Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE GRADO

Caracterización de cielos según la taxonomía de la Comisión Internacional de la Iluminación mediante redes de aprendizaje profundo

Grado en Ingeniería Mecánica

ALUMNO: Kasin Sooksong

DIRECTORES: Andrés Suárez García

CURSO ACADÉMICO: 2019-2020

Universidade Vigo

Centro Universitario de la Defensa
en la Escuela Naval Militar

TRABAJO FIN DE GRADO

Caracterización de cielos según la taxonomía de la Comisión Internacional de la Iluminación mediante redes de aprendizaje profundo

Grado en Ingeniería Mecánica

Intensificación en Tecnología Naval

Cuerpo General / Infantería de Marina

Universidade Vigo

RESUMEN

La Comisión Internacional de la Iluminación (Commission internationale de l'éclairag, CIE) creó la clasificación de cielos para distinguir entre distintas nubosidades. En el ámbito de la eficiencia energética es importante, puesto que se podrían crear sistemas que dejasen pasar más o menos luz natural, repercutiendo directamente en el gasto energético. De forma habitual, en la clasificación del cielo según la taxonomía CIE, se utiliza el dispositivo "Sky Scanner" valorado en varios miles de euros. En este TFG, se emplearon los modelos de redes neuronales convolucionales de aprendizaje profundo creado en el Google Colab, utilizando Tensorflow junto con las bibliotecas Keras, Numpy, Pandas, Sci-kit Learn, Matplotlib y Seaborn para sustituir la función del Sky Scanner. El modelo creado se entrenó con las imágenes captadas de la cámara "Sieltec SONA 201D" desde la Escuela Politécnica Superior de la Universidad de Burgos durante todo el año de 2017. Se emplearon dos arquitecturas AlexNet y VGG con la intención de sacar la mejor configuración probando varios parámetros evaluando con las métricas como la matriz de confusión, la exactitud, la precisión, el valor F1 y la exhaustividad. Además, se realizaron pruebas con los datos originales desbalanceados en cuanto tamaño de categorías y se creó un conjunto de datos para evitar los efectos negativos del desbalanceo. Además, se realizaron extensiones de pruebas como la clasificación de tres tipos de cielos: nublado, parcialmente nublado y despejado. Los resultados finales de este TFG no son como se esperaba debido a las imágenes que se ha intentado clasificar que son muy similares y han sido difícil de distinguir incluso con los ojos humanos.

PALABRAS CLAVE

Sky Scanner, aprendizaje profundo, Google Colab, TensorFlow, redes neuronales convolucionales

AGRADECIMIENTOS

Me gustaría agradecerle a mi tutor, Don Andrés Suárez García, por su implicación y apoyo en este trabajo fin de grado.

CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1 INTRODUCCIÓN Y OBJETIVOS

1.1 Introducción

La iluminación solar es una parte importante de la vida humana. A lo largo de la historia, La humanidad ha aprendido a utilizar la luz del día para mejorar la comodidad de la vida humana. La luz del día también es la parte más importante para el desarrollo de la innovación y las tecnologías. Es necesario determinar las condiciones de la luz del día por las condiciones del cielo y los lugares.

La Comisión Internacional de la Iluminación (Commission internationale de l'éclairag, CIE) es la organización internacional científica no lucrativa que sus objetivos son proporcionar los conocimientos e investigación sobre la iluminación solar y estandarizar las metodologías relacionadas a la iluminación.

La distribución de la iluminación solar depende de tiempo, clima y la posición del sol a lo largo del día. Con estas condiciones se ha generado las condiciones de cielo desde los cielos totalmente nublado hasta los cielos totalmente despejado.

La taxonomía de los 15 cielos estándares de CIE depende de la distribución solar relativa calculados con los parámetros como la posición del sol, el ángulo Zenith , la diferencia entre el azimut del elemento en el cielo y el azimut del meridiano del sol categorizando desde la clase 1 que es el más nublado hasta la clase 15 que se considera como la clase de cielo más despejado (Figura 1-1) [1].

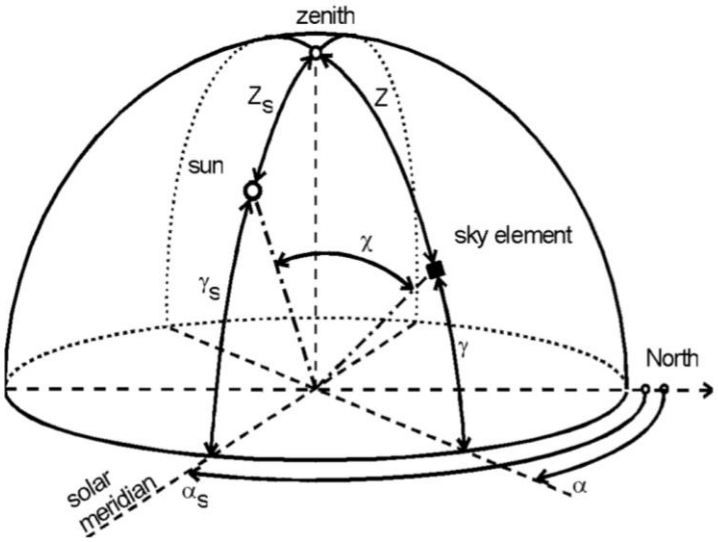


Figura 1-1 el concepto del cielo CIE. (Fuente: [1])

La medición y clasificación de los cielos se puede realizar mediante el dispositivo “Sky scanner”. Sin embargo, este es un dispositivo de alto coste. Se podría intentar obtener la misma clasificación a partir de imágenes de la bóveda celeste. El modelo de dispositivo utilizado para captar las imágenes que se ha utilizado en este trabajo fin de grado es el modelo de Sky Scanner “EKO MS-321LR” con la cámara para captar las imágenes “Sieltec SONA 201D” establecido en la Escuela Politécnica Superior de la Universidad de Burgos, en la provincia de Burgos. Las imágenes pertenecen en todo el año en 2017.

1.2 Objetivos del trabajo fin de grado

Se ha contemplado la dificultad de empleo de dispositivo “Sky Scanner” debido al precio alto del dispositivo y la incapacidad de movilización del mismo. Por ello, se ha desarrollado el presente trabajo con el objetivo principal de sustituir el “Sky Scanner” por las imágenes de la bóveda celesta, utilizando un modelo de redes neuronales de aprendizaje profundo como nexo de unión entre los registros obtenidos por ambos dispositivos. Además, se han utilizado las herramientas más económicas posibles como el terminal Google Colab y Tensorflow que les ofrecen a los usuarios de Google de forma gratuita. Ello agilizó las simulaciones, puesto que el hardware necesario para procesar este tipo de redes es altamente costoso de varios miles de euros.

Otros objetivos que se han contemplado en este trabajo fin de grado son:

- Describir los posibles obstáculos al emplear las redes neuronales de aprendizaje profundo en este tipo de trabajos.
- Determinar los valores óptimos de los parámetros considerados para poder ser referenciado a los trabajos similares en el futuro.
- Seleccionar las herramientas y estructuras adecuadas de redes neuronales de aprendizaje profundo.

Y otros objetivos que se han contemplado y se esperan lograr son los siguientes:

- Ser inspiración de todos los interesados en el campo de inteligencia artificial y aprendizaje de máquina que son las bases de tecnología del presente y futuro.
- Ser motivación prestar atención a los cambios de las tecnologías que podrían afectar a la vida a corto-medio plazo.

2 ESTADO DEL ARTE

2.1 La inteligencia artificial

2.1.1 La inteligencia artificial y su evolución

La inteligencia artificial (IA) se puede definir como aquella inteligencia exhibida por artefactos científicos construidos por humanos. Un sistema artificial posee inteligencia cuando es capaz de llevar a cabo tareas que, si fuesen realizadas por lógicas equivalentes a humanos, se diría que este es inteligente. Se considera que el origen de la IA se remonta a los intentos del hombre desde la antigüedad por incrementar sus potencialidades físicas e intelectuales, creando artefactos con automatismos y simulando la forma y las habilidades de los seres humanos.

La primera referencia de la creación de la IA es en el año 1950. El científico informático llamado Alan Turing es considerado uno de los padres de la ciencia de computación y la informática moderna. Es conocido por la concepción del Test de Turing [2], un criterio según el cual puede juzgarse la inteligencia de una máquina si sus respuestas en la prueba son indistinguibles de las de un ser humano. Esta prueba es aceptable para las métricas de inteligencia artificial. En esa época, se define por primera vez la palabra "Inteligencia Artificial" por el científico informático John Mccarthy en la conferencia de Darthmouth que tiene lugar en el verano de 1956 en la universidad de Darthmouth. En 1965, un grupo de investigadores de la Universidad de Stanford, liderado por el profesor Edward Feigenbaum, inventaron el primer sistema de expertos llamado Dendral o Dendral Heuristic [3]. El sistema fue creado para hacer las predicciones de las sustancias químicas utilizando la base de datos químicos.

Los períodos de 1974 a 1980 y 1987 a 1993 fueron nombrados como el invierno de la IA. Esos años fueron la época en que muchos investigadores de la IA se enfrentaron a muchas dificultades por la falta de fondos para las investigaciones y la falta de confianza de muchos de los sectores industriales. Después de 1990, la nueva era de la tecnología de la IA emergió junto con la expansión de la Internet. Muchos investigadores pudieron acceder a los conocimientos recientes para inventar nuevas tecnologías de la IA. Además, con las nuevas innovaciones creadas durante esos años, la base de datos de conocimientos de la IA se expandió mucho. En 1997, fue la primera vez que la máquina basada en la IA llamada Deep Blue de IBM [4] pudo ganar a un campeón mundial de ajedrez, Garry Kasparov, en competición de ajedrez. A partir de entonces, el uso de la IA en los juegos empezó a aceptarse y se inició el desarrollo de la complejidad de la IA para resolver los problemas complejos.

Después del año 2000, se han creado muchas innovaciones basadas en la tecnología de la IA para los consumidores: el robot Asimo de Honda [5] que puede imitar los gestos humanos, los vehículos no tripulados impulsados por la IA compleja, el Alpha Go [6] [7] de Google que pudo ganar al campeón mundial humano en la competición de Go, etc.

La IA tiene muchas influencias y efectos en la sociedad. En 2016, se anunció la asociación sobre la IA [8] para beneficiar a las personas y a la sociedad y comprometerse a hacer un uso responsable de la IA. Sus miembros fundadores son Amazon, Google, Facebook, Deepmind, Microsoft e IBM. Esta coalición sin fines de lucro fue anunciada para encontrar la mejor manera de utilizar las tecnologías de la IA, proporcionando el conocimiento a la sociedad y para controlar la seguridad del uso de la IA.

2.1.2 El Aprendizaje Máquina (Machine Learning)

El aprendizaje máquina o aprendizaje automático es la ciencia de la programación de la computadora para que puedan aprender por sí mismo desde los datos proporcionados y así poder trabajar con las datos del futuro.

En 1959, El Sr. Arthur Samuel, Profesor de la Universidad de Stanford, dio la definición de aprendizaje automático como "campo de estudio que le da a las computadoras la capacidad de aprender sin ser programadas explícitamente" [9]. El uso del aprendizaje de máquina es diferente a una programación estructurada que tenemos que alimentar la entrada a un programa y obtener los datos de salida utilizando las funciones del programa. En el aprendizaje máquina, solamente necesita los datos de entrada y los de salida. La máquina relaciona automáticamente los datos de entrada con los de salida, aprendiendo y siendo capaz de predecir nuevos datos de salida a partir de nuevos datos de entrada.

2.2 Las redes neuronales de aprendizaje profundo

2.2.1 El concepto de las redes neuronales artificiales

La red neuronal humana está formada por una gran cantidad de neurnas. La neurona se podría definir como la unidad de procesamiento más pequeña del ser humano. Cada neurona está construida por tres componentes principales que son Dendrita (Dendrites), Soma o cuerpo celular (Cell body) y el Axón (Axon). Las neuronas están conectadas entre sí por Sinapsis (Synapse) que funciona como punto de conexión ubicada en Axón y Soma (Figura). Las principales funciones de una neurona son recibir, transmitir y procesar las señales neuronales de entrada de otras neuronas. Una neurona transmite la señal a través de la sinapsis química y eléctrica en el proceso llamado neurotransmisión, que es el principal proceso que activa el neurotransmisor (señales químicas) y el Potencial de Acción (señales eléctricas) para transmitir información dentro de la red neuronal.

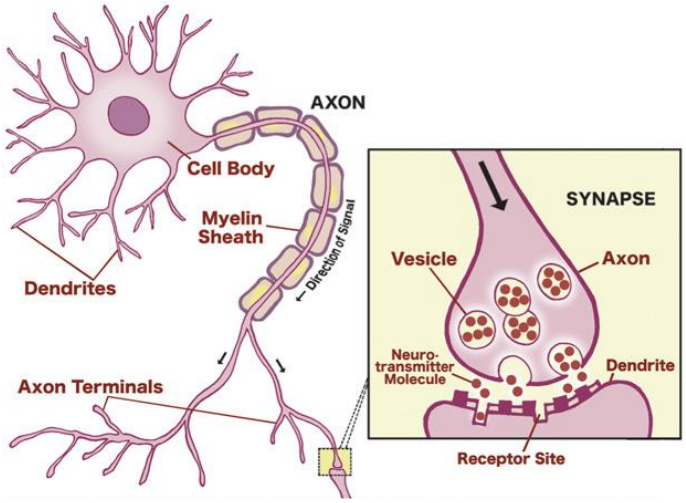


Figura 2-1 Estructura de una neurona. (Fuente: http://www.urbanchildinstitute.org/why-0-3/baby-and-brain)

El modelo matemático de la red neuronal artificial se basa en el estudio de las funciones y la estructura de la neurona humana, lo que significa que puede ser capaz de guardar el conocimiento mediante un proceso de aprendizaje. El conocimiento que el modelo ha aprendido se guarda en forma de valor llamado peso. Cada vez que el modelo aprende, el valor de peso se modificará. Este proceso es similar al proceso de aprendizaje humano.

En la red neuronal artificial, el procesamiento se lleva a cabo en las pequeñas unidades llamadas nodos o perceptrones que es el modelo matemático equivalente a una neurona humana. Los componentes principales de un nodo son. (Figura 2-2)

- **Datos de entrada**: equivalente a las señales de entrada de una neurona humana
- **Datos de salida**: equivalente a las señales de salida de una neurona humana. Son valores de salida de una función de activación aplicada a la función de suma
- **Los valores de peso**.
- **Sesgo o Bias**: un valor constante que se añade para ajustar el valor de salida junto con la función de suma
- **Función de suma**: se define como la suma de todos los valores de entrada multiplicado por los pesos sinápticos.
- **Función de activación**: la función que determina la salida de un nodo. Las funciones utilizadas frecuentes son ReLu, Sigmoid, TanH, Logistic, etc

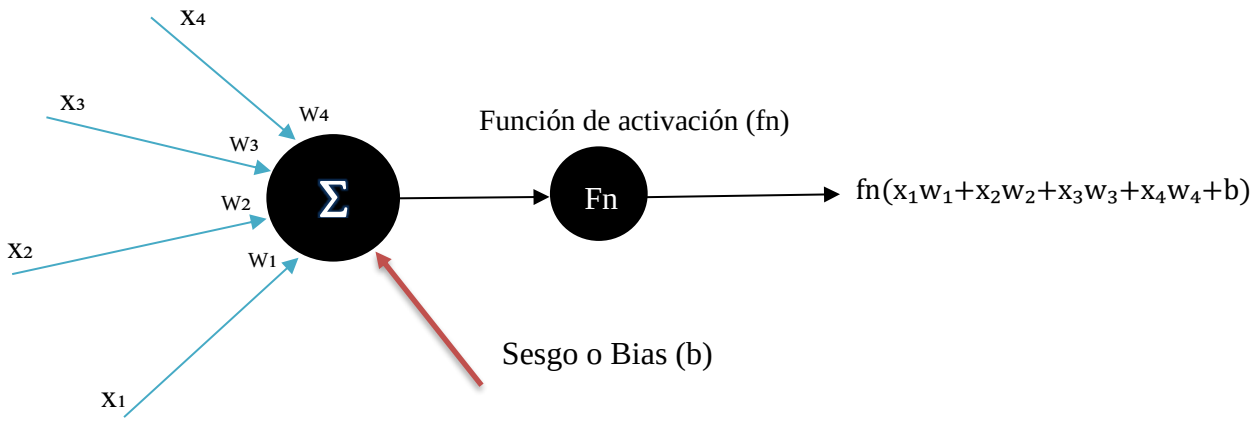


Figura 2-2 La estructrura de un Perceptrón. (Fuente: Elaboración propia)

Cada nodo está conectado con otros a través de enlaces. En estos enlaces, los valores de salida del nodo anterior son multiplicados con un valor de peso, agrupándose posteriormente mediante la función de suma. Para transmitir el valor de salida de un nodo, se aplica la función de activación sobre la función de suma y el valor de Sesgo. Una capa que contiene varios nodos enlazados con los otros de las otras capas.

2.2.2 Perceptrón

Por definición, el perceptrón es igual a una neurona artificial. Por tanto, un perceptrón multicapa es un tipo de red neuronal artificial con varias capas. Las capas pueden clasificarse en 3 tipos(Figura 2-3)

- **capa de entrada (input layer)**: es la capa que recibe los datos iniciales del proceso
- **capa oculta (hidden layers)**: son las capas entre la capa de entrada y salida
- **capa de salida (output layer)**: la última capa de la red

Además, el perceptrón multicapa se utiliza la retro propagación o la propagación hacia atrás como algoritmo para el entrenamiento de este tipo de redes

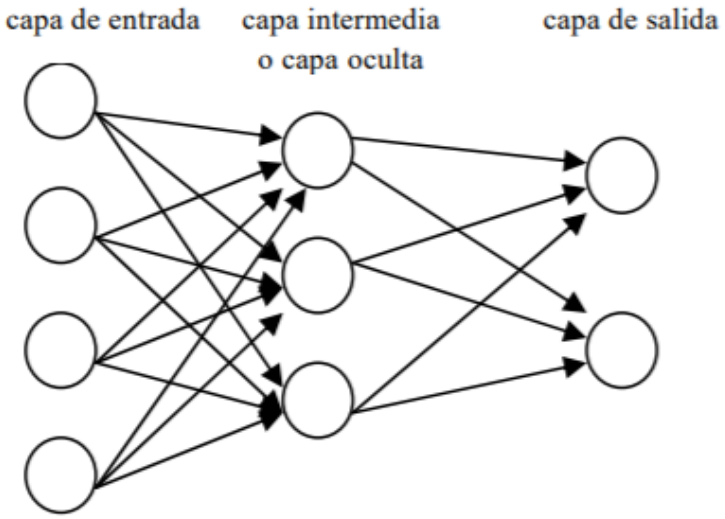


Figura 2-3 La arquitectura de Perceptrón Multicapa. (Fuente : [10])

2.2.3 El proceso de aprendizaje

El aprendizaje profundo es inspirado en el funcionamiento de la red neuronal del cerebro. Los métodos de aprendizaje simplemente son los métodos de entrenamiento del modelo para obtener un nuevo valor de peso mediante el optimizador hasta que el modelo puede funcionar con el nivel aceptable.

Se puede clasificar en 3 tipos [11]

- **El aprendizaje supervisado** : es el aprendizaje que alimentan los valores de entrada y los valores de resultados al modelo para que el modelo modifique los valores de pesos y Bias.
- **El aprendizaje no supervisado** : es el aprendizaje que el modelo no puede saber los valores de resultados, pero el modelo se adapta dependiendo de tendencia y comportamiento de los datos.
- **El aprendizaje por refuerzo**: el método de aprendizaje utilizando el premio o la penalización para controlar la dirección de aprendizaje del modelo

2.3 Las redes neuronales convolucionales

2.3.1 Definición y su evolución

Las redes neuronales convolucionales son uno de los tipos de redes neuronales el cual tienen la función similar a la Corteza visual del cerebro de un mamífero [12].

En 1968, David Hubel y Torsten Weisel publicaron su investigación titulada "Campos receptivos de neuronas individuales en la corteza estriada del gato" [13].Colocaron los electrodos en la zona de la corteza visual primaria del cerebro del gato. Descubrieron el procesamiento de la corteza visual después de su observación. Los resultados de la investigación inspiraron el desarrollo de las redes neuronales convencionales.

En 1980, el Sr. Kunihiko Fukushima publicó su estudio sobre una red neuronal llamada Neocognitrón [14] que se inspiraba en las funciones de la corteza visual. Este estudio es uno de los más interesantes que utiliza las técnicas similares a las redes neuronales actuales.

De 2010 ha tenido lugar el concurso anual ImageNet Large Scale Visual Recogmition Challenge (ILSVCR) [15] de las redes neuronales para clasificar y reconocer las imágenes de la base de datos ImageNet. La arquitectura AlexNet [16] desarrollada por Alex Krizhevsky ganó el concurso del 2012 [15] utilizando la red neural convolucional profunda. El modelo sólo obtuvo un 26,2% de tasa de errores Top-5 [15], lo que consideró la gran mejora en comparación con las arquitecturas en ese momento. Desde entonces, se han desarrollado muchas arquitecturas en la investigación e industria, por ejemplo, ResNet [17], Inception [18], SENet [19], etc. que se basan en la red neural convolucional profunda.

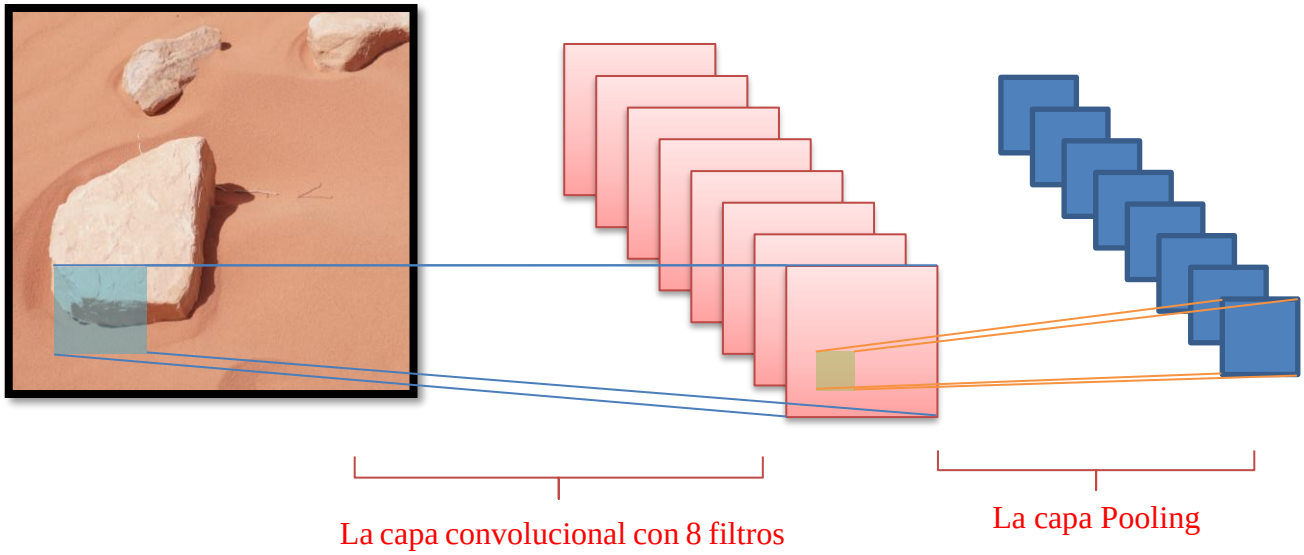


Figura 2-4 Estructura clásica de las redes neuronales convolucionales. (Fuente: Elaboración propia)

2.3.2 Los elementos básicos de una estructura de las redes convolucionales

Las imágenes son vectores multidimensionales, por lo que se supone que las redes neuronales convolucionales reducen el número de parámetros y adaptan la arquitectura de la red a las tareas de visión. Una red neuronal convolucional normalmente construida por muchas capas con una tarea específica. A continuación, se describen las típicas capas que se componen una red convolucional**.**

- **La capa convolucional**
  La capa convolucional se utiliza para reemplazar para reducir el gran número de parámetros que un perceptrón multicapa. Existe un filtro o núcleo (Kernel) que se coloca sobre algunos de los píxeles de la imagen de entrada dependiendo de las dimensiones del tamaño del núcleo. El núcleo va a hacer una operación matemática llamada “Convolución” con los píxeles de entrada para extraer y crear un mapa de características.

