes neuronales convolucionales (CNNs) preentrenadas, como AlexNet, GoogleNet o SqueezeNet, explicadas en el capítulo tres. Como se ha indicado previamente, estas redes han sido entrenadas previamente en grandes conjuntos de datos (ImageNet) y pueden extraer características relevantes que permiten distinguir las matrículas de otras partes de la imagen. El uso de modelos preentrenados es una práctica común en la transferencia de aprendizaje, aprovechando su capacidad para identificar patrones complejos en las imágenes.

Antes de clasificar las regiones candidatas, es necesario redimensionar las imágenes de estas regiones a un tamaño estándar, que la red preentrenada espera. Por ejemplo, AlexNet y SqueezeNet esperan imágenes de 227x227x3 píxeles, mientras que GoogleNet espera imágenes de 224x224x3 píxeles. Este redimensionado asegura que las características extraídas sean consistentes con las que la red ha aprendido a identificar durante su entrenamiento inicial.

Luego, cada región redimensionada se somete a un proceso de clasificación utilizando la red preentrenada. El resultado, como se ha indicado previamente, es que la red produce una etiqueta y un valor de confianza para cada región. La etiqueta indica si la región es una matrícula o no, y el valor de confianza representa la probabilidad asignada por la red a esa clasificación. Solo las regiones que son clasificadas como matrícula y que tienen un alto valor de confianza son consideradas para la selección final.

Entre las regiones que han sido clasificadas como matrícula, se selecciona la que tiene la mayor probabilidad. Este paso es crucial, ya que garantiza que la región elegida sea la más probable de contener la matrícula correcta. La precisión en esta selección depende de la robustez del modelo de clasificación y de la calidad de las imágenes procesadas.

4.3.6. Reconocimiento Óptico de Caracteres (OCR)

Para realizar el reconocimiento óptico de caracteres (OCR) en la región de la matrícula previamente detectada, se sigue un proceso que incluye la mejora de la precisión en la detección de caracteres individuales y la unificación de estas regiones para facilitar la lectura por parte del módulo OCR. La técnica implementada en el OCR es la explicada

en el capítulo dos.

El primer paso en el OCR es convertir la imagen de la matrícula a una escala de grises. Esto reduce la complejidad de la imagen al eliminar la información de color, dejando solo la intensidad de los píxeles. Luego, la imagen se binariza utilizando un umbral determinado, convirtiendo la imagen a blanco y negro para distinguir mejor los caracteres del fondo.

Para detectar las regiones de la imagen que contienen caracteres, se utiliza la técnica de Regiones Extremas Maximamente Estables (MSER) (Matas et al., 2002). MSER es una técnica robusta para detectar regiones conexas en una imagen que son invariables a cambios de iluminación y escala. Estas regiones son candidatas a contener caracteres.

Una vez detectadas las regiones MSER, se calculan varias propiedades geométricas de estas regiones, como la caja delimitadora (*bounding box*), la excentricidad, la solidez, la extensión y el número de Euler. Estas propiedades se utilizan para filtrar las regiones que no cumplen con las características típicas de los caracteres, eliminando falsos positivos que no son relevantes.

Las cajas delimitadoras de las regiones de caracteres se expanden ligeramente para asegurar que los caracteres completos estén contenidos dentro de ellas. Luego, se calcula la relación de superposición entre las cajas delimitadoras para identificar y unificar aquellas que pertenecen al mismo grupo de texto. Esto se realiza mediante la construcción de un grafo de componentes conectados, donde cada nodo representa una caja delimitadora y las aristas representan la superposición significativa entre ellas.

Después de identificar los grupos de texto, se ajustan las cajas delimitadoras para cada grupo, determinando los límites mínimos y máximos en las direcciones x e y. Este paso asegura que las cajas unificadas abarcan correctamente todas las regiones de texto sin dejar fuera ningún carácter.

Finalmente, el módulo OCR se aplica a la imagen binarizada utilizando las cajas delimitadoras ajustadas. El OCR reconoce los caracteres dentro de estas regiones y genera el texto correspondiente. Para mejorar la precisión, se define un conjunto de caracteres esperados (dígitos y letras mayúsculas) y se especifica el layout del texto.

4.3.7. Subida de la información a la nube

Una vez que se ha identificado la matrícula del vehículo en movimiento, el siguiente paso es subir esta información a una plataforma en la nube para su almacenamiento y análisis. En este caso, se utiliza ThingSpeak, una plataforma de IoT que permite agregar, visualizar y analizar datos en tiempo real.

Para enriquecer los datos subidos y hacerlos más representativos de una situación real, aparte de la matrícula detectada se simulan una serie de valores adicionales que podrían ser típicos en un entorno de tráfico, como la calle, el tipo de vehículo, la hora y minutos, día de la semana y día del mes. Estos valores adicionales ayudan a crear un contexto más completo y útil para el análisis posterior.

Primero, se define el canal de ThingSpeak al que se subirán los datos. Cada canal en ThingSpeak tiene un ID único y claves API (de lectura y escritura) que controlan el acceso

al canal. El ID del canal identifica de manera única el destino de los datos, mientras que las claves API aseguran que solo usuarios autorizados puedan leer o escribir en el canal.

La información recolectada, junto con los valores simulados, se sube a ThingSpeak utilizando una función que escribe datos en el canal especificado. En este contexto, los datos son subidos a varios campos del canal, cada uno correspondiente a matrícula, tipo, calle, hora, minuto, día de la semana y día del mes. La clave API de escritura se utiliza para autenticar la solicitud de subida y asegurar que los datos sean almacenados correctamente.

Durante el proceso de escritura, se incluyen pausas para asegurar que la operación se complete correctamente. Las pausas permiten que el servidor de ThingSpeak procese la solicitud de subida y almacene los datos sin interrupciones.

Después de subir los datos, se realiza una verificación para confirmar que la información se ha escrito correctamente en ThingSpeak. Esta verificación se realiza leyendo los datos del canal utilizando la clave API de lectura. Al leer los datos recientemente escritos y comprobar su exactitud, se asegura la integridad del proceso de subida y se valida que la información almacenada es correcta y completa.

4.3.8. Procesado de los datos

Terminado el proceso de recolección de datos, ahora es posible descargar un archivo csv desde la plataforma de Thingspeak y procesar estos datos para poder explotarlos.

El objetivo a conseguir a partir de estos datos es conseguir predecir la situación del tráfico, para ello se ha utilizado un dataset de Kaggle de muestra. El conjunto de datos incluye columnas como la hora, la fecha, el día de la semana, los recuentos para cada tipo de vehículo (coches, bicicletas/motos, buses, camiones), una columna *Total* representa el recuento total de todos los tipos de vehículos y por último una columna que indica la situación del tráfico categorizada en cuatro clases: Intenso, alto, normal y bajo. Cada elemento contiene los datos detectados en rangos de 15 minutos para todo el día.

4.3.8.1. Transformación de los datos

Para ello la primera fase que habría que realizar es modificar los datos en crudo subidos a ThingSpeak. Ya que se hace uso del dataset de Kaggle (Aman, 2024) este punto es meramente teórico. Lo primero que tendríamos que hacer es empezar dividiendo los datos por calles para entrenar los modelos específicamente para analizar el comportamiento del flujo de vehículos en cada una de las calles.

El segundo paso consistiría en dividir los datos en los rangos de quince minutos para cada dia del mes y sacar mediante el tipo de vehículo el contador para cada tipo y el contador total de ese rango de tiempo, con lo que se tendrían igualados nuestros datos en crudo con los del dataset.

4.3.8.2. Preparación del dataset

En lo que sigue se analiza y se habla sobre el datset de Kaggle. Para preparar el dataset lo primero que se ha hecho es reemplazar todas las columnas que son categóricas con palabras por números.

Después, el procesamiento ha consistido en pasar las columnas de la hora a minutos ya que originalmente venía en este formato: 12:00:00 AM. Se ha elegido minutos y segundos porque en ningún valor de la columna se incluye ningún segundo que no sea 0, así que se pueden obviar.

Por último, también se han normalizado las columnas del contador de coches, bicicletas, buses, camiones y total utilizando el MinMaxScaler que las deja en un rango entre 0 y 1.

4.3.8.3. Entrenamiento de los modelos de IA

Con este dataset de datos pre-procesados estamos frente a un problema de clasificación de aprendizaje automático supervisado. Por lo que los principales modelos candidatos a usar son: KNeighborsClassifier, DecisionTreeClassifier, RandomForestClassifier, GaussianNB, SVC y MLPClassifier (Duda et al., 2001).

Aunque el problema consiste en clasificar la situación del tráfico en cuatro clases, se han utilizado modelos de regresión (SVR lineal, SVR con kernel, DecisionTreeRegressor y RandomForestRegressor) debido a que el tráfico no aumenta de forma discreta o en escalones, sino que se trata de un fenómeno continuo. Estos modelos de regresión permiten predecir valores numéricos que representan la intensidad del tráfico, y posteriormente, se establecen umbrales o puntos de corte para asignar cada valor predicho a una de las cuatro clases definidas. De esta manera, se aprovecha la capacidad de los modelos de regresión para capturar la naturaleza continua del tráfico, mientras que la clasificación final se realiza en función de los umbrales establecidos.

Además de los enfoques de clasificación y regresión, se ha explorado un enfoque de aprendizaje no supervisado mediante la técnica de clustering. Para ello, se ha eliminado la columna que contiene las etiquetas de las clases de tráfico previamente definidas. El objetivo es evaluar cómo se comportaría un sistema que no cuente con esta información previa y deba agrupar los datos en clusters o grupos basándose únicamente en las características o variables de entrada. Mediante el uso de técnicas de clustering, como el algoritmo k-means, se busca identificar patrones inherentes en los datos y agruparlos en k *clusters* distintos. Esta aproximación permitirá analizar si los clusters resultantes reflejan de manera natural las diferentes situaciones de tráfico, o si revelan nuevas perspectivas o agrupaciones interesantes que no se habían considerado previamente.

