”TRANSFORMER PARA NOVATOS: RECONOCIMIENTO DE ESPECIES DE AVES MEDIANTE UN ViT”

José Pérez Rodríguez
Dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
josperrod9@alum.us.es

Alberto Monedero Martín
Dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
albmonmar3@alum.us.es

***Resumen*—En éste trabajo, se han realizado tareas de investigación y documentación, así como de una posterior implementación del modelo Transformer y el modelo de Red Neuronal Convolucional, con el objetivo, no solo de aprender acerca de dichas técnicas, sino también de obtener competencias relacionadas con el aprendizaje autónomo(como el uso de Pytorch) y la producción de un artículo científico. Mediante este artículo explicaremos en la mayor medida posible todos los conocimientos obtenidos durante esta ardua tarea de investigación pasando de una explicación teórica acerca del funcionamiento de los modelos anteriormente nombrados, hasta una descripción de toda la implementación necesaria para el funcionamiento de estos modelos.**

***Palabras clave*—Inteligencia Artificial, identificación de especies de pájaro, preprocesado de imágenes, transformer, atención, red neuronal convolucional.**

I. Introducción

En el campo del Deep Learning, han existido avances significativos en la última década, pero es en 2017, cuando una nueva publicación, ”Attention is all you need” [1] transforma nuestra concepción de lo que la Inteligencia Artificial es capáz de hacer. Aparecen así los Transformers, y desde ese momento, muchos logros en el campo de la Inteligencia Artificial se apoyan en ésta tecnología,mejorando el ”state of the art”, como por ejemplo Tesla, con su sistema de conducción Auto Pilot o GPT3, que permite generar textos que simulan la redacción humana.

El objetivo de éste trabajo es el de desarrollar un sistema capáz de realizar predicciones sobre un dataset de imágenes de aves, con el objetivo de identificar a que tipo de ave pertenece cada imágen. Para ello, se han implementado dos soluciones, utilizando en primer lugar el Transformer, y en segundo lugar, una red convolucional, para poder así comparar el desempeño de ambos mecanismos en la resolución de un mismo problema.

En cuanto al contenido del documento en cuestión, éste se divide en las siguientes secciones:
-II. Principios Teóricos: En esta sección, se explican los conceptos teóricos en los que se basa El Transformer.

-III. Metodología: Sección dedicada a describir el método implementado en el trabajo, aterrizando dichos conceptos teóricos a la solución propuesta.

-IV. Resultados: En esta sección se detallan tanto los experimentos realizados como los resultados obtenidos.

-V. Conclusiones: Sección en la que se indican las conclusiones obtenidas con la realización del trabajo.

II. Principios teóricos

Durante nuestra investigación, varios mecanismos han sido usados tanto para la creación del Transformer, como para la red convolucional usada en la comparativa del rendimiento. A continuación, describiremos todos estos mecanismos desde un punto de vista teórico. Recomendamos los videos del canal Dot CSV acerca del Transformer [2] para entender mejor el enfoque teórico de dichos conceptos.

A. Attention

Un mecanismo de atención, en el contexto de las redes neuronales, consiste en una operación matemática que recibe como inputs un conjunto de vectores, que pueden representar texto, imágenes o cualquier tipo de datos con el que trabajemos (dicha codificación vectorial se conoce como Embedding), y nos da como resultado otro conjunto de vectores. Este resultado dependerá, obviamente, del tipo de mecanismo de atención que utilizemos. Vamos a ver algunos ejemplos.