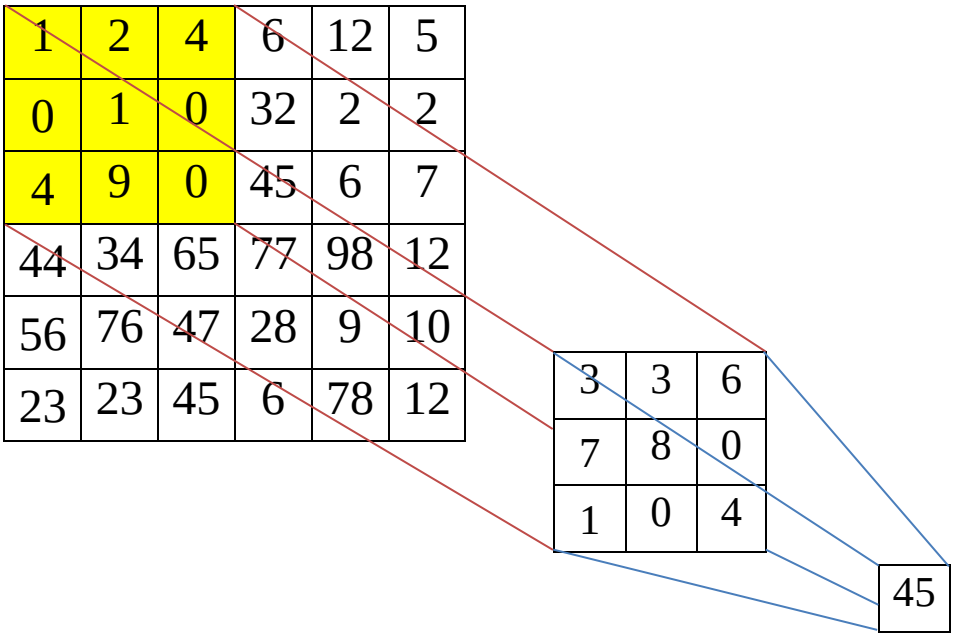


Figura 2-5 La función de capa convolucional utilizando un filtro. (Fuente: Elaboración propia)

Existe además las configuraciones importantes como Stride y Padding. Stride es el número de paso que usa un núcleo para deslizarse y Padding es el marco que se añade a la imagen de entrada para hacer la imagen más grande con un valor "0". La razón de usar el Padding es alterar la dimensionalidad de la salida de las capas convolucionales.

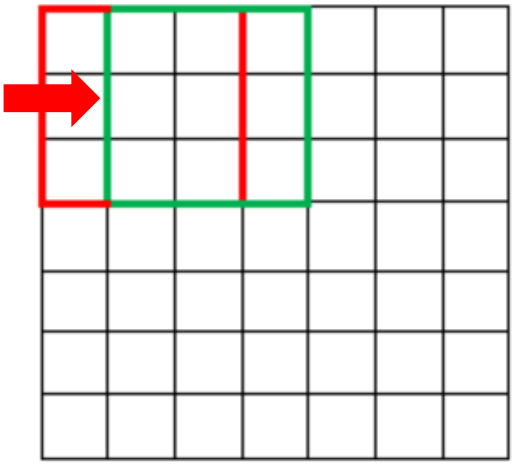


Figura 2-6 La función de un Stride. (Fuente: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/)

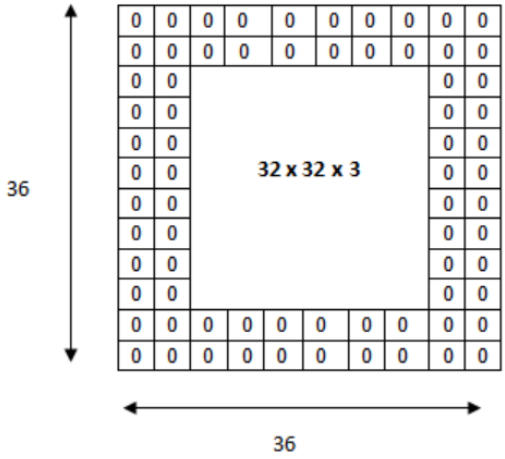


Figura 2-7 La función de Padding. (Fuente: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/)

La siguiente ecuación es la fórmula sobre las dimensiones de entrada, hiperparámetros y dimensiones de salida:

$$n_{out} = \frac{n_{in} - k + 2p}{s} \tag{2.3.2}$$

- n_{in} es dimensión de etrada (altura o anchura)
- n_{out} es dimensión de salida (altura o anchura)
- k es dimensiones de un filtro
- p es dimensiones de Padding
- s es dimensiones de Stride

- **La capa Pooling**
  Tal y como se dijo, la capa convolucional proporciona el Mapa de características. Así que la función de la capa Pooling es aplicar una reducción de dimensiones en el "mapa de caracteristicas" de entrada. Así se reduce aún más el número de parámetros y la complejidad computacional del modelo manteniendo la profundidad [20]. Existe varios tipos de la capa Pooling. Los ejemplos son MaxPooling, la que es utiliza para este trabajo fin de grado, MinPooling y AveragePooling.

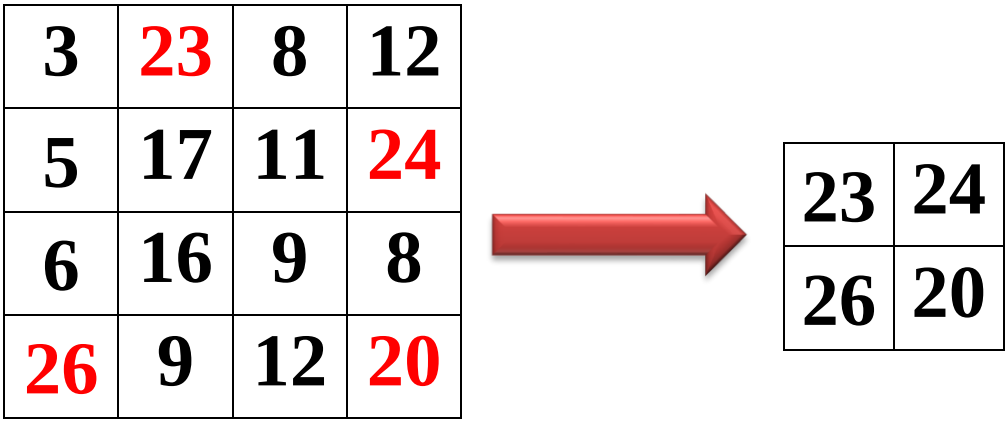


Figura 2-8 La operación de la capa Max Pooling. (Fuente: Elaboración propia)

- **Las capas Fully-connected layers**
  Son las últimas capas que se encuentra en las redes neuronales. La primera capa que se encuentra suele ser la capa de aplanamiento (Flatten) que tiene función para ordenar las dimensiones de los datos de entrada antes de alimentar a la siguiente capa. Las siguientes capas son capas ocultas de la arquitectura similar a perceptrón multicapa hasta la última capa de salida final. En este trabajo fin de grado se utiliza "Softmax" como última capa.

2.3.3 los elementos de adición utilizados en este trabajo fin de grado

- **ReLu(Rectified Linear Unit)**
  ReLu es una función lineal utilizada en redes neuronales como función de activación. ReLu ha sido diseñada como una simple función lineal que puede dar un valor de salida de 0 si el valor de entrada es menor o igual que 0 y si el valor de entrada es mayor que 0, la función siempre da el valor positive como muestra en Figura 2-9.

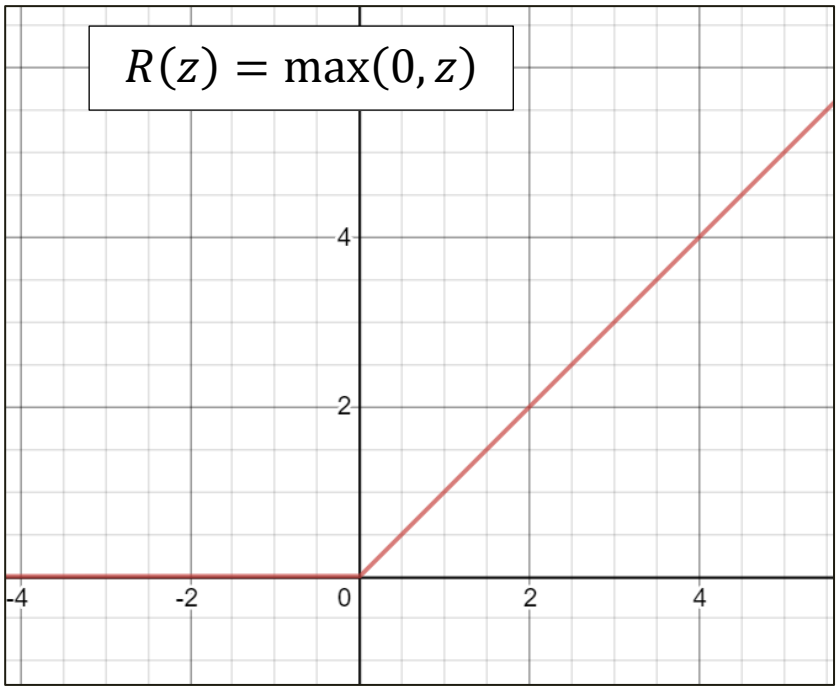


Figura 2-9 Función ReLu. (Fuente: Elaboración propia)

ReLu es muy utilizada con las ventajas que la red pueda reducir la complejidad computacional y optimizar los resultados en general [21] [22] [23].

- **Softmax**
  Softmax es una capa de función de activación que suele colocarse en la última etapa de las redes convolucionales de clasificación sobre todo las redes que clasifican las imagenes de tipo multi clase [24]**.** La capa Softmax asigna probabilidades decimales a cada clase en un caso de clases múltiples. Esas probabilidades decimales deben sumar a 1 [24].

- **La función de Pérdida(Loss function) – Categorical Cross Entropy Loss** [25]
  La función de pérdida es una función matemática usada para mapear los múltiples valores de eventos de entrada al único número real. La función de pérdida se utiliza en las redes neuronales artificiales para evaluar el rendimiento de la red comparando los valores de salida predichos por la red con los valores reales que la red debería haber predicho. En este TFG, se ha seleccionado la función de Pérdida de Entropía Cruzada (Cross Entropy Loss) para el clasificador que es popular entre las tareas de clasificación. La combinación de Softmax y la pérdida de entropía cruzada se denomina pérdida de entropía cruzada categórica o pérdida de Softmax [24].

- **Técnica de Regularización – Dropout** [26]
  La regularización es un conjunto de las técnicas utilizadas para evitar o minimizar los efectos de sobreajuste (Overfitting). El sobreajuste es el fenómeno que el modelo tiene una tasa de error inesperada en los datos de validación debido a que el modelo sólo se ajustará a aprender los casos particulares que le enseñamos y será incapaz de reconocer nuevos datos de entrada. Muchas veces le introducimos los datos con anomalías o ruidos en algunas dimensiones que el modelo llega a aprender. Cuando el modelo se sobreentrena y se le ocurre, el modelo solamente es válido con las imágenes de entrenamiento, pero no es capaz de clasificar las nuevas. Existen varias técnicas como regularización L1/L2 [27], Dropout [26], Cutmix [28], etc. En este tragbajo fin de grado, se ha utilizado la técnica Dropout [26] para regularizar los efectos de sobreajuste. El Dropout [26] se aplica en una capa para quitar las neuronas de

forma aleatoria dependiendo de los porcentajes introducidos. La desaparición de nodos le obliga al modelo cambiar sus iteraciones de predicciones produciendo los efectos de redundancias en la red que se mejora las predicciones del modelo y además se produce los efectos de regularización al quitar los nodos aleatoriamente.

- **La Normalización de datos por lotes(Batch Normalization)** [29]
  La Normalización de datos por lotes o Batch Normalization es una técnica de optimizar el proceso de aprendizaje que se ha elegido para este trabajo fin de grado. Lo que hace la Batch Normalization es normalizar y ordenar los datos con las operaciones matemáticas utilizando el valor de media y la desviación estándar. Además, contiene la función "Shift and Scale" para la adaptación con distintas redes. "Shift and Scale" es la función que contiene dentro de "Batch Normalization" debido a que no todos los modelos pueden trabajar con los resultados optimizados utilizando la normalización de Batch Normalization. Por tanto, se añade los parámetros "Gamma" y "Sesgo" para ajustar los valores de la normalización (**Figura** 2- 2-10). Las ventajas de Batch Normalization son optimización de la tasa de aprendizaje, reducir la complejidad de la red y regularizar.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Figura 2-10 los contenidos de BatchNormalization. (Fuente: [29])

2.3.4 Las arquitecturas de redes neuronales convolucionales utilizadas en este trabajo fin de grado

- AlexNet (2012) [16]
  En este trabajo fin de grado, se ha utilizado la arquitectura AlexNet (2012) en varias etapas con la adaptación propia. AlexNet es una de las clásicas arquitecturas de redes neuronales convolucionales que actualmente aún es popular entre las varias arquitecturas. Este modelo ganó el concurso de ImageNet (ILSVRC) en 2012 con la tasa de errores TOP-1 de 0.265 [15]. Fue el primer ganador utilizando la red neuronal convolucional profundo en este concurso. Esta arquitectura fue diseñada por Alex Krishevsy. En este estudio, el diseño es simple comparado con los modernos modelos. Utilizaron 5 capas convolucionales, 5 capas Pooling y 3 capas completamente conectadas(Fully connected). Además utilizaron ReLu como función de activación, Dropout para regularizar el modelo y fue entrenado con las imágenes de base de datos ImageNet.

- VGG (2014) [30]
  Además, se ha utilizado la arquitectura VGG (2014) en varias etapas con la adaptación propia. VGG fue creado en 2014 por Karen Simonyan y Andrew Zisserman de la Universidad de Oxford.También participó el concurso ILSVRC con la mejor tasa de error de 7.3%(aunque el ganador de ese año fue GoogLeNet de Google [18] con la tasa de error de 6.7% [15]). El diseño fue simple utilizado 19 capas convolucional con los filtros de tamaño 3x3, Stride y Padding de 1 junto con las capas MaxPooling de 2x2 de tamaño con Stride de 2.

2.3.5 Los optimizadores

Los optimizadores son los protagonistas en el proceso de aprendizaje. Un optimizador tiene función de optimizar los valores de peso y/o sesgo y devolver al modelo en cada ronda de aprendizaje para reducir el valor de pérdida salido de la función de pérdida (Loss Function). En este trabajo fin de grado, se ha utilizado el optimizador ADAM [31]. El optimizador ADAM es un optimizador utilizado lo bueno de los optimizadores "RMSprop" [32] y "El optimizador de Momentos" [32]. Es capaz de modificar la tasa de aprendizaje y acumular los momentos de las iteraciones anteriores para minimizar los errores en el descenso de gradiente.

2.3.6 Los procesos de aprendizaje utilizado en este trabajo fin de grado

Las técnicas aplicadas en este trabajo fin de grado han sido la retro propagación y el descenso de gradiente [33]. Los modelos de la CNN van a predecir las etiquetas a partir de las imágenes. En el proceso de entrenamiento, la primera vez que el modelo tiene que predecir la imagen, el modelo va a realizar la predicción utilizando pesos y bias aleatorios. Los resultados predichos se comparan con los resultados reales mediante la función de pérdida (Loss function). El descenso del gradiente es la técnica que se utiliza para encontrar el valor mínimo absoluto de la función de pérdida en la cual posee una gran cantidad de parámetros, intentando descender para encontrar el punto más bajo de la función de pérdida. En este proceso los optimizadores son los principales protagonistas. Los optimizadores se utilizan para corregir los pesos y/o Sesgo y enviar los nuevos valores a los parámetros mediante la técnica de retropropagación [34]. Los parámetros guardan los nuevos valores de peso y/o Sesgo, predice los resultados de nuevo y el proceso de entrenamiento va a repetir el mismo proceso, intentando minimizar el valor de pérdida (Loss), mostrando en Figura 2-11 y Figura 2-12.

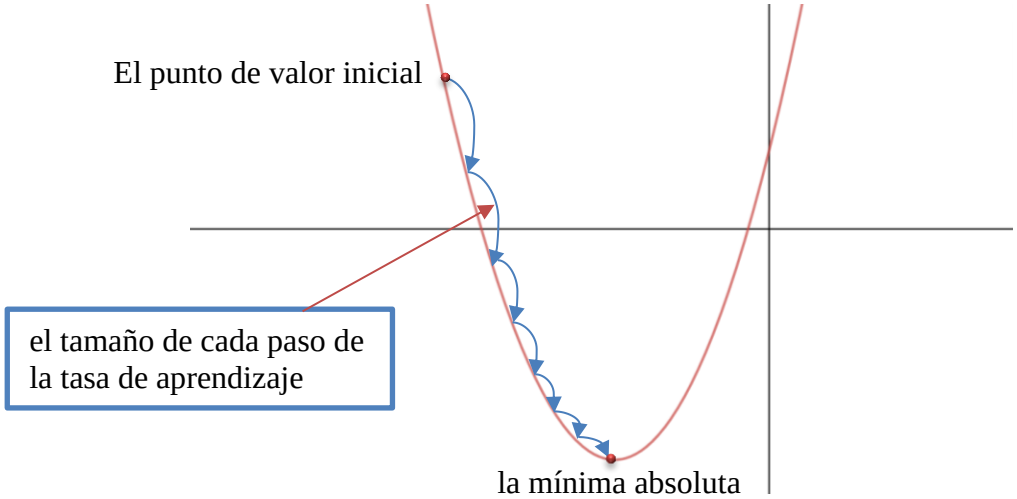


Figura 2-11 El descenso de gradiente de una función para encontrar la mínima absoluta. (Fuente: Elaboración propia)

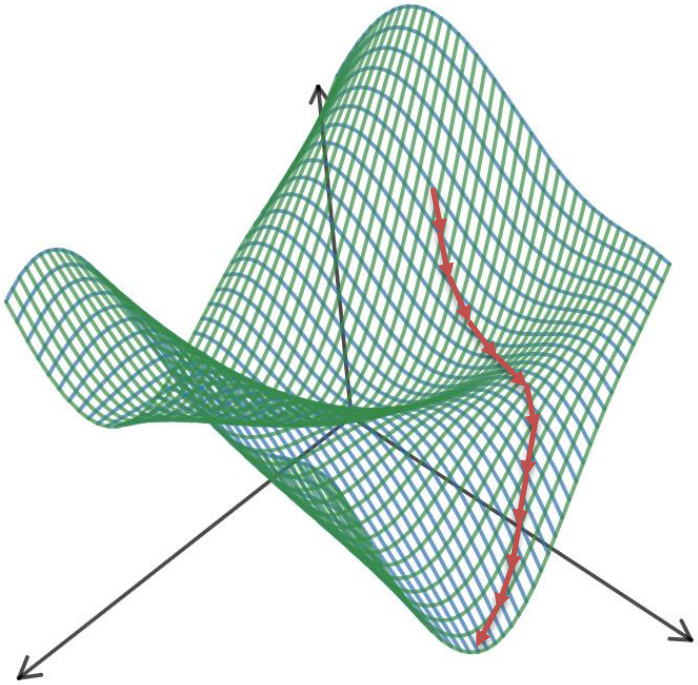


Figura 2-12 La metodología del descenso de gradiente de una función de varias variables. (Fuente: Elaboración Propia)

2.4 Google Colaboratory

Google Colaboratory o en el nombre más corto “Colab” es un terminal online que permite a los usuarios que tienen una cuenta de Google acceder para codificar y ejecutar los códigos de forma gratuita. Google Colab se basa en Jupyter Notebook, que se ejecuta en la nube de los servidores de Google [35], lo que ofrece a los usuarios muchos recursos de buena calidad sin necesidad de configuración. Además, con Google Colab los usuarios pueden almacenar sus archivos en Google Drive y compartirlos con otros usuarios. Google Colab puede ofrecer CPU, GPU y TPU a los usuarios que pueden elegir el procesador para sus tareas específicas. El CPU es Intel(R) Xeon(R) CPU @ 2.30GHz y el GPU es Tesla K80 de NVIDIA. El TPU de Colab es el TPU del propio desarrollo de Google que acelera el entrenamiento de los modelos de aprendizaje automático [36] [37].

Los usuarios pueden acceder al uso de Colab hasta 12 horas de procesamiento seguidas con limitación que los recursos disponibles en el Colab varían a lo largo del tiempo para adaptarse a las fluctuaciones de la demanda por tanto Google Colab no puede garantizar la disponibilidad de recursos para los usuarios [38].

En este trabajo fin de grado, Se considera útil emplear el Google Colab para elaborar el proyecto debido a sus recursos como el TPU de Google, el acceso fácil al servidor de la plataforma Google Cloud Computing, el Tensorflow preinstalado con sus bibliotecas y el precio gratis.

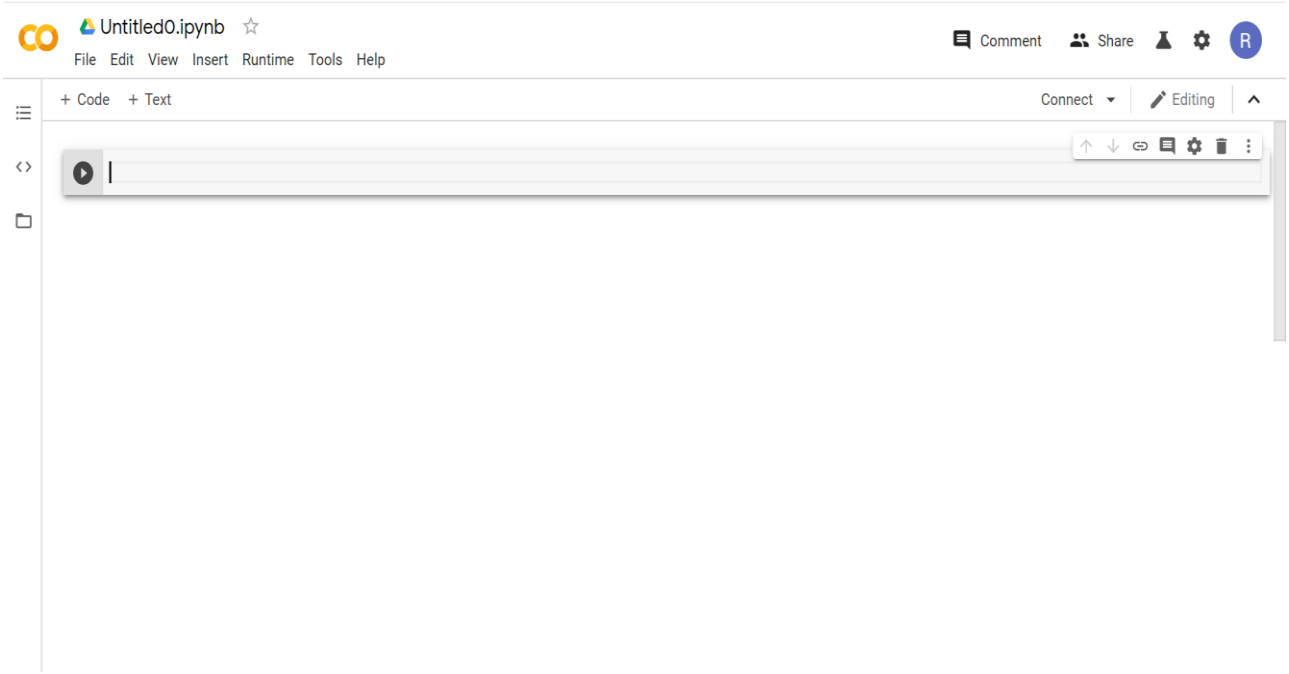


Figura 2-13 El terminal de Google Colab en el navegador Google Chrome. (Fuente: Elaboración propia)

2.5 Lenguaje de programación “Python” [39]

El lenguaje de programación elegido en este trabajo fin de grado ha sido el lenguaje “Python”. La idea del desarrollo de Python fue creada en 1989 por el Sr. Guido Van Rossum en el Stichting Mathematisch Centrum en Holanda procedente del lenguaje ABC. En 1991, la primera versión de Python (Python 0.9.0) fue publicado en USENET. En 1994, la versión 1.0.0 de Python fue lanzado con las aplicaciones como MapReduce, Lambda y Filter.

Python es un lenguaje de programación de alto nivel, interpretado y orientado a objetos, con semántica dinámica. Python es un lenguaje muy popular utilizado en varias tareas como Inteligencia artificial, Ciencia de datos (Data Science), Web. Actualmente, el uso de Python es muy extendido por la interfaz que facilita la programación, la rapidez de ejecución y la integración con otros sistemas fácilmente [40].

Figura 2-14 Logo de Python (Fuente: www.phython.org)

2.6 Tensorflow y las bibliotecas utilizadas en este trabajo fin de grado

Tensorflow [41] es una plataforma de código abierto de extremo a extremo para el aprendizaje automático creada por Google. Tensorflow puede proporcionar flexibilidad y bibliotecas completas para los usuarios que facilitan el desarrollo del proyecto de aprendizaje automático. Tensorflow está probado y apoyado en códigos Python 2 y Python 3. También Tensorflow puede funcionar en muchos sistemas operativos como Windows, Linux, MacOs, Android. Google también ha desarrollado la plataforma Tensorflow para dispositivos móviles. Tensorflow está preinstalada en Google Colab que facilita a los usuarios que tienen cuenta de Google [42].

En este trabajo fin de grado se ha utilizado las bibliotecas basadas del lenguaje Python que Tensorflow ofrece. Las bibliotecas utilizadas son

- **Keras** [43]
  Keras es la interfaz de programación de aplicación de programación(API) basada en Python utilizada en las capas superiores de Tensorflow, CNTK o Theano.Keras ha sido desarrollado para facilitar el proyecto enfocado en Redes Neuronales.

- **Numpy** [44]
  Numpy es una biblioteca basada en el lenguaje Python utilizada para facilitar la programación de los procesos matemáticos en los proyectos científicos. En este trabajo fin de grado, se ha utilizado Numpy para facilitar los cálculos de vectores y tensores.

