1. Introducción

1.1 Planteamiento del problema y objetivos

Uno de los microfósiles más utilizados para estudios biostratigráficos, evolutivos, paleogeográficos, y paleocológicos de la era Paleozoico son los conodontes (Albanesi, 2017), los cuales son muy abundantes, diversos y resistentes a los procesos diagenéticos (Molina, 2002). Estos pequeños fósiles formaban parte del aparato oral del animal conodonte que tenía una forma era alargada y una columna vertebral rudimentaria (Navas-Parejo and Martínez Pérez, 2017). Además, se caracterizaban por tener tres morfologías bien diferenciadas (coniformes, ramiformes, y pectiniformes) ocupando 3 posiciones diferentes (P, S y M) dentro del aparato oral del animal conodonte. En particular, el elemento en posición P es el que tiene valor taxonómico para la clasificación de los distintos taxones de conodontes (Sweet, 1988).

La calidad del registro fósil conodonte frecuentemente está afectada por sesgos de distinta naturaleza que pueden falsear su distribución espacio-temporal (von Bitter and Purnell, 2005). Entre ellos se encuentran los sesgos en la identificación de los elementos conodontales (Purnell and Donoghue, 2005) debido a que la asignación de estos microfósiles a un determinado taxón está fundamentalmente basada en el criterio de los taxónomos. Esto supone un cierto grado de subjetividad en la identificación de especies difícil de superar sin la utilización de métodos computacionales que automaticen esta tarea. Entre los escasos trabajos de clasificación automática de conodontes que existen, se encuentra el reciente estudio de Duan (2023) que utiliza redes neuronales convolucionales para clasificar algunas especies pertenecientes a un solo género. Por lo tanto, el objetivo del presente estudio es desarrollar un código que analice una mayor cantidad de especies e identifique a que género pertenecen las distintas imágenes incluidas en la base de datos.

2. Herramientas computacionales utilizadas

2.1. Construcción de la base de datos

La base de datos se obtuvo a partir de una exhaustiva búsqueda bibliográfica de fotografías de conodontes (elementos P) publicadas en trabajos digitales de revistas indexadas. Un total de 200 imágenes correspondientes a 200 especies repartidas de manera equilibrada en 50 géneros de conodontes coniformes fueron seleccionadas. Luego las láminas donde se encontraban esas especies fueron copiadas a partir de una captura de pantalla y se pegaron en Paint donde se cortaron las imágenes seleccionadas para incluirlas en la base de datos (**Fig. 1**). Además, esas imágenes se convirtieron a escala de grises y el fondo se igualo a un color blanco utilizando Photoshop. Por otro lado, el peso de las mismas se redujo igualando el tamaño de todas las imágenes utilizando Bulk Imagen Resizer (2.0) para disminuir su tiempo de procesamiento. Cabe destacar que ese número de imágenes se incrementó mediante la rotación de las mismas en 3 ángulos diferentes (5°, 10°, y 15°) de manera automática utilizando algoritmos descriptos posteriormente aquí. En particular, se seleccionaron ángulos de

rotación leves para que la imagen original no se distorsione demasiado y no perder información importante. El aumento del número de datos mediante esa técnica se realizó con el objetivo de aumentar el rendimiento del modelo ya que la base de datos inicialmente obtenida probablemente no es suficiente para obtener un alto rendimiento (Shorten and Khoshgoftaar, 2019). Finalmente, este procedimiento permitió obtener un total de 800 imágenes siendo cada género representado por 16 imágenes correspondientes a 4 especies diferentes. Cabe destacar que las imágenes rotadas y originales se normalizaron y redimensionaron automáticamente para facilitar la manipulación de las mismas durante el entrenamiento del modelo.