1) Hard-Attention: El mecanismo de atención más sencillo de entender es el conocido como hard attention mechanism. En este tipo de atención generaremos un número de vectores a la salida igual a los de la entrada (de ahora en adelante asumiremos que este es siempre el caso a no ser que se indique lo contrario) en el que cada output atenderá únicamente a su correspondiente vector a la entrada. O dicho de otra forma, este mecanismo de atención no produce nada nuevo, produce

a la salida lo mismo que recibe a la entrada. Consideramos el conjunto de vectores siguiente.

$$\vec{a} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \vec{b} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \quad \vec{c} = \begin{pmatrix} -0.5 \\ -0.5 \end{pmatrix}$$

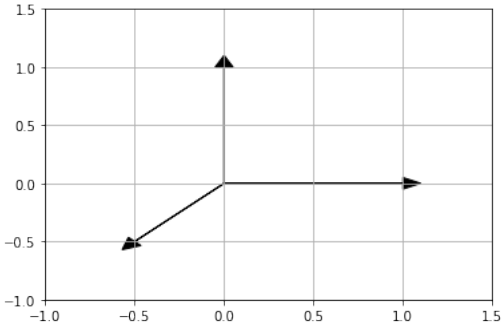

Fig. 1. Disposición de los vectores en el espacio

Un mecanismo de hard attention prestará atención a un único vector. Podemos representarlo como un vector en el que, en cada posición, tenemos el peso relativo de cada vector a la entrada. En este caso, todos los valores serán 0 excepto el que se encuentre en la mismo posición del vector al que queremos prestar atención. Para aplicar nuestro mecanismo de atención, simplemente multiplicamos nuestro conjunto de vectores por el vector de atención. En este caso solo pondremos atencion al vector $\vec{a}$.

$$\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \times \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ -0.5 & -0.5 \end{pmatrix}$$

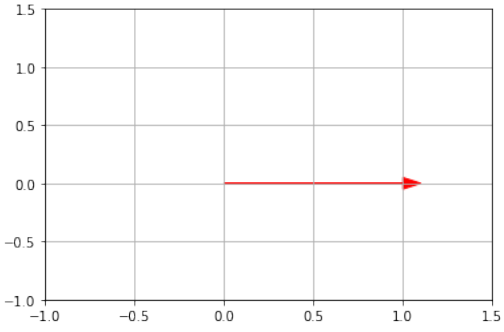

Fig. 2. Vectores multiplicados por vector one-hot

Podemos aplicar este mecanismo en una sola operación a todos los vectores generando una matriz de atención. En el caso de hard attention, esta matriz es la identidad.

$$I = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Multiplicando nuestra matriz de atención por la matriz que contiene todos los vectores de entrada, obtenemos los vectores de salida. En este caso, repetimos, obtendremos exactamente el mismo conjunto de vectores ya que cada vector a la salida atiene únicamente a un vector a la entrada, aquel que está en su misma posición.

$$I \times \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ -0.5 & -0.5 \end{pmatrix}$$

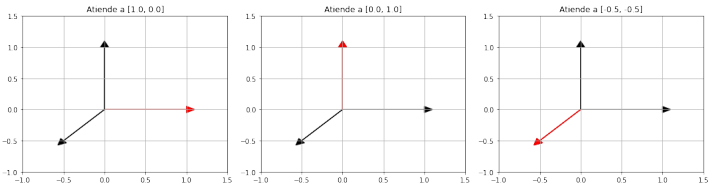

Fig. 3. Vectores multiplicados por la matriz Identidad

2) Soft-Attention: Ahora que hemos explicado qué es un mecanismo de atención y visto un ejemplo sencillo, vamos a ver otro tipo de mecanismo un poco más flexible. Si en el caso de la atención fuerte, cada vector generado presta atención simplemente a un único vector en la entrada, en el caso de la atención débil vamos a permitir prestar atención a todos los vectores a la entrada. Así pues, cada vector generado será una combinación de los inputs. En el siguiente ejemplo, cada vector generado presta un 80% de atención al vector en la entrada en su misma posición y un 10% al resto.

