Universidad de Valladolid

Facultad de Ciencias

Grado en Física

Clasificación automática de imágenes del cielo mediante Inteligencia Artificial

Autor: Víctor Marcos Garrachón

Tutores: Roberto Román Diez y Juan Carlos Antuña Sánchez

Curso Académico: 2023-2024

Resumen

Conocer la cobertura nubosa es de gran importancia para realizar estudios del clima. Es por ello que a lo largo de este Trabajo Fin de Grado se desarrollará un algoritmo de Deep Learning para automatizar el proceso de clasificación de imágenes del cielo según el número de octas. En primer lugar, se introducen conceptos como las diferentes capas de la atmósfera, algunas propiedades físicas de la misma, los gases que la componen, los aerosoles atmosféricos y las nubes. Posteriormente se introducen conceptos teóricos sobre la Inteligencia Artificial y más concretamente sobre las redes neuronales y en particular sobre las Redes Neuronales Convolucionales. A continuación se muestra el procedimiento que se llevó a cabo: clasificación manual de imágenes, separación de imágenes en los conjuntos de entrenamiento, validación y test y por último se indica la arquitectura de la Red Neuronal utilizada. En el último apartado se desarrolla un análisis estadístico de los resultados obtenidos por la Red Neuronal.

Abstract

Knowing cloud coverty is of great importance for conducting climate studies. Therefore, throughout this Final Degree Project, a Deep Learning algorithm will be developed to automate the process of classifying sky images according to the number of octas. First, concepts such as the different layers of the atmosphere, some of its physical properties, the gases that compose it, atmospheric aerosols, and clouds are introduced. Subsequently, theoretical concepts about Artificial Intelligence, specifically neural networks, and particularly Convolutional Neural Networks, are introduced. Next, the procedure carried out is shown: manual classification of images, separation of images into training, validation, and test sets, and finally, the architecture of the Neural Network used is indicated. In the last section, a statistical analysis of the results obtained by the Neural Network is developed.

Índice general

Capítulo 1

Motivación y objetivos

En los útimos años, el clima esta siendo objeto de muchos estudios de investigación científica, debido, en parte, a la preocupación sobre el cambio climático y su impacto en el planeta. Con el objetivo de entender y predecir cómo está cambiando el clima, es necesario recopilar datos sobre diferentes factores meteorológicos y climáticos como pueden ser la temperatura, la velocidad y dirección del viento, la humedad o la presión atmosférica. La cobertura nubosa, que representa la cantidad de cielo que se encuentra cubierto por nubes, es también fundamental para poder realizar este tipo de estudios.

Existen diversas formas de determinar la cobertura nubosa en una determinada zona:

- Observaciones satelitales: Los satélites meteorológicos equipados con cámaras o sensores infrarrojos pueden capturar imágenes de la cobertura nubosa desde el espacio. Estas imágenes proporcionan una vista global de la cobertura de nubes en grandes áreas geográficas.
- Radares meteorológicos: Los radares meteorológicos emiten ondas electromagnéticas que interactúan con las partículas de agua en las nubes. Analizando el retorno de estas señales, se puede determinar la ubicación, altura y densidad de las nubes.
- Observaciones visuales: Consiste en analizar la cobertura nubosa observando directamente el cielo de forma visual.
- Fotografías del cielo: Se analiza la cobertura nubosa a partir de imágenes del cielo con cámaras situadas normalmente en la superficie terrestre.

Para cuantizar la cubierta de nubes se hace uso de las octas, que representa los octavos del cielo cubierto por nubes. Se trata de una escala que va desde el número 0 hasta el 8 e indica el grado de cobertura por nubes de la atmósfera. Es decir, según el número de octas, consideraremos que el cielo se encuentra más o menos cubierto por nubes:

- Cielo despejado: 0 octas.
- Cielo poco nuboso: de 1 a 3 octas.
- Cielo nuboso: 4 o 5 octas.
- Cielo muy nuboso: 6 o 7 octas.
- Cielo totalmente cubierto: 8 octas.

1.1. Planteamiento del problema y objetivo del Trabajo Fin de Grado

Tradicionalmente, para determinar la cobertura nubosa en octas, se anotan manualmente los valores de las octas a partir de observaciones visuales directas. Sin embargo, si se requiere observar el cielo con una frecuencia alta, esta tarea resulta complicada. Es por ello que se utilizan cámaras, que capturan imágenes del cielo con una mayor frecuencia. Como se ha mencionado, en ocasiones, es de utilidad tomar imágenes del cielo repetidamente con una frecuencia relativamente alta, como podría ser, por ejemplo, cada 5 minutos. Esto a lo largo de varias semanas genera una cantidad considerablemente grande de imágenes, por lo que determinar el número de octas de cada una de ellas puede ser algo tedioso. Sin embargo, el probema al que nos enfrentamos es al de clasificación de imágenes (entre numeros del 0 al 8 octas según el grado de cobertura del cielo) y con el surgimiento de las inteligencias artificiales, y más concretamente con el Deep Learning y las Redes Neuronales Convolucionales (CNN), se ha conseguido crear clasificadores de imágenes que logran una precisión elevada.

Es por ello que **el objetivo de ete Trabajo Fin de Grado es** el de automatizar el proceso de clasificación de imágenes del cielo según su cobertura nubosa en octas. El método a proceder será el siguiente: se diseñará y entrenará una Red Neuronal Convolucional para que sea capaz de indicar el número de octas que contiene una imágen del cielo. Posteriormente, se realizará un estudio estadístico para determinar como de eficaz es esta Red Neuronal.

Previamente a explicar la metodología desarrollada para crear este clasificador automático, desarrollaremos dos aspectos teóricos de importancia en este TFG:

- La atmósfera terrestre.
- La Inteligencia Artificial (IA), donde nos centraremos en explicar las Redes Neuronales, y más concretamente las CNN.

Capítulo 2

La atmósfera terrestre

2.1. Capas de la atmósfera terrestre

La atmósfera terrestre se define como una capa de gases que rodea la Tierra [1]. Debido a la gravedad y a la compresibilidad de los gases, la mayor parte de la masa estos (el 80 %) se encuentra en la troposfera[2]. Sin embargo, las proporciones de los diferentes gases, lo que coloquialmente se conoce como aire, se mantienen casi inalterables hasta los 80-100 Km de altitud. El límite superior de la atmósfera se estima alrededor de los 10.000 Km de altura donde la concentración de gases es tan baja (prácticamente despreciable) que se asemeja a la del espacio exterior.

La atmósfera se clasifica según las siguientes capas:

- La troposfera: Comienza en la superficie terrestre y el límite superior se encuentra a unos 12 Km, aunque varía entre el ecuador y los polos, ya que en el ecuador este límite se encuentra más arriba. Se trata de la capa más importante para la meteorología. En ella se sitúan las nubes, ya que la troposfera contiene prácticamente todo el vapor de agua atmosférico. En la troposfera también se concentran prácticamente todos los aerosoles, partículas sólidas y líquidas en suspensión, tales como el polvo mineral procedente de los desiertos, partículas de humo de quema forestal, y también procedentes de la contaminación antropogénica. Este polvo actúa como núcleos de condensación que facilitan el paso del vapor de agua atmosférico a agua líquida. En esta capa hay importantes flujos convectivos de aire, verticales y horizontales, producidos por las diferencias de presión y temperatura que dan lugar a los fenómenos meteorológicos (precipitaciones, viento, nubes). El aire de la troposfera se calienta a partir del calor emitido por la superficie terrestre[3]. La temperatura de la troposfera es máxima en su parte inferior, alrededor de 15 ºC de media, y a partir de ahí comienza a descender con la altura según un Gradiente Térmico Vertical (GTV) de 6,5 ºC de descenso cada Km que se asciende en altura (la temperatura baja 0,65 ºC cada 100m de altura) hasta llegar a -70 ºC en el límite superior de la troposfera: la tropopausa.

- La estratosfera[4]: se extiende desde la tropopausa hasta los 50 Km de altura, límite de la estratosfera llamado estratopausa. En esta capa se genera la mayor parte del ozono atmosférico que se concentra entre los 15 y 30 Km de altura llamándose a esta zona capa de ozono u ozonosfera. La temperatura asciende con la altura hasta llegar próximo a los 0 ºC en la estratopausa. Este incremento de temperatura está relacionado con la absorción por el ozono de la radiación solar ultravioleta (UV), por lo que esta capa actúa como pantalla protectora frente a los perjudiciales rayos ultravioleta. Dentro de esta capa hay movimientos horizontales de aire, pero no verticales como sucede en la troposfera.

- Mesosfera (50-80 km): La temperatura disminuye hasta alcanzar los -90 ºC en su límite

superior llamado mesopausa. La mayoría de los meteoroides se desintegran en esta capa.

- Termosfera o ionosfera[5] (80-500 km): se denomina así porque gran parte de las moléculas presentes están ionizadas por la absorción de las radiaciones solares de alta energía (rayos gamma, rayos X y parte de la radiación ultravioleta), provocando que el nitrógeno y el oxígeno pierdan electrones quedando ionizados con carga positiva, los electrones desprendidos originan campos eléctricos por toda la capa. La interacción de las partículas subatómicas procedentes del Sol con los átomos ionizados da lugar a fenómenos luminosos llamados auroras polares (aurora boreal en polo norte y aurora austral en polo sur) que suceden cerca de los polos magnéticos. En la ionosfera rebotan las ondas de radio y televisión usadas en las telecomunicaciones. La temperatura de la termosfera va ascendiendo en altura al absorber las radiaciones de alta energía, pudiendo alcanzar más de 1000 ºC.

- Exosfera (500-10.000 km): Tiene una bajísima densidad de gases hasta llegar a ser similar a la del espacio exterior (casi vacío) con lo que el cielo se oscurece (no hay prácticamente materia que absorba o disperse la luz).

La atmósfera sirve de protección a los seres vivos, tanto a los que se encuentran en la superficie terrestre, como los que se encuentran en zonas próximas a esta y en los océanos. Esta protección consiste en la absorción por parte de la atmósfera de gran parte de la radiación más energética, y por tanto la más dañina para los seres vivos (rayos X, rayos gamma y radiación UV), que se dirige a la Tierra. Estos procesos de absorción se producen, como ya se ha mencionado, principalmente en la termosfera, aunque gran parte de la radiación ultravioleta es absorbida por la capa de ozono ubicada en la estratosfera.

Además, la atmósfera realiza una función reguladora sobre el clima y la temperatura de la Tierra a través de varios procesos. Durante el día la atmósfera absorbe y dispersa de vuelta al espacio una parte de la radiación solar incidente, evitando que alcance la superficie terrestre e incremente notablemente la temperatura del planeta. Además la atmósfera también absorbe, tanto de día como de noche, parte de la radiación térmica infrarroja (IR) emitida por la superficie terrestre; una fracción de esta radiación absorbida es re-emitida de vuelta a la superficie (efecto invernadero), evitando el descenso brusco de la temperatura en la superficie. Al proceso descrito anteriormente se le une la circulación atmosférica, que tiende a compensar las diferencias de temperatura en diferentes partes del planeta originadas por la variación geográfica de la insolación.

2.2. Propiedades físicas de la atmósfera

Algunas propiedades físicas de interés de la atmósfera son las siguientes:

- Presión atmosférica[6]: Se define como la fuerza por unidad de superficie que ejerce el peso de la columna de aire que se encuentra sobre un punto específico. La presión atmosférica disminuye rápidamente con la altura. Experimentalmente, la presión se mide habitualmente con un barómetro. En los mapas meteorológicos, la presión atmosférica se suele representar mediante isobaras, que son líneas que unen puntos de igual presión. Las diferencias de presión en la atmósfera es la responsable de la aparición de vientos. La presión atmosférica en función de la altura viene dada por la siguiente ecuación:

$$P_h(h) = P_0 e^{-\frac{mgh}{KT}} \tag{2.1}$$

siendo P_0 la presión a nivel del suelo, g la gravedad, k la constante de Boltzman, T la temperatura y m la masa del aire.

- Temperatura: Constituye el elemento meteorológico más importante en la determinación del tipo de clima. A continuación explicaremos la relación existente entre esta magnitud física y los gases situados en la atmósfera. Los gases atmosféricos interaccionan con la radiación que incide sobre ellos y que proviene principalmente del Sol. Interaccionan principalmente mediante dos mecanismos: absorción y dispersión Rayleigh [7]. Según el tipo de gas, se absorverán unas bandas espectrales u otras. Los gases emiten radiación debido a esta absorción y a la temperatura a la que se encuentran. Aquellos gases que absorben y remiten hacia la superficie radiación térmica infrarroja, son los conocidos como gases de efecto invernadero, y su efecto neto es el de elevar la temperatura de la superficie y la atmósfera a un valor mayor del que habría sin estos gases. Entre los gases de efecto invernadero se encuentran el vapor de agua, el dióxido de carbono y el metano. Un aumento en las concentraciones de este tipo de gases, como el reportado en el dióxido de carbono conduce al fenómeno conocido como calentamiento global, de gran importancia en el cambio climático y que contribuye a un aumento de la temperatura del planeta.[8]

- Humedad atmosférica: Se define como la cantidad de vapor de agua que se encuentra en la atmósfera. La humedad atmosférica se puede expresar de diferentes formas. La humedad absoluta se define como la masa de vapor de agua por masa de aire seco. La humedad relativa normalmente se expresa como un porcentaje y representa la humedad absoluta dividido la máxima humedad a una temperatura dada. LaTeXhumedad específica es la relación entre la masa de vapor de agua y la masa total de una parcela de aire húmedo [9].

- Transmitancia atmosférica: La transmitancia atmosférica es una magnitud que se cuantifica como el porcentaje de intensidad lumínica que es capaz de atravesar la atmósfera a una longitud de onda dada. Su expresión matemática es:

$$T(\,\%) = 100\frac{I}{I_o} \tag{2.2}$$

  La figura 2.1 muestra la transmitancia atmosférica para un cielo despejado. También se representa que moléculas producen esa absorción de la radiación

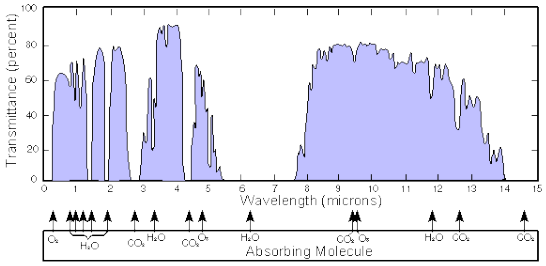


Figura 2.1: Transmitancia de la atmósfera
[10]

La atmósfera está compuesta por gases, pero en ella también encontramos otros componentes como son los aerosoles y las nubes. En las siguientes secciones se desarrollará estos conceptos:

2.3. Gases

Hasta los primeros 80-100 Km la composición del aire es homogénea, manteniéndose las mismas proporciones en los gases (aunque la concentración de gases decrece). A esta capa se

la conoce como homosfera. A partir de esta altura la composición varía habiendo gases que predominan según una altura determinada, llamándose a esta capa heterosfera. La composición media del aire seco en la homosfera es la siguiente [11]:

- Nitrógeno (N_2): 78,084 %.
- Oxígeno (O_2): 20,947 %.
- Argón (Ar): 0,934 %.
- Dióxido de Carbono (CO_2): 0,035 %.
- Otros: Neón (Ne), Helio (He), Criptón (Kr), Hidrógeno (H_2), Xenón (Xe), Metano (CH_4), Ozono (O_3), Óxidos de Nitrógeno(NOx), etc : 0,03 %.