Capítulo 5

Resultados obtenidos

Una vez que se ha explicado el diseño e implementación de la aplicación, a continuación se procede a valorar los resultados más significativos teniendo en cuenta los diferentes módulos que la componen y la solución integrada proporcionada. El capítulo se estructura de forma que en primer lugar se ofrecen resultados sobre el entrenamiento de las CNN utilizadas, sección 5.1. En la sección 5.2 se muestra una etapa de procesamiento de una de las imágenes, con sus distintas fases y resultados. Es importante destacar que en ambas secciones se utilizan datos conteniendo matrículas de vehículos, de forma que en cada una de estas dos secciones se indica el número de imágenes utilizadas, así como las secuencias de vídeo captadas con la cámara de un dispositivo móvil que actúa de sensor en el contexto IoT. No obstante, debido a la naturaleza de estos datos y en base a las leyes de protección vigentes, los resultados que se muestran se analizan mostrando un vehículo que no necesita garantizar tal protección. Lo mismo es aplicable en el caso de la matrícula mostrada en la figura 2.8 previa. En este sentido, conviene aclarar que las explicaciones relativas relacionadas con las imágenes que se muestran son perfectamente válidas para el conjunto de imágenes utilizadas y analizadas. Como colofón a este capítulo en la sección 5.3 se ofrecen resultados relativos al procesamiento de los datos en la plataforma ThingSpeak, específica de IoT. En este caso se trata de una simulación, donde se explican las posibilidades de extensión de la aplicación desarrollada, de forma que en dicha plataforma se almacenan datos, que pueden procesarse en cualquier momento. En este caso concreto, los datos que se muestran son relativos a flujo de vehículos circulando en vías urbanas o interurbanas en movimiento. Se almacena el tipo de vehículo y el número de ellos, que constituyen el flujo de tráfico. Esta identificación del tipo de vehículo se puede llevar a cabo perfectamente mediante los mismos modelos de CNN descritos en este trabajo. También cabe la posibilidad de utilizar el procedimiento de reconocimiento de matrículas, mediante la correspondiente adaptación, para almacenar este dato en la plataforma. Para el análisis de los datos se utilizan técnicas de regresión, cuyos resultados también se ofrecen en esta misma sección. Finalmente, conviene indicar que el procesamiento de estos datos procedentes de la plataforma ThingSpeak, pueden ser procesados internamente dentro de dicha plataforma, mediante la utilización de funciones embebidas en ella, o bien externamente, que es como realmente se realiza en este caso.

Para los desarrollos de la aplicación y sus funcionalidades asociadas se ha utilizado MATLAB como lenguaje de programación. Los datos recuperados de la nube a través de la plataforma ThingSpeak se procesan con Python y librerías asociadas. Los procesamientos locales se han realizado con un procesador Intel Core i7 6700U y 8GB RAM. Los detalles relativos a la ubicación del código y su instalación con vistas a su ejecución se encuentran

descritos en el Apéndice A.

5.1. Entrenamiento de las redes

A continuación, se informa sobre una sesión de entrenamiento para AlexNet. GoogleNet y SqueezeNet no se menciona por repetición de contenido.

Como se ha comentado anteriormente, estos modelos de redes neuronales están pre-entrenados con el conjunto de datos ImageNet (Russakovsky et al., 2015). Este entrenamiento se ha llevado a cabo para clasificar imágenes en 1000 categorías basadas en dicho conjunto de datos, que contiene más de 14 millones de imágenes. En este contexto, uno de los primeros pasos a seguir es la redefinición de la capa de salida, de modo que en lugar de las 1000 categorías originales, se utilicen las categorías necesarias para esta aplicación específica.

Previamente se ha descrito que es necesario entrenar dos modelos: uno para la detección de vehículos en movimiento y otro para la detección de matrículas. Ambos modelos siguen un proceso de reentrenamiento similar, pero con clases diferentes. El modelo para la detección de vehículos incluye las siguientes clases: asfalto, autobús, camión, coche, moto, líneas y muro. Por otro lado, el modelo para la detección de matrículas abarca estas clases: matrícula, asfalto, acera, coche, árbol y cielo.

Teniendo en cuenta estas nuevas categorías, la capa de salida se redefine para proporcionar las 6 ó 7 clases en lugar de las 1000 iniciales. Posteriormente, se inicia el entrenamiento de los modelos utilizando los pesos pre-entrenados con ImageNet, ajustándolos durante el proceso para adaptarse a las nuevas categorías. Este proceso se conoce como transferencia de aprendizaje (*transfer learning*).

Para entrenar las redes se han utilizado 115 imágenes distintas de vehículos en las que aparecían sus matrículas además de otros elementos que rodean al vehículo. Posteriormente se crearon 6 carpetas para que las redes aprendieran los 6 distintos tipos de elementos que existían en las imágenes a procesar: matricula, coche, cielo, acera, árbol y asfalto. Cada carpeta se rellenaba con recortes de dichos elementos a partir de las imágenes recogidas. La carpeta matrícula estaba compuesto de 99 recortes de matrículas, la carpeta coche de 71 recortes de distintos elementos de los vehículos, la carpeta cielo de 58 recortes de cielo, la carpeta acera de 60 recortes de aceras, la carpeta árbol de 50 recortes distintos de árboles, y la carpeta asfalto de 39 recortes de asfalto. Todos los recortes se han sacado de las imágenes previamente mencionadas.

Para entrenar las redes neuronales del segundo modelo, se utilizaron 120 imágenes de vehículos que incluían sus matrículas y otros elementos del entorno. Luego, se crearon 6 carpetas para que las redes aprendieran a identificar los 6 tipos diferentes de elementos presentes en las imágenes: matrícula, coche, cielo, acera, árbol y asfalto. Cada carpeta se llenó con recortes de estos elementos tomados de las imágenes originales. La carpeta de matrículas contenía 100 recortes de matrículas, la carpeta de coches tenía 80 recortes de diferentes partes de vehículos, la carpeta de cielo contenía 70 recortes de cielo, la carpeta de acera tenía 70 recortes de aceras, la carpeta de árboles contenía 50 recortes de árboles, y la carpeta de asfalto tenía 40 recortes de asfalto. Todos los recortes se obtuvieron de las imágenes previamente mencionadas. Para el modelo destinado a la detección de los

vehículos se hizo lo mismo pero con sus respectivas clases.

Durante el entrenamiento, se presentan dos gráficas que muestran su evolución, como se puede observar en la figura 5.1. Estas gráficas reflejan el avance de las epochs (iteraciones), ya que cada epoch incluye un número determinado de imágenes que participan en el total de iteraciones. En la gráfica superior, se puede ver cómo progresa la precisión de la red, es decir, el porcentaje de aciertos durante el entrenamiento. A medida que el entrenamiento avanza, la precisión mejora consistentemente hasta estabilizarse cerca del 100 %, lo cual es un indicador muy positivo, dado que el objetivo es alcanzar el 100 %.

En la gráfica inferior, representada en rojo, se muestran los valores de la función de pérdida a lo largo de las epochs. Esta función mide el error y evalúa la desviación entre las predicciones de la red y los valores reales del entrenamiento. A diferencia de la precisión, la función de pérdida sigue una trayectoria descendente, comenzando en un valor alrededor de 2.3 y disminuyendo rápidamente hasta estabilizarse cerca de cero. Esto también es una señal positiva, ya que alcanzar un error mínimo (cero) es el objetivo ideal. Al final del entrenamiento, se logra un 100 % de precisión tras completar el máximo número de epochs programadas, con un tiempo total de 7 minutos y 16 segundos, alcanzando hasta 340 iteraciones utilizando una sola CPU y una tasa de aprendizaje constante de 0.0001. En resumen, estos resultados indican un entrenamiento altamente satisfactorio.

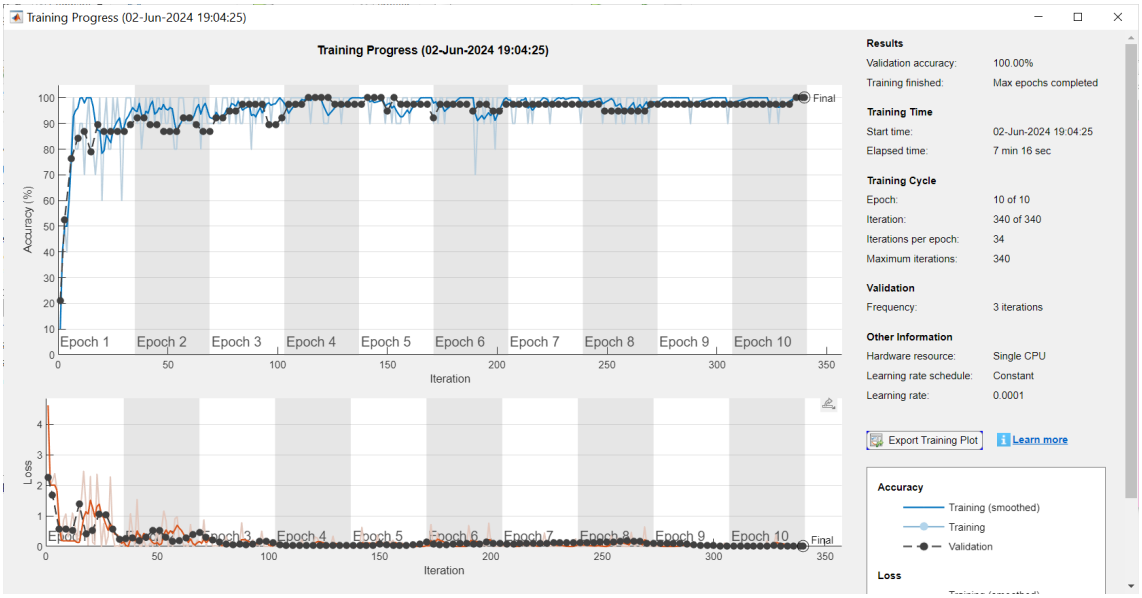


Figura 5.1: Gráfica del progreso de entrenamiento de AlexNet

Además de los gráficos que muestran la evolución de la precisión y la función de pérdida, Matlab proporciona más datos tras la fase de entrenamiento de las redes. En la figura 5.2 se presenta una serie de pruebas con imágenes del conjunto de test, mostrando el porcentaje de aciertos entre los distintos elementos de la imagen durante el entrenamiento con AlexNet. Exceptuando el caso de acera y asfalto, se obtiene un 100 % de probabilidad de pertenencia a la clase correspondiente.