$$A = \begin{pmatrix} 0.8 & 0.1 & 0.1 \\ 0.1 & 0.8 & 0.1 \\ 0.1 & 0.1 & 0.8 \end{pmatrix}$$

$$A \times \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ -0.5 & -0.5 \end{pmatrix} = \begin{pmatrix} 0.75 & 0.05 \\ 0.05 & 0.75 \\ -0,3 & -0.3 \end{pmatrix}$$

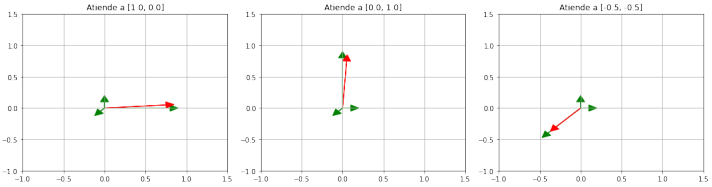

Fig. 4. Vectores multiplicados por la matriz de atención

En verde podemos ver la contribución de cada vector al resultado, mientras que en rojo podemos ver los vectores generados por nuestro mecanismo de atención. En este caso,

son muy similares a los originales ya que estamos prestando mucha atención a éstos mismos. Sin embargo, el resto de vectores pueden influir en el resultado.

3) Self-Attention: Hasta este punto la matriz de atención ha sido elegida manualmente como si de un hiperparámetro se tratara, pero para el tipo de problema al que nos enfrentamos necesitamos que la propia matriz de atención se autoconfigure pues es el objetivo de los problemas que el transformer debe solucionar, es por esto necesario el mecanismo de auto-atención o también conocido como *Scaled Dot-Product Attention*.

En este último mecanismo, cada vector es responsable de decidir por si mismo cuánta atención prestar al resto. Para ello calcularemos la similitud entre vectores. Cuanto más parecidos sean dos vectores, más atención habrá entre ellos y viceversa. En una tarea de traducción de texto, por ejemplo, a la hora de generar una palabra un mecanismo de self attention permitiría prestar más atención a aquellas palabras más relacionadas a la entrada, y no desperdiciar computación con aquellas que no tienen importancia. Esta similitud la calculamos multiplicando los vectores por si mismos y aplicando una función softmax. Este mecanismo de atención consiste en tres conjuntos de vectores K, Q y V, llamados respectivamente keys, queries y values. Utilizaremos K y Q para calcular la matriz de atención, la cual aplicaremos a V.

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

En este caso escalamos el producto de Q y K con la raíz cuadrada de su dimensión, d_k.Para que este sistema sea capaz de aprender, calcularemos los diferentes vectores utilizando perceptrones: Q = W_qX, K = W_kX, K = W_kX. De esta manera, nuestro sistema será capaz de calcular la mejor representación de X para obtener un alineamiento óptimo en el mecanismo de atención.

4) Multi-head Attention: Los autores de "Attention is all you need" [1] proponen una mejora al mecanismo de atención ya explicado, conocida como Auto-atención por Multi-cabeza.

La Auto-atención por Multi-cabeza surge con el objetivo de mejorar la capacidad de representación de datos. En el contexto de atención, esto se traduce en repetir un número determinado de veces (heads o cabezas) el mecanismo de atención por producto escalar.

A grandes rasgos, repetimos el mecanismo de atención aplicando diferentes proyecciones a la hora de obtener nuestras queries, keys y values. Una vez aplicada la atención a cada cabeza, concatenamos los resultados y aplicamos una nueva capa lineal para obtener el resultado final.