- **Pandas** [45]
  Pandas es una biblioteca de multiusos basada en el lenguaje Python principalmente se aplica en los proyectos de datos, por ejemplo, la limpieza de datos, visualizar los datos, crear los modelos etc. En este trabajo fin de grado, se ha utilizado Pandas para visualizar los datos de entrada y sus etiquetas.

- **Matplotlib** [46]
  Matplotlib es una biblioteca basada en el lenguaje Python utilizada para generar los datos de forma gráfica como histogramas, diagramas, esquemas etc. En este trabajo fin de grado, se ha utilizado Matplotlib para visualizar las gráficas de entrenamiento de los modelos.

- **Scikit learn** [47]
  Scikit learn es API basada en el lenguaje Python que ofrece a usuarios varias herramientas para los proyectos de aprendizaje de máquina extremo a extremo. En este trabajo fin de grado, se ha empleado las métricas de evaluación y la herramienta de distribución de datos como "train_test_split".
- **Seaborn** [48]
  Seaborn es una biblioteca instalada en la capa superior de Matplotlib para crear las gráficas de estadisticas. En este trabajo fin de grado, se ha utilizado Seaborn para visualizar las cantidades de dataset y crear los mapas de calor de las matrices de confusión.

2.7 TPU (Tensor Processing Unit)

Las unidades de procesamiento tensorial (TPU) [49] son los circuitos integrados personalizados específicos de aplicación (ASIC) que se utilizan para acelerar los procesamientos del aprendizaje automático. El TPU puede ofrecer rendimiento de pico hasta 92 teraoperaciones/segundo (TOPS) [50]. La TPU se ha creado con las arquitecturas especificas propias de Google que es diseño de procesamiento matricial especializado de las cargas del aprendizaje profundo. Dentro de las redes neuronales, existen gran cantidad de multiplicaciones matriciales, por ejemplo, las multiplicaciones en las capas convolucionales de las regiones locales extraídas sobre las imágenes que los datos dentro de las redes son almacenados de forma tensorial (matriz de orden alta). Una TPU puede acelerar el procesamiento del modelo y el ahorro de energía mejor que el GPU que ofrece el Google Colab y algunos CPU utilizados del nivel de servidores.

Según la página web oficial de Google Cloud Platform (GCP) [49], los usos de TPU no son adecuados para algunas ejecuciones como los siguientes

- Programas de álgebra lineal que requieren ramificaciones frecuentes o dominados por el álgebra a nivel de los elementos
- Las cargas de trabajo que acceden a la memoria de manera dispersa
- Las cargas de trabajo que requieren aritmética de alta precisión. Por ejemplo, la aritmética de doble precisión no es adecuada para las TPU.
- Cargas de trabajo de redes neuronales que contienen operaciones personalizadas de Tensorflow escritas en C++. De manera más específica, las operaciones personalizadas en el cuerpo del ciclo de entrenamiento principal no son adecuadas para las TPU.

2.8 Técnicas de pre-procesado aplicadas de reducción de efectos del dataset desequilibrado y las imágenes insuficientes

La utilización del dataset desequilibrado puede reducir la capacidad de entrenamiento de los modelos debido de la falta de las imágenes de las clases minoritarias que el modelo no puede aprender suficientemente de ellas causando que los valores de pesos no sean optimizados como en las clases mayoritarias [51], por tanto, Se considera importante emplear las técnicas para reducir los posibles efectos negativos del uso del dataset desequilibrado.

En este Trabajo Fin de Grado, se ha utilizado las técnicas para minimizar los efectos de los datos desequilibrados como:

- **El aumento de datos (Data Augmentation)**
  Los datos insuficientes pueden causar algunos efectos que reduzcan la capacidad del modelo por la falta de datos para aprender. Por tanto, las predicciones de resultado no pueden alcanzar los resultados optimizados. El aumento de datos es la técnica que se aplica sobre los datos que se consideran insuficiente para las validación y pruebas de las redes neuronales. Consiste en aumentar la cantidad de imágenes generando las imágenes existentes volteando, girando, extendiendo, añadiendo ruidos, etc como en la Figura 2-15.

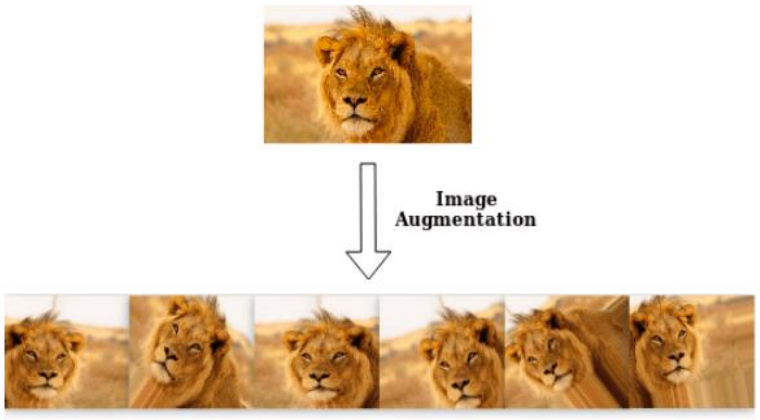


Figura 2-15 **El proceso de Data Aucmentation. (Fuente: https://towardsdatascience.com/machinex-image-data-augmentation-using-keras-b459ef87cd22)**

- **Sobre-muestreo aleatorio (Random Oversampling)**
  El dataset desequilibrado puede reducir la efectividad de entrenar de las redes neuronales [52] [53]. Se aplica la técnica efectiva como Sobre muestreo aleatorio en las clases minoritarias de un dataset desequilibrado para generar los datos intentando equilibrar la cantidad de datos entre las clases.

- **Sub-muestreo aleatorio (Random Undersampling)**
  Se aplica la técnica efectiva Submuestreo aleatorio [52] [53] en las clases mayoritarias de un dataset insuficiente para quitar los datos intentando equilibrar la cantidad de datos entre las clases.

2.9 Métricas utilizadas para la evaluación del modelo de redes neuronales

2.9.1 Matriz de confusión (Confusion Matrix)

Matriz de confusión es la tabla que permite visualizar el rendimiento del modelo comparando los resultados del modelo y los datos reales. Cada columna de la matriz muestra los valores reales mientras las filas muestran los valores de predicción del modelo. Los valores de la diagonal son los que cuando la predicción del modelo coincide con los reales.

		realidad	
		Positivo	Negativo
predicciones	Positivo	Verdadero Positivo	Falso Positivo
	Negativo	Falso Negativo	Verdadero Negativo

Figura 2-16 Matriz de confusión de una clasificación binaria.
(Fuente: Elaboración propia)

Verdadera positiva (VP): La predicción del modelo es “verdadera” y el valor real es “verdadera”

Verdadera negativa (VN): La predicción del modelo es “Falsa” y el valor real es “Falso”

Falsa positiva (FP): La predicción del modelo es “Verdadero” y el valor real es “Falso”

Falsa negativa (FN): La predicción del modelo es “Falsa” y el valor real es “Verdadero”

Las matrices de confusión que se ha empleado en este trabajo fin de grado son la matriz de confusión de multiclase que se clasifica por 15 etiquetas reales y otras 15 etiquetas de predicción(Figura 2-17) y la matriz de confusión de multiclase que se clasifica por 3 etiquetas reales y otras 3 etiquetas de predicción(Figura 2-18).

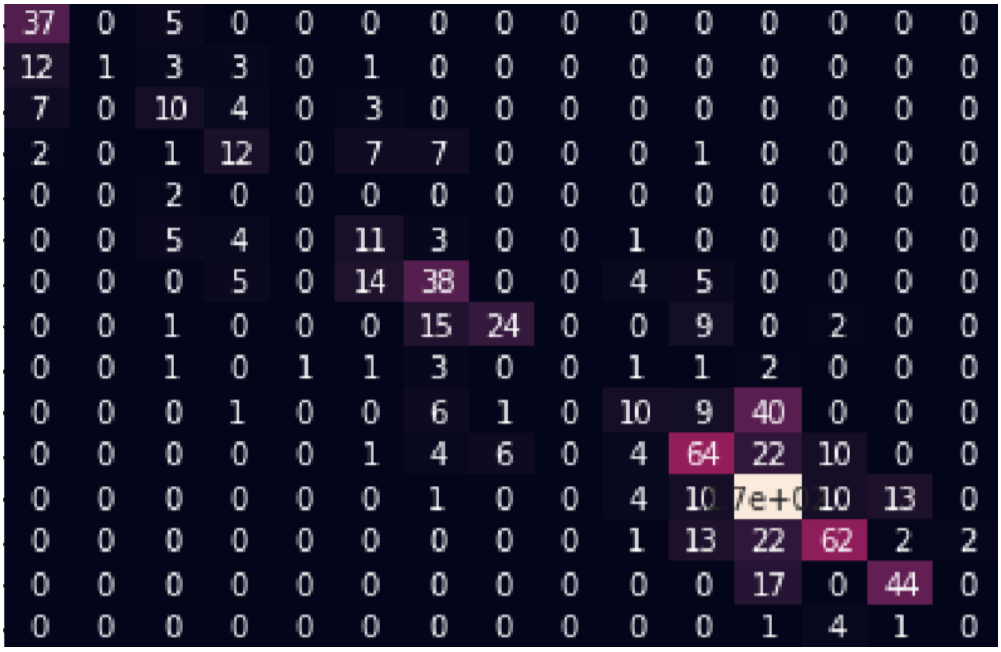


Figura 2-17 Una matriz de confusión de 15 clases utilizada en este TFG. (Fuente: Elaboración propia)

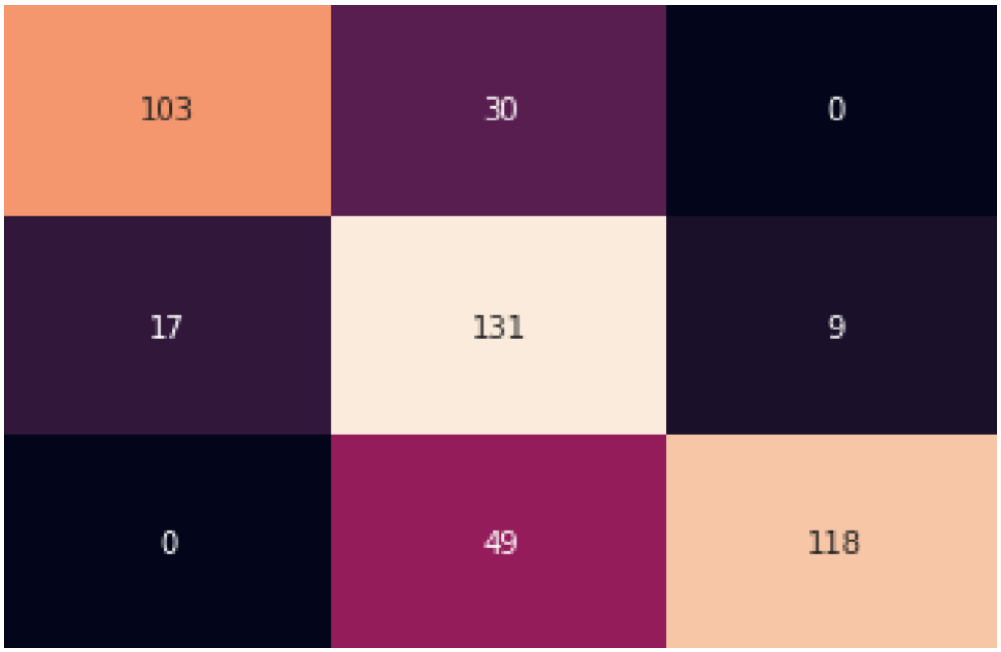


Figura 2-18 Una matriz de confusión de 3 clases utilizada en este TFG. (Fuente: Elaboración propia)

2.9.2 Exactitud (Accuraycy)

La fracción o número de las predicciones correctas entre todas las predicciones

$$\text{Exactitud} = \frac{(VP + VN)}{VP + VN + FP + FN} \tag{1.9.2}$$

2.9.3 Precisión (Precision)

Intuitivamente la capacidad del clasificador para etiquetar correctamente una muestra positiva

$$\text{Precisión} = \frac{VP}{VP + FP} \tag{2.9.3}$$

2.9.4 Exhaustividad (Recall)

Intuitivamente la capacidad del clasificador para encontrar todas las muestras positivas.

$$\text{Exhaustividad} = \frac{VP}{VP + FN} \tag{2.9.2}$$

2.9.5 Valor F-beta (F-β)

Puede ser interpretado como una media armónica ponderada de la Precisión y el Recall. Una medida de F-beta alcanza su mejor valor en 1 y la peor puntuación en 0. En este trabajo fin de grado utilizamos F-1 que significa el valor de Beta es 1 así que no se introduce más peso a ni la Precisión ni el Exhaustividad.

$$F_\beta = (1 + \beta^2)\frac{\text{Precisión} \times \text{Exhaustividad}}{\beta^2\text{Precisión} + \text{Exhaustividad}} \tag{2.9.3}$$

La fórmula de F-1 score será

$$F1 = 2 \times \frac{\text{Precisión} \times \text{Exhautividad}}{\text{Precisión} + \text{Exautividad}} \tag{2.9.5}$$

2.9.6 TOP-n accuracy score

El significado de TOP-N accuracy es el acierto cuando la clase correcta está en las n probabilidades de las predicciones del modelo. En este trabajo fin de grado, se utiliza TOP-1 y TOP-5 como las métricas para evaluar el modelo.

Se ha utilizado la biblioteca Sci-kit Learn [47] para emplear las métricas [54] para evaluar el modelo de las clasificaciones de multiclase. Hay que indicar el tipo de ponderación para combinar los resultados entre las clases. Las opciones para nuestro caso son.

- **weighted**: se ejecuta teniendo en cuenta el desequilibrio de las clases calculando el promedio de las métricas multiplicando las métricas por los pesos de cada clase
- **macro:** se calcula el promedio de las métricas sin tener en cuenta los pesos de cada clase
- **micro**: se calcula de forma global similar a las métricas de clasificación binaria.

En este trabajo fin de grado, se ha empleado “weighted” para los dataset desequilibrados y “macro” para los dataset equilibrados. No se ha empleado “micro” por un problema de valores idénticos obtenidos en las pruebas de este trabajo.

3 DESARROLLO DEL TFG

3.1 Pre-procesado

Se creó el dataset de imágenes de un dispositivo “Sky Scanner” que se acumuló desde La Escuela Politécnica Superior de la Universidad de Burgos durante todo el año 2017. El dataset que se recibió del proveedor tiene un gran problema de igualdad o equilibrio del tamaño las 15 clases. Antes de la realización de la etapa Pre-procesado, se modificó el tamaño de resolución a 400x400 con calidad de 80JPG para acelerar el proceso. Se utilizó el TPU para todas las etapas y la capacidad de memoria RAM extendida de 35,35 Gb.

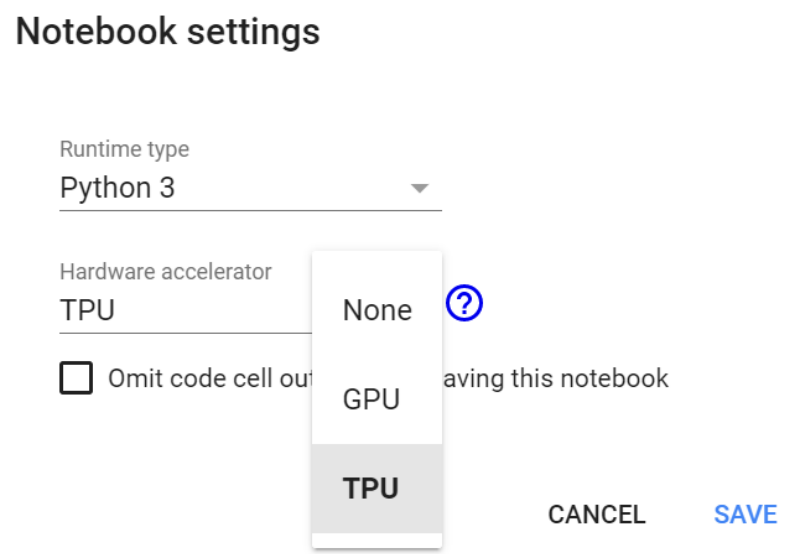


Figura 3-1 Las opciones de procesadores de Google Colab. (Fuente : Elaboración propia)

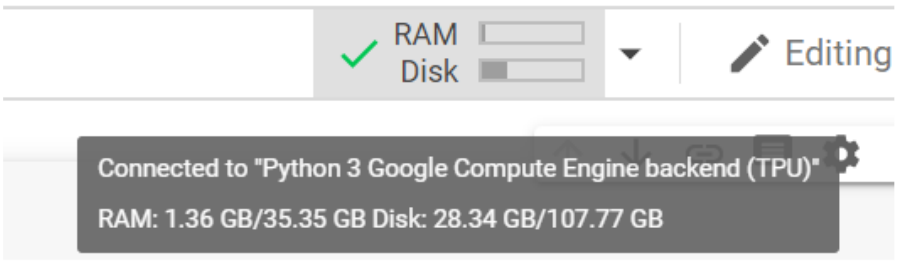


Figura 3-2 el bloque que indica la capacidad de RAM del procesador elegido. (Fuente: Elaboración propia)

En este trabajo fin de grado, Se ha utilizado Tensorflow 2.1.0

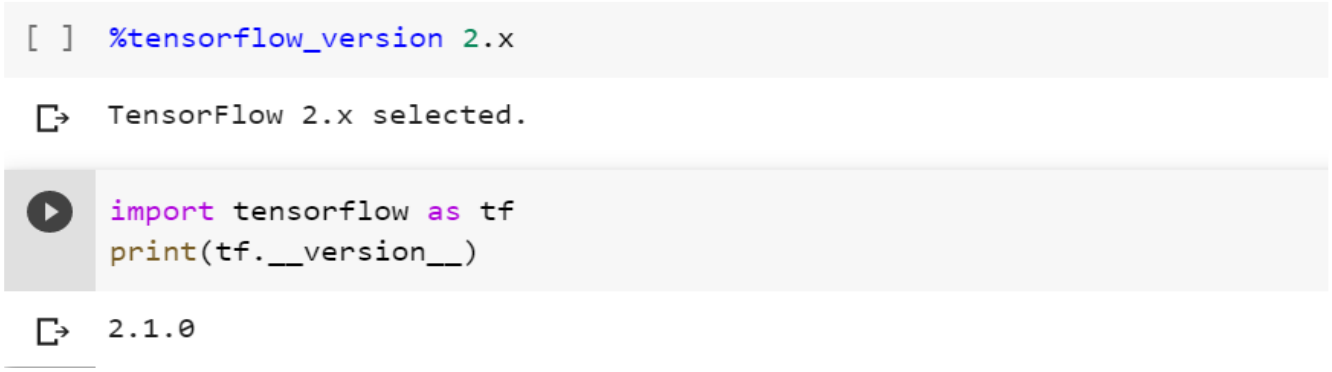


Figura 3-3 El empleo de Tensorflow 2.1.0 en el Google Colab. (Fuente: Elaboración propia)

Se ha empleado las bibliotecas como Keras, Numpy, Pandas, Matplotlib, Seaborn, Sklearn. De la biblioteca Keras, se ha importado las herramientas para el pre-procesasdo como “Preprocessing.image”. Se ha importado las herramientas para crear las redes neuronales convolucionales como “Sequential, Dense, Flatten, Conv2D, MaxPool2D, BatchNormalization, Dropout”. Se ha importado el optimizador Adam De la biblioteca Sklearn, se ha importado las herramientas como “train_test_split” y las metricas como “accuracy_score, precision_score, f1_score, recall_score, confusion_matrix”.

Se ha utilizado el archivo CSV que indica las clases de las imágenes de forma “One-hot encoded”. El ejemplo del significado de “One-hot encoded” es si una imágen está en clase 10 de las 15 clases aparece el número 1 en la matriz de tamaño [1x10] en la posición 10 como se muestra en la figura

```
clase 10 de 15 clases =>  [0,0,0,0,0,0,0,0,0,1,0,0,0,0,0]
```

Figura 3-4 El ejemplo de One-hot encode. (Fuente: Elaboración propia)

3.1.1 El dataset desequilibrado

Se puede ver la cantidad de las imágenes del dataset es 8266 imágenes en total y además se puede visulizar los nombres de imágenes con la indicación de clase en forma One-hot encoded.

```
[ ] datasets = pd.read_csv('/content/CIE Class_Original to use.csv')
```

```
datasets.head()
```

	NAME	CIEclass	CIE1	CIE2	CIE3	CIE4	CIE5	CIE6	CIE7	CIE8	CIE9	CIE10	CIE11	CIE12	CIE13	CIE14	CIE15
0	CIE_10_20170101_113000	10	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
1	CIE_10_20170101_123000	10	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
2	CIE_10_20170102_103000	10	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
3	CIE_10_20170105_113000	10	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
4	CIE_10_20170105_114500	10	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0

Figura 3-5 Los códigos empleados y los resultados para visualizar el archivo CSV. (Fuente: Elaboración propia)

Se puede visualizar la cantidad de las imágenes del dataset por clases. Se puede ver lo desequilibrado del dataset. La clase con más imágenes es la clase 12 y la clase con menos imágenes es la clase 5.

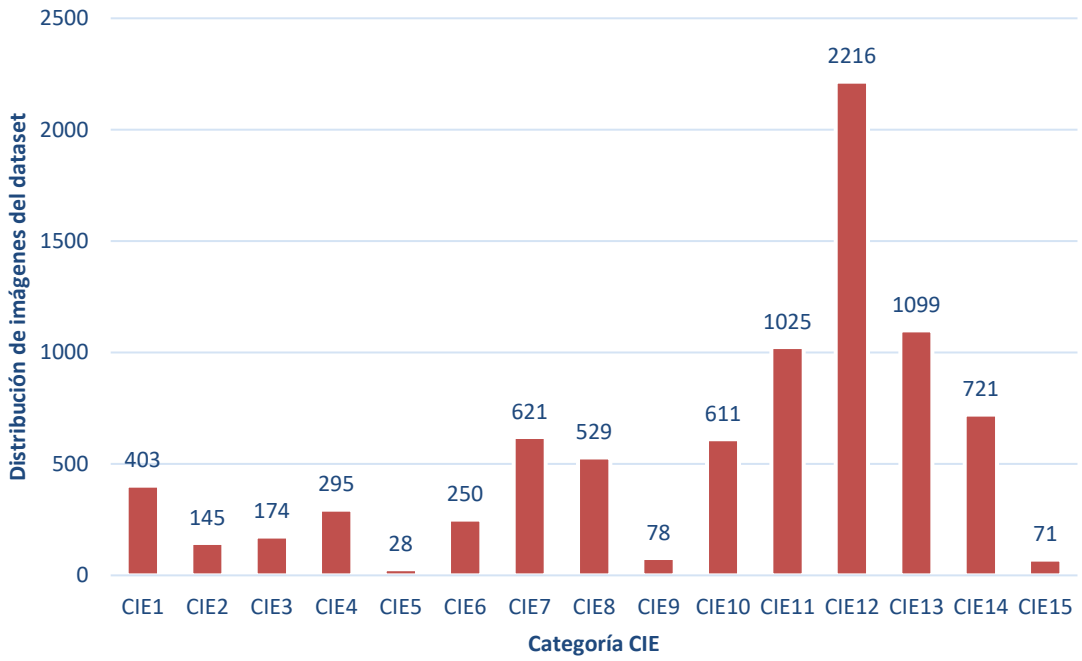


Figura 3-6 La distribución de las imágenes del dataset desequilibrado por la clase. (Fuente: Elaboración propia)

La siguiente etapa, se ha transformado las imágenes a la matriz de números de colores debido a que la visión computadora no es igual a los ojos humanos. En este trabajo fin de grado, se ha utilizado la escala grises en todas las etapas para reducir la consumición de RAM del Google Colab que se considera suficiente para procesar el modelo. Se necesita reducir los parámetros dividiendo los números de colores por 255 que los números de colores de los pixeles puede ser entre 0 a 255 para reducir el consumo de los recursos de Google Colab.