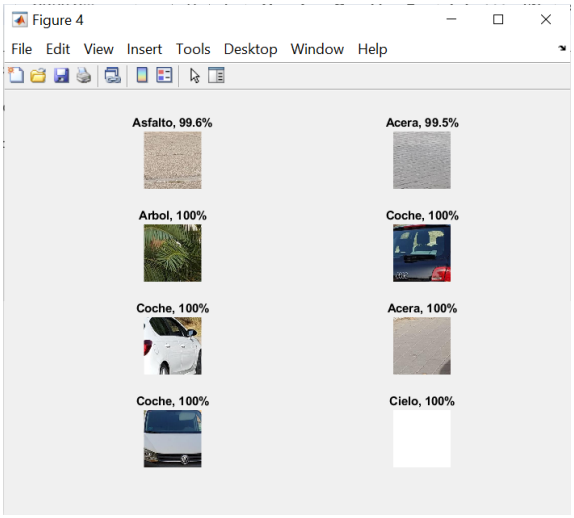


Figura 5.2: Tanda de pruebas del modelo generado con AlexNet

Otra información que Matlab ofrece al finalizar el entrenamiento son las matrices de confusión. Estas matrices utilizan un conjunto de datos del dataset para realizar el test, evaluando distintos elementos de las imágenes (acera, matrícula, coche, etc.) que ya están clasificados (etiquetados) para predecir su categoría con la red entrenada. De esta manera, dado un elemento de una imagen, la red entrenada lo clasifica en una categoría, comparando luego esta clasificación con la real de la imagen. Esta comparación genera la matriz de confusión, que proporciona información sobre las clasificaciones correctas e incorrectas. Por ejemplo, si la red clasifica incorrectamente una acera como cielo, se trata de un error. En el caso del entrenamiento de AlexNet, como se muestra en la figura 5.3, las filas representan las clases reales de los elementos y las columnas la clasificación obtenida por la red. La diagonal principal de la matriz indica dónde coinciden las clases reales con las predichas por la red. En el caso de AlexNet, la red ha acertado en todas las clasificaciones del test, logrando una media de acierto del 100 % para cada clase.

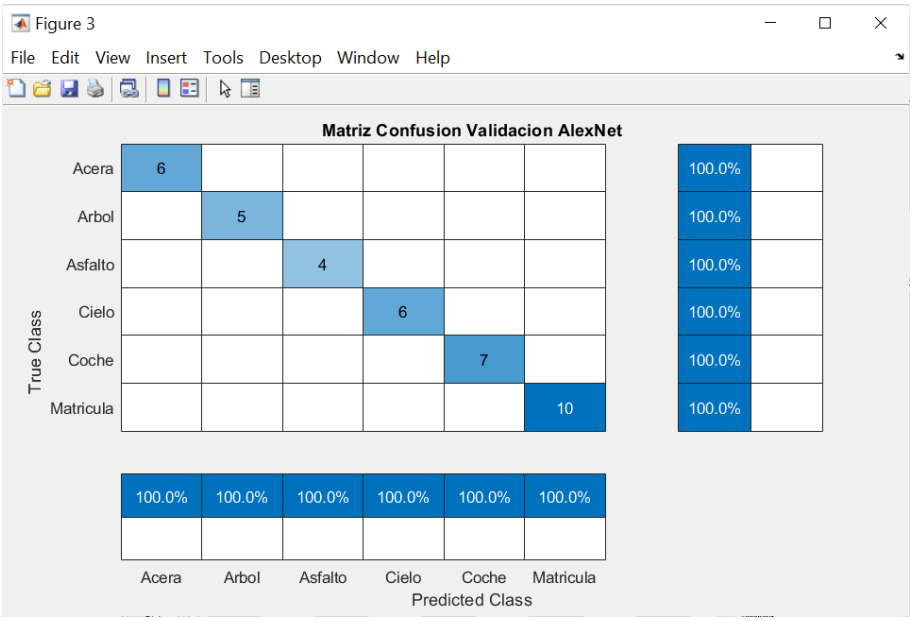


Figura 5.3: Matriz de confusión AlexNet

5.2. Resultados del proceso

Una vez tenemos las redes entrenadas se pasa a mostrar el proceso de identificación. Recordando este mismo proceso descrito en el capítulo anterior, en primer lugar se procede a detectar las zonas entre los fotogramas que contienen vehículos en movimiento para así posteriormente conseguir las regiones de estos vehículos en las que se ubican las matrículas, para a continuación detectar los caracteres, mediante el OCR.

En la imagen de la derecha correspondiente a la figura 5.4 aparece el fotograma que se encuentra analizando actualmente el proceso, en él se pueden visualizar los vectores del flujo del movimiento (flujo óptico) que en este caso los valores de su magnitud son relativamente bajos debido a que el vehículo se movía de forma lenta. Este campo vectorial determina la región concreta dentro de la imagen donde se ha detectado un movimiento significativo. El siguiente paso consiste en seleccionar esa región en movimiento, que en el caso de la Figura 5.4 se corresponde con la región recortada a partir de la imagen de la parte derecha y que contiene exactamente el vehículo, eliminando partes innecesarias de la imagen original. Este recorte es el que se muestra en la imagen de la parte izquierda de la figura 5.4. Tras la identificación del vehículo se procede a identificar posibles regiones candidatas a ser la matrícula, detectándose la región correspondiente, tal y como aparece en la parte central de la Figura 5.4.

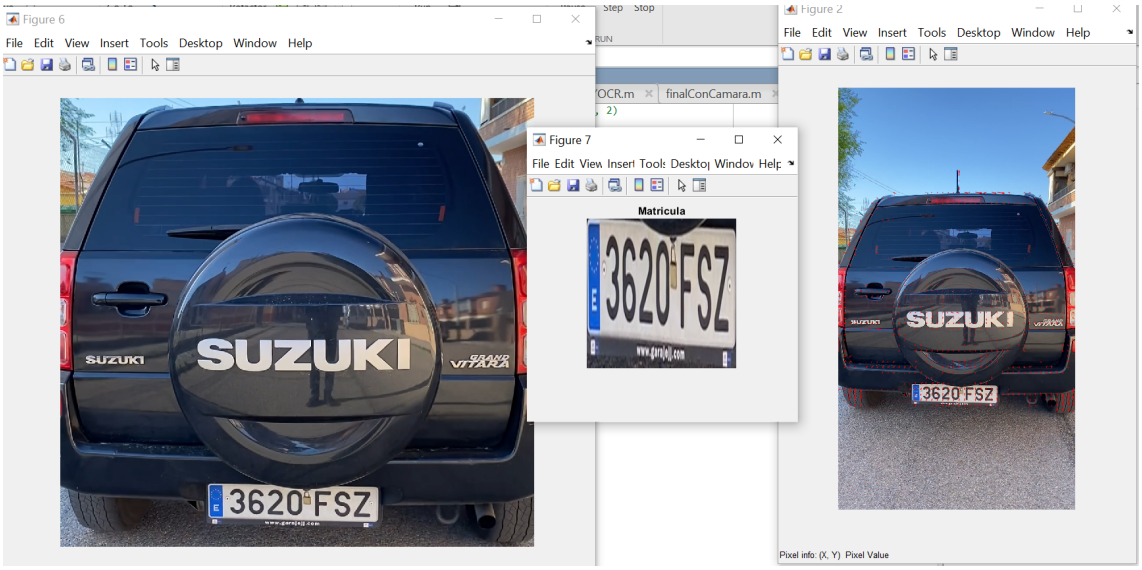


Figura 5.4: Resultados intermedios del proceso

La figura 5.5 sintetiza el resultado de todo el proceso descrito en el capítulo cuatro, en dicha figura se identifica un cuadrado delimitador (*bounding box*) un cuadrado con la identificación del vehículo, los vectores del flujo del movimiento y arriba a la izquierda la matrícula detectada. Estos vectores de movimiento son los indicados previamente durante la descripción de la figura 5.4 y que en este caso se aprecian con mayor nivel de detalle.