Tendríamos que considerar también el vapor de agua, que varía bastante de unas zonas a otras, entre el 1 % y el 4 %.

2.4. Aerosoles atmosféricos

Los aerosoles atmosféricos son las partículas, sólidas o líquidas, en suspensión en la atmósfera. A diferencia de los gases, cuya composición es estable y bien conocida en las distintas zonas del planeta, los aerosoles presentan una gran variabilidad temporal y geográfica en sus propiedades químicas, físicas y ópticas. Esto se debe, principalmente, a la amplia variedad de tipos de aerosol, causada por las numerosas fuentes de origen de los mismos alrededor del mundo. El tamaño de los aerosoles puede variar desde las pocas centenas de nanómetros, hasta las decenas de micras. Así, según el tamaño de los aerosoles, se pueden clasificar en finos, si su radio es menor de $0,5\mu m$ y gruesos, si su radio supera esta longitud. Incluso cuando se trata de un único tipo de aerosol, no todos tienen el mismo tamaño, sino que existe una distribución de tamaños. La distribución de tamaños de los aerosoles se suele asemejar a una distribución bimodal, un modo asociado al fino y otro al grueso, cada una de ellas siguiendo una distribución normal respecto al logaritmo del radio. La ecuación 2.3 describe esta distribución.

$$\frac{dV}{d(lnr)} = \frac{VC_F}{\sqrt{2\pi}\sigma_F} exp\left(\frac{(lnr - lnR_F)^2}{(2\sigma_F)^2}\right) + \frac{VC_F}{\sqrt{2\pi}\sigma_F} exp\left(\frac{(lnr - lnR_C)^2}{(2\sigma_C)^2}\right) \tag{2.3}$$

donde VC es la concentración en volumen total, R el radio modal de la distribución σ la desviación estándar y C y F hacen referencia al modo grueso y fino [12].

Debido al origen de los aerosoles, estos se suelen clasificar en dos tipos:

- Antropogénicos: Aerosoles producidos por la actividad humana. Son principalmente de tipo fino, por lo que permanecen más tiempo en la atmósfera, pudiendo recorrer grandes distancias.
- De origen natural: Algunos ejemplos son la sal marina, el polvo mineral proveniente de zonas desérticas, cenizas volcánicas... Suelen ser más gruesas, por lo que se depositan más rapidamente.

Aunque los aerosoles se pueden encontrar distribuidos verticalmente por toda la columna atmosférica (como puede ser el caso de los aerosoles expulsados por erupciones volcánicas), los que se encuentran a nivel de la superficie juegan un papel muy importante, ya que afectan a la calidad del aire. Además, son de importancia en lo que respecta al cambio climático. Los aerosoles, al igual que las partículas de aire, absorben y dispersan radiación. Sin embargo, en

vez de producirse scattering de Rayleigh, se produce scatering de Mie. Este favorece más en el sentido de incidencia de la radiación. Si los aerosoles son muy absorbentes, éstos pueden calentar las masas de aire en las que se encuentran, contribuyendo a un mayor calentamiento de la atmósfera y, por tanto, al calentamiento global. Por otro lado, si son muy poco absorbentes, lograrán que una mayor parte de la radiación que llega a la Tierra sea dispersada de vuelta al espacio, consiguiendo un efecto neto de enfriamiento, contrarrestando parcialmente el calentamiento global causado por los gases de efecto invernadero [13]. Es decir, mientras que unos aerosoles contribuyen al calentamiento global, otros lo contrarestan.

Los aerosoles también actúan como núcleos de condensación de las nubes. Esto implica que un cambio en las propiedades de los aerosoles, o en la concentración de éstos, puede modificar propiedades de las nubes tales como su tiempo de vida o su albedo, afectando al balance radiativo de la Tierra pero también al ciclo hidrológico y a la precipitación.[26]

2.5. Nubes

Como hemos mencionado, las nubes juegan un papel de gran importancia tanto en meteorología como en climatología. Es por ello que en esta sección se definirán las nubes, se explicará por qué son de tanta importancia meteorológica, se explicará el ciclo hidrológico y se dará una clasificación de las mismas.

Según la Organización Meteorológica Mundial, una nube es un hidrometeoro (meteoro constituido por partículas de agua: líquidas, sólidas o de ambas) suspendidas en la atmósfera y que, por lo general, no tocan el suelo. Las nubes suelen ser el único componente más facilmente apreciable de la atmósfera además de ser el indicador más evidente del estado del tiempo. [14]

Las nubes juegan un papel importante en varios aspectos de la atmósfera y la tierra, causando influencia por ejemplo en:

- Impacto sobre la radiación solar: las nubes influyen en la cantidad de radiación solar que llega a la superficie terrestre. Dependiendo de su tipo y densidad, las nubes pueden reflejar mayor o menor cantidad de radiación solar de vuelta al espacio
- Modulación de la temperatura: Durante el día, las nubes pueden bloquear la radiación solar directa, lo que puede mantener las temperaturas más frescas. Por la noche, absorben parte de la radiación de onda larga emitida por la superficie terrestre, reemitiendo una parte de vuelta, lo que puede evitar que las temperaturas desciendan demasiado.
- Influencia en la circulación atmosférica: Las nubes afectan la circulación atmosférica al alterar la distribución del calor y la humedad en la atmósfera. Esto puede tener efectos significativos en los patrones de viento y la distribución de las precipitaciones en diferentes regiones.

El ciclo hidrológico describe la circulación y cambios de estado del agua a través de los océanos, la atmósfera y la tierra. El ciclo hidrológico comienza con la evaporación del agua de los mares, ríos y lagos, cambiando de estado líquido a gaseoso. Al elevarse, el vapor de agua se enfría y se condensa, formando así las nubes. Cuando las nubes se saturan (o bien de gotas de agua o de cristales de hielo) tiene lugar la precipitación (en forma de lluvia, granizo, nieve o aguanieve). Una vez que se ha producido la precipitación, el agua fluye por la superficie de la tierra según la fuerza de la gravedad. Así el agua se acumula en arrollos, ríos, lagos y finalmente acaba en los mares y océanos. Parte del agua procedente de las precipitaciones, se filtra apareciendo corrientes subterráneas.

Entender el ciclo del agua es crucial para poder gestionar las reservas de agua, pronosticar patrones climáticos así como mitigar los efectos del cambio climático.

Se puede clasificar las nubes en tres grupos (ver figura 2.2), según la altura a la que se encuentran:

- Nubes bajas: En este grupo encontramos los *stratocumulos*, los *stratus*.
- Nubes medias: En este grupo encontramos los *altocumulus* y los *altostratus*.
- Nubes altas: *Cirrus*, *cirrucumulus* y *cirrostratos*.
- Los *nimbostratus* y los *cumulonimbus* son nubes que se enuentran a la vez en las tres alturas. Las nubes *cumulus* ocupan las alturas de las nubes bajas y de las medias.

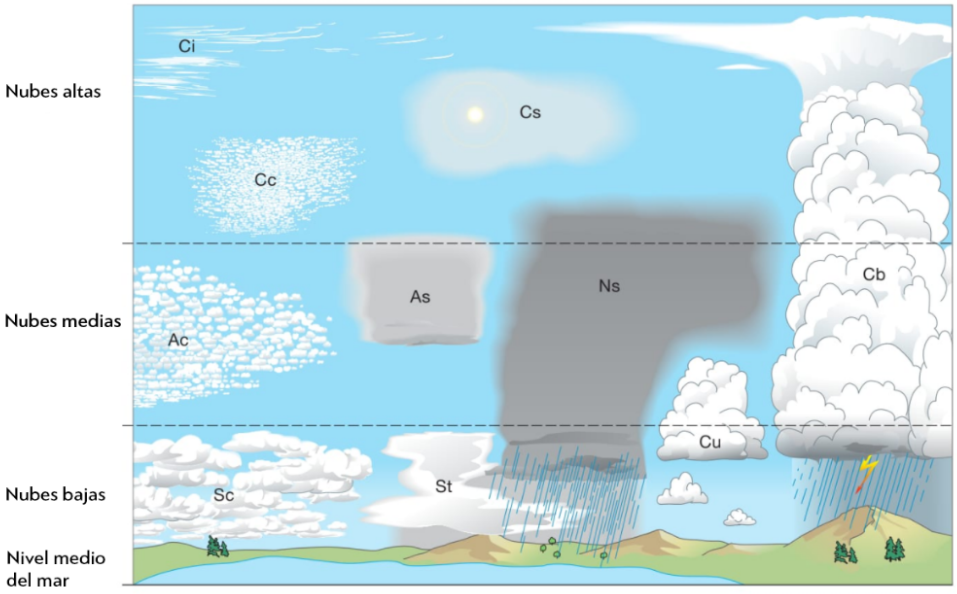


Figura 2.2: Tipos de nubes.
[15]

Capítulo 3

Inteligencia Artificial

La inteligencia artificial se define como el conjunto de tecnologías que permiten a las máquinas realizar tareas que se asocian a las humanas. Aunque generalmente el término Inteligencia Artificial se asocia a los conceptos de Aprendizaje Automático ("Machine Learning") o de Aprendizaje Profundo ("Deep Learning") la IA es un término más general que engloba otras áreas tecnológicas como son la visión por computadora ("Computer Vision"), la robótica o el procesamiento del lenguaje natural [27].

3.1. Machine Learning y Deep Learning

El Aprendizaje Automático, "Machine Learning", o ML, se define como una subrama de la IA en la que, mediante la utilización de algoritmos, las máquinas son capaces de aprender a partir de datos. Mediante la utilización del Aprendizaje Automático, se da a las máquinas capacidad para aprender sin haberse explícitamente programado para ello. Así, las máquinas, a partir de los datos, son capaces de extraer patrones y relaciones que luego pueden utilizar para realizar tareas específicas. Existen multitud de técnicas y modelos que permiten a las máquinas aprender, entre ellos los modelos de regresión, árboles de decisión ("Decision Trees") modelos probabilísticos, las máquinas de vectores soporte (SVP, "support vector machine") o las redes neuronales. Los algoritmos tradicionales de ML suelen tener una estructura más sencilla que los de Deep Learning[28].

El Aprendizaje Profundo, "Deep Leaning", o DL, se define como una subrama del Aprendizaje Automático en la que se persigue crear modelos que sean capaces de comprender conceptos abstractos y/o complejos a partir de otros más sencillos. Los algoritmos de DL se basan en redes neuronales artificiales profundas ("Deep Neural Networks") o DNN. Estas redes profundas se diferencian de las redes neuronales en que presentan una mayor complejidad, ya que poseen más de tres capas (el concepto de capas se desarrollará en el siguiente aparatado)[16].

Además, es necesario un conjunto de datos bastante mayor para entrenarlas. Por ejemplo, el desarrollo de automóviles sin conductor requiere de millones de imágenes. También es necesario un mayor tiempo de entrenamiento y una gran potencia de cálculo. En ocasiones se utiliza el Cloud Computing ("Computación en la nube") para reducir el tiempo de entrenamiento de una red neuronal, reduciéndose este tiempo de unas semanas a apenas unas horas. Los modelos de DL requieren de una gran cantidad de datos para ser entrenadas [17].

El aprendizaje profundo está siendo cada vez más utilizado, ya que está logrando resultados que antes eran imposibles. Algunos ejemplos de utilización actuales de este tipo de redes neuro-

nales los podemos encontrar en sistemas de conducción autónoma, investigación médica como el análisis de imágenes médicas, reconocimiento de voz, clasificación de vídeos, reconocimientos faciales, en predicción de las preferencias de un cliente... En investigación en física, tanto el ML como el DL actualmente también están siendo cada vez más utilizados [18].

3.2. Redes neuronales

En esta sección se estudiará las redes neuronales artificiales (ANN), también conocidas como redes neuronales simuladas (SNN) o, simplemente, Redes Neuronales.

Una red neuronal artificial consiste en la interconexión de un conjunto de unidades elementales llamadas neuronas. En la siguiente sección se estudiarán estas, así como las funciones de activación.

3.2.1. Las neuronas

Las neuronas artificiales son unidades de cálculo que pretenden imitar el funcionamiento de las neuronas biológicas. Constituyen las unidades fundamentales con la cual se construye las redes neuronales artificiales.

Una neurona artificial opera de la siguiente manera: las neuronas reciben múltiples datos (entradas) y cada una tiene asociado un peso, de forma que la neurona multiplica cada dato de entrada por un peso y realiza la suma de todos esos resultados. Los pesos muestran la importancia de cada entrada en la neurona. Además, la neurona tiene asociada un valor conocido como sesgo (bias) que suma al resultado anterior. Al resultado obtenido tras esta suma ponderada se le aplica una función llamada función de activación. Es decir, la neurona realiza las siguientes operaciones:

$$z(x) = \sum_{j=1}^{n} x_j w_j^i + b \tag{3.1}$$

donde x_j representa el valor de entrada j y w_j^i representa el peso asociado al dato de entrada j y situado en la capa i dentro de una red neuronal. b simboliza el sesgo. Después, al valor obtenido se le aplica una función de activación:

$$y = y(z(x)) \tag{3.2}$$

Existen varios tipos de funciones de activación, siendo las más utilizadas la función sigmoide, la función tangente hiperbólica y la función rectificadora. Para la red neuronal utilizada en este TFG se hizo uso de la función rectificadora (ReLU), cuya expresión matemática es:

$$y(z) = max(0, z) \tag{3.3}$$

y de la función Softmax, la cual es una generalización de la función logísitica:

$$\begin{gathered} \sigma(z) : \mathbb{R}^{\mathbb{K}} \to [0,1]^K, K \in \mathbb{N} \\ \sigma(z)_j = \frac{e_j^z}{\sum_{k=1}^{K} e_k^z} \quad para \quad j = 1, 2, ..., K \end{gathered} \tag{3.4}$$

Durante el entrenamiento de una red neuronal, se tiene como objetivo ajustar el valor de los pesos de las diferentes neuronas para que realice una cierta tarea de la forma más optima posible. Sin entrar mucho en detalle, en el entrenamiento de una red neuronal se calcula el error entre las predicciones realizadas por ella y el valor real y se trata de minimizar ese error reajustando los valores de los pesos de las neuronas utilizando algoritmos de optimización como podría ser el descenso del gradiente.[19]

3.2.2. Arquitectura de una red neuronal

Las redes neuronales están formadas por capas, que son grupos de neuronas. El valor de entrada de las neuronas de una capa determinada son los valores de salida del conjunto de neuronas de la capa anterior, de forma que las neuronas situadas en la capa i calculan la suma ponderada a partir de los valores de salida de las neuronas situadas en la capa $i-1$.