Con esto conseguimos que cada cabeza se centre en aspectos diferentes, pudiendose así tener en cuenta muchas dimensiones distintas del contexto

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O$$
$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

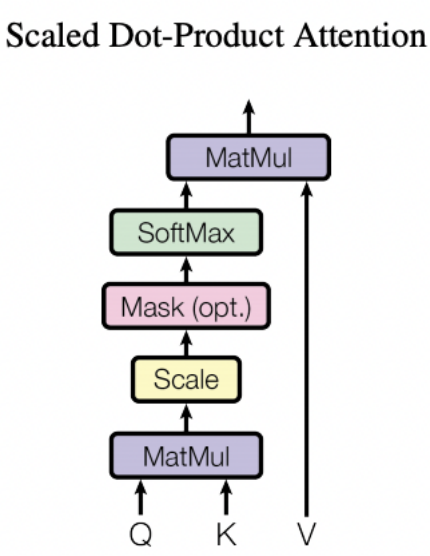

Fig. 5. Mecanismo de Auto-Atención de un Transformer

B. Codificación Posicional

Un aspecto relevante para entender el funcionamiento del Transformer es el de la codificación posicional. Esto se debe, a que a diferencia de las redes neuronales recurrentes en las que cada dato de una secuencia es procesado de forma secuencial, en un transformer los datos de la secuencia se procesan de manera simultánea. Al hacerse de dicho modo, es primordial implementar un mecanismo que añada información sobre la posición de cada dato en el vector de secuencia. Toda la codificación posicional que nos indicará la posición de cada dato en nuestra secuencia se calcula con una única funcion sinusoidal.

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

Fig. 6. Funciones senoidales que modelan la codificación posicional

$PE_{(pos,i)}$ representa la codificación de la posición en la secuencia, y la dimensión oculta . Estos valores, concatenados para todas las dimensiones ocultas, se añaden a las propiedades de entrada originales, y constituyen la información de posición. Distinguimos entre (i mod 2 = 0) y (i mod 2 = 1), que son las distintas dimensiones ocultas donde aplicamos un seno/coseno respectivamente. La salida de esta función sumada a los vectores de entrada, permite al modelo atender fácilmente a las posiciones relativas de los datos de entrada.

C. Estructura del Transformer:Arquitectura Encoder/Decoder

La arquitectura del Transformer, se basa en un Encoder y un Decoder. El encoder se encarga de analizar el contexto de la

secuencia de entrada, y el decoder es el encargado de generar la secuencia de salida a partir de dicho contexto.

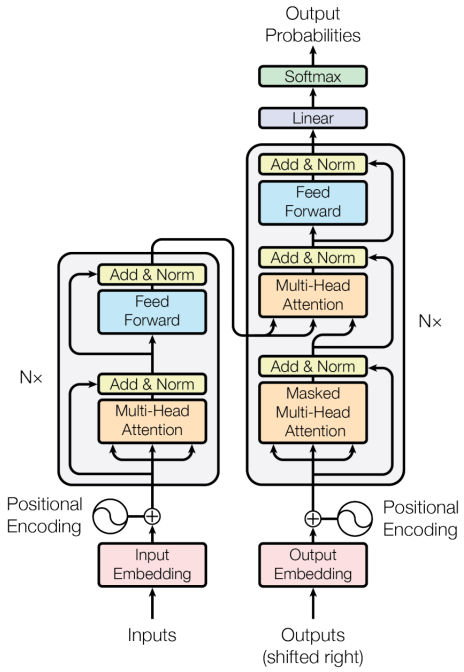

Fig. 7. Arquitectura del Transformer

El codificador consta de N bloques idénticos que se aplican en secuencia. Tomando como entrada los datos codificados a vectores junto a su codificación posicional, se pasa primero por un bloque de Atención Multi-cabeza. La salida se suma a la entrada original utilizando una conexión residual, y aplicamos una Normalización de Capa consecutiva sobre la suma. En conjunto, se calcula LayerNorm(x+Multihead(x,x,x)) (x siendo Q, K y V la entrada a la capa de atención). La conexión residual es crucial en la arquitectura de Transformer por dos razones:

i) Los Transformers están diseñados para ser muy profundos. Algunos modelos contienen más de 24 bloques en el codificador. Por lo tanto, las conexiones residuales son cruciales para permitir un flujo de gradiente suave a través del modelo.

ii) Sin la conexión residual, se pierde la información sobre la secuencia original. Recordemos que la capa de atención multicabezal ignora la posición de los elementos en una secuencia, y sólo puede aprenderla basándose en las características de entrada. Eliminar las conexiones residuales significaría que esta información se pierde después de la primera capa de atención (después de la inicialización), y con una consulta y un vector clave inicializados aleatoriamente, los vectores de salida para la posición no tienen relación con su entrada original. Es probable que todas las salidas de la atención representen información similar/igual, y no hay posibilidad de que el modelo distinga qué información proviene de qué elemento de entrada.