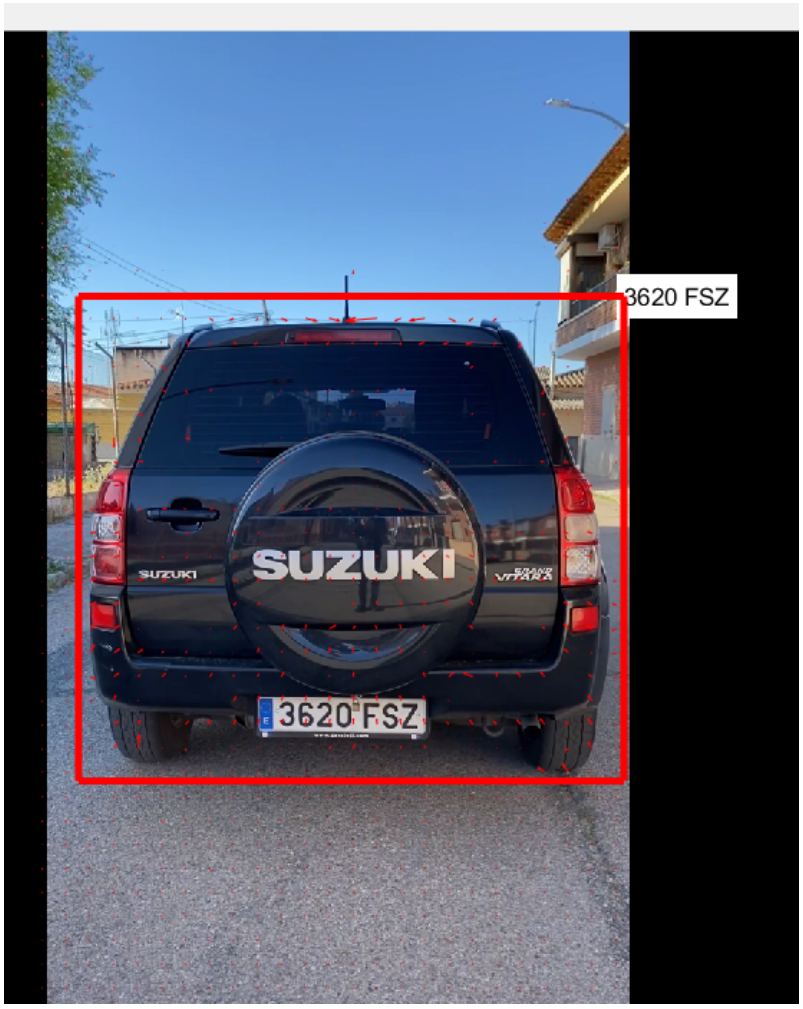


Figura 5.5: Resultado del proceso

En la mayoría de las ocasiones el proceso es capaz de detectar la matrícula correctamente pero en la figura 5.6 podemos ver un ejemplo de cómo es una mala lectura de la matrícula en uno de los fotogramas del video. En la figura se ve como la identificación del vehículo ha sido correcta tanto como la de la región de la matrícula, el fallo se ha producido a la hora de la detección de los carácteres por parte del OCR. En su mayoría los fallos suelen llegar por parte de este mismo módulo, ya que es una pieza que no ha sido reentrenada para detectar los caracteres de las matrículas sino que es un OCR que aporta MATLAB para uso general. Uno de los puntos de mejora sería el de crear un OCR propio para la detección de matrículas.

Como se ha indicado previamente, el objetivo principal del trabajo es proporcionar una solución conceptual planteada para la identificación de matrículas y su reconocimiento bajo el paradigma IoT, y por tanto, no es evaluar el desempeño de los distintos componentes por separado, es decir por un lado las CNN y por otro el OCR.

No obstante, desde el punto de vista de los resultados, cabe mencionar que, con la finalidad de analizar la aplicación en su conjunto, se han capturado cinco vídeos con una duración aproximada de 5 minutos cada uno en varias calles de la ciudad de Madrid, a razón de 30 frames por segundo con un dispositivo móvil, tipo *Samsung Galaxy S8*, como elemento sensor dentro del paradigma IoT. Los vídeos contienen secuencias de vehículos en movimiento desde diferentes perspectivas y vistas frontales tanto delanteras como traseras, procurando que aparezca la matrícula y un primer plano del vehículo como fundamental, junto con otros elementos estructurales del entorno. En algunos casos, debido al movimiento del propio vehículo, las regiones de la matrícula no han aparecido. En términos generales, se trata de imágenes diferentes a las utilizadas para el entrenamiento de las redes neuronales.

A modo de resumen y con carácter general, se puede concluir con los resultados porcentuales que se muestran en la tabla 5.1, los cuales se han determinado en base al criterio del experto humano, quien determina la correcta identificación o no. Para ello se han utilizado 532 frames extraídos aleatoriamente de las mencionadas secuencias de vídeos.

Tipos de identificaciones	**% aciertos**
Regiones conteniendo vehículos	98.70
Regiones conteniendo la matrícula dentro de un vehículo previamente identificado	96.12
Reconocimiento correcto de los caracteres que componen la matrícula	89.20

Tabla 5.1: Porcentaje de aciertos en diferentes tipos de identificaciones

A partir de estos resultados, resulta fácil deducir que la identificación de los vehículos en movimiento se realiza con un alto porcentaje de acierto, lo cual es debido a que estos aparecen de forma significativa en primer plano, tal y como se muestra en las figuras 5.4 a 5.6. Solamente en aquellos casos en los que la velocidad del vehículo es excesivamente baja, no se detecta movimiento y por tanto, no se identifica la existencia del vehículo, si bien esto no es el objetivo de esta detección. Como mejora del proceso de detección se propone añadir un módulo, por ejemplo de detección de objetos que identifique vehículos parados o con movimiento lento.

El grado de acierto en el reconocimiento de la región conteniendo la región de la matrícula, una vez detectado el vehículo, también resulta eficiente, identificando fallos de reconocimiento correspondientes a regiones que se confunden con la matrícula, como es el caso de la marca "Suzuki" en las imágenes previas o rótulos e inscripciones, principalmente en vehículos de reparto. En lo que respecta al reconocimiento de los caracteres de la matrícula, el porcentaje se sitúa por debajo del 90 %. Se trata de una situación característica achacable al OCR.

En efecto, en la Figura 5.4 se ve cómo la identificación del vehículo ha sido correcta tanto como la de la región de la matrícula, el fallo se ha producido a la hora de la detección de los caracteres por parte del OCR. Así pues, en su mayoría los fallos suelen llegar por parte de este mismo, ya que es un elemento que no ha sido reentrenado para detectar los caracteres de las matrículas sino que es el propio OCR que aporta MATLAB para uso general. Uno de los puntos de mejora sería el de crear un OCR propio para la detección de matrículas.

Figura 5.6: Ejemplo de fallo al leer

5.3. Procesamiento de los datos

En esta sección se describe el posterior procesamiento de los datos conseguidos mediante el proceso de reconocimiento descrito previamente. Para ello se utilizan la mencionada plataforma IoT ThingSpeak, que es sobre la que se centra este proceso.

5.3.1. Thingspeak

Si se accede a la plataforma Thingspeak se pueden ver todos los canales creados para el almacenamiento y procesamiento de los datos tal y como aparece en la figura 5.7, desde aquí hay que acceder al canal creado para la subida de los datos del tráfico, en él la plataforma ThingSpeak nos permite visualizar gráficas para cada campo del canal aparte de crear las nuestras propias y más opciones.

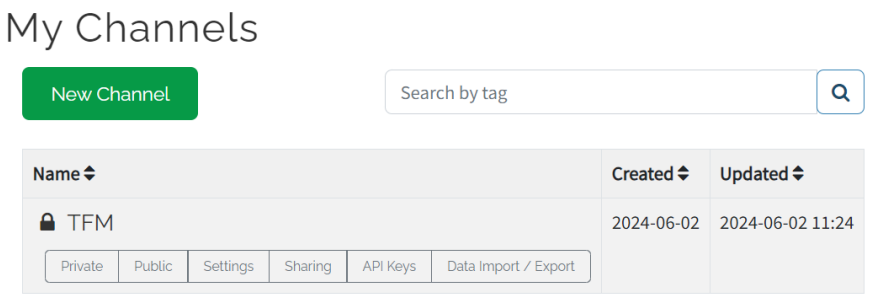


Figura 5.7: Canales de ThingSpeak

Una vez dentro en la pestaña de *Data Import/Export* se puede descargar un csv con todos los datos del canal para procesarlos posteriormente, tal y como se explicó en el capítulo anterior.

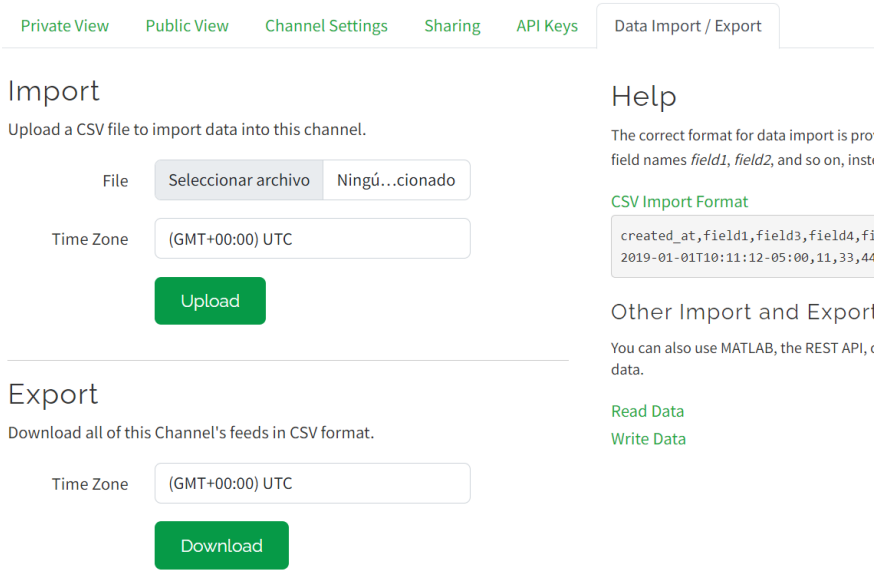


Figura 5.8: Exportar csv de ThingSpeak

5.3.2. Análisis del dataset

Llegados a este punto, ahora se trata de valorar las posibilidades y potencialidades de la plataforma ThingSpeak dentro de la solución conceptual desde la perspectiva IoT. Conviene, reseñar que los datos almacenados en los canales mencionados no se corresponden exactamente con la identificación de las matrículas de vehículos explicadas hasta el momento. Por el contrario, lo que se muestra es una ampliación simulada de los procesamientos previos. En tal sentido, la aplicación se extendería de forma que por cada tipo de vehículo en circulación se identificaría la naturaleza del mismo y su categoría, procediendo a realizar un conteo del número de vehículos que transitan por una determinada vía urbana o interurbana, con el propósito de realizar una monitorización del flujo de tráfico. La detección del tipo de vehículos se realizaría mediante los mismos modelos de redes explicados en el presente trabajo, de forma que dichos modelos se pueden perfectamente entrenar para identificar vehículos del tipo: coches, bicicletas, buses o camiones. En todos los casos