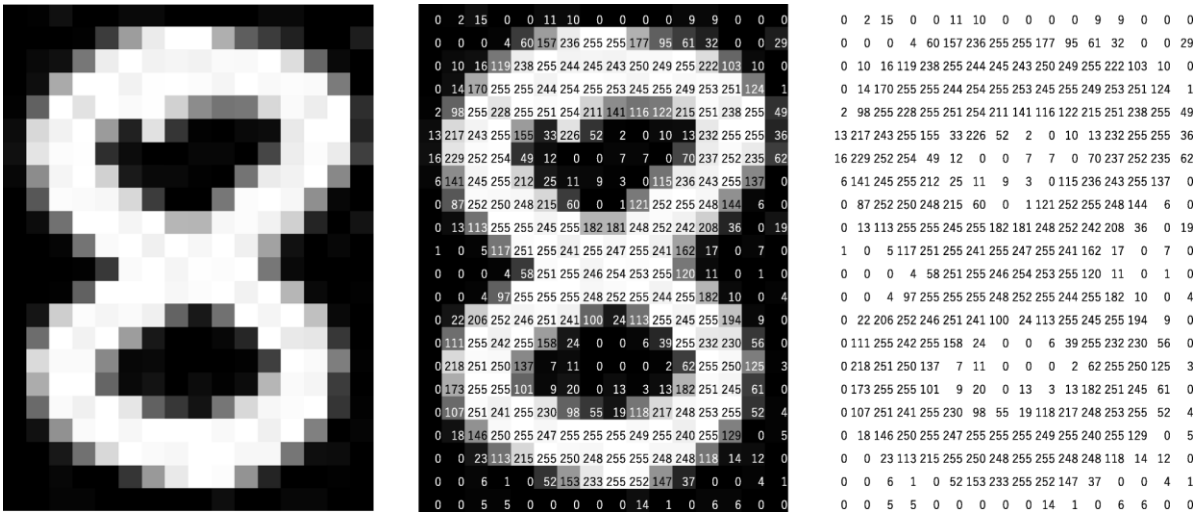


Figura 3-7 **La transformación de las imágenes a los números de colores de cada pixeles. (Fuente : https://mozanunal.com/2019/11/img2sh/)**

Se necesita la preparación de las etiquetas de las imágenes antes de alimentar al modelo. En este caso se nombra el conjunto de las etiquetas como “Y”

```
Y = datasets.drop(['NAME','CIEclass'],axis=1)
Y = Y.to_numpy()

Y.shape
```

Figura 3-8 La preparación de de las etiquetas de imágenes. (Fuente: Elaboración propia)

En las pruebas utilizadas el dataset desequilibrado, utilizamos la proporción de 80% del dataset para el entrenamiento, 10% para la validación del entrenamiento en cada epochs y otros 10% para la prueba del modelo final. Distribuimos las imágenes considerando que cada conjunto de los tres tiene que ser el mismo siempre para todos los epochs de entrenamiento y validación así que las de la prueba final tiene que ser el conjunto que no ha visto el modelo. Se considera importante barajar los datos antes de distribuir las imágenes.

3.1.2 El dataset equilibrado utilizando El aumento de datos, Sobre muestro aleatorio y Submuestreo aleatorio

Existen varios estudios que nos muestran los efectos negativos del uso de los datasets desequilibrados que pueden reducir el rendimiento de aprendizaje del modelo y además no dejan el uso de las métricas funcionen adecuadamente [53] [55] [51]. En este trabajo fin de grado, se intenta reducir los efectos negativos que pueda ocurrir utilizando las técnicas efectivas para equilibrar el dataset. Se elige las técnicas Aumento de datos, Sobre-muestro aleatorio y Sub-muestro aleatorio para el tratamiento. Antes del pre-procesado las imágenes, Se separa las imágenes de la prueba final del dataset para asegurar que en la prueba final el modelo pueda hacer las predicciones a todas las 15 clases. Luego, se equilibra las clases a igual tamaño de 250 imágenes, utilizando el Aumento de datos y Sobre-muestro con las clases minoritarias y Sub-muestro con las clases mayoritarias. Por tanto, las imágenes en total del conjunto de entrenamiento y validación será 3750 imágenes y del conjunto de pruebas final será 457 imágenes Los preprocesamientos son similares a los de dataset desequilibrado. Entre el conjunto de las imágenes de entrenamiento y validación, se reparte con la proporción de 80% de entrenamiento y 20 % de la validación.

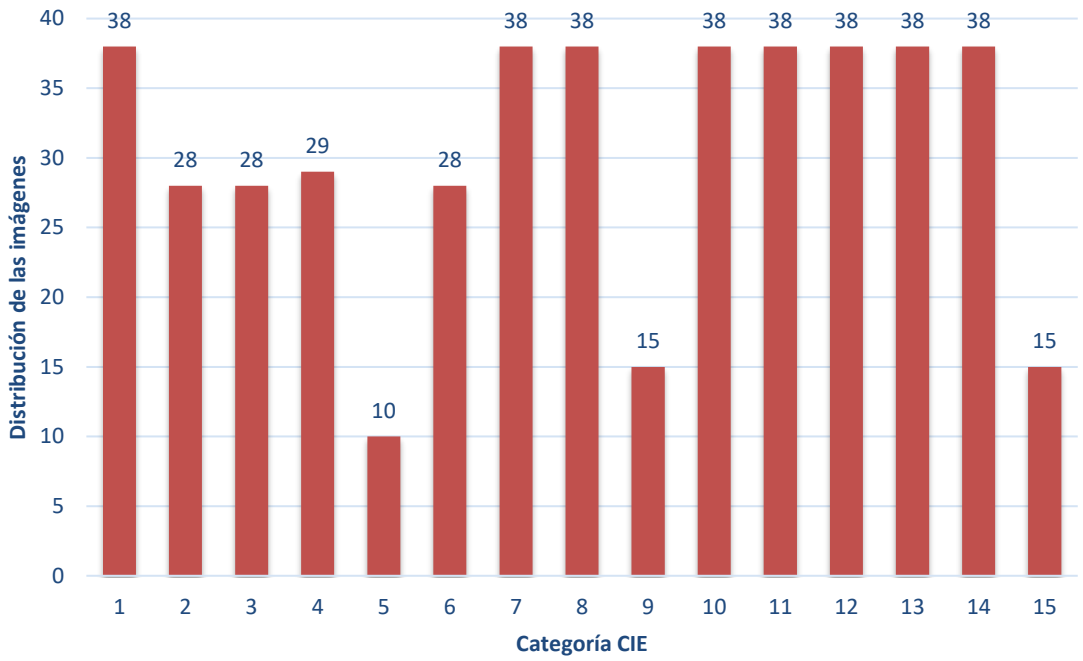


Figura 3-9 La distribución de las imágenes por clase para las pruebas finales del dataset equilibrado. (Fuente: Elaboración propia)

3.2 Diseño del modelo de las arquitecturas AlexNet y VGG con la adaptación propia

El diseño del modelo basado en la arquitectura AlexNet con la propia adaptación consisten en 5 bloques de las capas convolucionales, BatchNormalization, MaxPooling, Dropout y 4 capas completamente conectadas(Fully connected) aplicadas también las BatchNormalization y Dropout terminando con la capa de "Softmax" como la última. la estructura se muestra en los siguientes códigos empleando las bibliotecas

```
CNN = Sequential()
CNN.add(Conv2D(32,kernel_size=(3,3),padding='valid',use_bias=False,activation='relu',input_shape=X_train[0].sh
ape))
CNN.add(BatchNormalization())
CNN.add(MaxPool2D(2,2))
CNN.add(Dropout(0.3))
CNN.add(Conv2D(64,kernel_size=(3,3),use_bias=False,padding='valid',activation='relu'))
CNN.add(BatchNormalization())
CNN.add(MaxPool2D(2,2))
CNN.add(Dropout(0.3))
CNN.add(Conv2D(64,kernel_size=(3,3),use_bias=False,padding='valid',activation='relu'))
CNN.add(BatchNormalization())
CNN.add(MaxPool2D(2,2))
CNN.add(Dropout(0.3))
CNN.add(Conv2D(96,kernel_size=(3,3),padding='valid',use_bias=False,activation='relu'))
CNN.add(BatchNormalization())
CNN.add(MaxPool2D(2,2))
CNN.add(Dropout(0.4))
CNN.add(Conv2D(128,kernel_size=(3,3),use_bias=False,padding='valid',activation='relu'))
CNN.add(BatchNormalization())
CNN.add(MaxPool2D(2,2))
CNN.add(Dropout(0.5))
CNN.add(Flatten())
CNN.add(Dense(120,activation='relu',use_bias=False))
CNN.add(BatchNormalization())
CNN.add(Dropout(0.3))
CNN.add(Dense(120,activation='relu',use_bias=False))
CNN.add(BatchNormalization())
CNN.add(Dropout(0.4))
CNN.add(Dense(80,activation='relu',use_bias=False))
CNN.add(BatchNormalization())
CNN.add(Dropout(0.5))
CNN.add(Dense(15,activation='softmax'))
```

El diseño del modelo basado en la arquitectura VGG con la propia adaptación consisten en 5 bloques de las capas convolucionales que cada bloque contiene 2 capas convolucionales BatchNormalization, MaxPooling, Dropout con la excepción de los 2 primeros bloques que no tienen la capa MaxPooling y 2 capas completamente conectadas(Fully connected) aplicadas también las BatchNormalization y Dropout terminando con la capa de Softmax como la última. la estructura se muestra en los siguientes códigos empleando las bibliotecas. Así que, se aumentan los parámetros a optimizar mucho más que el modelo de AlexNet.

```
CNN = Sequential()
CNN.add(Conv2D(16,kernel_size=(3,3),padding='valid',activation='relu',input_shape=X_train[0].shape))
CNN.add(Conv2D(16,kernel_size=(3,3),padding='valid',use_bias=False,activation='relu'))
CNN.add(BatchNormalization())
CNN.add(Dropout(0.4))
CNN.add(Conv2D(32,kernel_size=(3,3),padding='valid',activation='relu'))
CNN.add(Conv2D(32,kernel_size=(3,3),padding='valid',use_bias=False,activation='relu'))
CNN.add(BatchNormalization())
CNN.add(Dropout(0.4))
CNN.add(Conv2D(32,kernel_size=(3,3),padding='valid',activation='relu'))
CNN.add(Conv2D(32,kernel_size=(3,3),padding='valid',use_bias=False,activation='relu'))
CNN.add(BatchNormalization())
CNN.add(MaxPool2D(2,2))
CNN.add(Dropout(0.5))
CNN.add(Conv2D(64,kernel_size=(3,3),padding='valid',activation='relu'))
CNN.add(Conv2D(64,kernel_size=(3,3),padding='valid',use_bias=False,activation='relu'))
CNN.add(BatchNormalization())
CNN.add(MaxPool2D(2,2))
CNN.add(Dropout(0.5))
CNN.add(Conv2D(128,kernel_size=(3,3),padding='valid',activation='relu'))
CNN.add(Conv2D(128,kernel_size=(3,3),padding='valid',use_bias=False,activation='relu'))
CNN.add(BatchNormalization())
CNN.add(MaxPool2D(2,2))
CNN.add(Dropout(0.5))
CNN.add(Flatten())
CNN.add(Dense(1000,activation='relu',use_bias=False))
CNN.add(BatchNormalization())
CNN.add(Dropout(0.4))
CNN.add(Dense(15,activation='softmax'))
```

3.3 Etapa 1 : Empleo del modelo AlexNet con el dataset desequilibrado

Las pruebas de esta etapa es intentar hallar los hiperparámetros optimizados para el entrenamiento de los modelos de las siguientes etapas. El modelo que se ha utilizado fue el AlexNet. Las pruebas que se han realizado en esta etapa son:

- **Las pruebas de las resoluciones**
  Al saber que los parámetros entrenables y el rendimiento del modelo se reduce directamente proporcional con el tamaño de la resolución de las imágenes [56] [57] [58], por tanto, Se realiza esta prueba para hallar la resolución adecuada para nuestro modelo comprobando con muchos variables. Uno de los variables relevantes que se considera como prioritario de esta prueba es la capacidad máxima de RAM ofrecida por el Google Colab. Igual si el pre-procesado supere 50% de la capacidad de RAM, no se considera aceptable.

  Se ha comprobado las resoluciones 400x400, 300x300, 256x256, 128x128.

- **Las pruebas de Epochs**
  una Epoch es un turno que se utiliza todos los datos del conjunto de entrenamiento y validación. Se realiza esta prueba para visualizar el comportamiento del modelo como El rendimiento del modelo comparando las epochs, La probabilidad de ocurrencia de sobreajuste.

  Se ha comprobado 10,20,50 y 100 epochs

- **Las pruebas del tamaño de lote (Batch size)**
  Batch size es el tamaño de muestras que se alimenta al modelo cada el descenso de gradiente. Se considera importante sabiendo que al aumentar el Batch size, se puede acelerar el entrenamiento por epoch y se considera importante para este trabajo fin de grado aunque la utilización de Batch size grande puede causar algunos efectos negativos al entrenamiento [59]. Se mide la duración de epochs como la duración máxima durante el entrenamiento sin contar el primer epoch.

  Se ha comprobado Batch size de 32,128,256

Los demás hiperparámetros que no estén en las pruebas como la tasa de apendizaje(valor igual a 0.001), se ha utilizado los valores iniciales ofrecidos por las bibliotecas. Se ha valorado el rendimiento del modelo con las métricas como Exactitud, Precisión, el valor F1, Exhaustividad, El valor de exactitud TOP-1 y TOP-5.

3.4 Etapa 2: Empleo del modelo AlexNet con los hiperparámetro y el dataset optimizados de la etapa anterior

En esta etapa, se ha utilizado los hiperparámetros más optimizados de las pruebas de la primera etapa aplicando en el dataset ya equilibrado. Los demás hiperparámetros que no estén en las pruebas como la tasa de aprendizaje(valor igual a 0.001),se ha utilizado los valores iniciales ofrecidos por las bibliotecas Se ha valorado el rendimiento del modelo con las métricas como Exactitud, Precisión, el valor F1, Exhaustividad, El valor de exactitud TOP-1 y TOP-5.

3.5 Etapa 3: Empleo del modelo VGG para comprobar el modelo con los parametros altos y las capas más profundas

En esta etapa, Se ha utilizado el modelo VGG para las pruebas con el objetivo de comparar el rendimiento entre el modelo de la cantidad de parámetros entrenables altos y el modelo con menor cantidad de parámetros entrenables. Al aumentar los parámetros, se aumenta los variables arreglables como peso y sesgo que, técnicamente, se podría aprender mejor el modelo pero siempre viene con problemas de sobreajuste.

Se ha realizado 2 pruebas, una sobre el dataset desequilibrado y la otra sobre el dataset equilibrado.

Se ha utilizado la resolución de 128x128, Epochs de 50 y el Batch size de 32 para las 2 pruebas que se considera suficiente para obtener la comparación esperada y que el modelo no sobrepase la capacidad máxima de RAM.

Los demás hiperparámetro que no estén en las pruebas como la tasa de aprendizaje(valor igual a 0.001), se ha utilizado los valores iniciales ofrecidos por las bibliotecas.

Se ha valorado el rendimiento del modelo con las métricas como Exactitud, Precisión, el valor F1, Exhaustividad, El valor de exactitud TOP-1 y TOP-5.

3.6 Etapa 4: Las predicciones de 3 tipos de cielo

Esta etapa es como una extensión de las pruebas. Al observar que es difícil clasificar las imágenes similares que, en algunos casos, será difícil clasificar por los ojos humanos pero según los resultados de las etapas anteriores como la matriz de confusión, se puede observar que los modelos pueden clasificar bien los cielos en 3 tipos según las características y claridad de cielo. En esta etapa, se realiza las pruebas de clasificación de los cielos en 3 tipos.

- “Cielo totalmente cubierto” (clase CIE 1-5)
- “Cielo parcialmente cubierto” (clase CIE 6-10)
- “Cielo despejado” (clase CIE 11-15)

Se elige las mejores configuraciones que se han aplicado en cada dataset de los 2 dataset según las métricas de TOP-1 y TOP-5 para las pruebas de esta etapa.

Se ha valorado el rendimiento del modelo con las métricas como Exactitud, Precisión, el valor F1, Exhaustividad, El valor de exactitud TOP-1 y TOP-5.

4 RESULTADOS / VALIDACIÓN / PRUEBA

4.1 Resultado de la etapa 1: Empleo del modelo AlexNet con el dataset desequilibrado

4.1.1 Prueba de la resolución 400x400 y 300x300

Con la resolución de 400x400, el pre-procesado consume la RAM de 35.35 y que finalmente provocando el choque de sesión y se reinició el tiempo de ejecución. Con la resolución de 300x300, el pre-procesado consume la RAM de 24.13 Gb que se ha superado el umbral de 50% de la capacidad máxima

Figura 4-1 El consumo de RAM al utilizar la resolución de 400x400. (Fuente: Elaboración propia)

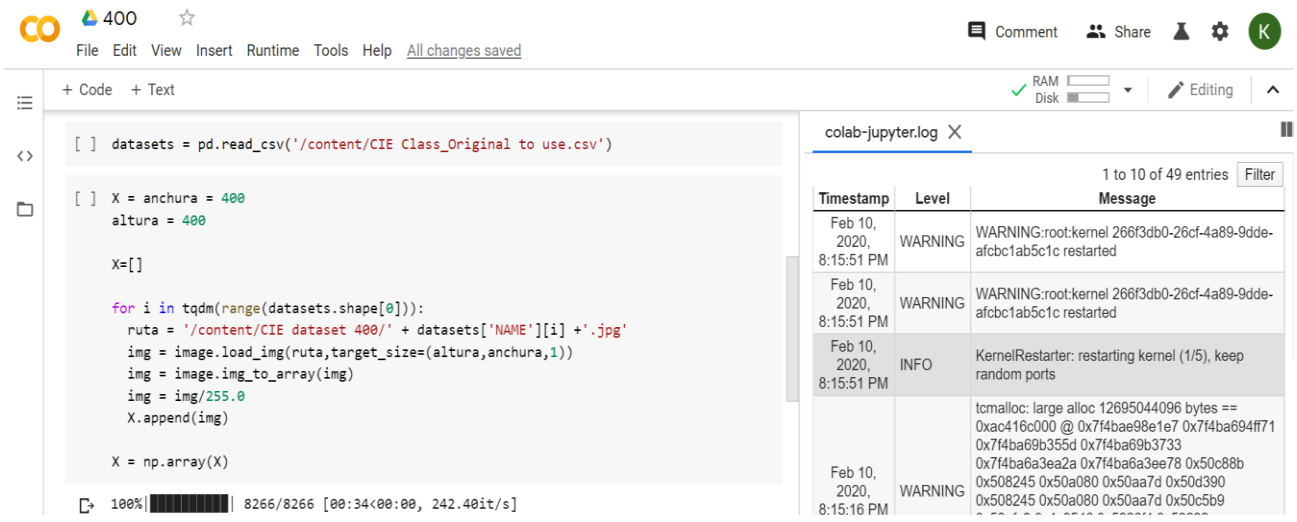


Figura 4-2 El choque de sesión al utilizar la resolución de 400x400. (Fuente: Elaboración propia)

Figura 4-3 El consumo de RAM al utilizar la resolución de 300x300. (Fuente: Elaboración propia)

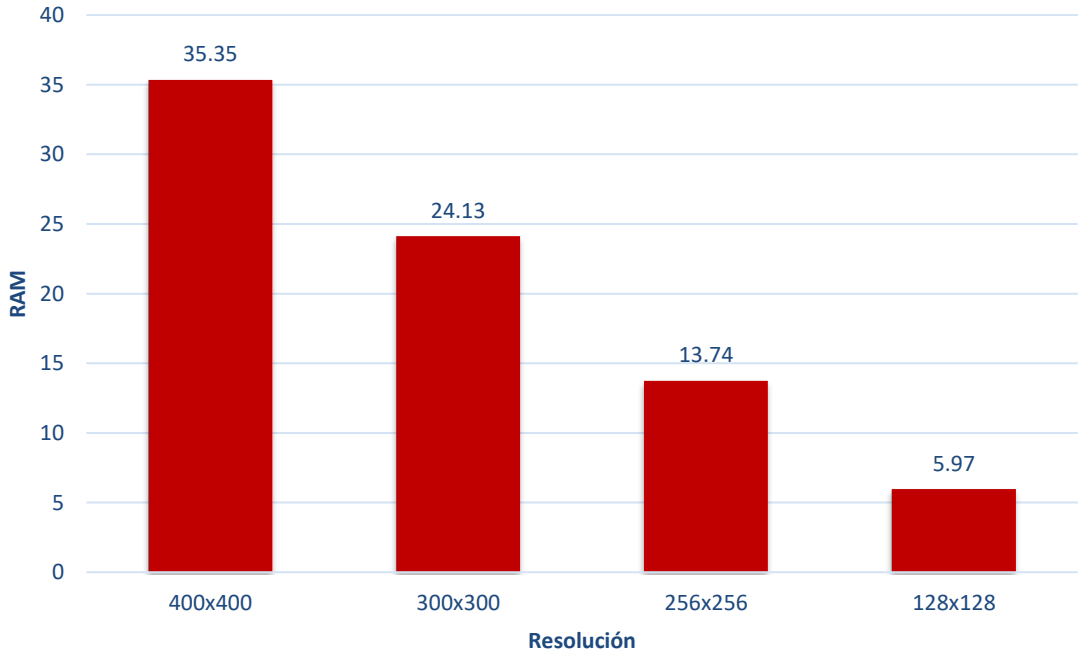


Figura 4-4 Los consumos de RAM por las resoluciones. (Fuente: Elaboración propia)

4.1.2 Prueba de la resolución 256x256

Con la resolución de 256x256, se consume 13.74 Gb de RAM que significa no se ha superado el 50% de la consumición de RAM así que podemos continuar probando los demás parámetros con esta.

Figura 4-5 El consumo de RAM al utilizar la resolución de 256x256. (Fuente: Elaboración propia)

Se realiza las pruebas con el modelo AlexNet con los siguientes hiperparámetros.

- Batch size de 32,128 y 256
- epochs de 10,20,50 y 100

Se ejecuta el modelo.Se puede observar los parámetros totales para la resolución de 256*256 son 803039 parámetros. Los parámetros entrenables son 801631 parámetros.Los parámetros no entrenables son 1408 parámetros.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 254, 254, 32)      864
_________________________________________________________________
batch_normalization (BatchNo (None, 254, 254, 32)      128
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 127, 127, 32)      0
_________________________________________________________________
dropout (Dropout)            (None, 127, 127, 32)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 125, 125, 64)      18432
_________________________________________________________________
batch_normalization_1 (Batch (None, 125, 125, 64)      256
max_pooling2d_1 (MaxPooling2 (None, 62, 62, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 62, 62, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 60, 60, 64)        36864
```

```
_________________________________________________________________
batch_normalization_2 (Batch (None, 60, 60, 64)        256
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 30, 30, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 30, 30, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 28, 28, 96)        55296
_________________________________________________________________
batch_normalization_3 (Batch (None, 28, 28, 96)        384
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 96)        0
_________________________________________________________________
dropout_3 (Dropout)          (None, 14, 14, 96)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 12, 12, 128)       110592
_________________________________________________________________
batch_normalization_4 (Batch (None, 12, 12, 128)       512
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 6, 6, 128)         0
_________________________________________________________________
dropout_4 (Dropout)          (None, 6, 6, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 4608)              0
_________________________________________________________________
dense (Dense)                (None, 120)               552960
_________________________________________________________________
batch_normalization_5 (Batch (None, 120)               480
_________________________________________________________________
dropout_5 (Dropout)          (None, 120)               0
_________________________________________________________________
dense_1 (Dense)              (None, 120)               14400
_________________________________________________________________
batch_normalization_6 (Batch (None, 120)               480
_________________________________________________________________
dropout_6 (Dropout)          (None, 120)               0
_________________________________________________________________
dense_2 (Dense)              (None, 80)                9600
_________________________________________________________________
batch_normalization_7 (Batch (None, 80)                320
_________________________________________________________________
dropout_7 (Dropout)          (None, 80)                0
_________________________________________________________________
dense_3 (Dense)              (None, 15)                1215
=================================================================
Total params: 803,039
Trainable params: 801,631
```

```
Non-trainable params: 1,408
```

Para la fase de entrenamiento y validación se ha utilizado el optimizador Adam, la función de pérdida de Entropía cruzada(Categorical Cross Entropy Loss). Se ha utilizado Exactitud(Accuracy) y Pérdida(Loss) para ver los compartimientos del modelo durante el entrenamiento y la validación mediante el comando "compile".