La normalización de capa también desempeña un papel importante en la arquitectura de Transformer, ya que permite un entrenamiento más rápido y proporciona una pequeña regularización. Además, asegura que las propiedades de los datos mantengan una magnitud similar entre los elementos de la secuencia.

Además de la Atención Multi-Cabeza, se añade al modelo una pequeña red feed-forward totalmente conectada, que se aplica a cada posición por separado y de forma idéntica a la que también se le aplica otra capa de normalización.

Tras esto pasamos las propiedades de los datos al decodificador, que como vemos en la imagen (Fig. 7) se realizan procedimientos similares al codificador por lo que no es necesario mucha más profundización en la arquitectura.

III. Metodología

A continuación se explica de forma detallada, como se ha implementado las soluciones, en primer lugar El Transformer y en segundo lugar la Red Neuronal Convolucional.

A. Implementación del Transformer

Para implementación del transformer hemos necesitado varias referencias para su realización puesto que desconociamos el funcionamiento de pyTorch y aunque poseyeramos los conocimientos teóricos la implementación era algo compleja. Es por eso que queremos referenciar algunos foros de información que nos resultaron útiles como el blog de ”SensioIA” [3], el blog de ”UvA Deep Learning Tutorials” [4] y el paper “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale” [5].

Los Transformers fueron originalmente creados con el propósito de procesar conjuntos, para aplicar los transformers a la secuencias de datos, simplemente debemos de añadir la codificación posicional a los vectores de entrada, y el modelo aprenderá por si solo que debe hacer. Así que, de la misma manera podríamos extrapolar ese mecanismo al campo de la clasificación de imágenes dando lugar así a la arquitectura Vision Transformer(ViT).

Comenzaremos pues explicando toda nuestra implementación pasando desde el uso y modificación de los datos de entrenamiento y de prueba, hasta los resultados que nos ofrece el ViT una vez entrenado:

1) Preparación del Dataset y Dataloader: En cuanto a la preparación de los datos de entrenamiento y prueba, a ambos se les aplican una serie de transformaciones.
En primer lugar se calcula la media y la desviación estándar de los datos de entrenamiento, ya que usaremos esta información para normalizar los datos correctamente.

A continuación se listan las transformaciones realizadas a los datos:

- *RandomHorizontalFlip:* Esta transformación la aplicamos solo a los datos de entrenamiento, y consiste en girar cada imagen horizontalmente con una probabilidad del 50 por ciento, ya que no queremos que la información de las imagenes dependa de su orientación.

- *RandomResizedCrop:* Esta transformación recorta la imagen dada a tamaño y relación de aspecto aleatorios. Este recorte se redimensiona después al tamaño dado, con el objetivo de que los valores reales de los píxeles cambien mientras que el contenido o la semántica general de la imagen permanece igual.

- *ToTensor:* Con esta transformación transformamos los datos al tipo Tensor, que es con el que se trabaja en PyTorch.

- *Normalize:* Finalmente, aplicamos la normalización de las imágenes, ya que ayuda a reducir la redundancia y la complejidad al examinar nuevos tipos de datos.

Por último, en cuanto a los DataLoaders, hemos utilizado en su creación una serie de parámetros que optimizan el rendimiento, estos son ”pin-memory” y ”drop-last”.