contabilizando el número de los que transitan, el día y la hora, junto con otros elementos de interés, tales como la situación instantánea del tráfico, en este caso "normal". Por otra parte, cabe la posibilidad de proceder a la identificación de las matrículas correspondientes, cuando corresponda, cuya identificación podría ser un dato adicional para incorporar en la plataforma ThingSpeak. De esta forma, se ofrece una solución conceptual ampliada, como extensión del trabajo realizado y ofreciendo las posibilidades de procesamiento de datos a través de ThingSpeak. Conviene reseñar que la plataforma ThingSpeak ofrece otros servicios complementarios tales como envío de notificaciones a través de servicios de mensajería, activación de reacciones de control programables por recepción de datos o por eventos ocurridos dentro de la propia plataforma de forma síncrona o asíncrona. Por tanto, ThingSpeak ofrece una amplia variedad de servicios en el contexto IoT. No obstante, el procesamiento de datos cuyos resultados se muestran a continuación, a diferencia de los procesos anteriores, se ha desarrollado con python y algunas de sus bibliotecas, incluyendo pandas, numpy y sklearn entre otras.

A partir de los resultados mostrados en el figura 5.9 se pueden observar 9 columnas, usando tres para representar el tiempo, cuatro con los contadores para cada tipo de vehículo además del total y por último la última con la situación del tráfico en ese momento.

	Time	Date	Day of the week	CarCount	BikeCount	BusCount	TruckCount	Total	Traffic Situation
0	12:00:00 AM	10	Tuesday	13	2	2	24	41	normal
1	12:15:00 AM	10	Tuesday	14	1	1	36	52	normal
2	12:30:00 AM	10	Tuesday	10	2	2	32	46	normal
3	12:45:00 AM	10	Tuesday	10	2	2	36	50	normal
4	1:00:00 AM	10	Tuesday	11	2	1	34	48	normal
...	...	...	...	...	...	...	...	...	...
5947	10:45:00 PM	9	Thursday	16	3	1	36	56	normal
5948	11:00:00 PM	9	Thursday	11	0	1	30	42	normal
5949	11:15:00 PM	9	Thursday	15	4	1	25	45	normal
5950	11:30:00 PM	9	Thursday	16	5	0	27	48	normal
5951	11:45:00 PM	9	Thursday	14	3	1	15	33	low

Figura 5.9: Datos del dataset de Kaggle

Ahora se comprueba que el dataset no contiene ningún valor nulo, 6 columnas son numéricas y 3 cadenas de texto. Si contamos cuántos valores existen en el dataset para cada día del mes vemos que hay 192.

Usando la librería seaborn se puede mostrar por ejemplo un gráfico *jointplot* en el que nos sitúa un punto en la gráfica donde el eje de abcisas y representa el número de vehículos, en el eje de ordenadas x se muestra el total y el color la situación del tráfico. Gracias a esto se visualiza de una manera muy gráfica cómo a medida que van subiendo el número de coches y el total, la situación del tráfico empeora.

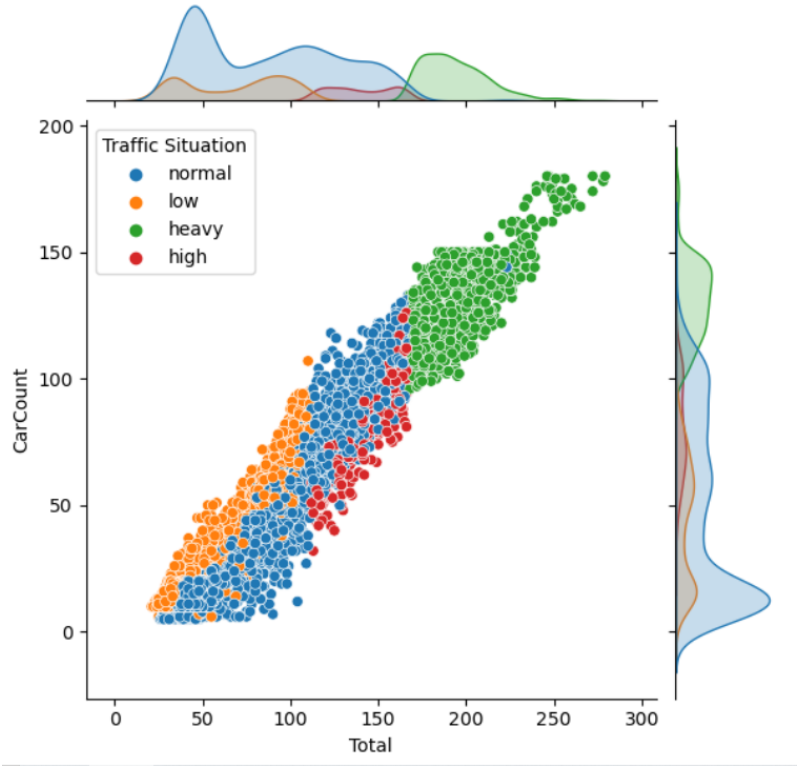


Figura 5.10: Gráfico de la situación del tráfico respecto a los coches y el total

Por último, en la figura 5.11 se muestra un mapa de calor de las correlaciones de Pearson entre todas las variables. La correlación de Pearson mide la relación lineal entre dos variables. Los valores de correlación oscilan entre -1 y 1.

- Un valor de 1 implica una correlación positiva perfecta. Esto significa que si una variable aumenta, la otra variable también aumenta en una proporción constante.
- Un valor de -1 implica una correlación negativa perfecta. Esto significa que si una variable aumenta, la otra variable disminuye en una proporción constante.
- Un valor de 0 implica que no hay correlación lineal entre las variables.

De esta forma, observando el mapa de calor, interesa fijarse en la última columna ya que es la que lo relaciona con la situación del tráfico. Podemos ver cómo las columnas del tiempo y el contador de camiones podrían ser eliminadas ya que no tienen mucho peso en la decisión final de la situación del tráfico. Mientras que el contador total, el de los coches, bicis y autobuses muestra que las variables están fuertemente correlacionadas. Igualmente para el resto del cuaderno estas variables se mantienen, ya que tampoco van a ralentizar o empeorar los modelos.

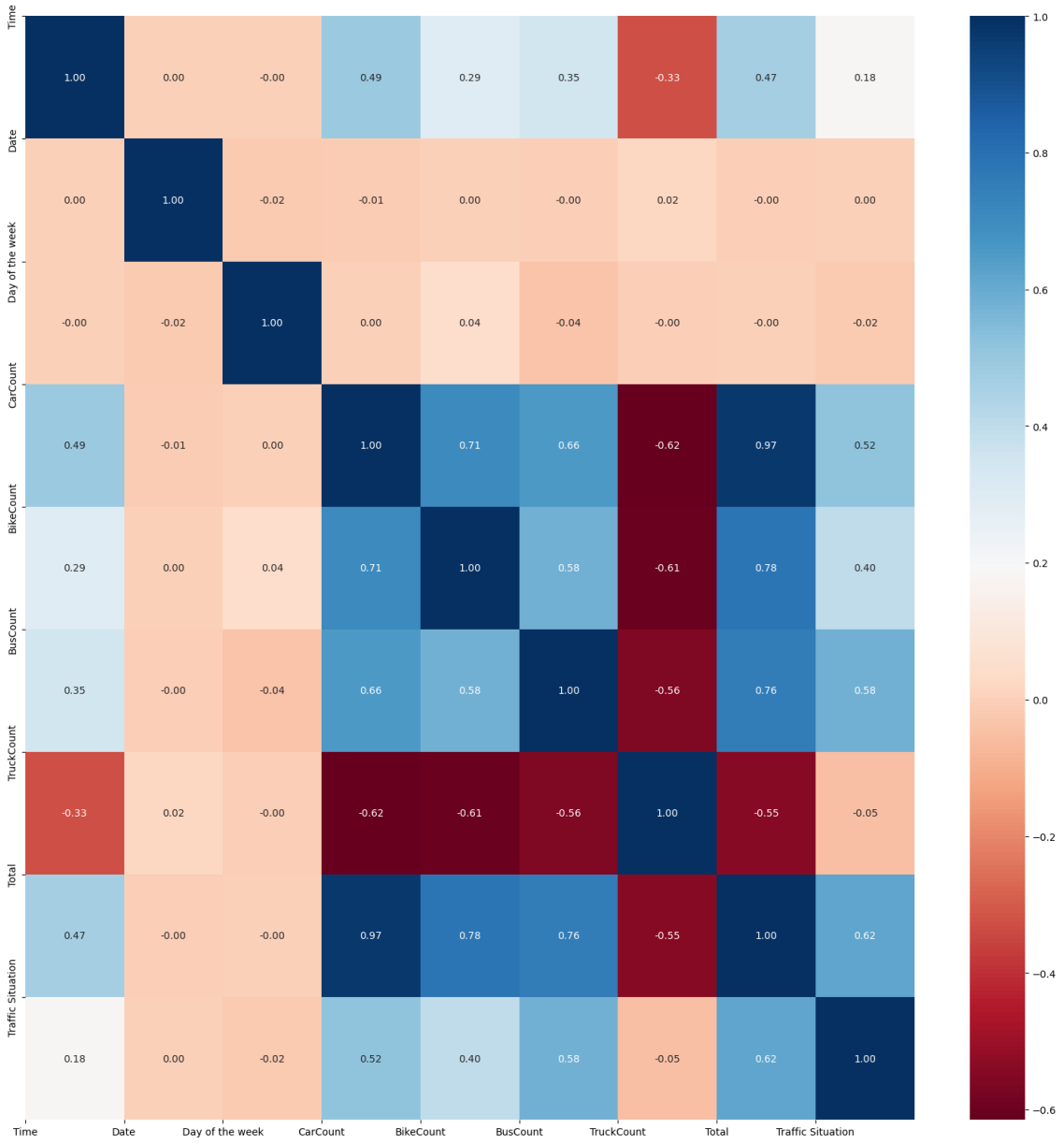


Figura 5.11: Mapa de calor de las correlaciones de Pearson

5.3.3. Modelos de IA

En los siguientes puntos, se explican los lodelos de inteligencia artificial entrenados para la predicción del tráfico. En ellos se detallan los hiperparámetros utilizados para cada modelo que se proporcionan a la función GridSearchCV, buscando la mejor combinación de estos para optimizar el rendimiento. Además, se presentan puntuaciones obtenidas para cada modelo, permitiendo una comparación de su eficacia.