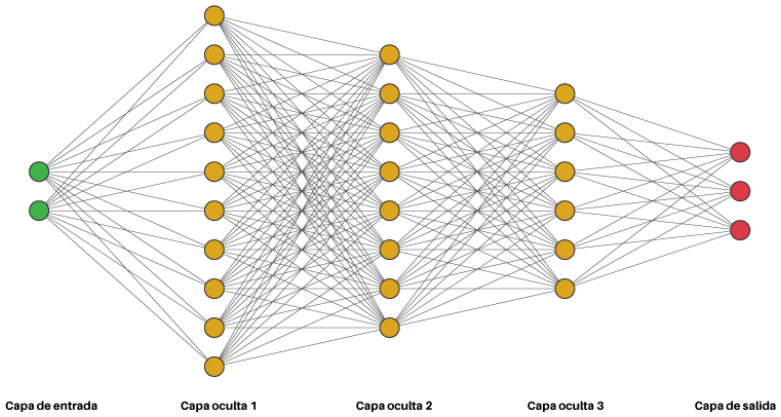


Figura 3.1: Estructura de una red neuronal.
[20]

Existen tres tipos diferentes de capas: la capa de entrada, formada por las neuronas a las que proporcionamos los datos de entrada, una o varias capas ocultas, y la capa de salida, que produce los valores finales de la red.

La figura 3.1 muestra la estructura de una red neuronal multicapa. Pero existen otros tipos de redes neuronales que, aunque tienen una estructura muy similar al de las redes multicapa, cada una de ellas tiene unas particularidades. Además, según la aplicación para la que se quieran utilizar, va a ser más útil utilizar redes neuronales de un tipo o de otro. Algunos tipos de redes son las Redes Neuronales Recurentes (RNN), las Redes Neuronales LSTM (Long Short-Term Memory), las Redes Generativas Adversariales (GAN) y las Redes Neuronales Convolucionales (CNN) [21].

3.2.3. Redes neuronales convolucionales

Las CNN han demostrado ser muy efectivas en tareas de visión por computadora, como reconocimiento de objetos, segmentación y clasificación de imágenes y muchas otras aplicaciones relacionadas con el procesamiento de imágenes[29]. Para este Trabajo Fin de Grado, utilizaremos este tipo de redes neuronales, ya que, nos enfrentamos a un problema de clasificación de imágenes. Para entender el funcionamiento de las capas de convolución, previamente tenemos que entender qué son imágenes digitales y los filtros o kernels.

Imágenes digitales: Las imágenes digitales en escala de grises las podemos entender como una función f definida:

$$f : [a, b] \times [c, d] \rightarrow [0, 255] \tag{3.5}$$

$[a, b]$ y $[c, d]$ son dos conjuntos discretos.

Es decir, se trata de una matriz, donde cada número representa un píxel y puede tomar un valor entre 0 (píxel completamente negro) y 255 (píxel completamente blanco) en el caso

de imágenes de 8 bits ($2^8 = 256$). Las imágenes a color no son más que una extensión de las imágenes a blanco y negro:

$$f : [a, b] \times [c, d] \rightarrow [0, 255] \times [0, 255] \times [0, 255] \tag{3.6}$$

Pero en lugar de utilizar un único canal, hacen uso de tres, cada uno de un color: rojo (R), verde (G) y azul (B). Como cada uno de ellos puede tomar un valor entre 0 y 255, podemos crear un total de $256^3 = 16{,}777{,}216$ colores.

Veamos un ejemplo a partir de la siguiente imagen. El código se realizó en Python y se mostrará en el **Anexo I**. En la figura 3.2a, se observan los tres canales (RGB) de una misma imagen. Cuanto más blanco sea un píxel de una de las imágenes, significa que mayor intensidad de ese canal tiene. Por ejemplo, en el cielo, de color azul, se observa que en el canal B esos píxeles tienen un gris más claro que en los otros dos canales.

(a)

(b)

Figura 3.2: Imagen tomada por el autor del TFG. (a): Tres canales R, G y B, y (b): Imagen resultante de la combinación de los tres canales.

Los filtros son matrices bidimensionales o tridimensionales que se utilizan en operaciones de convolución aplicadas a imágenes. Estas operaciones de convolución implican deslizar el filtro sobre la imagen y operar en cada posición para producir una nueva imagen filtrada.

La expresión matemática de un filtro de convolución es:

$$f(x, y) = (g * h)(x, y) = \sum_m \sum_n = g(x - m, y - n)h(m, n) \tag{3.7}$$

En la figura 3.3 se puede observar el efecto de la operación de convolución aplicada sobre una imágen.

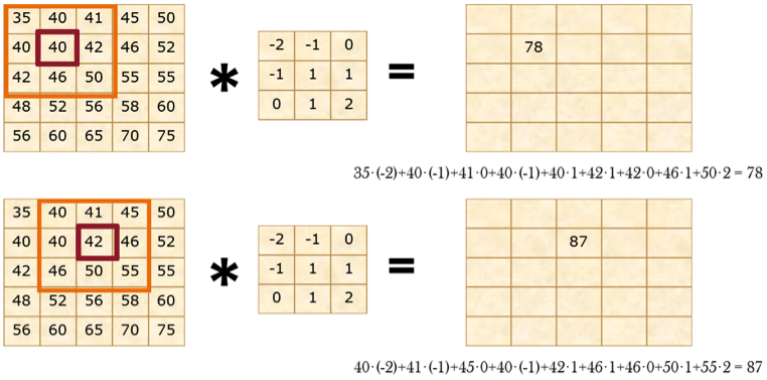


Figura 3.3: Filtros
[22]

Veamos en la figura 3.4 un ejemplo de aplicación de filtros en una imágen en la que, al aplicar un filtro sobre la imagen original, se observa otra imagen en la que quedan delimitados los bordes de los objetos. El código se muestra en el Anexo I.

(a) Imagen original (b) Filtro Canny edge

Figura 3.4: Ejemplo de aplicación de un filtro que detecta los bordes de objetos de una imagen. Imagen tomada por el autor del TFG.

Tipos de capas utilizadas en las Redes Neuronales Convolucionales:

- Capas de convolución: La característica principal de las Redes Neuronales Convolucionales es que estas presentan también capas de convolución. Estas capas aplican filtros a las imágenes de entrada para extraer características relevantes. De forma intuitiva, en estas capas, la red neuronal tendrá como objetivo el aprendizaje de los valores de diferentes filtros. Una red neuronal normal tiene como objetivo ajustar los pesos de las neuronas, mientras que en una capa de convolución, las redes neuronales tienen como objetivo ajustar los diferentes valores numéricos que componen estos filtros.
- Capas de agrupación: situadas entre las sucesivas capas convolucionales. Estas permiten reducir la dimensión de las imágenes mediante el uso de filtros. Existen de dos tipos: agrupaciones máximas o agrupaciones medias. Las agrupaciones máximas, más conocidas como maxpooling, sustituyen varios píxeles por el máximo de ellos, mientras que las agrupaciones medias sustituyen varios píxeles por el valor medio de todos ellos.
- Capas de activación: Aplican una función de activación a los valores de la capa anterior. Esto aporta no linealidad a la red.

- Capas de dropout: En estas capas se desactivan aleatoriamente ciertas entradas mientras el modelo se está entrenando. Tienen como objetivo evitar el sobreajuste (overfitting).
- Capa de salida: Proporciona la salida de la red, en nuestro caso será la cubierta de nubes, en octas, asociada a la imagen del cielo.

Capítulo 4

Instrumentación y metodología

4.1. Cámara de cielo

Como se ha explicado en la sección 3, para entrenar un modelo de aprendizaje automático es necesario un conjunto de datos. En este caso, se tratará de un conjunto de imágenes del cielo. Estas imágenes se han tomado con tres cámaras situadas en tres puntos diferentes de la ciudad de Valladolid. Una de ellas, denotada con el identificador C013 se encuentra en la Facultad de Ciencias de la Universiad de Valladolid (41°39'48"N 4°42'20"W). La cámara C014 se encuentra en el CEIP el Peral (41°36'56"N 4°45'18"W). Por último, la cámara C017 se encuentra en la Escuela Técnica Superior de Arquitectura de dicha universidad (41°38'58"N 4°44'27"W). Las tres son el mismo modelo de cámara: se trata del modelo OMEA-3C del fabricante *Alcor System*, con una modificación en la que se le incluyó un filtro tribanda adicional para reducir el ancho espectral de los canales. Esta cámara cuenta con un sensor IMX178 de SONY, con un tamaño de imagen de 3096x2080 píxeles. Además del filtro tribanda, tiene un filtro que bloquea la radiación infrarroja. Este modelo cuenta con una lente de ojo de pez y un sensor externo de temperatura y humedad. Contiene un sistema de calefacción interno para evitar la condensación interior y eliminar rápidamente las gotas de agua en caso de lluvia o rocío. Estas cámaras van conectadas a un ordenador en el que se ha desarrollado un aplicación de captura de imágenes. Están configuradas para que tomen una imágen cada 5 minutos durante el día y cada 2 minutos durante la noche [23].

En las imágenes de la figura 4.1 se muestra la cámara C013 situada en el tejado de la Facultad de Ciencias.

Figura 4.1: Imágenes de la cámara C013.

De la cámara C013 se dispuso de las imágenes tomadas los días 4 y 5 del mes de julio de 2023. De la cámara C014, las tomadas los días 1, 2 y 10 del mes de febrero de 2023. Por último, de la cámara C017, se dispuso de las imágenes tomadas los días 8, 9, 10 y 11 del mes de septiembre de 2023.

4.2. Preparación del data set

4.2.1. Etiquetado de las imágenes

Para poder realizar el entrenamiento de la red neuronal profunda, se necesita de un conjunto de imágenes considerablemente grande. Además, es necesario que estas imágenes sean variadas, es decir, hay que disponer de una cantidad suficiente de imágenes para cada octa. Sino, el entrenamiento no se realizará correctamente. Para que el modelo pueda aprender, necesitamos previamente clasificar nosotros manualmente nuestro conjunto de imágenes. Así, la primera tarea a realizar es el de asignar manualmente a cada imagen un número del 0 al 8 (según el número de octas que contenga). En las imágenes de la figura 4.2 se puede ver un ejemplo de imágenes para cada tipo de octa.

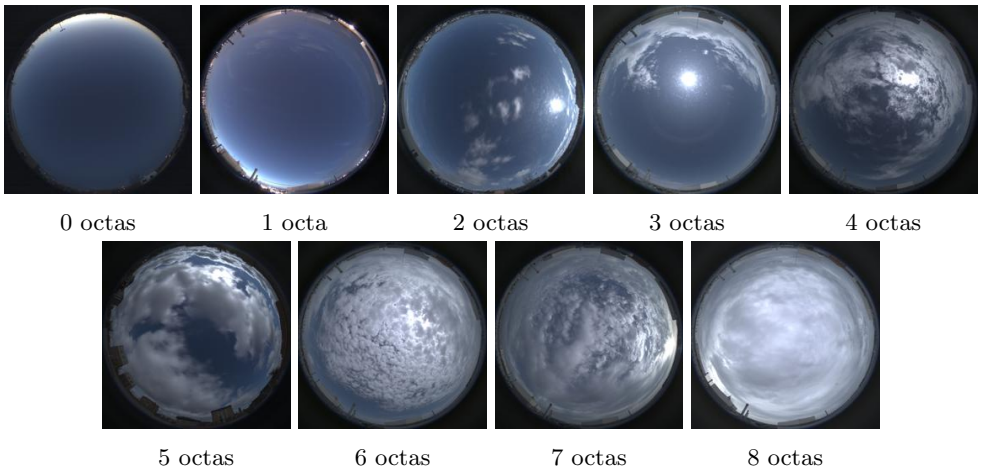


Figura 4.2: Conjunto de imágenes con diferentes cantidades de octas.

De todas las imágenes disponibles, se han clasificado manualmente, y por lo tanto, se han empleado para la realización del TFG, el siguiente número de imágenes para cada tipo de octa:

Número de Octas	**Cantidad de Imágenes**
0	800
1	331
2	152
3	124
4	129
5	88
6	101
7	314
8	1156
Total	2565

Cuadro 4.1: Distribución de imágenes según el número de octas.

Se aprecia que hay una gran variación del número de imágenes según el número de octas. Estas imágenes están agrupadas en carpetas según el número de octas.

4.2.2. Dividisión de los datos en conjuntos de entrenamiento, validación y prueba

En los algoritmos de prendizaje automático y de aprendizaje profundo, se utiliza un porcentaje de los datos de los que se dispone para entrenar y validar el modelo (normalmente el 80 % de todos los datos) y el porcentaje restante (normalmente el 20 %) para testear como de eficaz es el modelo una vez ya se ha entrenado. Así, se dividió nuestro conjunto de imágenes en dos data sets:

- Conjunto de imágenes de entrenamiento y validación: En este data set se encuetran las imágenes que se utlizarán para entrenar el modelo y las imágenes utilizadas en la validación del mismo durante el entrenamiento. Estas últimas tienen como objetivo de evitar el sobreajuste u overfitting del modelo. Decimos que un modelo está sobreajustado cuando es capaz de operar con mucha precisión con el conjunto de datos de entrenamiento, pero no realiza predicciones correctas con otros datos que no se hayan utilizado para entrenarlo (datos test). Así, el objetivo de los datos de validación es evitar que nuestro modelo esté sobreajustado. Con el objetivo de poder hacer una buena división entre los datos de entrenamiento y validación y los de prueba, se situó, de forma aleatoria, en este data set 70 imágenes de cada tipo de octa. Se escogió 70 porque corresponde al 80 % de las imágenes del número de octas de las que menos imágenes se dispone: de 5 octas. De estas imágenes se utilizará el 90 % para entrenamiento del modelo y el 10 % para validación.

- Conjunto de imágenes test. En este data set se encuentran las imágenes que se emplearán para estudiar cómo de eficaz es el modelo. Es decir, con los resultados dados por el modelo al aplicarle estas imágenes realizaremos un estudio estadístico. En él se dispone del resto de las imágenes que no se han utilizado para entrenar ni validar el modelo durante el entrenamiento. El número de imágenes test para cada número de octas viene dada por la tabla 4.2.

Número de Octas	Cantidad de Imágenes test
0	730
1	261
2	82
3	54
4	59
5	18
6	31
7	244
8	1086
Total	3195

Cuadro 4.2: Distribución de imágenes test según el número de octas.

Para realizar esta división de las imágenes en las carpetas de entrenamiento y validación y en la de test se realizó un código en Python, puesto que así podemos lograr una división aleatoria de las imágenes. El código se muestra en el **Anexo II**.

4.3. Arquitectura y entrenamiento de la red neuronal

En esta sección explicaremos el proceso seguido para programar y entrenar la red neuronal. El código fue realizado en Python, en la interfaz de usuario web JupyterLab. La librería de aprendizaje automático utilizada fue TensorFlow. El código está mostrado en el **Anexo III**.