2) PatchEmbedding: En cuanto a la representación de la información de una imagen, cada imagen de tamaño NxN deberá ser dividida en $(N/M)^2$ patches con un tamaño de MxM cada uno. Estos patches representan las palabras de entrada del Transformer. En nuestro caso, tras el tratado del anterior apartado, las imágenes tienen un tamaño de 32x32, por lo que al aplicar el PatchEmbedding con un tamaño de patch igual a 4, obtendremos secuencias de 64 patches con un tamaño de 4x4.

3) Self-Attention: Para la Atención Multicabeza, en un principio intentamos implementarla desde cero, pero no fuimos capaces de que funcionara correctamente, por lo que finalmente utilizamos el módulo nn.MultiheadAttention de Pytorch.
Además, hemos implementado el módulo ”AttentionBlock” en el que usamos la versión de normalización previa de los bloques del transformer propuesta por Ruibin Xiong et al en 2020 [6].
La idea es la de, en vez de aplicar la normalización de capa entre los bloques residuales, aplicarla como una primera capa en los bloques residuales. Esta reorganización de las capas admite un mejor flujo de gradiente y elimina la necesidad de una etapa de calentamiento.
Profundizando más en la estructura del Attention-Block(también conocido como Transformer Encoder) podemos ver que pasa por varias partes, en primer lugar, se aplica la capa de normalización antes nombrada y tras esto pasamos al mecanismo de Atención Multi-Cabeza que nos devolverá un tensor con la información ya procesada al que le aplicamos una suma residual para no perder la codificación posicional y los CLS Token que enviamos junto a los parches en la entrada. Tras esto volvemos a normalizar y procesamos los datos a través de un perceptrón multicapa(MLP) y volvemos a realizar una suma residual con la misma finalidad, todo esto se repite tantas veces como número de capas le hayamos asignado al ViT.

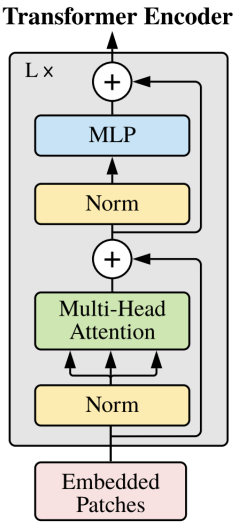

Fig. 8. ViT Encoder, imagen de Alexey Dosovitskiy et al 2020.

4) Implementación del modelo: En cuanto a la arquitectura del modelo implementado, nos hemos basado en la propuesta de Alexey Dosovitskiy en el paper ”An image is worth 16x16 words: Transformers for image recognition at scale” [4]. A continuación se muestra una imagen de la arquitectura del mismo a grandes rasgos:

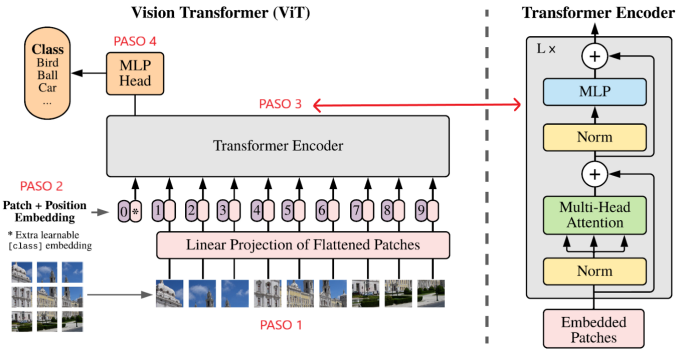

Fig. 9. Modelo VisionTransformer

En la imagen hemos definido varios pasos que nos ayudaran a entender de manera secuencial que es lo que ocurre dentro del Vit cada vez que procesamos un lote de imagenes:

i) El primer paso consiste en la realización del patch embedding(ya explicado en el subapartado 2), que nos