5.3.3.1. Modelos de regresión

Support Vector Regression (SVR) lineal

- El parámetro `'C'` controla la compensación entre el error de entrenamiento y la complejidad del modelo. Valores más altos de `'C'` permiten un mayor margen de error, lo que puede ser útil en datos ruidosos.
- El rango de valores [0.1, 1, 10] cubre diferentes niveles de regularización, desde una alta regularización (0.1) hasta una baja regularización (10).
- El valor R2 de 0.553749 indica un ajuste moderado del modelo.

Support Vector Regression (SVR) con kernel

- Además del parámetro `'C'`, se ha incluido el parámetro `'gamma'` que controla la influencia de cada muestra en el modelo.
- Valores más altos de `'gamma'` pueden hacer que el modelo se ajuste demasiado a los datos de entrenamiento (sobreajuste).
- El rango de valores [0.1, 1, 10] para `'gamma'` cubre diferentes niveles de influencia de las muestras.
- El valor R2 de 0.788120 indica un mejor ajuste que el SVR lineal, gracias a la capacidad del kernel para capturar relaciones no lineales.

Decision Tree Regressor

- El parámetro `'max_depth'` controla la profundidad máxima del árbol de decisión, lo que afecta la complejidad del modelo.
- Un valor de `'None'` significa que no hay límite en la profundidad, lo que puede llevar a sobreajuste.
- Los valores [5, 10] limitan la profundidad del árbol, lo que puede ayudar a prevenir el sobreajuste.
- El valor R2 de 0.995854 es prácticamente perfecto, por lo que el modelo se ajusta muy bien al tipo de datos del problema.

Random Forest Regressor

- El parámetro `'n_estimators'` determina el número de árboles en el bosque aleatorio.
- Más árboles pueden mejorar el rendimiento, pero también aumentan el costo computacional.
- El rango de valores [10, 50, 100] cubre diferentes tamaños de bosque.
- El valor R2 de 0.996026 es excelente, lo que indica que el modelo se ajusta muy bien a los datos.

5.3.3.2. Modelos de clasificación

K-Nearest Neighbors (KNN)

- El parámetro `'n_neighbors'` determina el número de vecinos más cercanos que se consideran para la clasificación.
- Valores más altos de `'n_neighbors'` pueden suavizar el modelo, pero también pueden ignorar detalles locales.
- El rango de valores [3, 5, 7] cubre diferentes niveles de suavizado.
- La precisión de 0.902168 es buena, pero otros modelos como los que se indican a continuación superan su rendimiento.

Decision Tree Classifier

- Al igual que en la regresión, el parámetro `'max_depth'` controla la profundidad máxima del árbol.
- Una precisión de 0.999253 es excelente, lo que sugiere que el árbol de decisión es capaz de capturar las relaciones complejas en los datos.

Random Forest Classifier

- Al igual que en la regresión, el parámetro `'n_estimators'` determina el número de árboles en el bosque aleatorio.
- Una precisión de 0.996079 es excelente, lo que indica que el bosque aleatorio es capaz de manejar bien la complejidad de los datos y evitar el sobreajuste.

Naive Bayes

- No se han especificado parámetros para este modelo, ya que se utiliza la implementación predeterminada del clasificador Naive Bayes en scikit-learn.
- Una precisión de 0.818523 es buena, pero otros modelos como los analizados aquí superan su rendimiento en este conjunto de datos específico.

Support Vector Machine (SVM) Classifier

- Al igual que en SVR, los parámetros `'C'` y `'gamma'` controlan la compensación entre el error de entrenamiento y la complejidad del modelo, y la influencia de cada muestra, respectivamente.
- El rango de valores [0.1, 1, 10] para `'C'` y `'gamma'` cubre diferentes niveles de regularización e influencia de las muestras.
- Una precisión de 0.952204 es muy buena, lo que demuestra la capacidad del SVM para manejar datos complejos y no lineales.

Red Neuronal

- El parámetro `'hidden_layer_sizes'` especifica el número de neuronas en las capas ocultas de la red neuronal.
- Los valores [(50), (100)] prueban redes con una capa oculta de 50 y 100 neuronas, respectivamente.
- El parámetro `'alpha'` controla la regularización de la red neuronal, evitando el sobreajuste.
- Los valores [0.0001, 0.05] cubren diferentes niveles de regularización, desde una baja regularización (0.0001) hasta una regularización moderada (0.05).
- Una precisión de 0.973675 es excelente, lo que demuestra la capacidad de la red neuronal para aprender patrones complejos en los datos.

5.3.3.3. Modelos de aprendizaje automático no supervisado

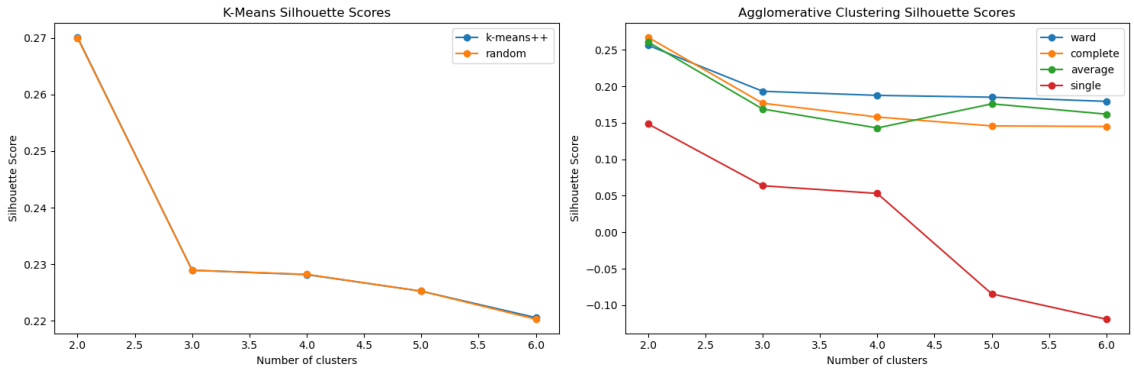


Figura 5.12: Gráfica con los valores de silueta

K-Means

- n_clusters_values = [2, 3, 4, 5, 6]: este parámetro determina el número de *clusters* a formar. Se han probado diferentes valores para explorar la estructura de los datos.
- `init_methods = ['k-means++', 'random']`: este parámetro controla el método de inicialización de los centroides de los *clusters*. `'k-means++'` es un método inteligente que intenta inicializar los centroides de manera dispersa, mientras que `'random'` los inicializa de forma aleatoria.

Los resultados muestran que, para K-Means, los métodos de inicialización `'k-means++'` y `'random'` tienen valores de silueta (silhouette) similares para diferentes valores de n_clusters. El valor de silueta más alto se obtiene con 2 *clusters* (0.27), lo que sugiere que esta podría ser una buena opción para agrupar los datos.

Clustering Jerárquico (*Agglomerative Clustering*)

- n_clusters_values = [2, 3, 4, 5, 6]: al igual que en K-Means, se prueban diferentes números de clusters.

- `linkage_criteria = ['ward', 'complete', áverage', 'single']`: este parámetro determina el método de vinculación utilizado para calcular la distancia entre clusters. `'ward'` minimiza la varianza dentro de los clusters, `'complete'` utiliza la distancia máxima entre puntos, `'average'` utiliza la distancia promedio, y `'single'` utiliza la distancia mínima.

En los resultados del *clustering* Jerárquico, se observa que el método de vinculación `'ward'` generalmente obtiene los valores de silueta más altos para diferentes valores de n_clusters. Concretamente, el valor de silueta más alto se obtiene con 2 *clusters* y el método `'ward'` (0.25), lo que sugiere que esta combinación de parámetros podría ser una buena opción para agrupar los datos.

El método de vinculación `'complete'` también muestra valores de silueta relativamente altos, especialmente para 2 *clusters* (0.256). Este método tiende a formar *clusters* más compactos y puede ser útil cuando se busca una separación clara entre los grupos.

Por otro lado, los métodos `'average'` y `'single'` generalmente obtienen valores de silueta más bajos, lo que indica que estos métodos de vinculación pueden no ser tan efectivos.

Es importante tener en cuenta que el valor de silueta es una métrica que evalúa la calidad de los *clusters* formados, considerando la cohesión dentro de los *clusters* y la separación entre ellos. Un valor de silueta más alto indica que los *clusters* están bien definidos y separados.

En resumen, se puede concluir que el mejor comportamiento a la hora de predecir es el de los modelos de clasificación o regresión, pero proporcionando los de clasificación mejores notas de media entre todos los modelos que se han probado. Los resultados del *clustering* también sugieren que dividir la predicción en dos clases en lugar de cuatro puede ser una opción más efectiva.