```
[ ]  CNN.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
```

Figura 4-6 El empleo de comando "compile". (Fuente: Elaboración propia)

Para Batch size de 128 con el epochs de 100, se ha provocado el choque de sesión pro superar la capacidad máxima de RAM y se reinició las sesión. También para todas las pruebas de Batch size de 256 se ha provocado el choque de sesión por superar la capacidad máxima de RAM. La siguiente tabla es la tabla de las métricas de evaluación del modelo de la fase de entrenamiento y validación.

Batch/Epochs	Accuracy	Precision	F1 score	recall score	t/epochs
32/10	0.2611	0.3125	0.2132	0.2611	229s
32/20	0.3204	0.3465	0.2797	0.3204	179s
32/50	0.5634	0.5653	0.5351	0.5634	190s
32/100	0.5659	0.5467	0.5356	0.5659	191s
128/10	0.3180	0.1949	0.2340	0.3180	178s
128/20	0.3808	0.4173	0.3070	0.3808	178s
128/50	0.3361	0.4644	0.3165	0.3361	228s
128/100	**Se ha superado la capacidad máxima de RAM**				
256/10					
256/20					
256/50					
256/100					

Tabla 4-1 los resultados del entrenamiento y validación de la resolución 256x256. (Fuente : Elaboración propia)

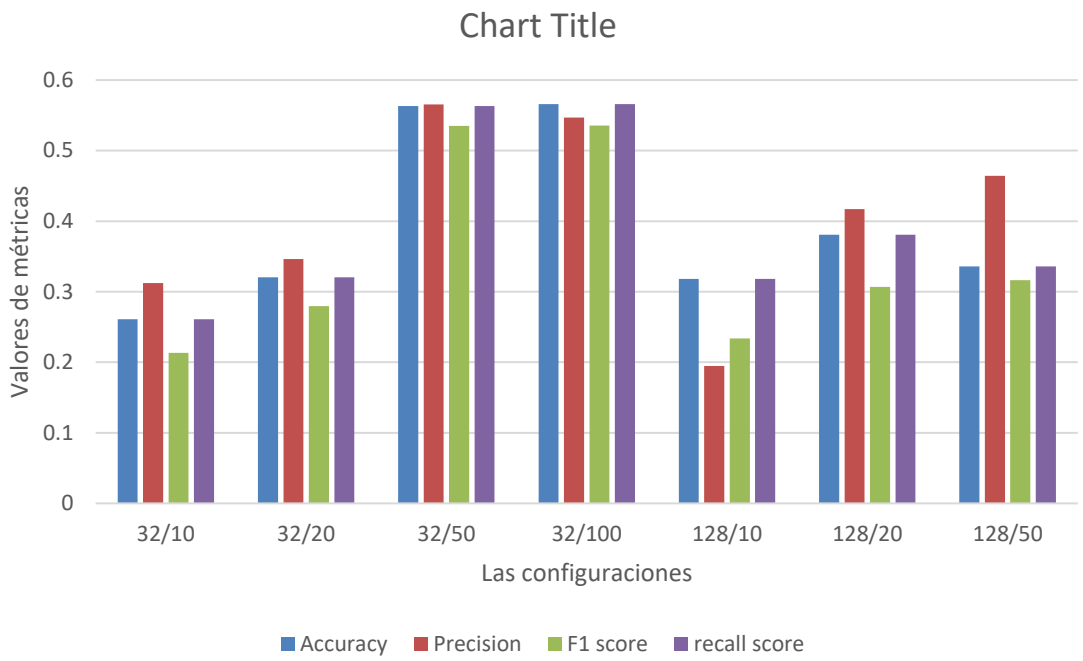


Figura 4-7 La gráfica de los resultados de la tabla 4-1. (Fuente: Elaboración propia)

Durante el entrenamiento se puede observar que al aumentar Batch size, el entrenamiento del modelo se empeora produciendo poca Exactitud y muchas Pérdidas sobre todo en los primeros epochs además se consume la cantidad de RAM innecesaria. Al aumentar Batch size, se acelera el entrenamiento y validación pero no se considera significativo además El Google Colab no puede garantizar los recursos [38], por tanto, el tiempo por epochs no es estable. Se mejora el entrenamiento directamente proporcional a número de epochs en todas las pruebas de Batch size. Se puede observar en la muestra de **Batch_size=32/epochs de 100** las señales de sobreajuste aproximadamente a 60 epochs(Figura 4-8 y Figura 4-9). Las mejor configuración según las métricas anteriores es la muestra de **Batch_size=32/epochs de 100**

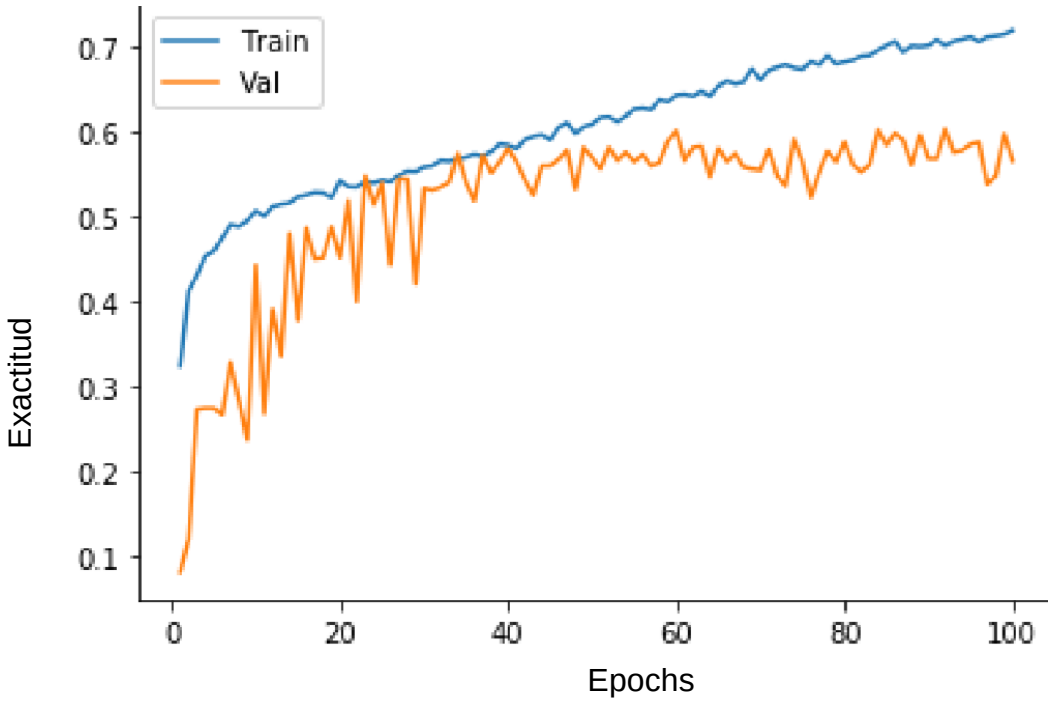


Figura 4-8 La grafica de Accuracy de entrenamiento y validación de Batch size de 32 y 100 epochs. (Fuente: Elaboración propia)

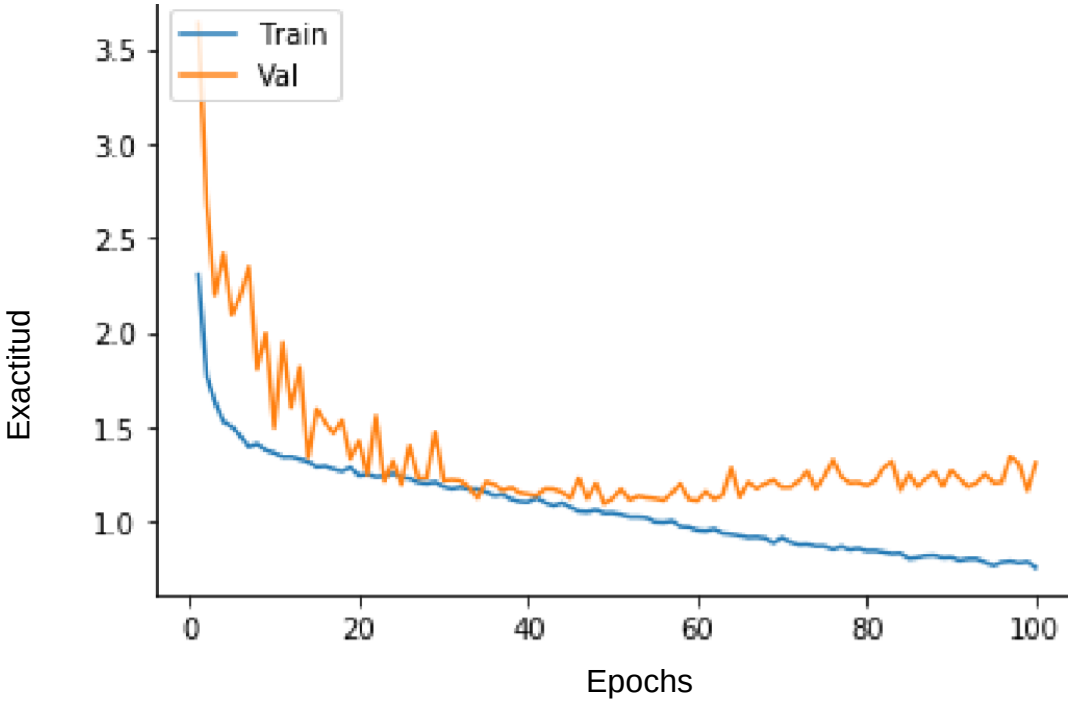


Figura 4-9 La grafica de las Loss de entrenamiento y validación de Batch size de 32 y 100 epochs. (Fuente: Elaboración propia)

La siguiente fase es la prueba final del modelo utilizando el conjunto de imágenes que no había visto el modelo para visualizar la efectividad del aprendizaje del modelo después del entrenamiento.

En las matrices de confusión de las pruebas finales(Figura 4-10). Se puede observar que en las clases minoritarias como clase 5 y clase 15, al modelo le cuesta hacer las predicciones incluso en la mejor configuración de esta etapa como la de 256x256, Batch size de 32 y 100 epochs.

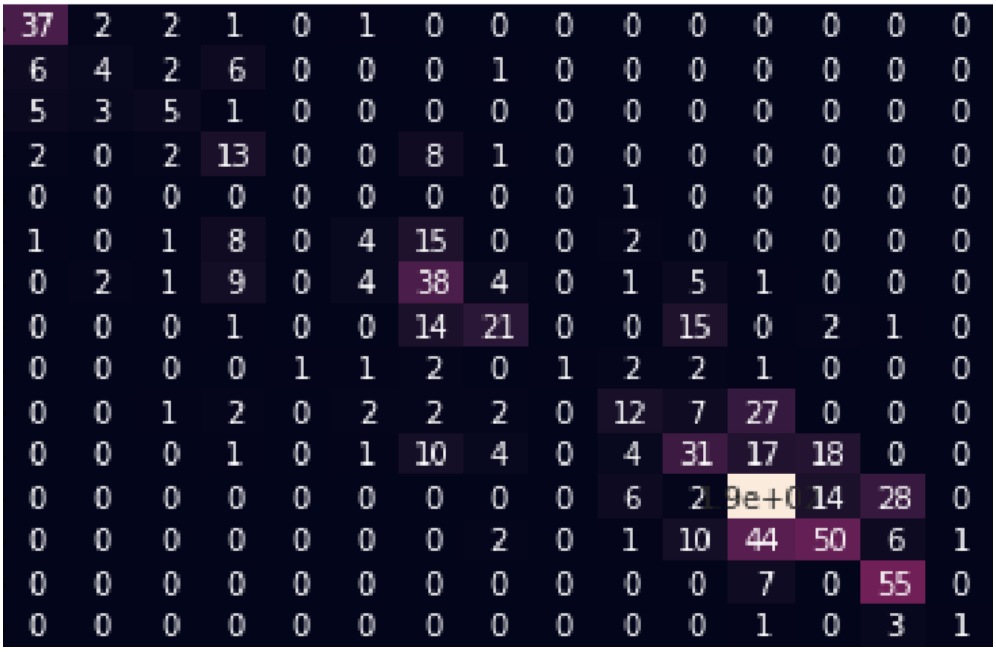


Figura 4-10 Matriz de confusión de la resolución 256x256,Batch sizede 32 y 100 epochs. (Fuente: Elaboración propia)

La siguiente tabla es la tabla de las métricas de evaluación del modelo de la fase de pruebas finales.

Batch/Epochs	Accuracy	Precision	F1 score	recall score	TOP-1	TOP-5
32/10	0.3010	0.3604	0.2510	0.3010	0.3010	0.9068
32/20	0.3603	0.3915	0.3194	0.3603	0.3603	0.8548
32/50	0.5695	0.5587	0.5468	0.5695	0.5695	0.9685
32/100	0.5610	0.5467	0.5356	0.5659	0.5610	0.9733
128/10	0.3579	0.2305	0.2744	0.3579	0.3579	0.6964
128/20	0.4062	0.4114	0.3375	0.4062	0.4062	0.8899
128/50	0.3748	0.5220	0.3598	0.3748	0.3748	0.8730
128/100	Se ha superado la capacidad máxima de RAM					
256/10						
256/20						
256/50						
256/100						

Tabla 4-2 Los resultados de las pruebas finales de la resolución 256x256. (Fuente: Elaboración propia)

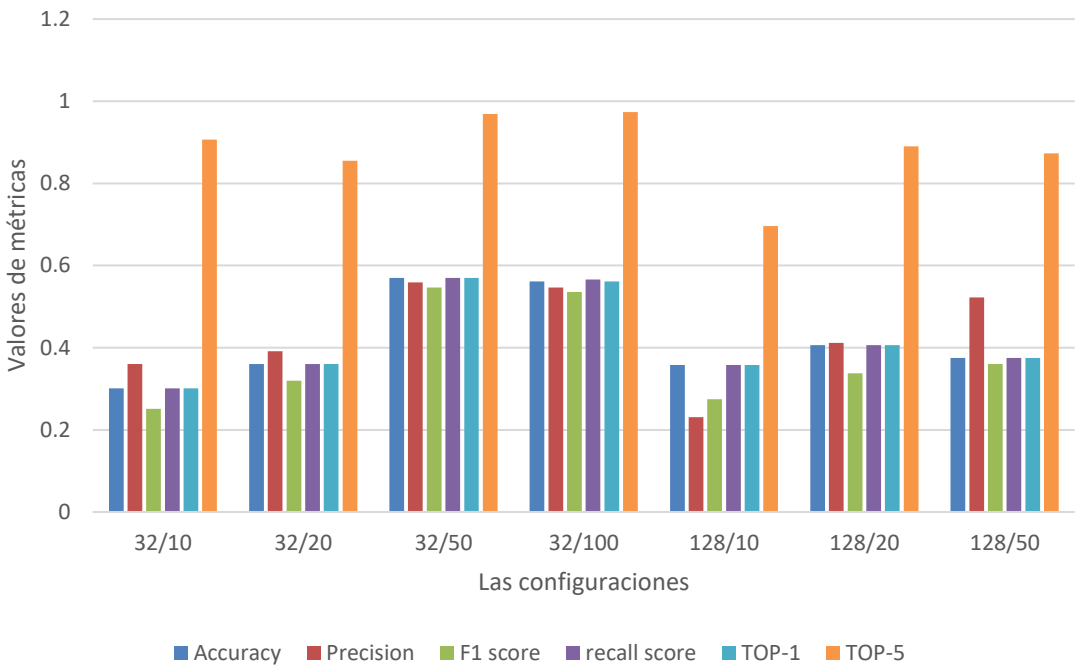


Figura 4-11 La gráfica de los resultados de la tabla 4-2. (Fuente: Elaboración propia)

La muestra de **Batch_size=32/epochs de 100** sigue siendo la mejor configuración para la prueba final aunque el valor de Exactitud o TOP-1 es menor que la muestra de **Batch_size=32/epochs de 50** pero debido a que el dataset es desequilibrado, por tanto, no es considerable utilizar Accuracy o TOP-1 por la falta de imágenes en alguna clase que se ha fijado en las matrices de confusión tanto la validación como la prueba final que hay mucho más predicciones correctas es las clases mayoritarias que en las clases minoritarias.

4.1.3 Prueba de la resolución 128x128

Con la resolución de 128x128, se consume 5.97 Gb de RAM que significa no se ha superado el 50% de la consumición de RAM así que podemos continuar probando los demás parámetros con esta.

Figura 4-12 La consumición de RAM al utilizar la resolución de 128x128. (Fuente: Elaboración propia)

Se realiza las pruebas con el modelo AlexNet con los siguientes hiperparámetros.

- Batch size de 32, 128 y 256
- epochs de 10, 20, 50 y 100

Se ejecuta el modelo. Se puede observar los parámetros totales para la resolución de 128x128 son 311.519 parámetros. Los parámetros entrenables son 310.111 parámetros. Los parámetros no entrenables son 1408 parámetros

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 126, 126, 32)      864
_________________________________________________________________
batch_normalization (BatchNo (None, 126, 126, 32)      128
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 63, 63, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 63, 63, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 61, 61, 64)        18432
_________________________________________________________________
batch_normalization_1 (Batch (None, 61, 61, 64)        256
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 30, 30, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 30, 30, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 28, 28, 64)        36864
_________________________________________________________________
batch_normalization_2 (Batch (None, 28, 28, 64)        256
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 14, 14, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 12, 12, 96)        55296
_________________________________________________________________
batch_normalization_3 (Batch (None, 12, 12, 96)        384
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 6, 6, 96)          0
_________________________________________________________________
dropout_3 (Dropout)          (None, 6, 6, 96)          0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 4, 4, 128)         110592
_________________________________________________________________
batch_normalization_4 (Batch (None, 4, 4, 128)         512
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 2, 2, 128)         0
_________________________________________________________________
dropout_4 (Dropout)          (None, 2, 2, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 512)               0
```

```
dense (Dense)                (None, 120)               61440
_________________________________________________________________
batch_normalization_5 (Batch (None, 120)               480
_________________________________________________________________
dropout_5 (Dropout)          (None, 120)               0
_________________________________________________________________
dense_1 (Dense)              (None, 120)               14400
_________________________________________________________________
batch_normalization_6 (Batch (None, 120)               480
_________________________________________________________________
dropout_6 (Dropout)          (None, 120)               0
_________________________________________________________________
dense_2 (Dense)              (None, 80)                9600
_________________________________________________________________
batch_normalization_7 (Batch (None, 80)                320
_________________________________________________________________
dropout_7 (Dropout)          (None, 80)                0
_________________________________________________________________
dense_3 (Dense)              (None, 15)                1215
=================================================================
Total params: 311,519
Trainable params: 310,111
Non-trainable params: 1,408
```

Para la fase de entrenamiento y validación se ha utilizado el optimizador Adam, la función de pérdida de Entropía cruzada(Categorical Cross Entropy Loss). Se ha utilizado Exactitud(Accuracy) y Pérdida(Loss) para ver los compartimientos del modelo durante el entrenamiento y la validación. La ejecución es igual a la de los entrenamientos y validaciones de la resolución 256x256.

La siguiente tabla es la tabla de las métricas de evaluación del modelo de la fase de entrenamiento y validación.

Batch/Epochs	Accuracy	Precision	F1 score	recall score	t/epochs
32/10	0.3712	0.3020	0.3074	0.3712	45s
32/20	0.4195	0.3583	0.3454	0.4195	52s
32/50	0.5139	0.5127	0.4858	0.5139	44s
32/100	0.5876	0.5869	0.5643	0.5876	45s
128/10	0.1354	0.1695	0.1057	0.1354	36s
128/20	0.3857	0.3931	0.3201	0.3857	40s
128/50	0.4957	0.5361	0.4603	0.4957	40s
128/100	0.5719	0.5807	0.5494	0.5719	40s
256/10	0.0507	0.0025	0.0049	0.0507	35s
256/20	0.0628	0.1256	0.0268	0.0628	40s
256/50	0.3784	0.3590	0.3171	0.3784	40s
256/100	0.5211	0.5409	0.4930	0.5211	39s

Tabla 4-3 Los resultados de entrenamiento y validación de la resolución 128x128.(Fuente: Elaboración propia)

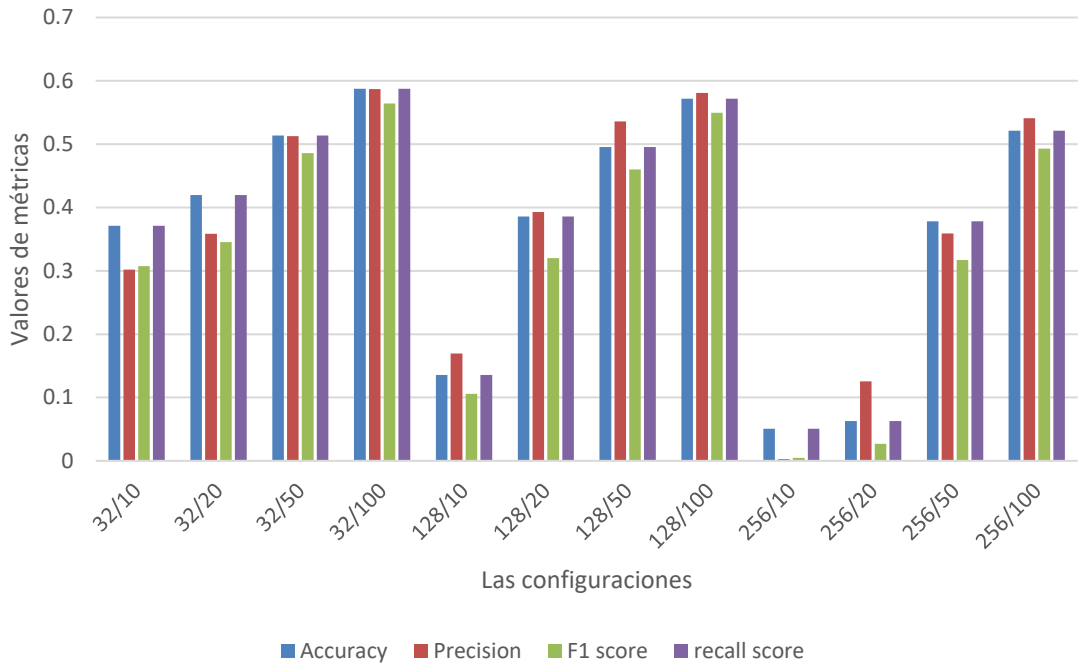


Figura 4-13 La gráfica de los resultados de la tabla 4-3. (Fuente: Elaboración propia)

Durante el entrenamiento se puede observar que al reducir resolución comparando con la resolución 256x256 no se reduce el rendimiento de entrenamiento significable y además ayuda a ahorrar la consumición de RAM. Al aumentar Batch size,se acelera el entrenamiento y validación pero no se considera significativo. El rendimiento del entrenamiento y validación se mejora directamente proporcional a número de epochs en todas las pruebas de Batch size. La mejor configuración de la fase de entrenamiento y validación según las métricas es **Batch_size=128/epochs de 100**,considerando las métricas más fiables como **el valor F1 y la Exhaustividad** [55] en este caso.

La siguiente fase es la prueba final del modelo utilizando el conjunto de imágenes que no había visto el modelo para visualizar la efectividad del aprendizaje del modelo después del entrenamiento.

La siguiente tabla es la tabla de las métricas de evaluación del modelo de la fase de pruebas finales.

Batch/Epochs	Accuracy	Precision	F1 score	recall score	TOP-1	TOP-5
32/10	0.3893	0.4436	0.3269	0.3893	0.3893	0.8476
32/20	0.4546	0.5424	0.3880	0.4546	0.4546	0.9286
32/50	0.5247	0.5183	0.4984	0.5247	0.5247	0.9721
32/100	0.5792	0.5692	0.5560	0.5792	0.5792	0.9697
128/10	0.1475	0.1705	0.1215	0.1475	0.1475	0.4655
128/20	0.4268	0.4563	0.3653	0.4268	0.4268	0.8814
128/50	0.5139	0.5201	0.4768	0.5139	0.5139	0.9625
128/100	0.5574	0.5696	0.5311	0.5574	0.5574	0.9721
256/10	0.0519	0.0027	0.0051	0.0519	0.0519	0.1293
256/20	0.0677	0.2532	0.0343	0.0677	0.0677	0.1995
256/50	0.4135	0.5248	0.3597	0.4135	0.4135	0.9250
256/100	0.5175	0.5156	0.4851	0.5175	0.5175	0.9637

Tabla 4-4 Los resultados de las pruebas finales de la resolución 256x256. (Fuente: Elaboración propia)

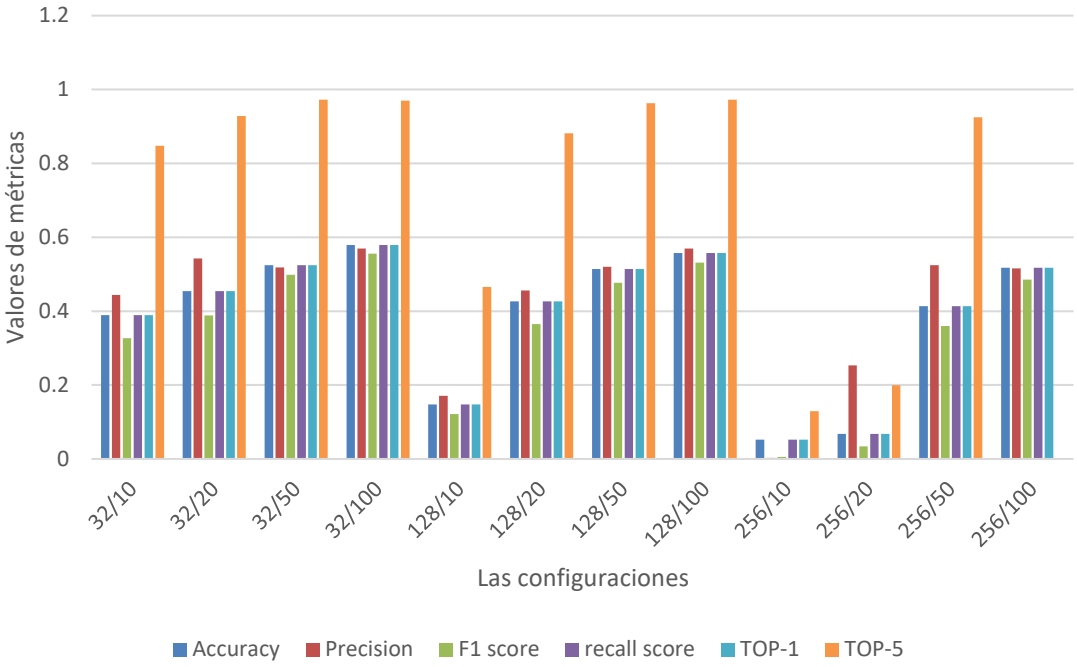


Figura 4-14 La gráfica de los resultados de la tabla 4-4. (Fuente: Elaboración propia)

Después de las pruebas finales, se puede observar que las muestras **Batch_size=32/epochs de 100 y Batch_size=128/epochs de 100** son las 2 configuraciones que tienen las mejores métricas. Si se considera las métricas como el valor F1 y la Exhaustividad, se puede observar que la muestra de **Batch_size=32/epochs de 100** tiene mejor configuración.