devuelve un tensor cuya estructura es la siguiente: [Batch_size, N_Patchs, Patch_size*Canales], estos tres elementos representan respectivamente el tamaño del lote en el que se van procesando las imágenes del dataset de entrenamiento, el número de parches en el que se divide cada imágen del lote, y el número de canales multiplicado por las dimensiones del parche(en nuestro caso 3*4*4)

ii) Para el segundo paso se procesarán el lote de imagenes parcheadas mediante una capa lineal (en nuestro caso,input_layer) con el objetivo de proyectar las imágenes a la dimensión del embbeding. A continuación se le añaden los CLS token que son definidos para usarlos posteriormente en la clasificación a través del MLP. A continuación, se añade la codificación posicional a cada parche.

iii) Para el tercer paso,se pasa por una capa de Dropout, que nos permite reducir la probabilidad de que el modelo se sobreajuste y a continuación se ejecutaría el AttentionBlock (ya explicado en el subapartado 3), tantas veces como capas se hayan definido para el transformer, en nuestro caso 6 veces.

iv) Finalmente, el tercer paso consiste en una capa MLP (Perceptrón multicapa), compuesta por una capa de normalización y una capa lineal, que toma el vector de características de salida del token CLS y lo asigna a una predicción de clasificación (Mapea dicho token de clasificación a una de las etiquetas del dataset).

B. Implementación de la Red Convolucional

Las Redes neuronales convolucionales, son un tipo de redes neuronales artificiales donde las neuronas corresponden a campos receptivos de una manera muy similar a las neuronas en la corteza visual primaria (V1) de un cerebro biológico. Este tipo de red es una variación de un perceptrón multicapa, sin embargo, debido a que su aplicación es realizada en matrices bidimensionales, son muy efectivas para tareas de visión artificial, como en la clasificación y segmentación de imágenes, que es nuestro caso.

Nuestra CNN tiene la siguiente arquitectura:

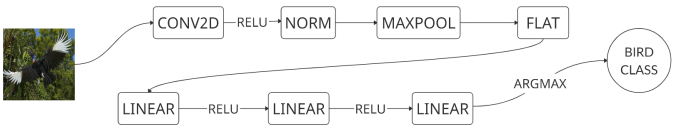

Fig. 10. Arquitectura CNN

En primer lugar se aplica una convolución, a la que le entran 3 valores como input (RGB) y devuelve 128 como output, utilizando una máscara de 5x5 y un salto (stride) de 4. Al usar un stride con valor mayor a 1, se obtiene una imagen resultante más pequeña, lo cual es útil en redes convolucionales ya que reduce la cantidad de datos a procesar. A la salida de dicha capa, se aplica una función de activación rectificador(RELU), y seguidamente se normalizan los valores utilizando una capa de normalización de lotes (BatchNorm2d), lo cual ayuda a acelerar el aprendizaje.

A continuación se aplica la operación Max Pooling, que consiste en dividir cada imagen en regiones(en nuestro caso de 4x4) y extraer el valor máximo de cada región, el cual se asignará al pixel correspondiente en la imagen de salida.
Conceptualmente, nos ayuda a analizar el contenido de una imagen por regiones para extraer la información más representativa de las mismas, lo cual es útil ya que reduce la cantidad de datos entre una capa y otra facilitando el procesamiento de las imágenes y el entrenamiento de la red, preservando la información más relevante. A la salida de dicha operación se aplica una capa flattenn, con la que aplanamos dicha salida, para adaptarla a la entrada de la siguiente capa.
Por último, tenemos tres capas lineales. Las dos primeras, usan como función de activación la función rectificador, y la última, tiene como salida un valor por cada clase (en nuestro caso 400) para cada imagen del lote, que representa la probabilidad de que la imagen pertenezca a dicha clase. Finalmente, esa salida se introduce en una función argmáx, para devolver por cada imagen del lote la clase para la que se obtuvo mayor probabilidad.