Capítulo 6

Conclusiones y Trabajo Futuro

6.1. Conclusiones

En el presente trabajo se ha planteado y desarrollado una solución conceptual dentro del paradigma IoT. Concretamente se trata de identificar características de vehículos en movimiento en vías urbanas o interurbanas, y como parte esencial la región de la matrícula y el reconocimiento de los caracteres que la conforman.

Con tal propósito se capturan datos procedentes de un sensor, en este caso una cámara de un dispositivo móvil. El procesamiento de estas imágenes se lleva a cabo mediante técnicas avanzadas de tratamiento de imágenes, utilizando procedimientos de detección del movimiento, identificación del vehículo y reconocimiento de la matrícula del vehículo.

Para el reconocimiento tanto del propio vehículo como de la región de la matrícula se utilizan modelos de redes neuronales convolucionales (CNN) pre-entrenadas, de forma que sólo se necesita una readaptación de los modelos para ajustarlos al tipo de datos requeridos y a las salidas deseadas, en cuanto al número de clase se refiere.

Una vez reconocida la región de la matrícula, el siguiente paso consiste en identificar los caracteres que la conforman, para ello se ha utilizado un OCR de carácter genérico.

Los resultados del procesamiento de las imágenes se almacenan en la nube, para ello se utiliza la plataforma ThingSpeak específica de IoT.

Todos estos procesos se han llevado a cabo a través de distintas funcionalidades convenientemente integradas, ofreciendo una solución IoT de concepto, y proporcionando distintas posibilidades de utilidad desde el punto de vista práctico y de futuro dentro de las denominadas ciudades inteligentes.

En cada uno de los distintos procesos se muestran los resultados más relevantes, siendo el objetivo el proporcionar dicha solución conceptual, antes que la evaluación de las diferentes técnicas, que constituyen un análisis fuera del contexto IoT del presente trabajo. Por tanto, desde este punto de vista, se han cumplido los objetivos inicialmente planteados

6.2. Trabajo futuro

Durante el desarrollo del proyecto se han identificado determinados aspectos que merecen una especial consideración de futuro, destacando los siguientes:

1. Investigar procesos de mejora en cuanto al tratamiento de las imágenes, en particular mejorar la eficiencia de las CNN y el OCR para adaptarlos lo mejor posible al entorno operativo.
2. Ampliar y explotar los recursos de procesamiento en la nube dentro de ThingSpaek, desarrollando scripts específicos de proceso.
3. Ampliar la base de datos de imágenes para los entornos considerados.
4. Añadir un módulo para la detección de vehículos parados o con un movimiento lento a través de detectores de objetos tipo YOLO u otros, también dentro del ámbito de las CNN.

Chapter 7

Introduction

The purpose of this Master's Thesis is to study, implement, and integrate computer vision and machine learning techniques for the detection of general features in moving vehicles, including the vehicle itself within the image and the recognition of the region that defines the plate and the characters that make it up, all under the Internet of Things (IoT) paradigm..

There is no doubt that the technological explosion we are immersed in greatly facilitates the development of increasingly powerful new technologies. In this regard, IoT emerges as a particularly interesting paradigm in various technological fields, including vehicle control from both traffic regulation and access to available resources, such as parking areas. These aspects are especially relevant today and are poised to be key in the future developments of Smart Cities.

In this context, various technology companies are focused on achieving these objectives. For instance, Vision (2024) focuses primarily on the development of "comprehensive computer vision solutions, both software and hardware, applied to vehicle control," utilizing intelligent techniques based on computer vision for license plate recognition in vehicle images. These techniques include Deep Learning (DL) and Optical Character Recognition (OCR).

Similarly, various strategies are directly focused on the IoT domain for vehicle traffic control, surveillance, and monitoring by identifying license plates as part of vehicle type identification. An example is Gigabyte (2024), a leading technology company proposing solutions for traffic monitoring, speed and traffic light control, or city and parking access. These technological solutions include advanced video processing, deep learning, and efficient data transmission within the IoT system. Connectivity among different sensory and processing systems is an essential feature of Smart Cities (AVUTEC, 2024).

There are solutions in the literature addressing the license plate recognition problem from different perspectives, largely through machine learning. Wang et al. (2019) review various techniques, including traditional learning methods like Scale-Invariant Feature Transform (SIFT) (Lowe, 1999) and Local Binary Patterns (LBP) (Ojala et al., 2002), as well as Deep Learning techniques (He et al., 2019; Peng et al., 2019). In these latter works, image processing occurs in two stages: first, recognizing candidate regions for license plates, and second, recognizing the characters composing the plate. This strategy

inspires the present work, which also features two stages progressing from a global analysis to refining the license plate itself.

Moreover, given the technological advancements in Deep Learning and IoT, several advanced solutions incorporate both paradigms, such as the previously mentioned strategies and other specific developments prioritizing edge processing (Ammar et al., 2023).

This work proposes a conceptual solution within the IoT paradigm, applying intelligent techniques based on Deep Learning to images captured by a color camera (sensor) with cloud-based information processing.

7.1. Motivation

Within the different solutions and strategies presented by technology companies and the previously mentioned references, the following essential characteristics stand out:

1. Image and/or video processing

2. Use of deep learning methods for recognition purposes, in this case, license plates

3. Integration of sensory technologies (image capture cameras) within an IoT ecosystem

The motivation for this work originates from the previous development presented in the Bachelor's Thesis for the 2022-23 academic year (Núñez et al., 2023). The author of this thesis is also a co-author of the aforementioned work Núñez et al. (2023). In the latter, license plate recognition was approached in a very preliminary manner using computer vision and deep learning techniques. Following this approach, specific needs and deficiencies were identified for its implementation as a truly operational system. This led to the idea of developing and configuring a system with certain applicability guarantees in real environments where connectivity between sensors, platforms, and cloud processing is the driving force for future developments in Smart Cities. This extension and improvement are proposed considering the IoT paradigm, addressing the identified deficiencies while also solving the detected shortcomings. However, developing a system of this nature is particularly challenging, especially considering that comprehensive intelligent systems within Smart Cities are still in an operational development phase, particularly regarding their connectivity.

It is under this framework and considering the previously expressed future perspectives that this work is situated and motivated. From a motivational standpoint, the integration of specific techniques for each technical characteristic mentioned earlier is proposed, particularly addressing the connectivity within the IoT paradigm. Each integrated technique requires its own adaptation process to address the overall problem of vehicle license plate identification, especially for moving vehicles. From the perspective of reusing techniques and methods, part of the work developed in Núñez et al. (2023) is utilized in developing the application presented, both from a theoretical-conceptual and functional-modular implementation standpoint. This approach is justified by leveraging existing and developed synergies while focusing on the necessary integration and connectivity in such a framework. This is why certain sufficiently mature concepts and developments proposed in Núñez et al. (2023) are reused in this work.

7.2. Objectives

As mentioned earlier, the primary objective of this work is to propose an intelligent conceptual solution within the IoT paradigm for recognizing license plates of moving and/or stationary vehicles.

Under this consideration, the following specific objectives arise:

1. Analyze IoT models that allow image capture, intelligent processing using DL techniques, including OCRs, and manage information through a specific IoT platform.

2. Study computer vision techniques for motion analysis in image sequences.

3. Study specific DL techniques for image recognition to detect both moving vehicles and the regions where license plates are located.

4. Study OCR techniques for recognizing license plate characters.

5. Verify the different developments and integrate the considered feasible techniques.

6. Extract conclusions and possible options for the present and future.

7.3. Work Plan

The work plan describes the set of basic activities aimed at achieving the previous objectives. In any case, regarding the approach of these activities, the need for their review as they progress is considered to detect and correct deficiencies in an iterative process.

In this regard, the planned and carried out activities are as follows:

1. Structuring the technological solution proposed in the work, defining and scoping the objectives, and determining the timelines or milestones during development.

2. Studying existing technologies from the following perspectives: image processing methods, DL-based recognition techniques, and data management through an IoT platform.

3. Developing a procedure based on motion detection in image sequences to identify areas of interest related to moving vehicles.

4. Developing methods based on Convolutional Neural Networks within the DL paradigm for recognizing regions of interest in images, including the area where the license plate is located and subsequently recognizing the characters forming it.

5. Configuring an IoT platform for managing the information extracted from image sequences.

6. Integrating the previously indicated developments and configurations.

7. Writing and drafting the thesis.

7.4. Organization of the Thesis

The thesis is structured, according to the previous considerations, as follows: Chapter Two describes the theoretical concepts related to image processing, distinguishing between basic and advanced techniques. Chapter Three introduces the concepts associated with Deep Learning, including various convolutional neural network models. Chapter Four describes the modular design of the complete IoT system with its different integrated parts. Chapter Five presents the results obtained through the developed application. Finally, Chapter Six discusses the conclusions and future work.

Chapter 8

Conclusions and Future Work

8.1. Conclusions

In this work, a conceptual solution within the IoT paradigm has been proposed and developed. Specifically, the aim is to identify characteristics of vehicles in motion on urban or interurban roads, the essential part of which is the region of the number plate and the recognition of the characters that make up the number plate.

To this end, data is captured from a sensor, in this case, a mobile device camera. The processing of these images is carried out using advanced image processing techniques, utilizing procedures for motion detection, vehicle identification, and license plate recognition.

For recognizing both the vehicle itself and the license plate region, pre-trained convolutional neural network (CNN) models are used, requiring only a readaptation of the models to adjust them to the required data types and desired outputs in terms of the number of classes.

Once the license plate region is recognized, the next step is to identify the characters that compose it, for which a generic OCR is used.

The results of the image processing are stored in the cloud, using the IoT-specific platform ThingSpeak.

All these processes have been carried out through various conveniently integrated functionalities, offering a conceptual IoT solution and providing various practical and future utility possibilities within the so-called smart cities.

In each of the different processes, the most relevant results are provided, with the objective being to provide this conceptual solution rather than evaluating the different techniques, which constitutes an analysis beyond the IoT context of this work. Therefore, from this point of view, the initially set objectives have been met.