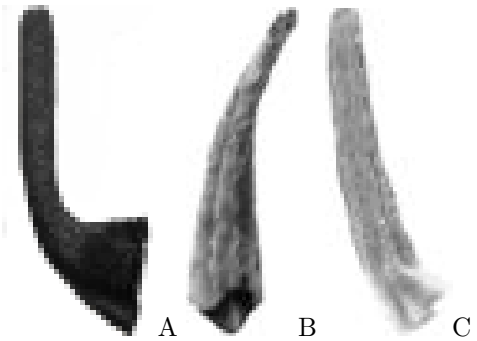

Fig. 1. Algunas de las imágenes incluidas en la base de datos. **A.** *Acanthodus humachensis*. **B.** *Gapparodus cuneata* **C.** *Decoriconus mercurius*.

2.2. Características de los métodos y herramientas computacionales utilizadas

Las redes prototípicas de pocos disparos (few shot learning) permite clasificar nuevas imágenes en función de su similitud con representaciones prototípicas generadas a partir de unas pocas imágenes de entrenamiento (Ji *et al.*, 2020). En el caso de que se dispongan de pocos datos como en el presente estudio, ese enfoque es considerado más simple y más eficiente que otros algoritmos de metaaprendizaje tradicionales (Snell *et al.*, 2017). Por lo tanto, aquí se seleccionó un modelo que combina características prototípicas con características aprendidas por la red neuronal para clasificar los distintos géneros de conodontes. Entre los lenguajes de programación disponibles se seleccionó Phyton porque es el más utilizado en el aprendizaje automático (machine learning) por su estructura de datos de alto nivel, legibilidad, portabilidad y versatilidad. Además, es un software libre que es fácil de aprender ya que su sintaxis es sencilla y permite crear un código a partir de unas pocas líneas siendo adecuado y accesible para principiantes (Raschka *et al.*, 2020). El código Phyton se escribió y se ejecutó en un entorno de Google Colab. Esta herramienta fue seleccionada porque no necesita de una configuración previa para desarrollar el código y no tiene costos adicionales. Además, ese navegador es considerado uno de los más aptos para tareas de aprendizaje automático ya que permite acelerar el tiempo de ejecución para entrenar una red neuronal convolucional (convolutional neural networks) (CNN) (Carneiro *et al.*, 2018). Entre las librerías importadas para utilizar funciones

predefinidas relacionadas con el procesamiento de las imágenes se encuentran el modulo **os** que permite manipular archivos, directorios etc., la librería **OpenCV** (**cv2)** que permite analizar y manipular las imágenes y el módulo **NumPy** que permite realizar operaciones matemáticas sobre los datos obtenidos a partir de las imágenes. Por otro lado, entre las librerías importadas para el modelado se encuentran **Tensorflow**, **Keras** y **Sklearn** que contienen funciones predefinidas relacionados con la construcción, el entrenamiento y evaluación de los modelos de aprendizaje automático y redes neuronales. Finalmente, la librería **Matplotlib** fue utilizada para crear y guardar graficos.

2.3. Descripción de los algoritmos utilizados

2.3.1. Preparación de las imágenes antes del entrenamiento

Se crearon un total de 7 bucles for para iterar sobre los elementos de las listas. La estructura de cada uno de esos bucles for se caracteriza por incluir una variable de iteración que en cada repetición toma el valor de un elemento de la lista. Además cada iteración del bucle construye una ruta con la función os.path.join que apunta a una carpeta especifica previamente definida para guardar cada uno de esos elementos. En algunos bloques se definieron otras variables aparte de la variable de iteración como en el bucle for que itera sobre una lista de 3 ángulos de rotación (5°, 15° y 20°) para rotar cada una de las imágenes de la base de datos en esos ángulos. Allí se definieron las variables nombradas como matriz_rotación e imagen_rotada. La primera utiliza la función cv2.getRotationMatrix2D() cuyo primer argumento define el punto central de rotación de la imagen numero_rotacion, 1). En este caso se utilizó la mitad del ancho (imagen.shape[1] / 2, y la mitad de la altura imagen.shape[0] / 2) de cada imagen como centro de rotación. El segundo argumento numero_rotación es el ángulo de rotación en grados que se aplicará a cada imagen y el último argumento 1 indica que se mantendrá la escala original de las imágenes. La variable imagen_rotada utiliza la función cv2.warpAffine() que aplica la transformación de la matriz matriz_rotación a la imagen original. Los argumentos de esa función son 'imagen' que es la imagen que se va a rotar; 'matriz_rotacion' que es la matriz de transformación que describe la rotación deseada y por ultimo '(imagen.shape[1], imagen.shape[0]' que especifica el tamaño de salida de las imágenes. Finalmente, la función cv2.warpAffine() devuelve las imágenes resultantes después de aplicar la rotación especificada por la matriz de rotación.