IV. Resultados

En esta sección, nos dedicaremos a comparar tanto el ViT como la CNN con el fin de ver su rendimiento y averiguar cual de los dos modelos es mejor. Para ello, hablaremos de como ha sido tanto el entrenamiento como las pruebas realizadas con el dataset de test.

A. Entrenamiento

- En cuanto al entrenamiento del ViT fueron necesarias 120 épocas con un tamaño del batch de 128 para obtener una función de pérdida(CrossEntropy) de 0.001 y una accuracy sobre el dataset de validación de 0.999, la duración del entrenamiento rondo entorno a las 6 horas(a partir de ahí el modelo se empezaba a sobreajustar).
- Para la CNN fueron necesarias 50 épocas con un tamaño del batch de 256 que nos permitió obtener una función de pérdida de 1.4 y una accuracy sobre el dataset de validacion de 0.71,la duración del entrenamiento rondo entorno a las 4 horas(a partir de ahí el modelo se empezaba a sobreajustar).

B. Testing

Para los datos de testing proporcionados por kaggle, al no poder obtener las etiquetas de cada especie, solo pudimos utilizar como medida la accuracy que realizábamos a través de

la clasificación por lo que los datos obtenidos para la accuracy por nuestros mejores modelos fueron:

- 0.802 para el ViT
- 0.551 para la CNN

Como vemos las predicciones realizadas por el transformer superaron por bastante a las de la red convolucional, pudiendo comprobar aquí su verdadera potencia.

V. Conclusiones

Durante el desarrollo de este trabajo, hemos podido investigar y profundizar en el área del Deep Learning, tanto de forma teórica, informándonos con papers, blogs, video-tutoriales, etc., como de forma práctica, enfrentándonos directamente con la implementación de los modelos propuestos.
Gracias a la parte teórica, hemos obtenido conocimientos relacionados con el Álgebra Lineal, algoritmos de atención y la lógica detrás del Transformer y las redes convolucionales.
Por otra parte, la implementación nos ha ayudado a obtener competencias con el uso de Python y las tecnologias específicas para la IA, como la librería Pytorch. Además, hemos aprendido a trabajar con tensores, una estructura de datos ampliamente utilizada en este campo, cuya función es trasladar y transformar la información en función de las distintas herramientas que utilicemos (distintos tipos de capas, datasets, dataloaders...).

Como conclusión, pese a que los resultados obtenidos con el ViT sean superiores a los de la CNN, creemos que estos datos no son concluyentes, ya que por un lado, la implementación utilizada para el modelo ViT es bastante más compleja que la utilizada en la CNN, y además, debido a que es nuestra primera experiencia implementando este tipo de tecnologías, no podemos asumir que ambas esten igual de bien implementadas. Lo mismo ocurre con el entrenamiento, ya que no podemos asegurar que ambos modelos han sido entrenados con la misma efectividad. Sin embargo, tras los conocimientos obtenidos durante la investigación, pensamos que la arquitectura del ViT tiene un potencial superior y que en un futuro cercano, e incluso en este mismo instante se estén creando tecnologías basadas en el Transformer que podrían revolucionar el mundo.

Referencias

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar,Jakob Uszkoreit,Llion Jones, Aidan N. Gomez, Łukasz Kaiser¡ Illia Polosukhin,"Attention is all you need", 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

[2] "videos acerca del Transformer de Dot CSV: https://www.youtube.com/watch?v=aL-EmKuB078

[3] Blog de "SensioIA" - https:https://juansensio.com/blog

[4] Blog de "UvA Deep Learning Tutorials" - https://uvadlc-notebooks.readthedocs.io/en/latest/index.html

[5] "AN IMAGE IS WORTH 16X16 WORDS: TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE" Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn,Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer,Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby,equal technical contribution, Brain Team.

[6] Ruibin Xiong,Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, Tie-Yan Liu "On Layer Normalization in the Transformer Architecture"Proceedings of the 37 th International Conference on Machine Learning, Online, PMLR 119, 2020.