Una vez realizada la división entre los datos de entrenamiento y validación y los datos test, se procedió a programar y entrenar la CNN. Para ello, el primer paso fue cargar todas las imágenes del conjunto de entrenamiento y validación en nuestro entorno de trabajo y se realizó finalmente la separación entre las imágenes de entrenamiento y las de validación. También se definió la función de coste o pérdida: se utilizó la función de entropía cruzada (crossentropy), esta es una medida de la diferencia entre dos distribuciones de probabilidad: la distribución verdadera y la distribución predicha. Para clasificación múltiple, la entropía cruzada viene dada por la expresión matemática siguiente:

$$H(p,q) = -\frac{1}{N}\sum_{j=1}^{N}\sum_{i=1}^{C} p_{ij}\log(q_{ij}) \tag{4.1}$$

donde N es el número de muestras, C el número de clases, p_{ij} es la probabilidad verdadera de la clase i para la muestra j, es decir, puede tomar el valor 1 o 0, y q_{ij} es la probabilidad predicha por el modelo para la clase i para la muestra j.

Estructura del modelo: El paso siguiente fue establecer la estructura del modelo. Se provó con varias estructuras hasta que se obtuvo resultados suficientemente buenos con una de ellas. En la figura 4.3 se muestra la arquitectura del modelo que finalmente fue considerado.

```python
# Definición del modelo
model = tf.keras.Sequential([
    # Capa convolucional 1
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),
    # Capa convolucional 2
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    # Capa convolucional 3
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    # Capa convolucional 4
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    # Capa de aplanamiento para conectar con las capas densas
    tf.keras.layers.Flatten(),
    # Primera capa densa
    tf.keras.layers.Dense(512, activation='relu'),
    # Capa de salida
    tf.keras.layers.Dense(num_classes, activation='softmax')  # Num_classes es el número de clases de salida
])
```

Figura 4.3: Modelo. Codigo basado en la librería TensorFlow (tf) en el que se define la arquitectura del modelo de CNN propuesta para el trabajo.

Es decir, el modelo está compuesto por las siguientes capas:

- Capa de entrada: Se trata de una capa de convolución que consta de 32 filtros 3×3 con función de activación ReLU. Además, en esta capa se redimensionó el tamaño de nuestras imágenes a imágenes de 224×224 píxeles y se mantuvieron los 3 canales (RGB).
- Primera capa oculta: Se trata de una capa de agrupación (máxima). Utiliza un filtro $2x2$. Es decir, este filtro sustituye agrupaciones de 4 píxeles de una foto por el valor máximo de ellos.
- Segunda capa oculta. Capa de convolución. Consta de 64 filtros 3×3 y la función de activación utilizada es la ReLU.
- Tercera capa oculta. Se trata de una capa de agrupación máxima igual a la anterior.
- Cuarta capa oculta. Capa de convolución de 128 filtros 3×3.
- Quinta capa oculta. Misma capa de agrupación máxima.
- Sexta capa oculta. Capa de convolución de 128 filtros 3×3.
- Séptima capa oculta: Misma capa de agrupación máxima.
- Octava capa oculta. Capa Flatten: Convierte los valores bidimensionales de las imágenes en un vector unidimensional.
- Novena capa oculta. Capa densa: Constintuída por 512 neuronas tradicionales con función de activación ReLU.
- Capa de salida: Capa densa formada por 9 neuronas y función de activación softmax. Esta última capa contiene 9 neuronas porque cada una de ellas simboliza una de las categorías del problema, es decir, las octas (de 0 a 8). Así, para una imagen de entrada, supongamos de 4 octas por ejemplo, tras recorrer toda la red neuronal, esta última capa nos devuelve 9 valores numéricos comprendidos entre 0 y 1. Estos valores están normalizados, de tal manera que la suma de los 9 valores es igual a 1, y está asociado a la confianza con la que la red identifica esa imagen con esa categoría. Si el funcionamiento de esta red neuronal

fuera correcto nos devolvería valores cercanos a 0 para las neuronas que simbolizan 0, 1, 2, 3, 5, 6, 7 y 8 octas y un valor próximo a 1 de la neurona que simboliza a la de 4 octas. Así, tomando el valor superior de los 9 valores numéricos que nos devuelve la red sabremos la predicción que esta ha logrado sobre la imagen de entrada.

Después de definir la arquitectura del modelo, se indicó en nuestro script que se hiciera uso del optimizador "adam" para realizar el entrenamiento de la red [30]. El optimizador es la función que trata de reducir los errores que la red neuronal tiene al realizar predicciones. Después se compiló el modelo y se le indicó que tuviera en cuenta los datos de validación para evitar el sobreajuste (realizando checkpoints). Acto seguido se entrenó el modelo. El tiempo de entrenamiento tuvo una duración aproximada de 10 minutos. El modelo realizó 25 épocas para su entrenamiento. Además, cuenta con 29049437 parámetros totales, de los cuales, 9683145 son ajustables.

Tras realizar el entrenamiento de la CNN, se pasó a analizar los resultados que esta predice. Para ello se llevó a acabo un análisis estadístico de los mismos. Esto está mostrado en el Capítulo 5.

Capítulo 5

Análisis de los resultados

Tras haber entrenado la red neuronal, se llevó a cabo un estudio para comprobar cómo de efectiva es. Para ello, se utilizó el conjunto de imágenes test, puesto que estas imágenes no se han empleado en el entrenamiento de la misma. Si se utilizaran las imágenes de entrenamiento para determinar como de efectivo es nuestro modelo, obtendríamos resultados que no serían correctos, ya que una red neuronal siempre obtiene mejores predicciones con los datos que se han utilizado para entrenarla. Por lo tanto, el primer paso a realizar fue cargar el conjunto de imágenes test en el Jupyter Notebook. El código Python realizado para obtener este análisis está mostrado en el Anexo III.

5.1. Tasa de acierto de la red neuronal

Se define la tasa de acierto de un modelo de clasificación como la proporción de muestras clasificadas correctamente respecto al total de muestras clasificadas:

$$\text{Tasa de acierto} = \frac{\text{Número de imágenes clasificadas correctamente}}{\text{Número total de imágenes utilizadas para realizar predicciones}} \tag{5.1}$$

La red neuronal entrenada obtiene a partir del conjunto de imágenes test una precisión de:

$$\text{Tasa de acierto} = 0{,}8084 \tag{5.2}$$

Es decir, el modelo acierta en un 80,84 % de los casos. Se puede comprobar que el valor de la precisión es considerablemente elevado. A continuación se compara este resultado con el resultado obtenido en TFGs realizdos con anterioridad sobre esta misma temática. En el TFG de Sergio Alegre Fernández [24] la tasa de acierto obtenida en sus diferentes modelos fue para todos sus modelos menor al 76 %. En el TFG relizado por Carolina Calvo Herrero[25], la tasa de acierto en sus modelos fue siempre menor al 67 % en todos sus modelos para imágenes tomadas durante la noche. Se observa que el resultado obtenido en el presente TFG es mayor que en cualquiera de los modelos realizados por los dos antiguos alumnos.

A continuación se muestra la tasa de acierto del modelo, considerando como aciertos, también cuando el modelo devuelve una octa más o una menos de lo que realmente tiene la imagen.

$$\text{Tasa de acierto } (\pm 1) = 0{,}9528 \tag{5.3}$$

En el TFG de Sergio Alegre Fernández la tasa de acierto (± 1) obtenida en sus diferentes modelos fue entre 0,946 y 0,967. En el TFG de Carolina Calvo Herrero, la tasa de acierto (± 1) en

sus modelos fueron entre 0,87 y 0,93. En este caso observamos que el modelo realizado en este TFG obtiene una mejor tasa de acierto (±1) que la que se obtiene en los modelos de Carolina, y bastante similar a la que se obtiene en los modelos de Sergio.

Sin embargo, la tasa de acierto es un valor estadístico que no proporciona información sobre el grado de error en una clasificación incorrecta. Por ejemplo, si el modelo recibe una imagen de 3 octas y clasifica incorrectamente la imagen, no es lo mismo que el modelo prediga que es una imagen de 4 octas a que indique que es una imagen de 7 octas. Para evaluar esta diferencia en los errores de clasificación, se puede utilizar, entre otras herramientas, la matriz de confusión.

5.2. Matriz de confusión

Otra herramienta para determinar la bondad de un modelo de clasificación es la matriz de confusión. Muestra la relación entre las etiquetas reales de las imágenes y las etiquetas predichas por el modelo. Así, las filas representan las clases predichas por el modelo, mientras que las columnas indican las clases reales de las imágenes. Cada celda de la matriz contiene el recuento y/o la frecuencia de muestras que pertenecen a una clase específica. En ocasiones se usan colores para resaltar los diferentes valores de la matriz. Para la red neuronal entrenada se ha obtenido la matriz de confusión de la figura 5.1.

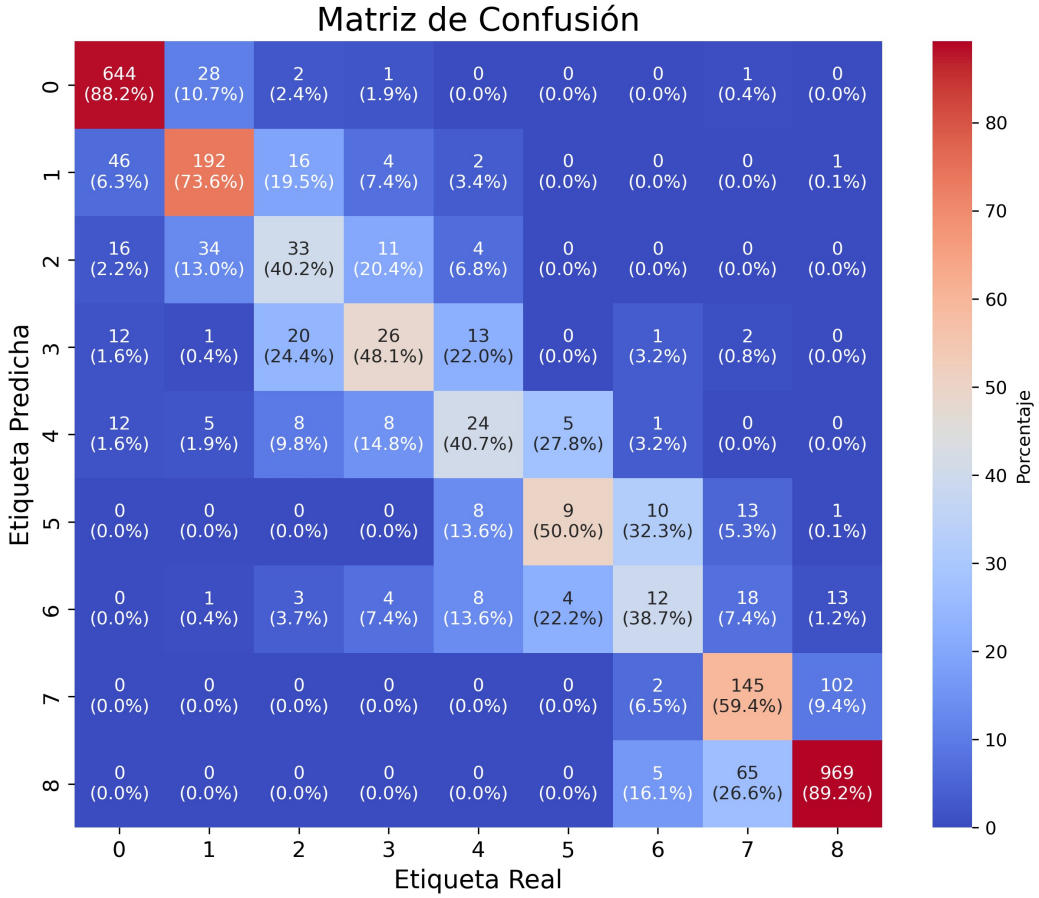


Figura 5.1: Matriz de confusión del modelo.

Se puede apreciar que las predicciones son considerablemente elavadas en los extremos. Para 0 y 8 octas se obtiene una precisión de prácticamente el 90 %. Sin embargo, para aquellas imágenes que tienen un número de octas central, el modelo realiza peores predicciones. Esto se debe a que, la distinción entre una imagen de, por ejemplo 4 octas y otra de 5 no es trivial. Se observa que para casillas alejadas de la diagonal principal, el porcentaje en prácticamente todas ellas es del 0 %. Así el modelo apenas realiza predicciones muy erráticas.

5.3. Distribución de diferencias

Se define un histograma como la representación gráfica que muestra la frecuencia con la que aparecen los valores en un conjunto de datos. A continuación, en la figura 5.2, se muestra el histograma obtenido a partir de la diferencia de los valores predichos por el modelo y los valores reales de cada una de las imágenes del conjunto de imágenes test.

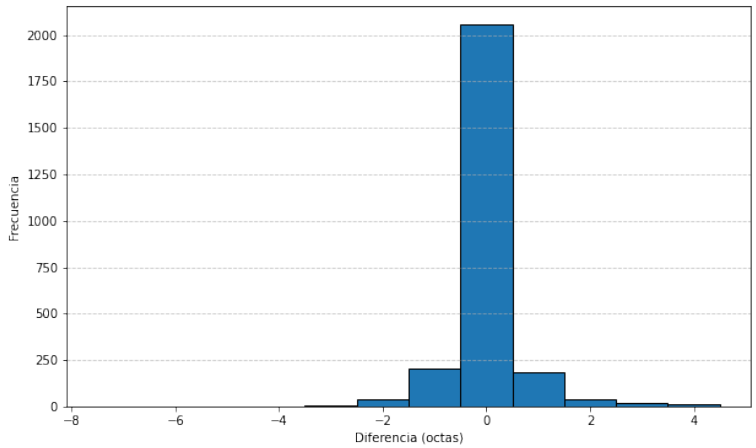


Figura 5.2: Histograma de las iferencias entre etiquetas predichas y reales

Se observa que se obtiene una frecuencia considerablemente grande para la diferencia 0. Para el resto de diferencias la frecuencia es bastante inferior y además disminuye progresivamente según se aleja de dicho valor 0. Se puede observar que la distribución es similar a una distribución normal. Veamos algunas variables estadísticas descriptivas que podamos obtener de este conjunto de datos formado por las diferencias de valores entre las etiquetas reales de las imágenes y las predichas.

La media:

$$\mu = 0,0300 \; octas \tag{5.4}$$

Al ser cercana a 0 el modelo tiene mucha exactitud, como también se ha comprobado en la tasa de acierto. Es decir, el modelo no sobrestima ni infraestima la cubierta nubosa.

La desviación estandar: Se trata de otro valor estadístico que mide la precisión, es decir, si hay mucha variabilidad en las predicciones.

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2} = 0,7165 \; octas \tag{5.5}$$

Al ser un valor menor a 1 se puede apreciar que el modelo es bastante preciso.

Con el objetivo de ver como se comporta el modelo para cada octa, se muestra en la figura 5.3 histogramas de manera similar a como hemos realizado la figura 5.2, pero para cada una de ellas.