Según los resultados de la primera etapa, se puede concluir que al aumentar Batch size afecta negativamente al entrenamiento del modelo provocando sobreajuste en los primeros epochs, Sobreconsumición de RAM aunque se acelera el proceso pero la diferencia de tiempo no se significable. Al aumentar epochs, optimiza positivamente al entrenamiento, validación y prueba final sobre todo con la resolución de 128x128. En las pruebas que se ha realizado con la resolución 128x128, no se ha observado ninguna señal de sobreajuste.

Para la siguiente etapa, Se elige los parámetros Batch size de 32, epochs de 100, 150, 200 y la resolución de 128 y 256 para las pruebas. Al aplicar la resolución de 256x256 habrá que tenerse en cuenta que pueda consumir alta cantidad de RAM.

4.2 Etapa 2 : Empleo del modelo AlexNet con los hiperparametro y el dataset optimizados de la etapa anterior

En esta etapa, Se ha utilizado los métodos, códigos y bibliotecas simlilares a la etapa anterior. Los cambios de las pruebas de esta etapa son el dataset y los hiperparametros elegidos de la primera etapa.

Se ha intentado realizar las pruebas con la resolución de 256x256 y con epochs de 150 y 200 pero La capacidad de RAM de nos permitió.

Se puede observar que en las pruebas de las 2 resoluciones, se ha aumentado el rendimiento en la fase del entrenamiento y la validación. Aparecen los números en todas las clases en el diagonal de la matriz de confusión con la cantidad mayor que otras posiciones sobre todo en las pruebas con número de epochs alto. Así que se ha aumentado los valores de la evaluación comparando con las pruebas del dataset desequilibrado. Se puede observar que con la resolución 128x128, se ha ocurrido el Overfitting en el epochs 130 aproximadamente.

resolución/epochs	Accuracy	Precision	F1 score	Recall score
128/50	0.5721	0.5797	0.5625	0.5828
128/100	0.704	0.7191	0.7231	0.7107
128/150	0.7053	0.7231	0.7040	0.7085
128/200	0.744	0.7525	0.7460	0.7477
256/50	0.6240	0.6896	0.6162	0.6247
256/100	0.744	0.7554	0.7452	0.7512
256/150	**Se ha superado la capacidad máxima de RAM**			
256/200				

Tabla 4-5 Los resultados de entrenamiento y validación de la etapa 2. (Fuente: Elaboración propia)

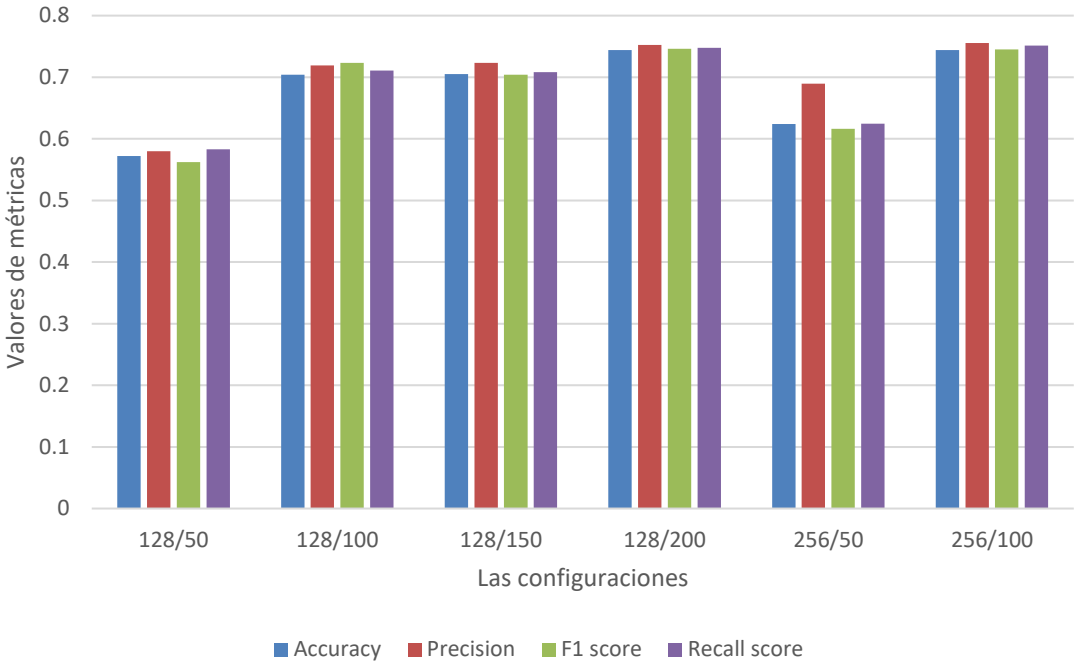


Figura 4-15 La gráfica de los resultados de la tabla 4-5. (Fuente: Elaboración propia)

Según la tabla, Se puede observar que la mejor configuración es la de **resolución = 256x256/epochs de 100** en todas las métricas. Incluso si comparamos esta configuración con **resolución = 128x128/epochs de 100 y 200**, debido al aumento de la resolución que afecta positivamente al entrenamiento hasta que provoca el sobreajuste. Si comparamos los resultados de las métricas a los de

primera etapa, se ha mejorado bastante en la fase de entrenamiento y validación utilizando el dataset equilibrado.

Para la siguiente fase que es las pruebas finales utilizando las imágenes que no había visto el model.

La siguiente tabla es la tabla de las métricas de la fase de prueba final.

resol/epochs	Accuaracy	Precision	F1 score	Recall score	TOP-1	TOP-5
128/50	0.4485	0.4251	0.4173	0.4550	0.4485	0.8949
128/100	0.4223	0.3726	0.3819	0.4346	0.4223	0.8336
128/150	0.3916	0.3569	0.3481	0.3926	0.3916	0.8621
128/200	0.4135	0.3670	0.3744	0.4148	0.4135	0.8512
256/50	0.4529	0.4522	0.4241	0.4574	0.4529	0.8905
256/100	0.4070	0.3592	0.3671	0.4091	0.4070	0.8884
256/150	**Se ha superado la capacidad máxima de RAM**					
256/200						

Tabla 4-6 Los resultados de las pruebas finales de la etapa 2. (Fuente: Elaboración propia)

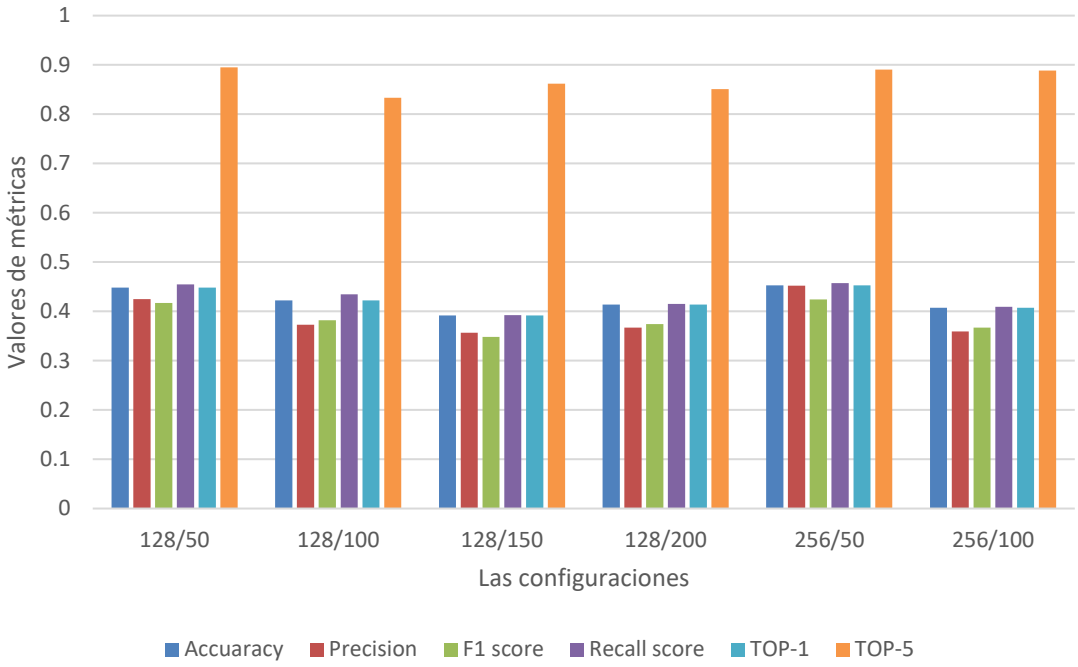


Figura 4-16 La gráfica de los resultados de la tabla 4-6. (Fuente: Elaboración propia)

Después de hacer la prueba final, se puede observar que el rendimiento del modelo según las métricas no puede indicar nada exactamente. El resultado del modelo en comparación con la primera etapa sigue siendo igual. Se puede observar que el tratamiento de sobre-muestro aleatorio que hemos aplicado, facilita al modelo para clasificar en la clase 15 que es la clase del cielo más despejado de todas. El modelo sigue teniendo el problema de confusión entre las imágenes de las clases cercanas que mostrando en las matrices de confusión y el valor de TOP-5 bastante alta (Podemos observar que todas las pruebas tienen el TOP-5 más de 0,80).

Se puede concluir que al aplicar las técnicas de tratamiento del dataset desequilibrado, se mejora el rendimiento del modelo durante el entrenamiento y validación. Cuando le enseñamos al modelo las

imágenes totalmente nuevas. El modelo tiene problema en clasificarlas. El modelo puede clasificar bastante bien las imágenes en las clases cercanas a la etiqueta real.

El problema que tiene el modelo ya no es el problema del desequilibrio de los datasets. Los problema que podemos visualizar ahora es el problema de que las imágenes de las clases cercanas son muy similares y le falta el dataset la variedad.

4.3 El resultado de la etapa 3: Empleo del modelo VGG para comprobar el modelo con los parámetros altos y las capas más profundas

Los métodos y códigos de ejecución de esta etapa son similar a la primera etapa. Los cambios son el modelo que se utiliza el model VGG para aumentar los parámetros entrenables. Los parámetros en total son 15.819.847 parámetros con la resolución de 128x128.Loas parámetros entrenables son 15.817.303 parámetros. Los parámetros no entrenables son 2.544 parámetros.

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_50 (Conv2D)           (None, 126, 126, 16)      448
_________________________________________________________________
conv2d_51 (Conv2D)           (None, 124, 124, 16)      2304
_________________________________________________________________
batch_normalization_34 (Batc (None, 124, 124, 16)      64
_________________________________________________________________
dropout_34 (Dropout)         (None, 124, 124, 16)      0
_________________________________________________________________
conv2d_52 (Conv2D)           (None, 122, 122, 32)      4640
_________________________________________________________________
conv2d_53 (Conv2D)           (None, 120, 120, 32)      9216
_________________________________________________________________
batch_normalization_35 (Batc (None, 120, 120, 32)      128
_________________________________________________________________
dropout_35 (Dropout)         (None, 120, 120, 32)      0
_________________________________________________________________
conv2d_54 (Conv2D)           (None, 118, 118, 32)      9248
_________________________________________________________________
conv2d_55 (Conv2D)           (None, 116, 116, 32)      9216
_________________________________________________________________
batch_normalization_36 (Batc (None, 116, 116, 32)      128
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 58, 58, 32)        0
_________________________________________________________________
dropout_36 (Dropout)         (None, 58, 58, 32)        0
_________________________________________________________________
conv2d_56 (Conv2D)           (None, 56, 56, 64)        18496
_________________________________________________________________
conv2d_57 (Conv2D)           (None, 54, 54, 64)        36864
_________________________________________________________________
batch_normalization_37 (Batc (None, 54, 54, 64)        256
_________________________________________________________________
max_pooling2d_16 (MaxPooling (None, 27, 27, 64)        0
_________________________________________________________________
```