Otro bucle en el cual se definen otras variables aparte de la variable de iteración es en el bucle en el que se construyen los prototipos de cada clase obtenidos a partir de las imágenes de soporte. En ese bloque primero se itera sobre cada etiqueta única presente en la base de datos representada por unique_labels. Luego se obtienen los índices de muestras con la etiqueta actual label_indices donde las etiquetas coinciden con la etiqueta actual en el bucle utilizando la función np.where(), cuyo argumento '(unique_labels == label)' compara cada elemento en el arreglo 'unique_labels' con el valor de 'label' Esa comparación retorna un arreglo booleano siendo 'True' en las posiciones donde 'unique_labels' es igual a 'label' y 'False' en las posiciones donde no se cumple esa condición. El argumento

[0] se utiliza para obtener el arreglo de índices de la tupla que contiene los valores booleanos devuelta por la función np.where(). A continuación, se define la variable label_samples para extraer las muestras correspondientes a la etiqueta actual utilizando la función support_set[] cuyo argumento '[label_indices] ' indica que se utilizaron los índices contenidos en 'label_indices' para realizar esa tarea. Luego se define la variable 'prototipo' para calcular el prototipo de la etiqueta actual mediante la función tf.reduce_mean(), El argumento de esa función 'label_samples' indica que la misma calcula el promedio de las muestras correspondientes a la etiqueta actual a lo largo del eje X '(axis=0)' contenidas en 'label_samples'. Por último, cada prototipo calculado se almacena en el diccionario de prototipos 'prototypes[label] = prototype' utilizando la etiqueta 'label' como clave para acceder posteriormente a ese prototipo.

2.3.2. Entrenamiento de la red neuronal

El modelo de red neuronal en Keras se definió como una pila lineal de capas iniciándose con una instancia de Sequential() a donde luego se pasa una lista de capas. La primera denominada keras.layers.Flatten() se encarga de aplanar la entrada en un vector unidimensional. El argumento '(input_shape=(500, 250, 3))' indica que se esperan entradas con forma (500, 250, 3) ya que las imágenes tienen un tamaño de 500x250 pixeles con 3 canales RGB. En la segunda capa denominada keras.layers.Dense sus 128 neuronas están conectadas con las neuronas de la capa anterior. El primer argumento '128' representa el número de neuronas en esa capa y el segundo argumento activation='relu' se utiliza para especificar la función de activación que se aplicará a las salidas. Luego se creó una tercera capa keras.layers.Dense() con 64 neuronas y también se utilizó la función de activación Relu. La última capa es también una capa densa, pero con 50 neuronas (que corresponden a las 50 clases del presente estudio) y en este caso se utiliza la función de activación sofmax activation='softmax'. En particular, esa función produce salidas que representan las probabilidades de pertenencia a cada una de las 50 clases disponibles. A continuación, se prosiguió a la compilación del modelo utilizando la función model.compile(). Entre los argumentos se encuentran el que define el optimizador a utilizar durante el entrenamiento optimizer='adam'. Además, para evaluar el rendimiento del modelo se utilizó la función de perdida Sparse Categorical indicada con el argumento Crossentropy loss='sparse_categorical_crossentropy'. Por otro lado, la métrica utilizada para evaluar el rendimiento del modelo durante el entrenamiento fue accuracy. Esta fue indicada en el argumento como metrics=['accuracy']. A continuación, la función model.fit() se utilizó para entrenar el modelo con los datos de soporte 'support_set y las etiquetas de soporte 'support_labels. Además, se especificó el número de épocas 'epochs=10' que es la cantidad de veces que se repetirá el proceso en el entrenamiento en el conjunto de datos completo. Otro parámetro especificado es el 'bach_size=32' el cual es el tamaño del lote el cual determina cuantos ejemplos se utilizan a la vez para actualizar los pesos del modelo durante cada iteración de entrenamiento.