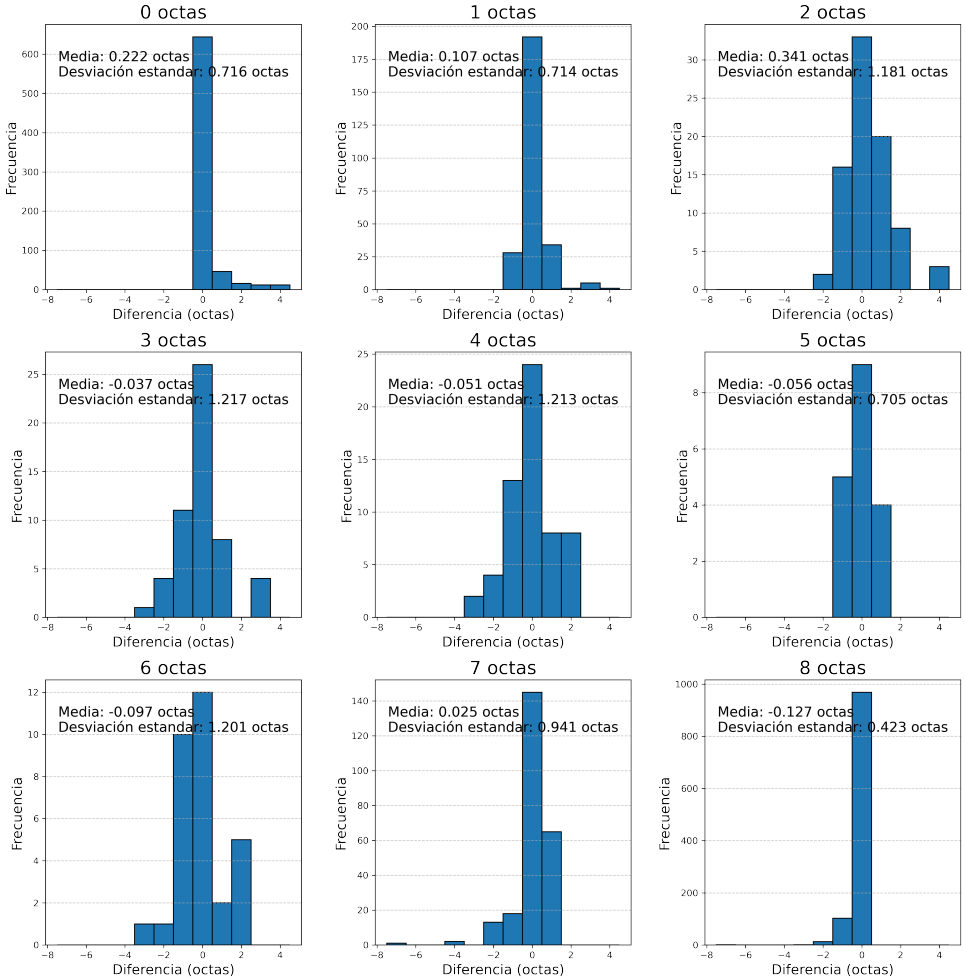


Figura 5.3: Histograma de las diferencias entre etiquetas predichas y reales para cada octa.

Se observa que para todas ellas, el valor 0 es el más repetido. Por consiguiente, como concluimos en la matriz de confusión, lo más probable para cada una de las octas es que realice una predicción correcta. Además, en prácticamente todas las octas, los siguientes valores más repetidos son el 1 o el -1, es decir, que si el modelo falla, lo más probable es que nos de una octa de más o una menos de lo que realmente tiene la imagen. La media es muy cercana a 0 en todas las octas, por lo tanto, podemos comprobar de nuevo que el modelo tiene una alta exactitud. En el caso de 2 octas, la media es de 0,341, lo que quiere decir que el modelo tiene una ligera sobreestimación para el caso de 2 octas. Para los casos de 3 y 4 octas se observa una mayor desviación estándar. Eso quiere decir que para imágenes de 3 y 4 octas será el caso en el que se puede cometer más errores a la hora de hacer predicciones.

Conclusiones

A lo largo de este Trabajo Fin de Grado se han desarrollado aspectos teóricos relacionados con la física de la atmósfera y con la Inteligencia Artificial y más concretamente con las Redes Neuronales Convolucionales. Posteriormente, se ha desarrollado un programa para automatizar el proceso de clasificación de imágenes del cielo según el número de octas. Para ello, se ha dividido el conjunto de imágenes disponibles en los conjuntos de entrenamiento, validación y test. Finalmente se ha entrenado la red neuronal.

Los principales resultados obtenidos mediante el uso de esta red neuronal han sido los mostrados a continuación. La tasa de acierto tiene un valor de 0.8084, la tasa de acierto (± 1) tiene un valor de 0.9528, la media de las diferencias entre los valores teóricos y los predichos tiene un valor de 0,030 octas y la desviación estándar tiene un valor de 0.7165 octas. Estos resultados nos permiten concluir que el modelo es muy exacto, y tiene una gran precisión. El modelo sobreestima el resultado para el caso de 2 octas en comparación con el resto de las octas.

Por último, como futuras líneas de trabajo se propone analizar este modelo de clasificación automática de imágenes del cielo mediante imágenes tomadas en otros lugares. Además, se propone también entrenar otra Red Neuronal para que nos indique si una imagen tiene gotas de agua. Esto podría resultar útil en días de lluvia, cuando hay alguna gota que permanece en la cúpula de la cámara.

Bibliografía

- [1]Loiacono M. "What Is... Earth's Atmosphere?"[Online] Disponible en: https://www.nasa.gov/general/what-is-earths-atmosphere/ Último acceso 28/06/2024.

- [2] "National Geographic - Atmosphere."[Online] Disponible en: https://education.nationalgeographic.org/resource/atmosphere/ Último acceso 28/06/2024.

- [3]"NIWA Taihoro Nukurangi"[Online] Disponible en: https://niwa.co.nz/atmosphere/layers-atmosphere Último acceso 30/06/2024.

- [4]"tutoroot Personalised Learning"[Online] https://www.tutoroot.com/blog/what-are-layers-of-atmosphere-structure-importance/ Último acceso 30/06/2024.

- [5]"UCAR Center for science education"[Online] Disponible en: https://scied.ucar.edu/learning-zone/atmosphere/layers-earths-atmosphere Último acceso 30/06/2024.

- [6]"MÉTÉO FRANCE"[Online] Disponible en: Link Último acceso 30/06/2024.

- [7],[12],[23] Antuña, J.C. (2021) Configuración y metodología para el uso de cámaras de todo cielo en la obtención de parámetros atmosféricos [Tesis de doctorado, Escuela de Doctorado de la Universidad de Valladolid].

- [8] [13] Intergovernmental Panel on Climate Change. (2021). Climate Change 2021. The Physical Science Basis. [Online] Disponible en: Link.

- [9]"Wikipedia Humidity"[Online] Disponible en: https://en.wikipedia.org/wiki/Humidity.

- [10]"Wikipedia Transmitancia"[Online] Disponible en: Link .

- [11] "National Oceanic and Atmospheric Administration."[Online] Disponible en: https://www.noaa.gov/jetstream/atmosphereÚltimo acceso: 30/06/2024.

- [14]"Organización Meteorológica Mundial" [Online] Disponible en: https://cloudatlas.wmo.int/es/clouds.html Último acceso: 30/06/2024.
- [15]"Organización Meteorológica Mundial"[Online] Disponible en: https://cloudatlas.wmo.int/es/useful-concepts.htmlÚltimo acceso: 30/06/2024.
- [16] Mancilla E.Ïnvgate"[Online] Disponible en: https://blog.invgate.com/es/ia-vs-machine-learning-vs-deep-learning-vs-redes-neuronalesÚltimo acceso: 1/07/2024.
- [17], [18], [19] Bosch, A., Casas-Roma, J., Lozano, T. (2019). "Deep learning: principios y fundamentos"
- [20]"Codificandobits"[Online] Disponible en: https://www.codificandobits.com/img/posts/2018-09-03/dnn.pngÚltimo acceso: 1/07/2024.
- [21]"KritiKat"[Online] Disponible en: https://kritikalsolutions.com/different-types-of-neural-networks-in-deep-learning/Último acceso: 1/07/2024.
- [22]Research Gate"[Online] Disponible en: Link Último acceeso 1/07/2024.
- [24] Alegre S. (2022). Trabajo Fin de Grado: "Clasificación automática de imágenes del cielo mediante inteligencia artificial".
- [25] Calvo C. (2023). Trabajo Fin de Grado: "Clasificación automática de imágenes del cielo mediante inteligencia artificial".
- [26] Intergovernmental Panel on Climate Change. (2013). Climate Change 2013. The Physical Science Basis. [Online] Disponible en: Link.
- [27] .ªnalyticsSteps"[Online] Disponible en: Link Último acceeso 5/07/2024.
- [28] Prosise J., Prosise A. (2022). .ªpplied Machine Learning and AI for Engineers. Solve Business Problems That Can't Be Solved Algorithmically".
- [29] Ferreras Extremo A (2021). "Estudio de algoritmos de redes neuronales convolucionales en dataset de imágenes médicas"
- [30] "GeeksforGeeks"[Online]. Disponible en: https://www.geeksforgeeks.org/adam-optimizer-in-tensorflow/. Último acceso 05/07/2024.

Anexo I

Canales R, G y B y aplicación de un filtro

[1]:
```
import cv2
import matplotlib.pyplot as plt
import numpy as np
```

[2]:
```
img_Gilbo = cv2.imread('Ruta/Gilbo.jpg')
```

[3]:
```
#Matplotlib lee RGB en lugar de BGR
imagen_rgb = cv2.cvtColor(img_Gilbo, cv2.COLOR_BGR2RGB)
plt.imshow(imagen_rgb)
plt.axis('off')
plt.show()
```

[4]:
```
canal_b, canal_g, canal_r = cv2.split(img_Gilbo)
canal = [canal_b, canal_g, canal_r]
```

```
   # Canal Rojo
plt.subplot(1, 3, 1)
plt.imshow(canal_r, cmap='gray')
plt.title("Canal R")
plt.axis('off')
    # Canal Verde
plt.subplot(1, 3, 2)
plt.imshow(canal_g, cmap='gray')
plt.title("Canal G")
plt.axis('off')
  # Canal Azul
plt.subplot(1, 3, 3)
plt.imshow(canal_b, cmap='gray')
plt.title("Canal B")
plt.axis('off')

plt.savefig("Ruta/canales_rgb.png", bbox_inches='tight', pad_inches=0, dpi=300)
plt.show()
```

[5]:
```
img_naranjo = cv2.imread('Ruta/caballos.jpg')
```

[6]:
```
#Apliquemos el filtro Canny a nuestra foto original
cany_imag = cv2.Canny(cv2.cvtColor(img_naranjo, cv2.COLOR_BGR2GRAY), 40, 40)
```

[7]:
```
plt.imshow(cv2.cvtColor(cany_imag, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.savefig("Cany_Filter.png", bbox_inches='tight', pad_inches=0)
plt.show()
```

Anexo II

Codigo para separar las imágenes en el data set de entrenamiento y validación y en el de test

```
import random
import os
import shutil

shutil.rmtree('Ruta/Train_Val')
os.mkdir('Ruta/Train_Val')
shutil.rmtree('Ruta/Test')
os.mkdir('Ruta/Test')

for i in range(0, 9):
    archivos_origen = os.listdir(f'Ruta/octas/{i}')
    archivos_Train_Val = random.sample(archivos_origen, 70)
    archivos_Test = [archivo for archivo in archivos_origen if archivo not in
    archivos_Train_Val]

    os.mkdir(f'C:/Users/victo/OneDrive/Fisica/Cuarto de carrera
    /TFG/Train_Val/{i}')
    for archivo in archivos_Train_Val:
            ruta_origen = os.path.join(f'Ruta/octas/{i}', archivo)

            ruta_destino = os.path.join(f'Ruta/Train_Val/{i}', archivo)

            shutil.copyfile(ruta_origen, ruta_destino)

    os.mkdir(f'Ruta/Test/{i}')

    for archivo in archivos_Test:
            ruta_origen = os.path.join(f'Ruta/octas/{i}', archivo)

            ruta_destino = os.path.join(f'Ruta/Test/{i}', archivo)

            shutil.copyfile(ruta_origen, ruta_destino)
```

Anexo III

Modelo y Analisis

[1]:
```
import tensorflow as tf
import matplotlib.pyplot as plt
import os
import seaborn as sns
from sklearn.metrics import confusion_matrix
import shutil
import numpy as np
```

```
C:\Users\victo\anaconda3\lib\site-packages\scipy\__init__.py:146: UserWarning: A
NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy
(detected version 1.26.4
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

[2]:
```
batch_size = 32
image_size = (224, 224)
```

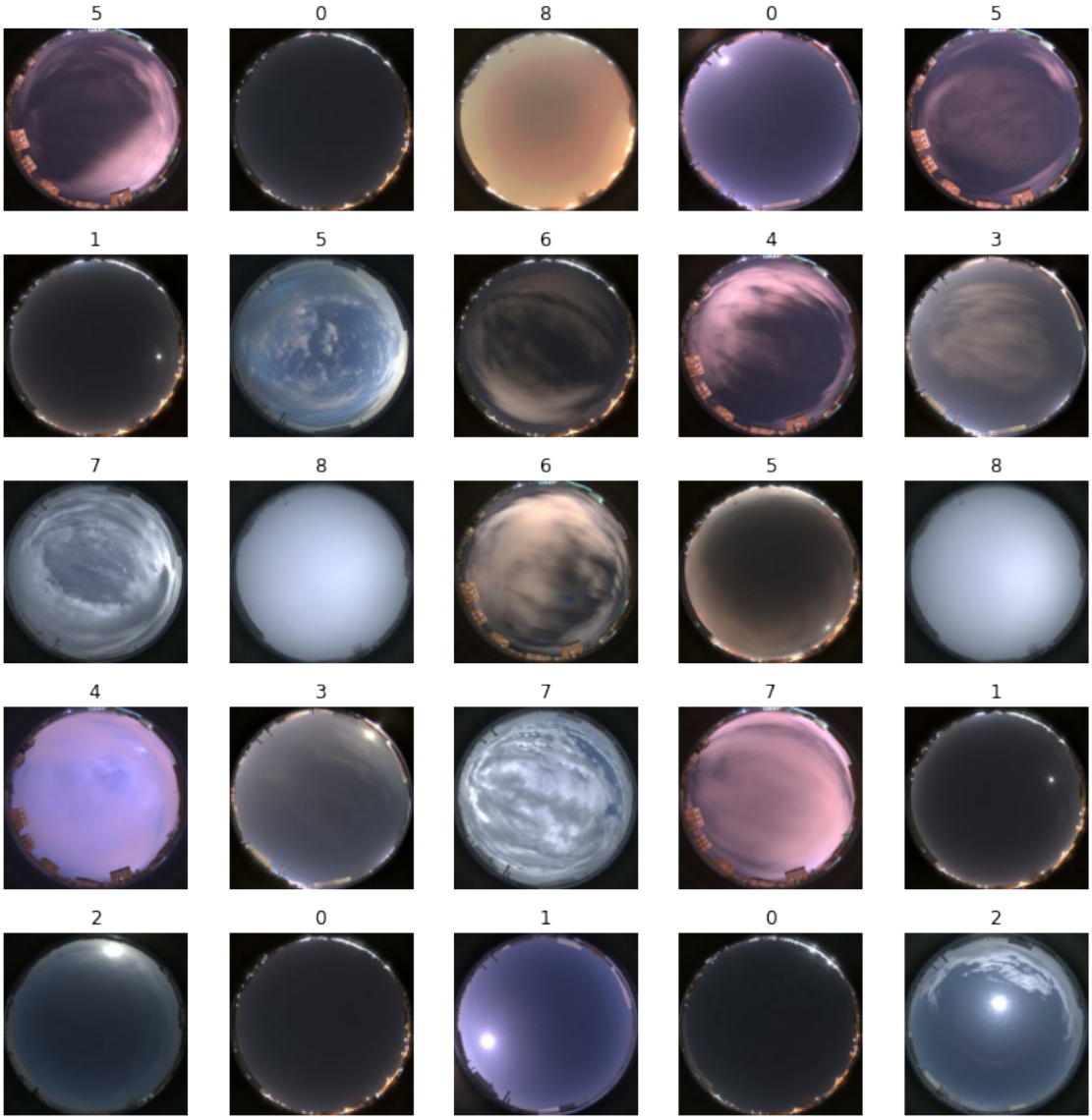
[3]:
```
#Código para leer las imágenes: Data set de entrenamiento
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
                'Ruta/Train_Val',     #Directorio donde se encuentran
                labels='inferred',                      #Para tomar cada␣
 ↪subdirectorio como una clase diferente
                label_mode='int',                       #Las etiquetas son␣
 ↪numeros enteros
                validation_split=0.1,                   #90% datos␣
 ↪entrenamiento, 10% validación
                subset="training",                      #Asignamos que este␣
 ↪data set es de entrenamiento
                seed=20220331,                          #Variable relacionada␣
 ↪con la aleatoriedad
                image_size=image_size,                  #Tamaño de las imágenes
                batch_size=batch_size,                  #Tamaño de los lotes de␣
 ↪entrenamiento
           )
```