```
dropout_37 (Dropout)         (None, 27, 27, 64)        0
_________________________________________________________________
conv2d_58 (Conv2D)           (None, 25, 25, 128)       73856
_________________________________________________________________
conv2d_59 (Conv2D)           (None, 23, 23, 128)       147456
_________________________________________________________________
batch_normalization_38 (Batc (None, 23, 23, 128)       512
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 11, 11, 128)       0
_________________________________________________________________
dropout_38 (Dropout)         (None, 11, 11, 128)       0
_________________________________________________________________
flatten_5 (Flatten)          (None, 15488)             0
_________________________________________________________________
dense_14 (Dense)             (None, 1000)              15488000
_________________________________________________________________
batch_normalization_39 (Batc (None, 1000)              4000
_________________________________________________________________
dropout_39 (Dropout)         (None, 1000)              0
_________________________________________________________________
dense_15 (Dense)             (None, 15)                15015
=================================================================
Total params: 15,819,847
Trainable params: 15,817,303
Non-trainable params: 2,544
```

La siguiente tabla es la tabla de los resultados de las métricas de la fase de entrenamiento y validación. Incluso la prueba final de la prueba 1

Prueba 1	Accuracy	Precision	F1 score	Recall score	TOP-1	TOP-5
Entrenamiento y validación	0.4933	0.4987	0.4821	0.4933	-	-
Prueba final	0.5126	0.5186	0.5047	0.5126	0.5126	0.9407

Tabla 4-7 Los resultados de entrenamiento y validación de la etapa 3. (Fuente: Elaboración propia)

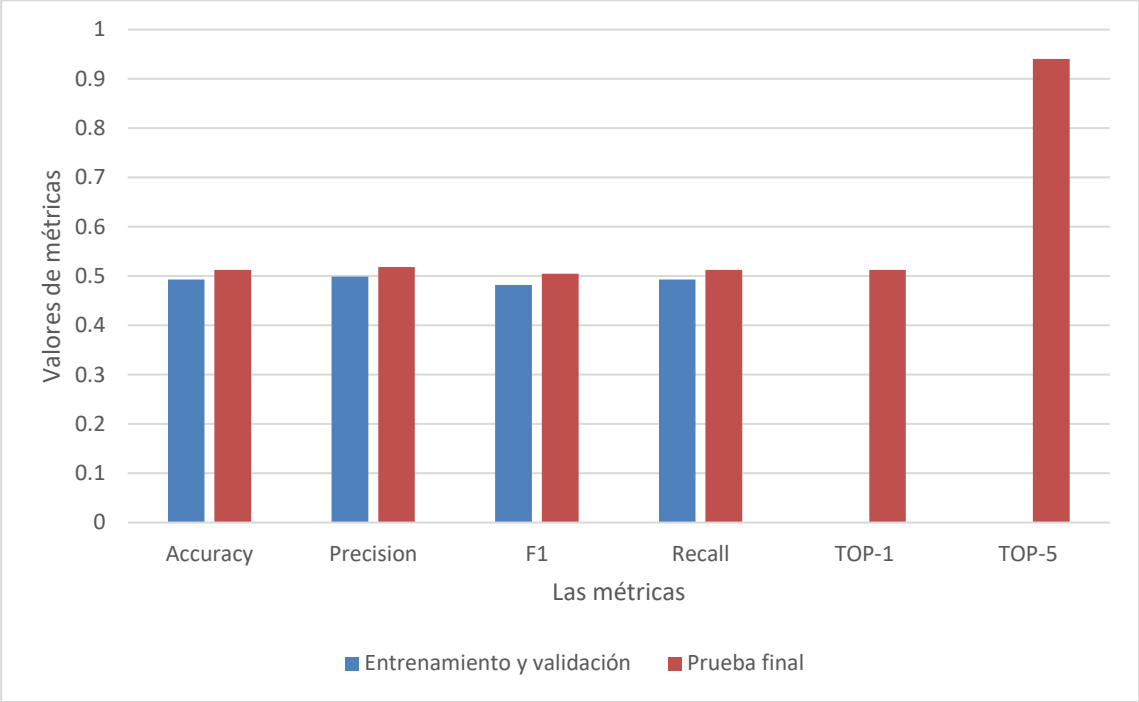


Figura 4-17 La gráfica de los resultados de la tabla 4-7. (Fuente: Elaboración propia)

La siguiente tabla es la tabla de los resultados de las métricas de la fase de entrenamiento y validación incluso La prueba final de la prueba 2

Prueba 2	Accuracy	Precision	F1 score	Recall score	TOP-1	TOP-5
Entrenamiento y validación	0.6746	0.6868	0.6804	0.6834	-	-
Prueba final	0.3894	0.3661	0.3564	0.3778	0.3894	0.8315

Tabla 4-8 los resultados de entrenamiento y validación de la etapa 3. (Fuente: Elaboración propia)

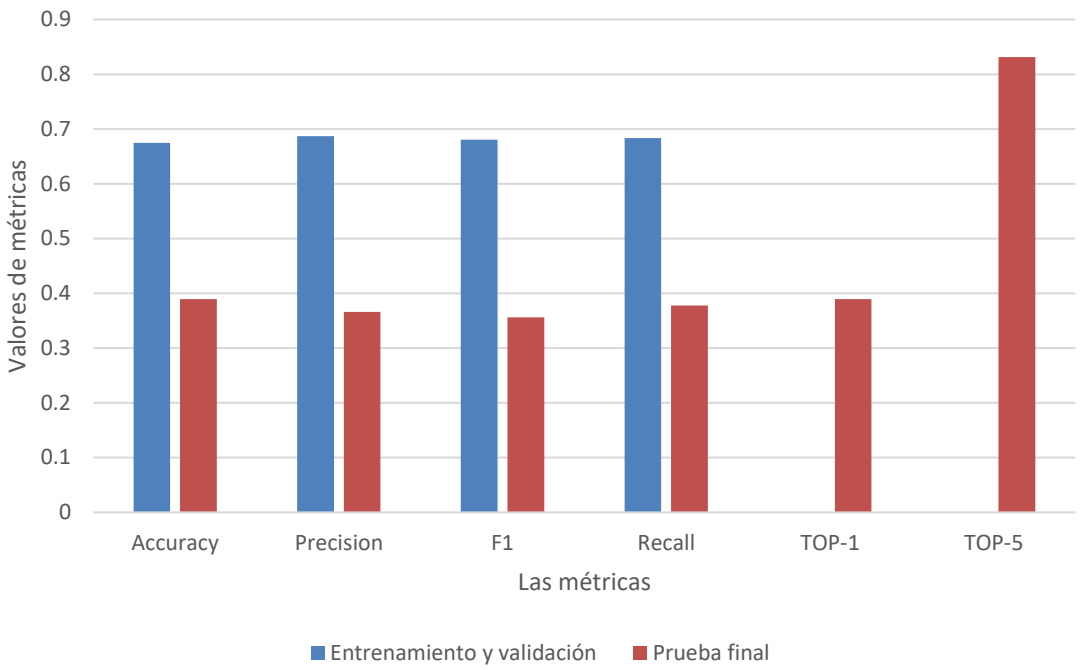


Figura 4-18 La gráfica de los resultados de la tabla 4-8. (Fuente: Elaboración propia)

Se puede visualizar que al aumentar los parámetros entrenables aplicando sobre los 2 datasets, Se ha ocurrido un gran sobreajuste a lo largo del proceso.

Las pruebas de esta etapa se comparan con las etapas anteriores. los siguientes son los que hemos observado durante la fase de entrenamiento y validación.

- En comparación de **la prueba 1** con la configuración de **128x128, batch_size=32 y epochs de 50 de la primera etapa**, La configuración de la primera etapa tiene mejor resultados en todas las métricas.

- En comparación de **la prueba 2** con la configuración de **128x128, batch_size=32 y epochs de 50 de la segunda etapa**, La configuración de la prueba 2 de la tercera etapa tiene mejor resultados en todas las métricas, aunque se sufre por el sobreajuste durante el entrenamiento

Las pruebas de esta etapa se comparan con las etapas anteriores. los siguientes son los que hemos observado durante la fase de la prueba final.

- En En comparación de **la prueba 1** con la configuración de **128x128, batch_size=32 y epochs de 50 de la primera etapa**, La configuración de la primera etapa y la de la prueba 1 son todas las métricas iguales aproximadamente

- En comparación de **la prueba 2** con la configuración de **128x128, batch_size=32 y epochs de 50 de la segunda etapa**, La configuración de la primera etapa puede sacar mejor resultados de la predicción en todas las métricas.

Según las pruebas de la tercera etapa, Los cambios de modelo que se han aplicado en esta etapa no resulta significativo en mejorar las predicciones, aunque se puede aplicar más técnicas de regularización para visualizar los comportamientos del modelo, pero hay que hacer más pruebas y

justificar los valores para encontrar un punto de equilibrio y evitar el subajuste. Además, el problema principal que se han descubierto y teníamos durante las pruebas de esta etapa es el dataset que tiene las imágenes muy similares que se considera el gran obstáculo para todas las etapas.

4.4 Resultado de la etapa 4: Las predicciones de 3 tipos de cielo

Las configuraciones que utilizamos en esta etapa para **el dataset desequilibrado** es **128x128,epochs de 100 y Batch size de 32** que es la mejor configuración de TOP-1 score y **256x256,epochs de 100 y Batch size de 32** que es la mejor configuración de TOP-5. Las 2 configuración son de la primera etapa.

Se visualiza los números de cada tipo por los siguientes diagramas de barra.

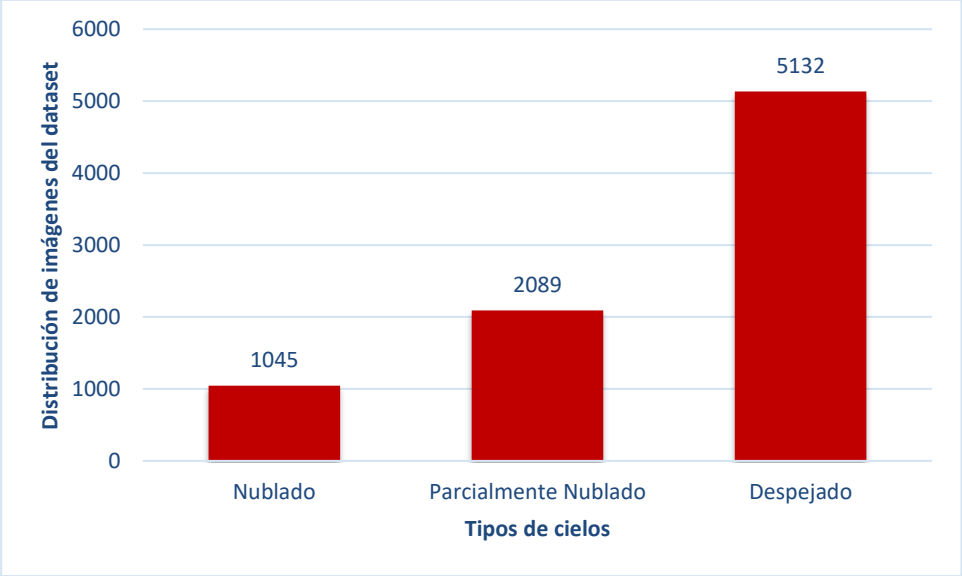


Figura 4-19 La distribución de imágenes por tipos en el dataset desequilibrado. (Fuente: Elaboración propia)

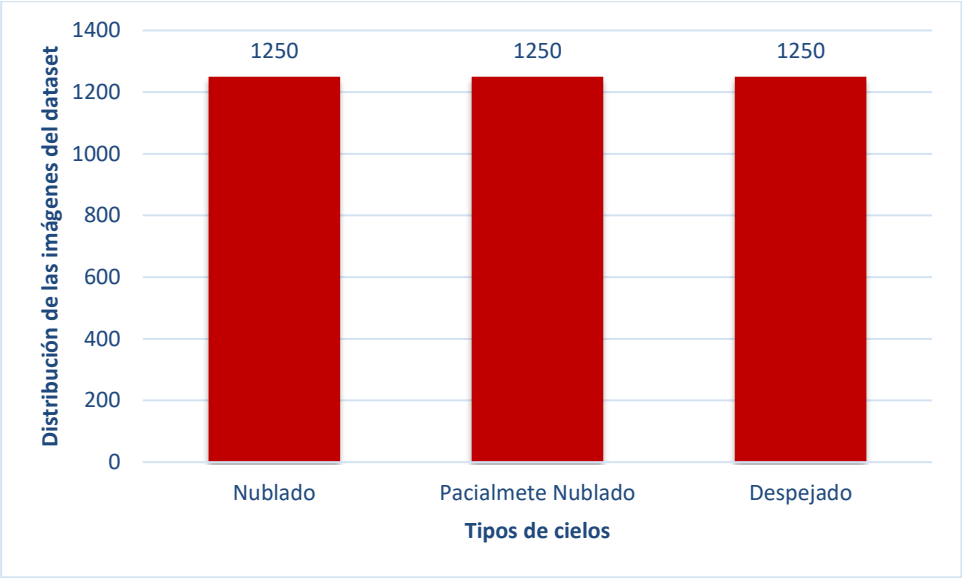


Figura 4-20 La distribución de imágenes por tipos en el dataset equilibrado. (Fuente: Elaboración propia)

La siguiete tabla es la tabla de los resultados de la fase de entrenamiento, validación y prueba final aplicando sobre el dataset desequilibrado.

- **128x128, epochs de 100 y Batch size de 32**

pruebas	Accuracy	Precision	F1	Recall
entre&val	0.8125	0.8360	0.8192	0.8125
prueba final	0.8246	0.8479	0.8307	0.8246

Tabla 4-9 Los resultados de 128x128, epochs de 100 y Batch size de 32 para el entrenamiento, validación y pruebas finales con el dataset desequilibrado para etapa 4. (Fuente: Elaboración propia)

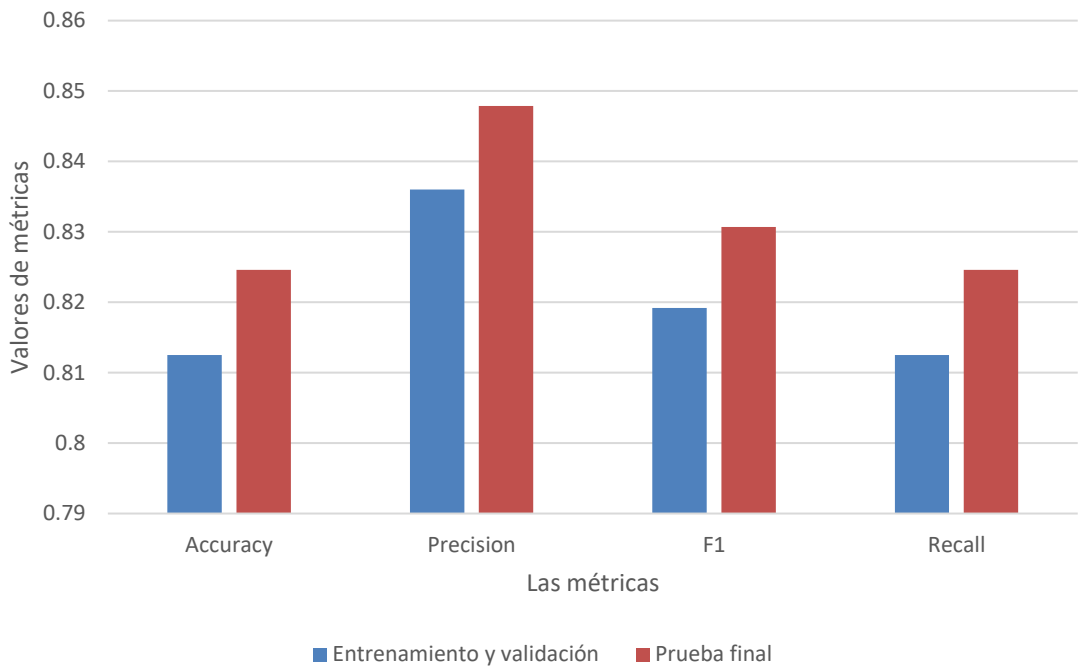


Figura 4-21 La gráfica de los resultados de la tabla 4-9. (Fuente: Elaboración propia)

- **256x256, epochs de 100 y Batch size de 32**
  Se ha realizado las pruebas con esta configuración, al intentar 4 veces terminar las ejecuciones no hemos podido alcanzar debido a que las pruebas han alcanzado los límites de RAM disponibles.

pruebas	Accuracy	Precision	F1	Recall
entre&val	Alcance al límite de RAM			
prueba final				

Tabla 4-10 Los resultados de 256x256, epochs de 100 y Batch size de 32 para el entrenamiento, validación y pruebas finales con el dataset desequilibrado para etapa 4. (Fuente: Elaboración propia)

Según los resultados anteriores, se puede observar que el modelo puede clasificar los 3 tipos de cielo hasta el nivel aceptable que son más de 80% de todas las métricas de las predicciones en pruebas finales. Lo más interesante es el modelo está confundido entre las clases **"5-10 cielo parcialmente cubierto" y "11-15 cielo despejado"** tanto en la validación como en las pruebas finales señalando en las matrices de confusión con los números altos de predicción debido a la existencia muchas imágenes similares en el dataset como se muestra en Figura 4-22 y **Figura**

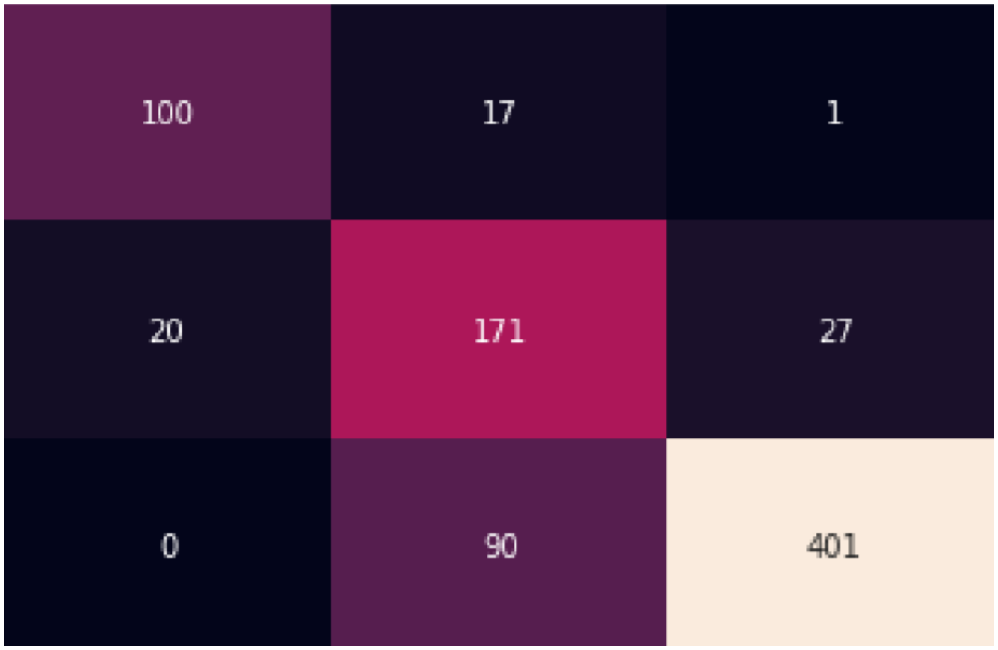


Figura 4-22 Matriz de confusión de 128x128, epochs de 100 y Batch size de 32 para el entrenamiento y validación con el dataset desequilibrado para etapa 4. (Fuente: Elaboración propia)

Figura 4-23 Matriz de confusión de 128x128, epochs de 100 y Batch size de 32 para las pruebas finales con el dataset desequilibrado para etapa 4. (Fuente: Elaboración propia)

Las siguientes pruebas son las pruebas aplicadas sobre **el dataset equilibrado**. Se utiliza la configuración de **128x128, epochs de 50 y Batch size de 32** de la segunda etapa que ha sacado los mejores valores de TOP-1 y TOP-5.

La siguiete tabla es la tabla de los resultados de la fase de entrenamiento, validación y prueba final aplicando sobre el dataset desequilibrado.

pruebas	Accuracy	Precision	F1	Recall
entre&val	0.8613	0.8706	0.8620	0.8617
prueba final	0.7702	0.8037	0.7769	0.7718

Tabla 4-11 Los resultados de 128x128, epochs de 50 y Batch size de 32 para el entrenamiento, validación y pruebas finales con el dataset equilibrado para etapa 4. (Fuente: Elaboración propia)

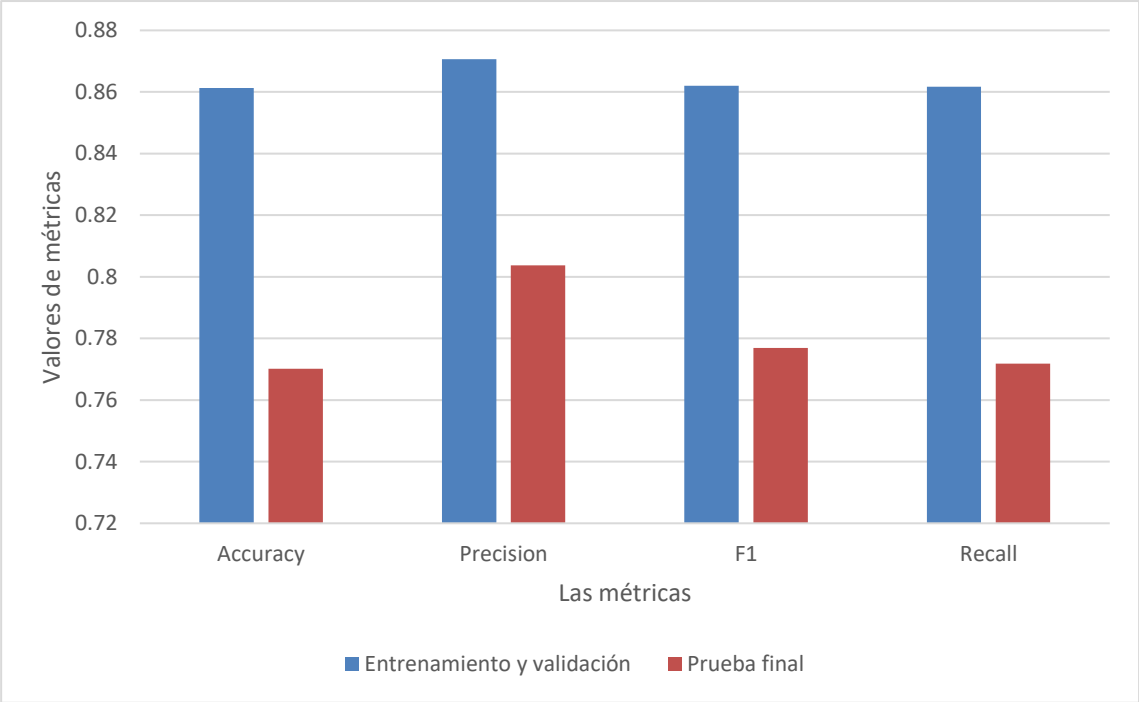


Figura 4-24 La gráfica de los resultados de la tabla 4-11. (Fuente: Elaboración propia)

Se puede observar que el nivel de las predicciones correctas aún es aceptable. Con esta configuración sigue teniendo problema de las imágenes similares en las clases parcialmente cubierto mostrando en su matriz de confusión en Figura 4-25 y Figura 4-26.

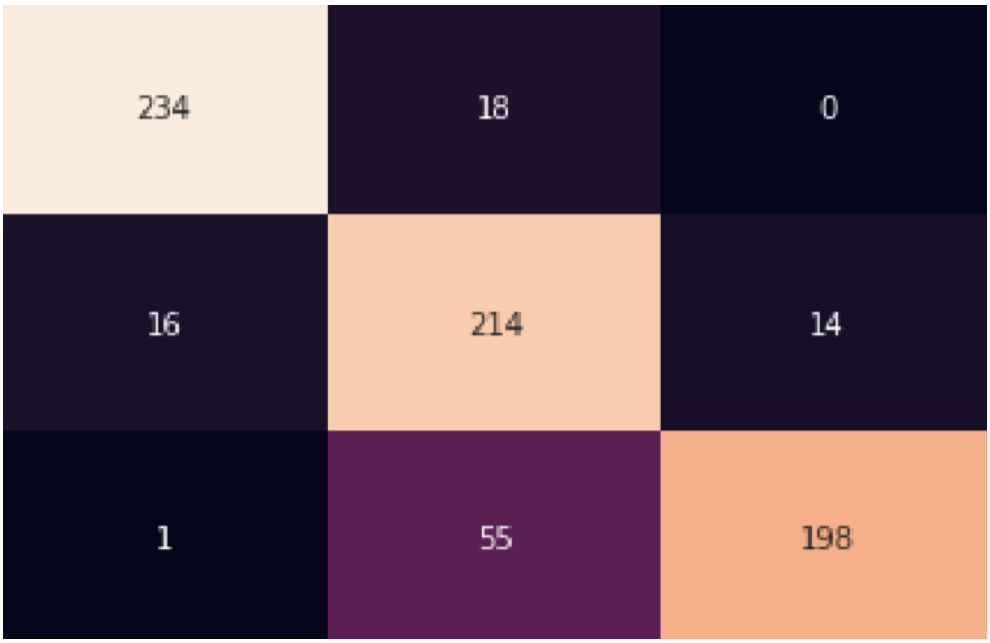


Figura 4-25 Matriz de confusión de 128x128, epochs de 50 y Batch size de 32 para el entrenamiento y validación con el dataset equilibrado para etapa 4. (Fuente: Elaboración propia)

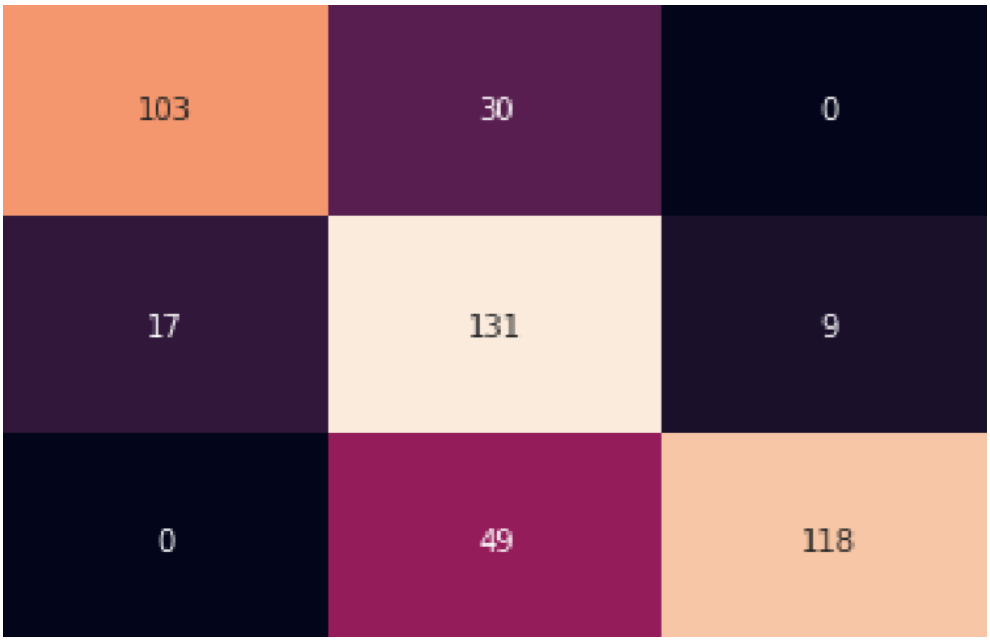


Figura 4-26 Matriz de confusión de 128x128, epochs de 50 y Batch size de 32 para las pruebas finales con el dataset equilibrado para etapa 4. (Fuente: Elaboración propia)

5 CONCLUSIONES Y LÍNEAS FUTURAS

En este apartado se explica las principales conclusiones que se ha obtenido en los apartados anteriores además se explica el alcance de este trabajo fin de grado comparando con los objetivos establecidos al principio y los límites principales que se ha obtenido durante las pruebas. Se presentan, así mismo, las líneas futuras que se puede seguir y ampliar con este trabajo fin de grado.

5.1 Conclusiones

Se ha podido observar que, al principio de trabajo, empezamos el trabajo con los límites iniciales como:

- El límite de la capacidad de RAM proporcionado por el Google Colab que nos limitan probar los modelos con altas resoluciones y altos números de parámetros entrenables.
- El dataset con las imágenes de las clases desequilibradas que nos dificultan en las pruebas porque no se ha podido asegurar que la distribución de las imágenes esté balanceada. La falta de imágenes en algunas clases también dificulta el aprendizaje del modelo.
- Las imágenes de las clases cercanas son muy similares y no se clasifica fácilmente incluso con los ojos humanos

Al empezar las pruebas con los 2 modelos de redes neuronales elegidos para clasificar los 15 tipos de cielo utilizando el dataset desequilibrado de 8.266 imágenes y equilibrado de 3.750 para el entrenamiento y validación. Se aplica 457 imágenes para la prueba final. Se ha observado que el uso del dataset desequilibrado con la resolución de 128x128, batch size de 32 y 100 epochs puede hacer predicción sacando mejor métricas de evaluación con Exactitud = 0,5792 , Precisión = 0,4575 , F1 = 0.4010 y Exhaustividad = 0,4031. Con el uso del dataset equilibrado con la resolución de 128x128, batch size de 32 y 50 epochs puede hacer predicción sacando mejor métricas de evaluación con Exactitud = 0,4485, Precisión = 0,4251, F1 = 0,4173 y Exhaustividad = 0,4550.

La clasificación de 3 tipos de cielo con el dataset desequilibrado utilizando la configuración de 128 x128,batch size de 32 y 100 epochs,el modelo puede hacer las predicciones sacando mejor métricas de evaluación con Exactitud = 0,8246 , Precisión = 0,8479 , F1 = 0,8307 y Exhaustividad = 0,8246.La clasificación con el dataset equilibrado con las configuraciones de 128x128,batch size de 32 y 50 epochs,el modelo puede hacer las predicciones sacando mejor métricas de evaluación con Exactitud = 0,7702 , Precisión = 0,8037 , F1 = 0,7769 y Exhaustividad = 0,7718.

Se puede observar los efectos del desequilibrio de las imágenes en todas las matrices de confusión que se confunde el modelo hacer las predicciones entre la clase correcta y las clases cercanas.

5.2 Líneas futuras

Se puede optimizar las predicciones del modelo en lugar de acumular los datos con las siguientes propuestas

- Mejorar las metodologías e infraestructuras para acumular las imágenes aplicando las nuevas técnicas y tecnologías que se utilizan en varios proyectos de Big Data. Luego, es necesario limpiar el conjunto de las imágenes para obtener el mismo número de imágenes por clase y así lograr un equilibrio en el número de imágenes por clase.
- Acumulación de más imágenes de cada clase con más variedad para que el modelo pueda aprender más con el dataset más grande

Actualmente, hay muchas empresas que tiene los servicios de procesamiento en el nube(Cloud Computing) ofrecidos a los clientes como Goolgle Cloud Platform [60], Amazon Web Services [61] y Huawei Public Cloud [62] . Al comprar los servicios, nos pueden ofrecer mejor calidad de Hardwares en la nube sin límite además nos pueden ofrecer los modelos de aprendizaje de máquina pre-entrenados y bien ajustado para clasificar adecuadamente que puede aumentar las predicciones correctas.

6 BIBLIOGRAFÍA

[1] S. Darula y R. Kittler, «CIE general sky standard difining luminance distributions,» de *The Canadian coference on vuilding energy simulation*, Montreal, 2002.

[2] A. Turing, «Computing Machinery and intelligence,» *Mind*, vol. 59, nº 236, pp. 433-460, 1950.

[3] R. K. Lindsay, B. G. Buchanan, E. A. Feigenbaum y J. Lederberg, Applications of Artificial intelligence for organic chemistry : The Dendral project, McGraw-Hil , inc., 1980.

[4] IBM, «IBM 100 : DEEP BLUE,» [En línea]. Available: https://www.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/.

[5] HONDA, «ASIMO Technical information,» Honda Motor Co.,Ltd., 2007.

[6] A. H. C. J. M. A. G. L. S. G. v. d. D. J. S. I. A. V. P. M. L. S. D. D. G. J. N. N. K. I. S. T. David Silver, «Mastering the game of Go with deep neural networks and tree search,» *Nature*, pp. 484-489, 2016.

[7] Deepmind, «Alphago,» nº https://deepmind.com/research/case-studies/alphago-the-story-so-far.

[8] p. o. AI, nº https://www.partnershiponai.org/.

[9] A. Géron, «The machine learning landscape : What is Machine Learning?,» de *Hands-on Machine Learning with Scikit-Learn,Keras, and TensorFlow : concepts,Tools and Techniques to build intelligent systems*, Sebastopol,CA, O'Reilly Media,Inc., 2019, p. 10.

[10] E. Argente, O. Sapena, V. Botti, J. M. Serra, A. Chica y A. Corma, «Aplicación de una red neuronal para la predicción de la reacción catálica isomerización del n-Octano,» de *Conferencia de la associación española para la inteligencia artificial (CAEPIA)*, 2001.

[11] P. Dangeti, Statistics for Machine Learning : Techniques for exploring supervised, unsupervised and reinforcement learning models with Python and R, Birmingham: Pack Publishing Ltd., 2017.

[12] L. Pessoa, A. Grunewald, H. Neumann y E. Littmann, «A biological neural network of visual cell Response : Static and Motion processing,» *Jounal of the Brazilian computer society*, vol. 4, nº 1, 1997.

[13] D. H. Hubel y T. N. Wiesel, «Receptive feilds,Binocular interaction and functional architecture in the cat's visual cortex,» *Journal Physics*, vol. 160, pp. 106-154, 1962.

[14] K. Fukushima, «Neocognitron: A Self-organizing Neural Network Model for a mechanism of pattern recognition unaffected by shift in position,» *Biological Cybernetics*, vol. 36, pp. 193-202, 1980.

[15] J. D. H. S. J. K. S. S. S. M. Z. H. A. K. A. K. M. B. A. C. B. a. L. F.-F. Olga Russakovsky, «ImageNet Large Scale Visual Recognition Challenge (ILSVRC),» *International journal of computer vision*, vol. 115, pp. 211-252, 2015.

[16] A. Krizhevsky, I. Sutskever y G. E. Hinton, «ImageNet classification with deep convolutional neural networks,» de *Advances in neural information processing systems*, 2012.

[17] K. He, X. Zhang, S. Ren y J. Sun, «Deep residual learning for image recognition,» Microsoft Research, 2015.

[18] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke y A. Rabinovich, «Going deeper with convolutions,» de *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, 2015.

[19] J. Hu, L. Shen, S. Albanie, G. Sun y E. Wu, «Squeeze and Excitation networks,» de *2018 IEEE Conference on Computer Vision and Pattern Recognition*, Salt Lake City, 2018.

[20] K. T. O'Shea y R. Nash, «An Introduction to Convolutional Neural Network,» 2015.

[21] G. Lin y W. Shen, «Research on convolutional neural network based on improved Relu piecewise activation function,» de *8th International Congress of Information and Communication Technology 2018*, Xiamen, 2018.

[22] C. E. Nwankpa, W. Ijoma, A. Gachagan y S. Marshall, «Activation Functions : Comparison of trends in practice and research for Deep Learning».

[23] A. Géron, «Introduction to Artificial Neural Networks with Keras : Multilayer perceptrons and Backpropagation,» de *Hands-on Machine Learning with Scikit-Learn, Keras and TensorFlow : Concepts, Tools and Techniques to buid intelligent systems*, Sebastopol,CA, O-Reilly Media,Inc., 2019, p. 288.

[24] A. Géron, «Chapter 10 : Introduction to Artificial Neural Networks with Keras : Classification MLPs, Chapter 4 : Traning Models : Softmax Regressing,» de *Hands-on Machine Learning with Scikit-Learn, Keras and TensorFlow : Concepts, Tools and Techniques to buid intelligent systems*, Sebastopol,CA, O-Reilly Media,Inc., 2019, pp. 149-152, 290-292.

[25] Keras, «Losses,» https://keras.io/losses/.

[26] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever y R. Salakhutdinov, «Dropout: A Simple Way to Prevent Neural Networks from Overfitting,» *Journal of machine learning research*, vol. 15, pp. 1929-1958, 2014.

[27] X. Zongben, Z. Hai, W. Yao, C. Xianyu y L. Yong, «L1/2 Regularization,» *Science China Information scieneces*, vol. 53, pp. 1159-1169, 2010.

[28] D. H. S. J. O. S. C. J. C. Y. Y. Sangdoo Yun, «CutMix : Regularization Strategy to Train Strong Classifiers with Localizable Features,» de *International conferece of computer vision 2019*, Seoul, 2019.

[29] S. Ioffe y C. Szegedy, «Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,» 2015.

[30] K. Simonyan y A. Zisserman, «Very Deep Convolutional Network for large-scale image recognition,» de *International Conference on Learning Representations(ICLR) 2015*, San Diego,CA, 2015.

[31] J. Ba y D. P. Kingma, «Adam : The method for stochastic optimization.,» de *The international conference on learning representation(ICLR) 2015*, San Diego,CA, 2015.

[32] S. Ruder, «An overview of gradient descent optimization algorithms,» 2017.

[33] S. Shalev-Shwartz y S. Ben-David, «Gradient Descent,» de *Understanding Machine Learning : From Theory to Algorithms*, New York,NY, Cambridge University press, 2014, pp. 185-193.

[34] S. Shalev-Schwartz y S. Ben-David, «SGD and Backpropagation,» de *Understanding Machine Learning : From Theory to Algorithms*, New York,NY, Cambridge University Press, 2014, pp. 277-281.

[35] GOOGLE, «Welcome to Colaboratory,» GOOGLE, [En línea]. Available: https://colab.research.google.com/notebooks/intro.ipynb#.

[36] GOOGLE, «Google Cloud : Cloud TPU,» GOOGLE, [En línea]. Available: https://cloud.google.com/tpu?hl=es-419.

[37] R. V. M. D. N. T. N. G.-B. B. V. H. C. D. A. P. P. R. f. Tiago Carneiro, «Performance Analysis of Google Colaboratory as a Tool for Accelerating Deep Learning Applications,» *IEEE Access*, 2018.

[38] GOOGLE, «Colaboratory : Frequently asked questions,» GOOGLE, [En línea]. Available: https://research.google.com/colaboratory/faq.html.

[39] Wannaphong, «Python 3 : ประวัติความเป็นมาของ Python,» 9 Septiembre 2017. [En línea]. Available: https://python3.wannaphong.com/2017/09/python.html.

[40] M. K. J. Michael Aivazis, «Python for Scientists and Engineers,» *Computing in Science & Engineering*, vol. 13, pp. 9-12, 2011.

[41] A. A. P. B. E. B. Z. C. C. C. Martín Abadi, «TensorFlow : Large-Scale Machine Learning on Heterogeneous Distributed Systems,» Preliminary white paper, 2015.

[42] TensorFlow, «Colab : An easy way to learn and use Tensorflow,» TensorFlow, 3 May 2018. [En línea]. Available: https://medium.com/tensorflow/colab-an-easy-way-to-learn-and-use-tensorflow-d74d1686e309.

[43] F. Chollet, «Keras,» 2015. [En línea]. Available: https://keras.io.

[44] S. C. C. G. V. Stefan van der Walt, «The Numpy Array : A structure of Numerical Computation,» *Computing in Science & Engineering* , vol. 13, nº 2, pp. 22-30, 2011.

[45] W. Mckinny, «Data Structures for Statistical Computing in Python,» de *the 9th Python in Science Conference,*, Austin,Texas, 2010.

[46] J. D. Hunter, «Matplotlib: A 2D Graphics Environment,» *Computing in Science & Engineering*, vol. 9, pp. 90-95, 2007.

[47] G. V. A. G. V. M. B. T. O. G. M. B. P. P. R. W. V. D. J. V. A. P. D. C. M. B. a. o. Fabian Pedregosa, «Scikit-Learn : Machine Learning in Python,» *Journal of Machine learning research* , vol. 12, pp. 2825-2830, 2011.

[48] SEABORN, «Seaborn : An introduction to seaborn,» SEABORN, [En línea]. Available: https://seaborn.pydata.org/introduction.html.

[49] G. C. COMPUTING, «Google cloud computing : Tensor processing unit(TPU),» 9 Diciembre 2019. [En línea]. Available: https://cloud.google.com/tpu/docs/tpus?hl=es-419.

[50] C. Y. N. P. a. o. Norman P. Jouppi, «In-Datacenter Performance Analysis of a Tensor Processing Unit,» de *The 44th International Symposium on Computer Architecture(ISCA)*, Toronto, 2017.

[51] A. R. B.-A. Aoautef Mahani, «Classification problem in imbalanced datasets,» IntechOpen, 2019.

[52] N. Japkowicz, «Learning form imbalanced datasets : A comparison of various strategies,» Daltech/Dalhousie University, Halifax, 2000.

[53] B. Rocca, «Towards data science : Handling imbalanced datasets in machine learning,» 27 Enero 2019. [En línea]. Available: https://towardsdatascience.com/handling-imbalanced-datasets-in-machine-learning-7a0e84220f28.

[54] S.-k. Learn, «Scikit-Learn : Metrics and scoring: quantifying the quality of predictions,» Scikit-Learn, 2007-2019. [En línea]. Available: https://scikit-learn.org/stable/modules/model_evaluation.html#classification-metrics.

[55] S. M. Lador, «Towards data science : What metrics should be used for evaluating a model on an imbalanced data set? (precision + recall or ROC=TPR+FPR),» 5 Septiembre 2017. [En línea]. Available: https://towardsdatascience.com/what-metrics-should-we-use-on-imbalanced-data-set-precision-recall-roc-e2e79252aeba.

[56] S. P. K. Gaurav Jaiswal, «Effects of Varying Resolution on Performance of CNN based Image Classification An Experimental Study,» *International Journal of computer sciences and engineering*, vol. 6, nº 9, pp. 451-456, 2018.

[57] B. C. Michal Koziarski, «Impact of low resolution on image recognition with deep neural networks : An experimental study,» *International Journal of applied math and Computer science*, vol. 28, nº 4, pp. 735-744, 2018.

[58] S. Dodge y L. Karam, «Understanding How Image Quality Affects Deep Neural Networks,» de *8th International Conference on Quality of Multimedia Experience (QoMEX) 2016*, Lisbon, 2016.

[59] T. R. Martinez y D. R. Wilson, «The general inefficiency of batch training for gradient descent learning,» *Neural Networks*, vol. 16, nº 10, pp. 1401-1544, 2003.

[60] G. C. PLATFORM, «Google cloud : Descripción general de Google Cloud Platform,» GOOGLE, 19 diciembre 2019. [En línea]. Available: https://cloud.google.com/docs/overview?hl=es.

[61] A. W. Services, «Información general sobre Amazon Web Services : Documentos tecnicos de AWS,» Amazon Web Services, Inc, 2018.

[62] H. T. Co..Ltd., «IDC Solutions white paper,» Huawei Technologies CO.,Ltd., Shenzhen, 2014.

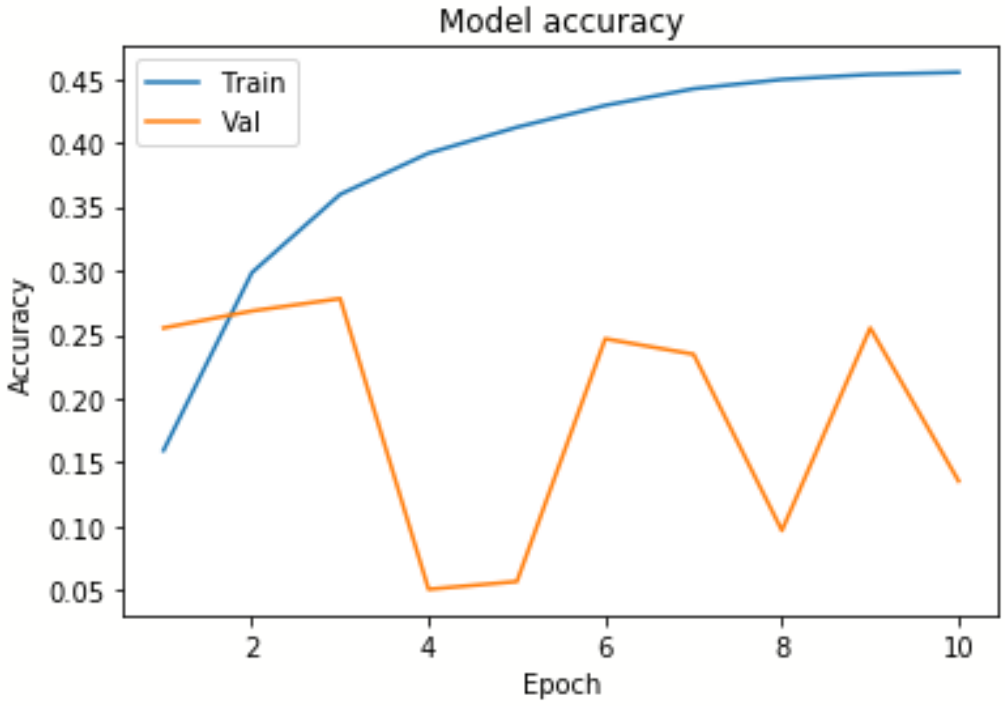
ANEXO I: LOS CÓDIGOS FUNDAMENTALES EMPLEADOS

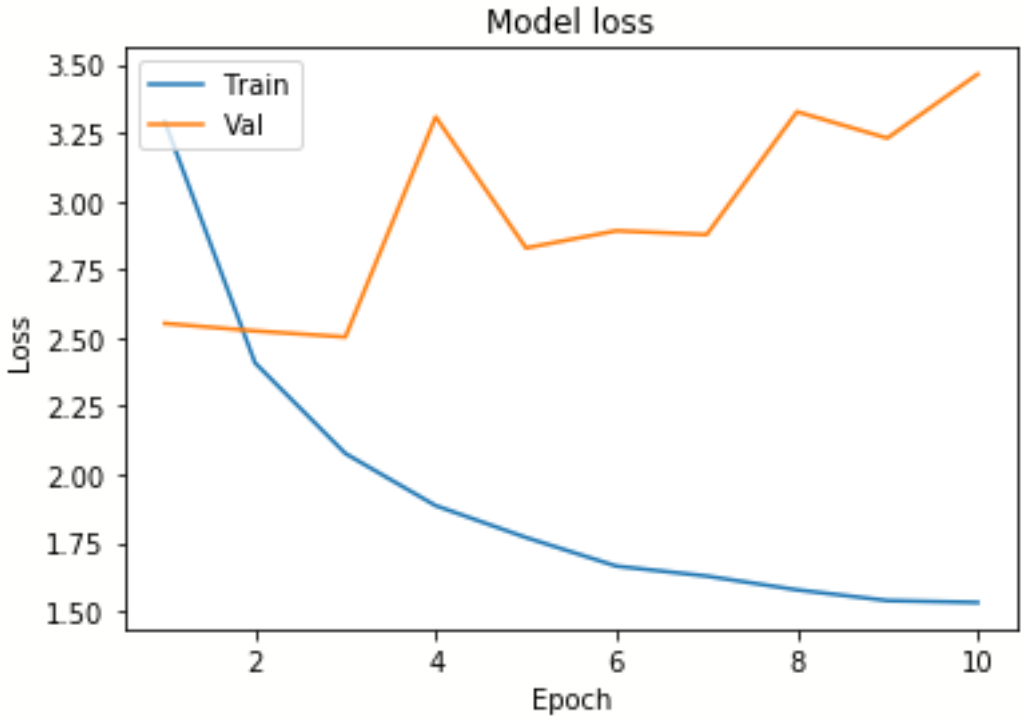
```
#importación de tensorflow 2.x que es TensorFlow 2.1.0
%tensorflow_version 2.x
import tensorflow as tf