Finalmente, el algoritmo utilizado para crear la gráfica de precisión en función de

las épocas incluyo la utilización de la función plt.plot() para definir su tamaño ‘(figsize=(10, 6))’. En este caso se seleccionó un ancho de 10 unidades y una altura de 6 unidades. Luego se utilizó la función plt.plot() para trazar la gráfica cuyo argumentos incluyen la variable ‘épocas’ y la variable ‘precisión’. Además, se utilizó un marcador circular marker='o' para los puntos en la línea y una línea sólida para conectar los puntos linestyle='-'. El título “Precisión a lo largo de las épocas” de la gráfica se estableció utilizando la función plt.title() cuyo argumento indica ese título 'Precisión a lo largo de las épocas'. Las funciones plt.xlabel() y plt.ylabel() fueron utilizadas para etiquetar el eje X ‘Época’ e Y ‘Precisión’, respectivamente. Por otro lado, se agregó una cuadricula al gráfico para facilitar la lectura de los valores mediante la función plt.grid() con el argumento ‘True’, el cual activa la grilla permitiendo su visualización.

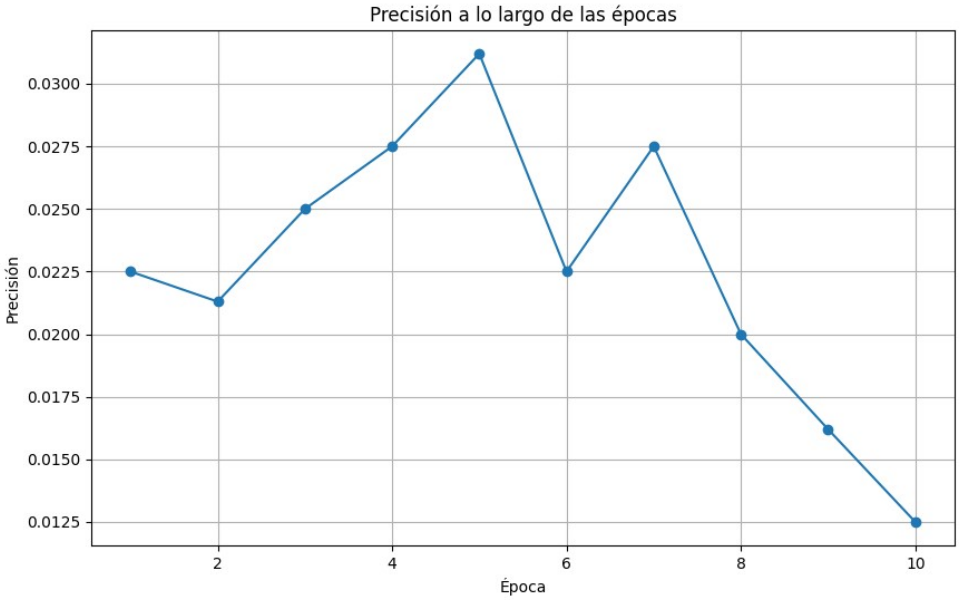

Fig. 2. Gráfico de precisión vs. numero de épocas.