```
Found 630 files belonging to 9 classes.
Using 567 files for training.
```

[4]:
```
#Código para leer las imágenes: Data set de validación
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
                'Ruta/Train_Val',
                labels='inferred',
                label_mode='int',
                validation_split=0.1,
                subset="validation",
                seed=20220331,
                image_size=image_size,
                batch_size=batch_size,
            )
```

```
Found 630 files belonging to 9 classes.
Using 63 files for validation.
```

[5]:
```
# Obtener un lote de datos de validación
for imagenes, etiquetas in train_ds.take(1):
    # Crear una nueva figura
    plt.figure(figsize=(10, 10))

    # Iterar sobre las primeras 25 imágenes y etiquetas en el lote
    for i in range(25):
        # Obtener la imagen y la etiqueta correspondiente
        imagen = imagenes[i]
        etiqueta = etiquetas[i]

        # Convertir la etiqueta a una cadena si es necesario
        etiqueta = str(etiqueta.numpy()) if isinstance(etiqueta, tf.Tensor)␣
 ↪else str(etiqueta)

        # Añadir una subparcela a la figura
        plt.subplot(5, 5, i + 1)
        plt.imshow(imagen.numpy().astype("uint8"))  # Mostrar la imagen como␣
 ↪una matriz numpy
        plt.title(etiqueta)                         # Mostrar la etiqueta como␣
 ↪título
        plt.axis("off")                             # Desactivar ejes

    # Ajustar el diseño y mostrar la figura
    plt.tight_layout()
    plt.show()
```



[6]:
```
#funciones de perdida
#categorical cross entropy
loss=tf.losses.SparseCategoricalCrossentropy()
"""
loss='mae' #num_classes = 1
loss='mse' #num_classes = 1
"""
```

[6]: "\nloss='mae' #num_classes = 1\nloss='mse' #num_classes = 1\n"

[7]:
```
#Numero clases:
num_classes = 9
```

[8]:
```
# Definición del modelo
model = tf.keras.Sequential([
    # Capa convolucional 1
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(224,␣
 ↪224, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),
    # Capa convolucional 2
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    # Capa convolucional 3
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    # Capa convolucional 4
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    # Capa de aplanamiento para conectar con las capas densas
    tf.keras.layers.Flatten(),
    # Primera capa densa
    tf.keras.layers.Dense(512, activation='relu'),
    # Capa de salida
    tf.keras.layers.Dense(num_classes, activation='softmax')  # Num_classes es␣
 ↪el número de clases de salida
])
```

```
C:\Users\victo\anaconda3\lib\site-
packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

[9]:
```
#optimizers
#Descenso del gradiente
#lr_schedule = 0.05
#optimizer = tf.keras.optimizers.SGD(learning_rate=lr_schedule)
#Optimizador adam
optimizer ='adam'
```

[10]:
```
# Directorio donde se guarda el modelo
copia_modelo = 'modelos/BACKUP_2_copia.keras'
```

[11]:
```
#compilar
model.compile(optimizer='adam', loss=loss, metrics=['accuracy'])
```

[12]:
```
#PARA OTRAS FUNCIONES (CALLBACKS)

# Para otros callbacks
# Guarda el último modelo con el mejor ajuste en val_ts
```

```
checkpoint = tf.keras.callbacks.ModelCheckpoint(
    copia_modelo,
    monitor="val_loss",
    verbose=1,
    save_best_only=True,
    save_weights_only=False,
    mode="auto"
)
shutil.rmtree('Ruta/log.txt')
os.makedirs('Ruta/log.txt')
# Guarda la información de cada época, residuos, etc.
csv_logger = tf.keras.callbacks.CSVLogger(
    'log.txt',
    append=True,
    separator=','
)
```

[13]:
```
#TE PARA LAS EPOCAS SI NO MEJORA EN 10 EPOCAS, y TE REDUCE LA LR SI NO MEJORA␣
 ↪EN 5
early_stopping = tf.keras.callbacks.EarlyStopping(monitor="val_loss",␣
 ↪patience=10)
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor="val_loss", patience=5)
```

[14]:
```
#ENTRENAR MODELO:
data_trainining = (train_ds)
model.fit(
    data_trainining,
    validation_data=val_ds,
    shuffle=True,
    epochs=300,
    callbacks=[checkpoint,early_stopping, reduce_lr,csv_logger],
)
```

```
Epoch 1/300
18/18            0s 1s/step -
accuracy: 0.1424 - loss: 33.2537
Epoch 1: val_loss improved from inf to 1.98676, saving model to
modelos/BACKUP_2_copia.keras
18/18            25s 1s/step -
accuracy: 0.1442 - loss: 32.3378 - val_accuracy: 0.1746 - val_loss: 1.9868 -
learning_rate: 0.0010
Epoch 2/300
18/18            0s 1s/step -
accuracy: 0.3148 - loss: 1.8763
Epoch 2: val_loss improved from 1.98676 to 1.75685, saving model to
modelos/BACKUP_2_copia.keras
18/18            43s 2s/step -
accuracy: 0.3157 - loss: 1.8732 - val_accuracy: 0.2698 - val_loss: 1.7569 -
```

```
learning_rate: 0.0010
Epoch 3/300
18/18            0s 1s/step -
accuracy: 0.3864 - loss: 1.6323
Epoch 3: val_loss improved from 1.75685 to 1.46099, saving model to
modelos/BACKUP_2_copia.keras
18/18            23s 1s/step -
accuracy: 0.3882 - loss: 1.6265 - val_accuracy: 0.4127 - val_loss: 1.4610 -
learning_rate: 0.0010
Epoch 4/300
18/18            0s 1s/step -
accuracy: 0.5442 - loss: 1.2854
Epoch 4: val_loss improved from 1.46099 to 1.45248, saving model to
modelos/BACKUP_2_copia.keras
18/18            23s 1s/step -
accuracy: 0.5445 - loss: 1.2842 - val_accuracy: 0.4286 - val_loss: 1.4525 -
learning_rate: 0.0010
Epoch 5/300
18/18            0s 1s/step -
accuracy: 0.5555 - loss: 1.1395
Epoch 5: val_loss improved from 1.45248 to 1.35213, saving model to
modelos/BACKUP_2_copia.keras
18/18            29s 2s/step -
accuracy: 0.5558 - loss: 1.1386 - val_accuracy: 0.4444 - val_loss: 1.3521 -
learning_rate: 0.0010
Epoch 6/300
18/18            0s 1s/step -
accuracy: 0.6687 - loss: 0.9558
Epoch 6: val_loss improved from 1.35213 to 1.28856, saving model to
modelos/BACKUP_2_copia.keras
18/18            30s 2s/step -
accuracy: 0.6666 - loss: 0.9601 - val_accuracy: 0.5714 - val_loss: 1.2886 -
learning_rate: 0.0010
Epoch 7/300
18/18            0s 1s/step -
accuracy: 0.6691 - loss: 0.9535
Epoch 7: val_loss did not improve from 1.28856
18/18            20s 1s/step -
accuracy: 0.6676 - loss: 0.9553 - val_accuracy: 0.5079 - val_loss: 1.3886 -
learning_rate: 0.0010
Epoch 8/300
18/18            0s 1s/step -
accuracy: 0.7052 - loss: 0.8877
Epoch 8: val_loss did not improve from 1.28856
18/18            21s 1s/step -
accuracy: 0.7048 - loss: 0.8847 - val_accuracy: 0.4444 - val_loss: 1.6056 -
learning_rate: 0.0010
Epoch 9/300
```

```
18/18            0s 1s/step -
accuracy: 0.7421 - loss: 0.7133
Epoch 9: val_loss did not improve from 1.28856
18/18            21s 1s/step -
accuracy: 0.7418 - loss: 0.7137 - val_accuracy: 0.5238 - val_loss: 1.3679 -
learning_rate: 0.0010
Epoch 10/300
18/18            0s 1s/step -
accuracy: 0.7604 - loss: 0.6358
Epoch 10: val_loss did not improve from 1.28856
18/18            21s 1s/step -
accuracy: 0.7600 - loss: 0.6362 - val_accuracy: 0.5397 - val_loss: 1.4036 -
learning_rate: 0.0010
Epoch 11/300
18/18            0s 1s/step -
accuracy: 0.8465 - loss: 0.4600
Epoch 11: val_loss did not improve from 1.28856
18/18            21s 1s/step -
accuracy: 0.8448 - loss: 0.4631 - val_accuracy: 0.5556 - val_loss: 1.7367 -
learning_rate: 0.0010
Epoch 12/300
18/18            0s 1s/step -
accuracy: 0.7781 - loss: 0.5883
Epoch 12: val_loss improved from 1.28856 to 1.28277, saving model to
modelos/BACKUP_2_copia.keras
18/18            21s 1s/step -
accuracy: 0.7808 - loss: 0.5820 - val_accuracy: 0.5873 - val_loss: 1.2828 -
learning_rate: 1.0000e-04
Epoch 13/300
18/18            0s 1s/step -
accuracy: 0.8697 - loss: 0.3488
Epoch 13: val_loss did not improve from 1.28277
18/18            22s 1s/step -
accuracy: 0.8708 - loss: 0.3473 - val_accuracy: 0.5556 - val_loss: 1.4107 -
learning_rate: 1.0000e-04
Epoch 14/300
18/18            0s 1s/step -
accuracy: 0.9179 - loss: 0.2682
Epoch 14: val_loss did not improve from 1.28277
18/18            21s 1s/step -
accuracy: 0.9177 - loss: 0.2690 - val_accuracy: 0.5873 - val_loss: 1.3432 -
learning_rate: 1.0000e-04
Epoch 15/300
18/18            0s 1s/step -
accuracy: 0.9027 - loss: 0.2561
Epoch 15: val_loss did not improve from 1.28277
18/18            22s 1s/step -
accuracy: 0.9029 - loss: 0.2565 - val_accuracy: 0.5873 - val_loss: 1.4226 -
```

```
learning_rate: 1.0000e-04
Epoch 16/300
18/18            0s 1s/step -
accuracy: 0.9103 - loss: 0.2448
Epoch 16: val_loss did not improve from 1.28277
18/18            38s 1s/step -
accuracy: 0.9108 - loss: 0.2450 - val_accuracy: 0.5556 - val_loss: 1.4461 -
learning_rate: 1.0000e-04
Epoch 17/300
18/18            0s 1s/step -
accuracy: 0.9336 - loss: 0.2156
Epoch 17: val_loss did not improve from 1.28277
18/18            20s 1s/step -
accuracy: 0.9329 - loss: 0.2166 - val_accuracy: 0.5873 - val_loss: 1.4246 -
learning_rate: 1.0000e-04
Epoch 18/300
18/18            0s 1s/step -
accuracy: 0.9314 - loss: 0.2443
Epoch 18: val_loss did not improve from 1.28277
18/18            20s 1s/step -
accuracy: 0.9317 - loss: 0.2433 - val_accuracy: 0.5873 - val_loss: 1.4422 -
learning_rate: 1.0000e-05
Epoch 19/300
18/18            0s 1s/step -
accuracy: 0.9356 - loss: 0.2276
Epoch 19: val_loss did not improve from 1.28277
18/18            21s 1s/step -
accuracy: 0.9358 - loss: 0.2271 - val_accuracy: 0.6032 - val_loss: 1.4740 -
learning_rate: 1.0000e-05
Epoch 20/300
18/18            0s 1s/step -
accuracy: 0.9332 - loss: 0.2363
Epoch 20: val_loss did not improve from 1.28277
18/18            23s 1s/step -
accuracy: 0.9336 - loss: 0.2351 - val_accuracy: 0.6032 - val_loss: 1.4925 -
learning_rate: 1.0000e-05
Epoch 21/300
18/18            0s 1s/step -
accuracy: 0.9465 - loss: 0.1984
Epoch 21: val_loss did not improve from 1.28277
18/18            21s 1s/step -
accuracy: 0.9460 - loss: 0.1992 - val_accuracy: 0.6032 - val_loss: 1.4917 -
learning_rate: 1.0000e-05
Epoch 22/300
18/18            0s 1s/step -
accuracy: 0.9431 - loss: 0.2090
Epoch 22: val_loss did not improve from 1.28277
18/18            21s 1s/step -
```

```
accuracy: 0.9428 - loss: 0.2091 - val_accuracy: 0.6032 - val_loss: 1.4941 -
learning_rate: 1.0000e-05
```