8.2. Future Work

During the development of the project, certain aspects have been identified that deserve special consideration for the future, highlighting the following:

1. Investigate improvement processes for image processing, particularly improving the efficiency of CNNs and OCR to best adapt them to the operational environment.
2. Expand and exploit cloud processing resources within ThingSpeak by developing specific processing scripts.
3. Expand the image database for the considered environments.
4. Add a module for detection of stationary or slow moving vehicles through YOLO type object detectors or others, also within the scope of CNNs.

Bibliografía

H. Aman. Kaggle dataset, 2024. URL `https://www.kaggle.com/datasets/hasibullahaman/traffic-prediction-dataset`. Accedido Abril 2024.

A. Ammar, A. Koubaa, W. Boulila, B. Benjdira, and Y. Alhabashi. A multi-stage deep-learning-based vehicle and license plate recognition system with real-time edge inference. *Sensors*, 23(4):2120, 2023. doi: 10.3390/s23042120.

AVUTEC. On.device anpr and iot connectivity, 2024. URL `https://avutec.com/smart-city/`. Accedido Octubre 2023.

D.H. Ballard and C.M. Brown. *Computer Vision*. Prentice-Hall, Englewood Cliffs, NJ, 1982.

J.L. Barron, D.J. Fleet, and S. Beauchemin. Performance of optical flow techniques. *International Journal of Computer Vision*, 12:43–77, 1994.

C.M. Bishop. *Pattern Recognition and Machine Learning*. Springer, NY, USA, 2006.

H. Chen, C.H. Tsai, G. Schroth, D.M. Chan, R. Grzeszczuk, and B. Girod. Robust text detection in natural images with edge-enhanced maximally stable extremal regions. In *Image Processing (ICIP), 2011 18th IEEE International Conference on*. IEEE, 2011.

M. Courbariaux, Y. Bengio, and J.P. David. Binaryconnect: Training deep neural networks with binary weights during propagations. *arXiv:1511.00363v3*, 2015.

E.R. Davies. *Computer Vision: Principles, Algorithms, Applications, Learning*. Academic Press, London, UK, 2018.

R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern Classification*. John Wiley and Sons, Inc., New York, US, 2001.

Gigabyte. License plate recognition in smart city, 2024. URL `https://www.gigabyte.com/Industry-Solutions/license-plate-recognition-in-smart-city`. Accedido Octubre 2023.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. Disponible on-line: `https://www.deeplearningbook.org/` (Accedido Diciembre 2020).

R.M. Haralick, S.R. Stenberg, and X. Zhuang. Image analysis using mathematical morphology. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(4):532–550, 1987.

B. He, J. Li, Y. Zhao, and Y. Tian. Part-regularized near-duplicate vehicle re-identification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3997–4005, 2019.

K. He and J. Sun. Convolutional neural networks at constrained time cost. *arXiv:1412.1710 [cs.CV]*, 2015.

M.K. Hu. Visual pattern recognition by moment invariants. *IRE Transactions on Information Theory*, 8(2):179–187, 1962.

F.N. Iandola, S. Han, M.W. Moskewicz, K. Ashraf, W.J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size. *arXiv:1602.07360v4 [cs.CV]*, 2016.

ImageNet. Imagenet, 2020. URL `http://www.image-net.org`. Accedido Diciembre 2020.

D.P. Kingma and J.L. Ba. Adam: A method for stochastic optimization. In *Proc. 3rd International Conference on Learning Representations (ICLR 2015)*, pages 1–15, 2014.

A. Krizhevsky, I. Sutskever, and G.E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2012.

Y. Li and H. Lu. Scene text detection via stroke width. In *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, pages 681–684, 2012.

B.T. Low, H.C. Lui, A.H. Tan, and H.H. Teh. Connectionist expert system with adaptive learning capability. *IEEE Transactions on Knowledge and Data Engineering*, 3:313–324, 1991.

D.G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 2, pages 1150–1157, Kerkyra, Greece, 1999. doi: 10.1109/ICCV.1999.790410.

ImageNet LSVRC. Large scale visual recognition challenge 2012 (ilsvrc2012), 2012. URL `http://www.image-net.org/challenges/LSVRC/2012/`. Accedido Marzo 2024.

ImageNet LSVRC. Large scale visual recognition challenge 2014 (ilsvrc2014), 2014. URL `http://image-net.org/challenges/LSVRC/2014/eccv2014`. Accedido Marzo 2024.

J. Matas, O. Chum, M. Urban, and T. Pajdla. Robust wide baseline stereo from maximally stable extremal regions. In *Proc. British Machine Vision Conference (BMVC)*, 2002.

Matlab. The mathworks, 2024a. URL `https://es.mathworks.com/`. Accedido Marzo 2024.

Matlab. Thingspeak, 2024b. URL `https://thingspeak.com/`. Accedido Mayo 2024.

BVLC AlexNet Model. Bvlc alexnet model, 2023. URL `https://github.com/BVLC/caffe/tree/master/models/bvlc_alexnet`. Accedido Marzo 2024.

BVLC GoogLeNet Model. Bvlc googlenet model, 2020. URL `https://github.com/BVLC/caffe/tree/master/models/bvlc_googlenet`. Accedido Marzo 2024.

V.S. Nalwa. *A Guided Tour of Computer Vision*. Addison-Wesley, Reading, MA, 1993.

L. Neumann and J. Matas. Real-time scene text localization and recognition. In *Proc. 2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3538–3545, 2012.

A. Nomura, H. Miike, and K. Koga. Detecting a velocity field from sequential images under time-varying illumination. In V. Capellini, editor, *Time-Varying Image Processing and Moving Object Recognition*, volume 3, pages 343–350. Elsevier, Amsterdam, 1994.

J.M. Núñez, S. Tapia, and A. Lloret. *Reconocimiento de matrículas de vehículos en imágenes mediante técnicas de Aprendizaje Profundo aplicado a TIC*. Docta Complutense, 2023. Accedido Octubre 2023. Disponible on-line: `https://docta.ucm.es/search?query=Reconocimiento%20de%20matr%C3%ADculas%20de%20veh%C3%ADculos`.

T. Ojala, M. Pietikäinen, and T. Mäenpää. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):971–987, 2002.

N. Otsu. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66, 1979.

G. Pajares and J.M. de la Cruz. *Visión por Computador: imágenes digitales y aplicaciones*. RA-MA, 2007.

G. Pajares, P.J. Herrera, and E. Besada. *Aprendizaje Profundo*. RC-Libros, Madrid, 2021.

J. Peng, H. Wang, T. Zhao, and X. Fu. Learning multi-region features for vehicle re-identification with context-based ranking method. *Neurocomputing*, 259:427–437, 2019.

S. Ruder. An overview of gradient descent optimization algorithms. *arXiv:1609.04747v2 [cs.LG]*, 2017.

O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A.C. Berg, and L. Fei-Fei. Imagenet large scale visual recognition challenge. *Int. J. Computer Vision*, 115:211–252, 2015.

J. Serra. *Image Analysis and Mathematical Morphology*. Academic Press, London, 1982.

R. Smith. An overview of the tesseract ocr engine. In *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, page 629–33, 2007. doi: 10.1109/ICDAR.2007.4376991.

M. Sonka, V. Hlavac, and R. Boyle. *Image Processing, Analysis and Machine Vision*. Chapman & Hall, Cambridge, 1995.

C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *Computing Research Repository*, 2014.

Lector Vision. Smart city solutions, 2024. URL `https://www.lectorvision.com/soluciones/smart-cities-es`. Accedido Octubre 2023.

H. Wang, J. Hou, and N. Chen. A survey of vehicle re-identification based on deep learning. *IEEE Access*, 7:172443–172469, 2019.

Apéndice A

Instrucciones de instalación

En este Apéndice se proporciona, en primer lugar el enlace al repositorio donde se ubican tanto el código como los modelos. En segundo lugar, se detallan, a modo de Manual de Usuario, los pasos a seguir para la instalación y ejecución de los módulos basados en los códigos correspondientes (Matlab y Python), como se ha indicado en el capítulo cinco.

A.1. repositorio del proyecto

La ubicación del repositorio, es decir el enlace a la web es el que se indica a continuación. Además, los dos modelos (1 y 2) necesarios para la ejecución del código Matlab se encuentran en enlaces compartidos de Google Drive.

Repositorio con el código: `https://github.com/jmnse8/TFM`.

Enlace al primer modelo: modelo 1

Enlace al segundo modelo: modelo 2

A.2. Manual de usuario

Los cuatro pasos secuenciales a seguir, tanto para la instalación del código como para su ejecución, son los que se indican a continuación. Se proporcionan las instrucciones necesarias para tal fin, teniendo en cuenta que los detalles específicos son los que se proporcionan en los capítulos cuatro y cinco.

Paso 1
Instalar Matlab con los recursos necesarios para deep learning y python.

Paso 2
Clonar el proyecto del repositorio.

Descargar los dos modelos de DL y moverlos a la carpeta *matlab*.

Paso 3

Crear una cuenta en ThingSpeak y posteriormente un canal.

Paso 4

En la carpeta de matlab se encuentra el código referente a la detección de las matrículas. Para usarlo se necesita los dos modelos subidos en *Google Drive*. También se tiene que añadir las claves del canal creado en ThingSpeak y sustituir las variables *ChannelID*, *readAPIKey* y *writeAPIKey*. Por último, hay que ejecutar el fichero llamado *Vehiculos_ ThingSpeak.m*.

Si se desea ejecutar la versión móvil, primero tienes que ejecutar el fichero *InicioCamaraMovimiento.m* que inicializará las variables y después ya se puede ejecutar el fichero *finalConCamara.m*.

En la carpeta llamada *python* se encuentra el *notebook* con el código para analizar el dataset.