3.Resultados

Los resultados muestran que se procesaron exitosamente los 32 lotes de datos programados para cada una de las 10 épocas. Por ejemplo, la pérdida (loss) en la primera época es relativamente alta (114.0110). Esta pérdida inicial alta sugiere que las predicciones del modelo están muy alejadas de las etiquetas reales, lo que indica un bajo rendimiento en esta época. En lo que refiere a la precisión (accuracy) es del 2.75%, lo que significa que el modelo acertó en un pequeño porcentaje de las predicciones en esta época. A medida que el entrenamiento avanza, la pérdida disminuye y la precisión aumenta (Fig. 2), lo que indica que

el modelo de clasificación de géneros de conodontes, está aprendiendo de los datos y mejorando su capacidad para hacer predicciones precisas. Sin embargo, cabe destacar que incluso al final de las 10 épocas, la pérdida sigue siendo relativamente alta y la precisión es modesta. Esto podría sugerir que el modelo necesita más ajustes (ej Aumentar el número de imágenes de soporte) o una arquitectura diferente para obtener un rendimiento mayor en la clasificación de los distintos géneros de conodontes.

Referencias

Albanesi, G., 2017. Clase conodonta: guía didáctica para el estudio de conodontes / Guillermo Luis Albanesi. - 1a ed. ampliada. -Córdoba: Universidad Nacional de Córdoba. Facultad de Ciencias Exactas, Físicas y Naturales, CD-ROM, DOC ISBN 978-950-33-1364-0 1. Geología. 2. Paleontología. 3. Fósiles. I. Título.CDD 560.

Carneiro, T., Medeiros Da Nobrega, R.V., Nepomuceno, T., Bian, G.-B., De Albuquerque, V.H.C., Filho, P.P.R., 2018. Performance Analysis of Google Colaboratory as a Tool for Accelerating Deep Learning Applications. IEEE Access, 6, 61677–61685.

Duan X., 2023. Automatic identification of conodont species using fine-grained convolutional neural networks. *Front. Earth Sci.* 10:1046327. https://doi.org/10.3389/feart.2022.104632.

Molina E., (Ed.), 2002. *Micropaleontologia. Prensa Universitarias de Zaragoza.* Textos docentes 93, 674 p.

Purnell , M. A., Donoghue , P. C. J., 2005. Between death and data: Biases in interpretationof the fossil record of conodonts. In: Purnell , M. A., and Donoghue , P. C. J. (Eds). *Special Papers in Palaeon-tology Series,Conodont biology and phylogeny: interpreting the fossil record*. Palaeontological Association, London, 218 pp.

Raschka, S., Patterson, J., Nolet, C., 2020. Machine Learning in Python: Main Developments and Technology Trends in Data Science, Machine Learning, and Artificial Intelligence. *Information*, 11, 193, https://doi.org/10.3390/info11040193.

Shorten, C., Khoshgoftaar, T.M., 2019. A survey on Image Data Augmentation for Deep Learning. J *Big Data,* 6, 60. https://doi.org/10.1186/s40537-019-0197-0.

Snell J., Swersky K., Zemel, R. S., 2017. Prototypical networks for few-shot learning. In P*roceedings of the 31st International Conference on Neural Information Processing Systems*. Curran Associates Inc., 4080–4090.

Sweet, W.C., 1988. The Conodonta: morphology, taxonomy, paleoecology, and evolutionary history of a long-extinct animal phylum. *Oxford Monographs on Geology and Geophysics*, Clarendon Press, Oxford, 212 pp.

Von Bitter , P. H., Purnell , M. A., 2005. An exper-imental investigation of post-depositional taphonomic bias inconodonts. In: Purnell, M. A. and Donoghue , P.

C. J. (Eds). *Special Papers in Palaeontology,Conodont Biology and Phylogeny: interpreting the fossil record*. Palaeontological Association, London, 218 p.