[14]:
```
<keras.src.callbacks.history.History at 0x1d88e124c10>
```

[15]:
```
# Carga el modelo guardado
modelo = tf.keras.models.load_model(copia_modelo)
# Muestra un resumen del modelo
modelo.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 222, 222, 32)	896
max_pooling2d (MaxPooling2D)	(None, 111, 111, 32)	0
conv2d_1 (Conv2D)	(None, 109, 109, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 54, 54, 64)	0
conv2d_2 (Conv2D)	(None, 52, 52, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 26, 26, 128)	0
conv2d_3 (Conv2D)	(None, 24, 24, 128)	147,584
max_pooling2d_3 (MaxPooling2D)	(None, 12, 12, 128)	0
flatten (Flatten)	(None, 18432)	0
dense (Dense)	(None, 512)	9,437,696
dense_1 (Dense)	(None, 9)	4,617

Total params: 29,049,437 (110.81 MB)

Trainable params: 9,683,145 (36.94 MB)

Non-trainable params: 0 (0.00 B)

Optimizer params: 19,366,292 (73.88 MB)

[16]:
```
#Cargamos el data set Test:
test_ds = tf.keras.preprocessing.image_dataset_from_directory(
                'Ruta/Test',  # Directorio donde se encuentran los datos de␣
 ↪prueba
                labels='inferred',                          # Para tomar cada␣
 ↪subdirectorio como una clase diferente
                label_mode='int',                           # Las etiquetas son␣
 ↪números enteros
                seed=20220331,                              # Variable relacionada␣
 ↪con la aleatoriedad
                image_size=image_size,                      # Tamaño de las imágenes
                batch_size=batch_size,                      # Tamaño de los lotes␣
 ↪de prueba
            )
```

```
Found 2565 files belonging to 9 classes.
```

[17]:
```
# Evaluar el modelo en el conjunto de datos de prueba
test_loss, test_accuracy = model.evaluate(test_ds)

# Imprimir la precisión en el conjunto de datos de prueba
print("Precisión en el conjunto de datos de prueba:", test_accuracy)
```

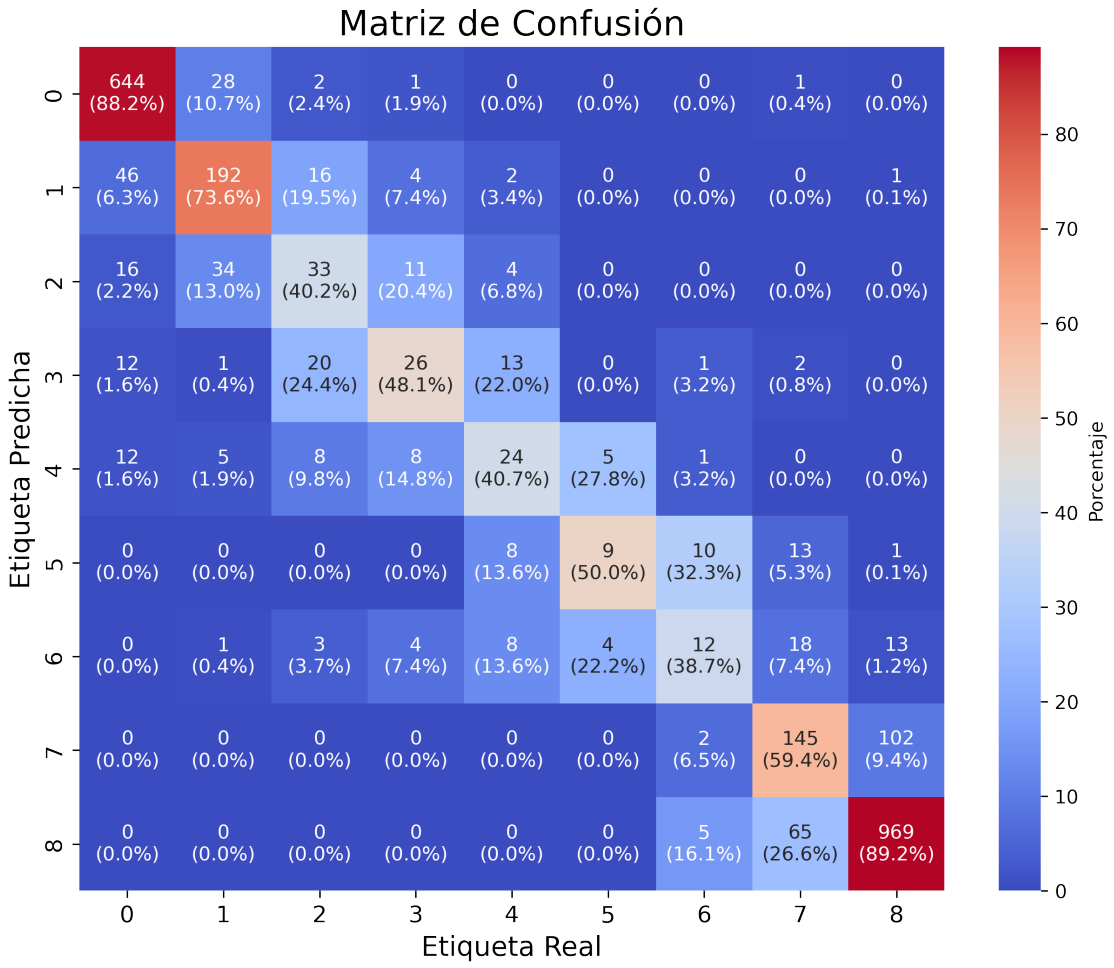
```
81/81             27s 338ms/step -
accuracy: 0.8084 - loss: 0.6895
Precisión en el conjunto de datos de prueba: 0.8007797002792358
```

[18]:
```
# Obtener la matriz de confusión
true_labels = []
predicted_labels = []

for images, labels in test_ds:
    true_labels.extend(labels.numpy())
    predicted_labels.extend(np.argmax(model.predict(images), axis=-1))

conf_matrix = confusion_matrix(predicted_labels, true_labels)

# Calcular el porcentaje de imágenes en cada cuadrado de la matriz de confusión
total_per_class = np.sum(conf_matrix, axis=0)  # Total de imágenes por clase
conf_matrix_percentage = conf_matrix / total_per_class[np.newaxis, :] * 100

# Crear una matriz de strings con el número de imágenes y el porcentaje
conf_matrix_labels = np.empty(conf_matrix.shape, dtype=object)
for i in range(conf_matrix.shape[0]):
    for j in range(conf_matrix.shape[1]):
        conf_matrix_labels[i, j] = f'{conf_matrix[i,␣
 ↪j]}\n({conf_matrix_percentage[i, j]:.1f}%)'
```

```
# Visualizar la matriz de confusión con porcentajes y números de imágenes
plt.figure(figsize=(10, 8), dpi=300)  # Aumentar el dpi para mejorar la␣
 ↪resolución
sns.heatmap(conf_matrix_percentage, annot=conf_matrix_labels, fmt='',␣
 ↪cmap='coolwarm', cbar=True, cbar_kws={'label': 'Porcentaje'})
plt.xlabel('Etiqueta Real', fontsize=14)
plt.ylabel('Etiqueta Predicha', fontsize=14)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.title('Matriz de Confusión', fontsize=18)
plt.savefig('Ruta/Matriz_densidad.jpg', bbox_inches='tight', dpi=300)  #␣
 ↪Guardar con mayor dpi
plt.show()
```

```
1/1             0s 442ms/step
1/1             0s 289ms/step
1/1             0s 328ms/step
1/1             0s 338ms/step
1/1             0s 329ms/step
1/1             0s 275ms/step
1/1             0s 292ms/step
1/1             0s 286ms/step
1/1             0s 289ms/step
1/1             0s 278ms/step
1/1             0s 281ms/step
1/1             0s 264ms/step
1/1             0s 306ms/step
1/1             0s 299ms/step
1/1             0s 275ms/step
1/1             0s 296ms/step
1/1             0s 268ms/step
1/1             0s 277ms/step
1/1             0s 286ms/step
1/1             0s 294ms/step
1/1             0s 314ms/step
1/1             0s 290ms/step
1/1             0s 272ms/step
1/1             0s 295ms/step
1/1             0s 286ms/step
1/1             0s 292ms/step
1/1             0s 294ms/step
1/1             0s 275ms/step
1/1             0s 281ms/step
1/1             0s 284ms/step
1/1             0s 278ms/step
1/1             0s 284ms/step
1/1             0s 296ms/step
```

```
1/1             0s 278ms/step
1/1             0s 301ms/step
1/1             0s 322ms/step
1/1             0s 314ms/step
1/1             0s 312ms/step
1/1             0s 292ms/step
1/1             0s 298ms/step
1/1             0s 301ms/step
1/1             0s 288ms/step
1/1             0s 349ms/step
1/1             0s 359ms/step
1/1             0s 321ms/step
1/1             0s 285ms/step
1/1             0s 279ms/step
1/1             0s 287ms/step
1/1             0s 289ms/step
1/1             0s 278ms/step
1/1             0s 302ms/step
1/1             0s 287ms/step
1/1             0s 289ms/step
1/1             0s 285ms/step
1/1             0s 288ms/step
1/1             0s 296ms/step
1/1             0s 290ms/step
1/1             0s 277ms/step
1/1             0s 289ms/step
1/1             0s 282ms/step
1/1             0s 275ms/step
1/1             0s 279ms/step
1/1             0s 312ms/step
1/1             0s 293ms/step
1/1             0s 295ms/step
1/1             0s 269ms/step
1/1             0s 277ms/step
1/1             0s 280ms/step
1/1             0s 273ms/step
1/1             0s 287ms/step
1/1             0s 267ms/step
1/1             0s 297ms/step
1/1             0s 280ms/step
1/1             0s 287ms/step
1/1             0s 280ms/step
1/1             0s 291ms/step
1/1             0s 278ms/step
1/1             0s 282ms/step
1/1             0s 298ms/step
1/1             0s 302ms/step
1/1             0s 236ms/step
```

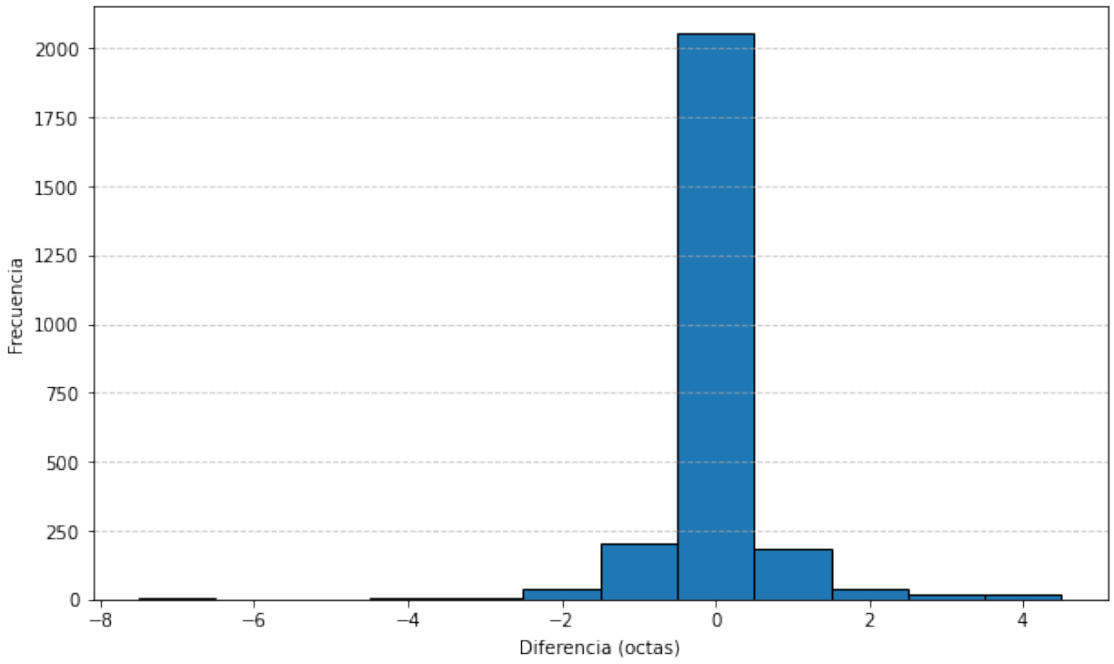


[19]:
```
# Inicializar las listas para etiquetas verdaderas y predichas
true_labels = []
predicted_labels = []

# Iterar sobre el dataset de prueba para obtener las etiquetas y predicciones
for images, labels in test_ds:
    true_labels.extend(labels.numpy())  # Convertir las etiquetas a un array de␣
 ↪numpy y añadirlas a true_labels
    predictions = model.predict(images)  # Obtener las predicciones del modelo
    predicted_labels.extend(np.argmax(predictions, axis=-1))  # Obtener el␣
 ↪índice de la clase predicha y añadirlo a predicted_labels

# Convertir las listas a arrays de numpy
true_labels = np.array(true_labels)
predicted_labels = np.array(predicted_labels)
```

```
# Calcular las diferencias
differences = predicted_labels - true_labels

# Crear un histograma de las diferencias
plt.figure(figsize=(10, 6))
plt.hist(differences, bins=range(int(np.min(differences)), int(np.
  ↪max(differences)) + 1), edgecolor='black', align='left')
plt.xlabel('Diferencia (octas)')
plt.ylabel('Frecuencia')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.savefig('Ruta/histograma_diferencias.png')
plt.show()
```

```
1/1             0s 293ms/step
1/1             0s 296ms/step
1/1             0s 343ms/step
1/1             0s 314ms/step
1/1             0s 297ms/step
1/1             0s 297ms/step
1/1             0s 319ms/step
1/1             0s 283ms/step
1/1             0s 348ms/step
1/1             0s 342ms/step
1/1             0s 396ms/step
1/1             0s 325ms/step
1/1             0s 352ms/step
1/1             0s 289ms/step
1/1             0s 289ms/step
1/1             0s 282ms/step
1/1             0s 284ms/step
1/1             0s 296ms/step
1/1             0s 302ms/step
1/1             0s 277ms/step
1/1             0s 275ms/step
1/1             0s 273ms/step
1/1             0s 283ms/step
1/1             0s 296ms/step
1/1             0s 299ms/step
1/1             0s 296ms/step
1/1             0s 286ms/step
1/1             0s 280ms/step
1/1             0s 283ms/step
1/1             0s 304ms/step
1/1             0s 296ms/step
1/1             0s 315ms/step
1/1             0s 307ms/step
1/1             0s 310ms/step
1/1             0s 285ms/step
```