#importación de las bibliotecas y herramientas
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Conv2D,MaxPool2D,BatchNormalization,Dropout,Dense,Flatten,Softmax
from tensorflow.keras.preprocessing import image
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import confusion_matrix,accuracy_score,precision_score,recall_score,f1_score
from sklearn.model_selection import train_test_split

#importación la visulización de progreso
from tqdm  import tqdm

#pre-procesado de la lista de imágenes y sus etiqutas del archivo CSV con el directorio
/content/CIE Class_Original to use.CSV
dataset = pd.read_csv('/content/CIE Class_Original to use.csv')

#unzip el dataset contenido de las imágenes con el directorio /content/DATASET.zip
!unzip /content/DATASET.zip

#pre-procesado de las imágenes del directorio extraido de /content/DATASET.zip antes de alimentar al modelo
(en el siguiente ejemplo se utiliza 256*256)
#el variable X es el tensor contenido de los arrays de las imágenes pre-procesado
anchura = 256
altura = 256

X=[]

for i in tqdm(range(datasets.shape[0])):
  ruta = '/content/DATASET/' + datasets['NAME'][i]
  img = image.load_img(ruta,target_size=(altura,anchura,1))
  img = image.img_to_array(img)
  img = img/255.0
  X.append(img)

X = np.array(X)
```

```
#pre-procesado de las etiquetas de las imágenes antes de alimentar las imágenes al modelo
#Se han quitado las columnas que no sean para las imágenes alimentadas al modelo
#La variable Y contiene las etiquetas
Y = datasets.drop(['NAME','CIE','Nublado','Parcial','despejado','TIPOS'],axis=1)
Y = Y.to_numpy()

#La distribución de las imágenes utilizando Scikit-Learn "train_test_split"
ratio_train = 0.8
ratio_val = 0.1
ratio_test = 0.1
x_remaining,X_test, y_remaining,Y_test = train_test_split(X, Y, test_size=ratio_test,shuffle=True,random_state=1)
ratio_remaining = 1 - ratio_test
ratio_val_adjusted = ratio_val / ratio_remaining
X_train, X_val, Y_train, Y_val = train_test_split(x_remaining, y_remaining, test_size=ratio_val_adjusted,random_state=
1)

#El modelo AlexNet(con adaptación propia)
CNN = Sequential()
CNN.add(Conv2D(32,kernel_size=(3,3),padding='valid',use_bias=False,activation='relu',input_shape=X_train[0].shape))
CNN.add(BatchNormalization())
CNN.add(MaxPool2D(2,2))
CNN.add(Dropout(0.3))
CNN.add(Conv2D(64,kernel_size=(3,3),use_bias=False,padding='valid',activation='relu'))
CNN.add(BatchNormalization())
CNN.add(MaxPool2D(2,2))
CNN.add(Dropout(0.3))
CNN.add(Conv2D(64,kernel_size=(3,3),use_bias=False,padding='valid',activation='relu'))
CNN.add(BatchNormalization())
CNN.add(MaxPool2D(2,2))
CNN.add(Dropout(0.3))
CNN.add(Conv2D(96,kernel_size=(3,3),padding='valid',use_bias=False,activation='relu'))
CNN.add(BatchNormalization())
CNN.add(MaxPool2D(2,2))
CNN.add(Dropout(0.4))
CNN.add(Conv2D(128,kernel_size=(3,3),use_bias=False,padding='valid',activation='relu'))
CNN.add(BatchNormalization())
CNN.add(MaxPool2D(2,2))
CNN.add(Dropout(0.5))
CNN.add(Flatten())
CNN.add(Dense(120,activation='relu',use_bias=False))
CNN.add(BatchNormalization())
CNN.add(Dropout(0.3))
CNN.add(Dense(120,activation='relu',use_bias=False))
CNN.add(BatchNormalization())
CNN.add(Dropout(0.4))
CNN.add(Dense(80,activation='relu',use_bias=False))
CNN.add(BatchNormalization())
CNN.add(Dropout(0.5))
CNN.add(Dense(15,activation='softmax'))
```

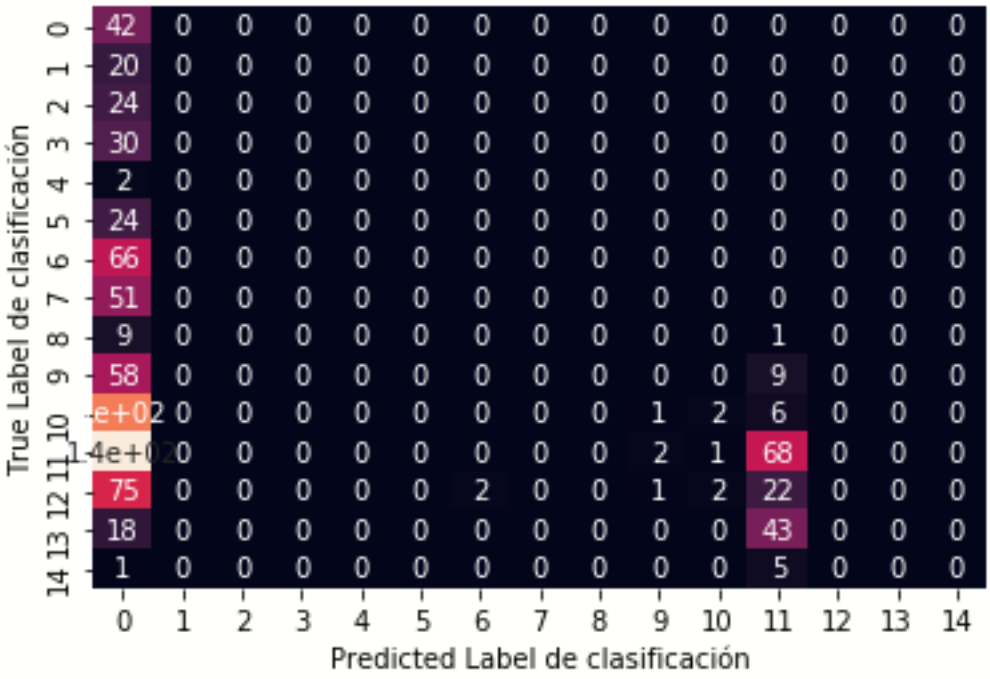
```
#El modelo VGG(con la adaptación propia)
CNN = Sequential()
CNN.add(Conv2D(16,kernel_size=(3,3),padding='valid',activation='relu',input_shape=X_train[0].shape))
CNN.add(Conv2D(16,kernel_size=(3,3),padding='valid',activation='relu',use_bias=False)
CNN.add(BatchNormalization())
CNN.add(Dropout(0.4))
CNN.add(Conv2D(32,kernel_size=(3,3),padding='valid',activation='relu'))
CNN.add(Conv2D(32,kernel_size=(3,3),padding='valid',activation='relu',use_bias=False)
CNN.add(BatchNormalization())
CNN.add(Dropout(0.4))
CNN.add(Conv2D(32,kernel_size=(3,3),padding='valid',activation='relu'))
CNN.add(Conv2D(32,kernel_size=(3,3),padding='valid',activation='relu',use_bias=False)
CNN.add(BatchNormalization())
CNN.add(MaxPool2D(2,2))
CNN.add(Dropout(0.5))
CNN.add(Conv2D(32,kernel_size=(3,3),padding='valid',activation='relu'))
CNN.add(Conv2D(32,kernel_size=(3,3),padding='valid',activation='relu',use_bias=False)
CNN.add(BatchNormalization())
CNN.add(MaxPool2D(2,2))
CNN.add(Dropout(0.5))
CNN.add(Conv2D(64,kernel_size=(3,3),padding='valid',activation='relu'))
CNN.add(Conv2D(64,kernel_size=(3,3),padding='valid',activation='relu',use_bias=False)
CNN.add(BatchNormalization())
CNN.add(MaxPool2D(2,2))
CNN.add(Dropout(0.5))
CNN.add(Conv2D(128,kernel_size=(3,3),padding='valid',activation='relu'))
CNN.add(Conv2D(128,kernel_size=(3,3),padding='valid',activation='relu',use_bias=False)
CNN.add(BatchNormalization())
CNN.add(MaxPool2D(2,2))
CNN.add(Dropout(0.5))
CNN.add(Flatten())
CNN.add(Dense(1000,activation='relu',use_bias=False)) CNN.add(BatchNormalization()) CNN.add(Dropout(0.4))
CNN.add(Dense(15,activation='softmax'))

#La indicación de La función de pérdidas,Optimizador y las métricas para visualizar el entrenamiento

CNN.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])

#El entrenamiento y los progresos
ModelValidation = CNN.fit(X_train,Y_train,epochs=10,validation_data=(X_val,Y_val),batch_size=128,verbose=1)
```

```
#Visualización de las gráficas de entrenamiento y validación de las pérdidas y la Exactitud
epoch = 10
def plot_learningCurve(ModelValidation, epoch):
  epoch_range = range(1, epoch+1)
  plt.plot(epoch_range, ModelValidation.history['accuracy'])
  plt.plot(epoch_range, ModelValidation.history['val_accuracy'])
  plt.title('Model accuracy')
  plt.ylabel('Accuracy')
  plt.xlabel('Epoch')
  plt.legend(['Train', 'Val'], loc='upper left')
  plt.show()

  plt.plot(epoch_range, ModelValidation.history['loss'])
  plt.plot(epoch_range, ModelValidation.history['val_loss'])
  plt.title('Model loss')
  plt.ylabel('Loss')
  plt.xlabel('Epoch')
  plt.legend(['Train', 'Val'], loc='upper left')
  plt.show()

print('training and validation curves')
print(plot_learningCurve(ModelValidation, epoch))
```

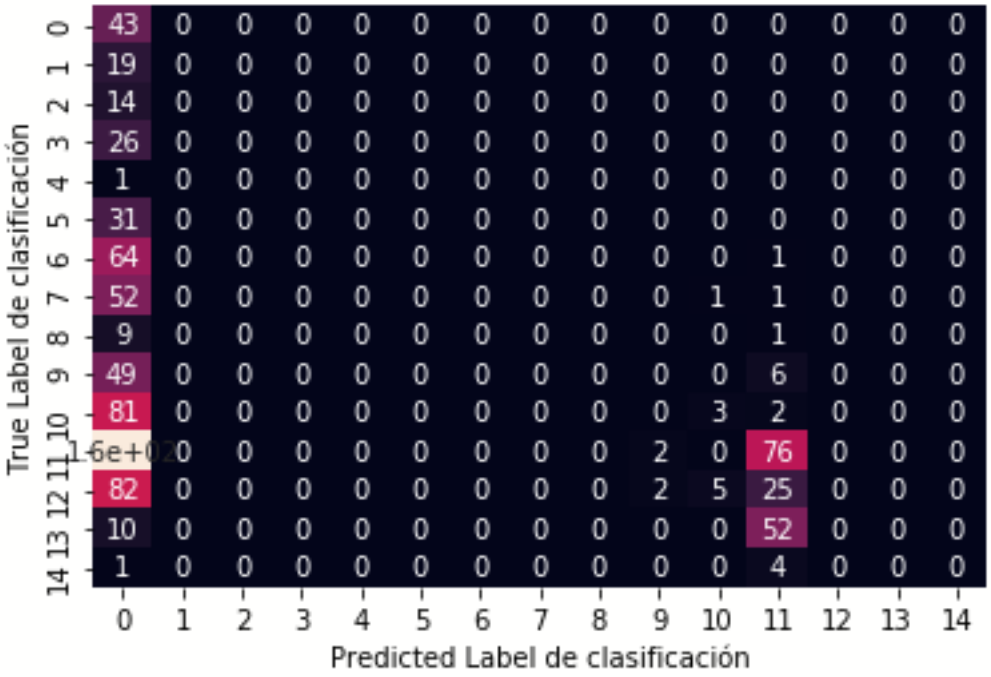





```
#Las predicciones de la validación
Y_pred_forValidation = CNN.predict(X_val)
Y_pred_oneDigit = np.argmax(Y_pred_forValidation,axis=1)
Y_val_oneDigit = np.argmax(Y_val,axis=1)
#matriz de confusión de la validación
Confusion_Matrix_for_validation = confusion_matrix(Y_val_oneDigit,Y_pred_oneDigit)
print(Confusion_Matrix_for_validation)
#mapa de calor de matriz de confusión
sns.heatmap(Confusion_Matrix_for_validation,annot=True,cbar=False)
plt.ylabel('True Label de clasificación')
plt.xlabel('Predicted Label de clasificación')
```




```
#Las métricas Exactitud,Precisión,F1,Exhaustividad con el "average" ajustado
print('accuracy for validation is :',accuracy_score(Y_val_oneDigit,Y_pred_oneDigit))
print('f1 weighted : ',f1_score(Y_val_oneDigit, Y_pred_oneDigit, labels=None, average="weighted"))
print('f1 micro : ',f1_score(Y_val_oneDigit, Y_pred_oneDigit, labels=None, average="micro"))
print('f1 macro : ',f1_score(Y_val_oneDigit, Y_pred_oneDigit, labels=None, average="macro"))
print('recall weighted : ',recall_score(Y_val_oneDigit, Y_pred_oneDigit, labels=None, average="weighted"))
print('recall micro : ',recall_score(Y_val_oneDigit, Y_pred_oneDigit, labels=None, average="micro"))
print('recall macro : ',recall_score(Y_val_oneDigit, Y_pred_oneDigit, labels=None, average="macro"))
print('precision weighted : ',precision_score(Y_val_oneDigit, Y_pred_oneDigit, labels=None, average="weighted"))
print('precision micro : ',precision_score(Y_val_oneDigit, Y_pred_oneDigit, labels=None, average="micro"))
print('precision macro : ',precision_score(Y_val_oneDigit, Y_pred_oneDigit, labels=None, average="macro"))


#la predicción final con las imágenes de pruebafinal
Y_testing = CNN.predict(X_test)
Y_testing_oneDigit = np.argmax(Y_testing,axis=1)
Y_test_oneDigit = np.argmax(Y_test,axis=1)
#matriz de confusión de la prueba final
Matrix_Confusion_forTesting = confusion_matrix(Y_test_oneDigit,Y_testing_oneDigit)
print(Matrix_Confusion_forTesting)
#mapa de calor de matriz de confusión
sns.heatmap(Matrix_Confusion_forTesting,annot=True,cbar=False)
plt.ylabel('True Label de clasificación')
plt.xlabel('Predicted Label de clasificación')
```




```
#Las métricas Exactitud,Precisión,F1,Exhaustividad con el "average" ajustado
print('accuracy for validation is :',accuracy_score(Y_test_oneDigit,Y_testing_oneDigit))
print('f1 weighted : ',f1_score(Y_test_oneDigit,Y_testing_oneDigit, labels=None, average="weighted"))
print('f1 micro : ',f1_score(Y_test_oneDigit,Y_testing_oneDigit, labels=None, average="micro"))
print('f1 macro : ',f1_score(Y_test_oneDigit,Y_testing_oneDigit, labels=None, average="macro"))
print('recall weighted : ',recall_score(Y_test_oneDigit,Y_testing_oneDigit, labels=None, average="weighted"))
print('recall micro : ',recall_score(Y_test_oneDigit,Y_testing_oneDigit, labels=None, average="micro"))
print('recall macro : ',recall_score(Y_test_oneDigit,Y_testing_oneDigit, labels=None, average="macro"))
print('precision weighted : ',precision_score(Y_test_oneDigit,Y_testing_oneDigit, labels=None, average="weighted"))
print('precision micro : ',precision_score(Y_test_oneDigit,Y_testing_oneDigit, labels=None, average="micro"))
print('precision macro : ',precision_score(Y_test_oneDigit,Y_testing_oneDigit, labels=None, average="macro"))

#TOP-1 y TOP-5
def top_n_accuracy(preds, truths, n):
    best_n = np.argsort(preds, axis=1)[:,-n:]
    ts = np.argmax(truths, axis=1)
    successes = 0
    for i in range(ts.shape[0]):
      if ts[i] in best_n[i,:]:
        successes += 1
    return float(successes)/ts.shape[0]

preds = Y_testing
truths = Y_test

print('top-1 is ',top_n_accuracy(preds,truths,n=1))
print('top-5 is ',top_n_accuracy(preds,truths,n=5))
```