```
1/1             0s 311ms/step
1/1             0s 285ms/step
1/1             0s 285ms/step
1/1             0s 320ms/step
1/1             0s 346ms/step
1/1             0s 316ms/step
1/1             0s 285ms/step
1/1             0s 284ms/step
1/1             0s 279ms/step
1/1             0s 281ms/step
1/1             0s 277ms/step
1/1             0s 294ms/step
1/1             0s 308ms/step
1/1             0s 303ms/step
1/1             0s 280ms/step
1/1             0s 288ms/step
1/1             0s 273ms/step
1/1             0s 280ms/step
1/1             0s 278ms/step
1/1             0s 295ms/step
1/1             0s 291ms/step
1/1             0s 273ms/step
1/1             0s 288ms/step
1/1             0s 285ms/step
1/1             0s 269ms/step
1/1             0s 266ms/step
1/1             0s 265ms/step
1/1             0s 265ms/step
1/1             0s 279ms/step
1/1             0s 281ms/step
1/1             0s 302ms/step
1/1             0s 289ms/step
1/1             0s 269ms/step
1/1             0s 270ms/step
1/1             0s 282ms/step
1/1             0s 270ms/step
1/1             0s 308ms/step
1/1             0s 294ms/step
1/1             0s 307ms/step
1/1             0s 314ms/step
1/1             0s 279ms/step
1/1             0s 294ms/step
1/1             0s 320ms/step
1/1             0s 276ms/step
1/1             0s 321ms/step
1/1             0s 95ms/step
```



[20]:
```
import scipy
#algunos valores estadísticos descriptivos
# Media (Promedio)
print(f"La media es: ",np.mean(differences))

# Mediana
print(f"La mediana es: {np.median(differences):.10f}")

# Moda
print(f"La moda es: {scipy.stats.mode(differences)[0][0]}")

# Varianza
print(f"La varianza es: ",np.var(differences))

# Desviación estándar
print(f"La desviación estandar es: ",np.std(differences))

# Rango
print(f"El rango es: ",np.std(differences))

#Asimetria:
print(f"La asimetría de la distribución es", scipy.stats.skew(differences))
```

```
La media es:  0.030019493177387915
La mediana es: 0.0000000000
La moda es: 0
```

```
La varianza es:  0.5133288495225501
La desviación estandar es:  0.7164697129136375
El rango es:  0.7164697129136375
La asimetría de la distribución es 0.610951392636848
```

[21]:
```python
# Precision con mas menos un fallo en octas
test_loss, test_accuracy = model.evaluate(test_ds)

# Obtener las etiquetas verdaderas y las predicciones del modelo
true_labels = []
predicted_labels = []

for images, labels in test_ds:
    true_labels.extend(labels.numpy())  # Convertir las etiquetas a un array de numpy y añadirlas a true_labels
    predictions = model.predict(images)  # Obtener las predicciones del modelo
    predicted_labels.extend(np.argmax(predictions, axis=-1))  # Obtener el índice de la clase predicha y añadirlo a predicted_labels

# Convertir las listas a arrays de numpy
true_labels = np.array(true_labels)
predicted_labels = np.array(predicted_labels)

# Calcular la precisión contando como acierto si la predicción está dentro de un rango de ±1 alrededor de la etiqueta verdadera
aciertos = np.sum((predicted_labels == true_labels) |
                  (predicted_labels == true_labels + 1) |
                  (predicted_labels == true_labels - 1))

precision_con_rango = aciertos / len(true_labels)

# Imprimir la precisión con el rango de ±1
print("Precisión en el conjunto de datos de prueba (incluyendo ±1):", precision_con_rango)
```

```
81/81            23s 287ms/step -
accuracy: 0.8071 - loss: 0.7023
1/1              0s 337ms/step
1/1              0s 275ms/step
1/1              0s 273ms/step
1/1              0s 271ms/step
1/1              0s 298ms/step
1/1              0s 273ms/step
1/1              0s 304ms/step
1/1              0s 269ms/step
1/1              0s 304ms/step
1/1              0s 279ms/step
1/1              0s 274ms/step
```

```
1/1             0s 269ms/step
1/1             0s 287ms/step
1/1             0s 293ms/step
1/1             0s 323ms/step
1/1             0s 321ms/step
1/1             0s 330ms/step
1/1             0s 324ms/step
1/1             0s 353ms/step
1/1             0s 312ms/step
1/1             0s 316ms/step
1/1             0s 343ms/step
1/1             0s 341ms/step
1/1             0s 354ms/step
1/1             0s 328ms/step
1/1             0s 292ms/step
1/1             0s 281ms/step
1/1             0s 277ms/step
1/1             0s 288ms/step
1/1             0s 328ms/step
1/1             0s 304ms/step
1/1             0s 294ms/step
1/1             0s 289ms/step
1/1             0s 279ms/step
1/1             0s 282ms/step
1/1             0s 286ms/step
1/1             0s 283ms/step
1/1             0s 280ms/step
1/1             0s 279ms/step
1/1             0s 349ms/step
1/1             0s 309ms/step
1/1             0s 282ms/step
1/1             0s 285ms/step
1/1             0s 277ms/step
1/1             0s 293ms/step
1/1             0s 281ms/step
1/1             0s 305ms/step
1/1             0s 267ms/step
1/1             0s 292ms/step
1/1             0s 289ms/step
1/1             0s 275ms/step
1/1             0s 274ms/step
1/1             0s 279ms/step
1/1             0s 271ms/step
1/1             0s 292ms/step
1/1             0s 323ms/step
1/1             0s 326ms/step
1/1             0s 309ms/step
1/1             0s 323ms/step
```

```
1/1             0s 314ms/step
1/1             0s 291ms/step
1/1             0s 331ms/step
1/1             0s 319ms/step
1/1             0s 328ms/step
1/1             0s 346ms/step
1/1             0s 280ms/step
1/1             0s 286ms/step
1/1             0s 284ms/step
1/1             0s 271ms/step
1/1             0s 275ms/step
1/1             0s 266ms/step
1/1             0s 298ms/step
1/1             0s 287ms/step
1/1             0s 300ms/step
1/1             0s 277ms/step
1/1             0s 281ms/step
1/1             0s 288ms/step
1/1             0s 280ms/step
1/1             0s 285ms/step
1/1             0s 295ms/step
1/1             0s 72ms/step
Precisión en el conjunto de datos de prueba (incluyendo ±1): 0.9528265107212476
```

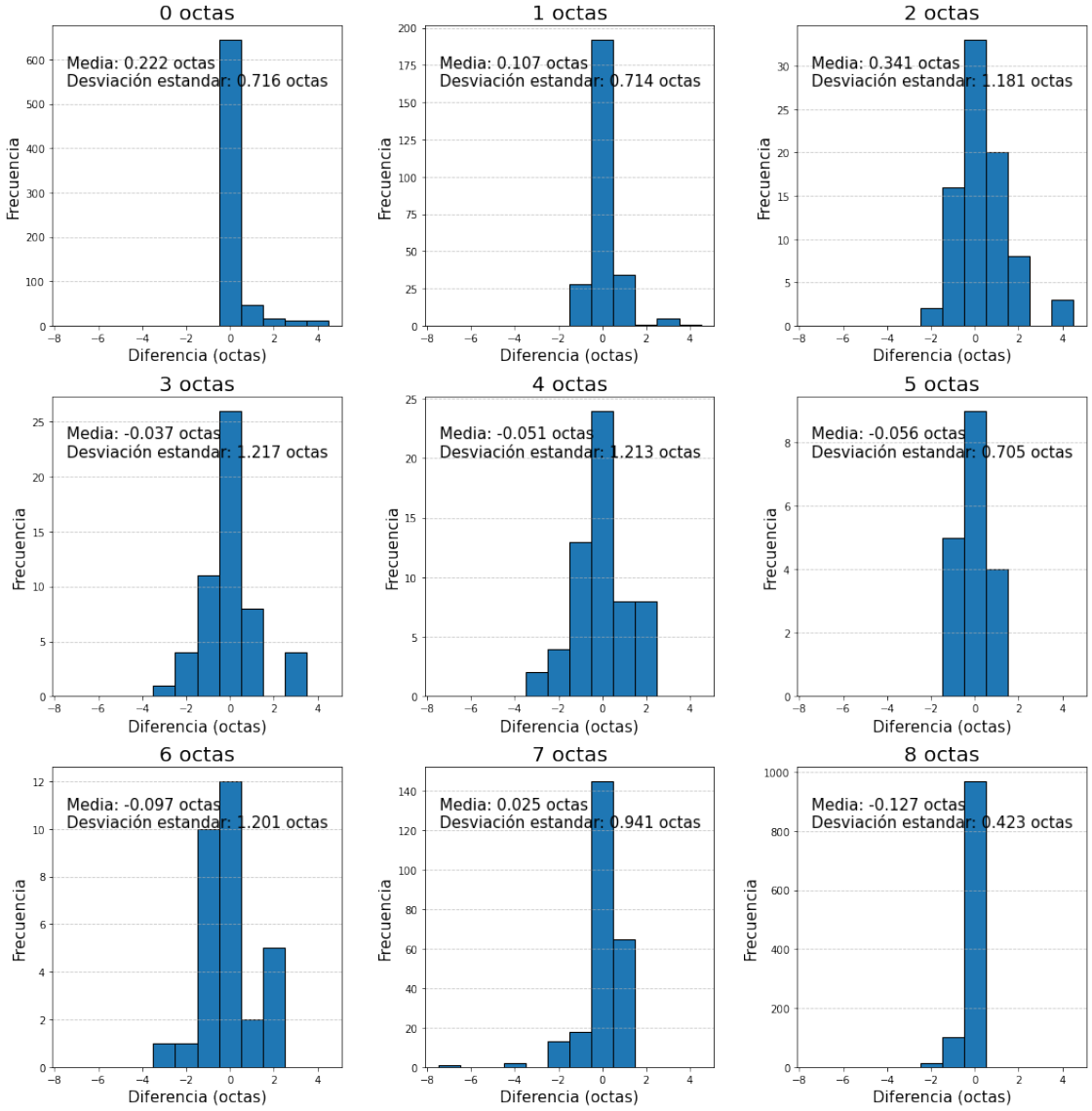
[22]:
```python
# Obtener las etiquetas verdaderas y las predicciones del modelo
true_labels = []
predicted_labels = []

for images, labels in test_ds:
    true_labels.extend(labels.numpy())  # Convertir las etiquetas a un array de␣
 ↪numpy y añadirlas a true_labels
    predictions = model.predict(images)  # Obtener las predicciones del modelo
    predicted_labels.extend(np.argmax(predictions, axis=-1))  # Obtener el␣
 ↪índice de la clase predicha y añadirlo a predicted_labels

# Convertir las listas a arrays de numpy
true_labels = np.array(true_labels)
predicted_labels = np.array(predicted_labels)

# Calcular las diferencias entre las etiquetas verdaderas y las predicciones
differences =predicted_labels - true_labels

# Obtener las etiquetas únicas
unique_labels = np.unique(true_labels)

# Crear un subplot para cada etiqueta
fig, axes = plt.subplots(3, 3, figsize=(15, 15))
```

```
# Iterar sobre cada etiqueta única
for i, label in enumerate(unique_labels):
    # Seleccionar solo las diferencias para la etiqueta actual
    label_differences = differences[true_labels == label]

    # Obtener el subplot correspondiente
    ax = axes[i // 3, i % 3]

    # Crear el histograma para la etiqueta actual
    ax.hist(label_differences, bins=range(int(np.min(differences)), int(np.
 ↪max(differences)) + 1), edgecolor='black', align='left')
    ax.set_title(f'{label} octas', fontsize=20)
    ax.set_xlabel('Diferencia (octas)', fontsize=15)
    ax.set_ylabel('Frecuencia', fontsize=15)
    ax.grid(axis='y', linestyle='--', alpha=0.7)

    # Calcular la media, mediana y varianza
    mean = np.mean(label_differences)
    des = np.std(label_differences)

    # Agregar texto con la media, mediana y varianza al gráfico
    ax.text(0.05, 0.9, f'Media: {mean:.3f} octas \nDesviación estandar: {des:.
 ↪3f} octas', transform=ax.transAxes, fontsize=15, verticalalignment='top')

# Ajustar el espaciado entre subplots
plt.tight_layout()

# Guardar la figura
plt.savefig('Ruta/histogramas_por_octa.png', dpi=300)

# Mostrar los histogramas
plt.show()
```

```
1/1             0s 274ms/step
1/1             0s 280ms/step
1/1             0s 274ms/step
1/1             0s 270ms/step
1/1             0s 288ms/step
1/1             0s 317ms/step
1/1             0s 271ms/step
1/1             0s 276ms/step
1/1             0s 272ms/step
1/1             0s 267ms/step
1/1             0s 269ms/step
1/1             0s 271ms/step
1/1             0s 285ms/step
```

```
1/1             0s 318ms/step
1/1             0s 281ms/step
1/1             0s 264ms/step
1/1             0s 282ms/step
1/1             0s 302ms/step
1/1             0s 344ms/step
1/1             0s 322ms/step
1/1             0s 291ms/step
1/1             0s 301ms/step
1/1             0s 280ms/step
1/1             0s 332ms/step
1/1             0s 338ms/step
1/1             0s 297ms/step
1/1             0s 273ms/step
1/1             0s 290ms/step
1/1             0s 272ms/step
1/1             0s 274ms/step
1/1             0s 276ms/step
1/1             0s 273ms/step
1/1             0s 270ms/step
1/1             0s 280ms/step
1/1             0s 275ms/step
1/1             0s 277ms/step
1/1             0s 282ms/step
1/1             0s 272ms/step
1/1             0s 268ms/step
1/1             0s 263ms/step
1/1             0s 300ms/step
1/1             0s 298ms/step
1/1             0s 275ms/step
1/1             0s 278ms/step
1/1             0s 282ms/step
1/1             0s 274ms/step
1/1             0s 277ms/step
1/1             0s 289ms/step
1/1             0s 264ms/step
1/1             0s 269ms/step
1/1             0s 303ms/step
1/1             0s 272ms/step
1/1             0s 278ms/step
1/1             0s 263ms/step
1/1             0s 278ms/step
1/1             0s 283ms/step
1/1             0s 290ms/step
1/1             0s 279ms/step
1/1             0s 318ms/step
1/1             0s 284ms/step
1/1             0s 290ms/step
```

```
1/1             0s 331ms/step
1/1             0s 347ms/step
1/1             0s 379ms/step
1/1             0s 333ms/step
1/1             0s 320ms/step
1/1             0s 330ms/step
1/1             0s 365ms/step
1/1             0s 338ms/step
1/1             0s 287ms/step
1/1             0s 280ms/step
1/1             0s 284ms/step
1/1             0s 273ms/step
1/1             0s 271ms/step
1/1             0s 277ms/step
1/1             0s 261ms/step
1/1             0s 278ms/step
1/1             0s 293ms/step
1/1             0s 285ms/step
1/1             0s 278ms/step
1/1             0s 88ms/step
```


0 octas
Media: 0.222 octas
Desviación estandar: 0.716 octas
1 octas
Media: 0.107 octas
Desviación estandar: 0.714 octas
2 octas
Media: 0.341 octas
Desviación estandar: 1.181 octas
3 octas
Media: -0.037 octas
Desviación estandar: 1.217 octas
4 octas
Media: -0.051 octas
Desviación estandar: 1.213 octas
5 octas
Media: -0.056 octas
Desviación estandar: 0.705 octas
6 octas
Media: -0.097 octas
Desviación estandar: 1.201 octas
7 octas
Media: 0.025 octas
Desviación estandar: 0.941 octas
8 octas
Media: -0.127 octas
Desviación estandar: 0.423 octas
Frecuencia
Diferencia (octas)

[ ]: