Inteligencia Artificial aplicada a la clasificación de especies de aves

José María García[1] **y Eloy Moreno** [1]

[1] *Universidad de Sevilla, Sevilla, España*

June 27, 2022

Durante estos últimos años, el mundo de la tecnología ha sufrido grandes avances en la mayoría de sus ámbitos debido a los múltiples y revolucionarios descubrimientos obtenidos gracias al esfuerzo de miles de personas. En este artículo vamos a hablar de uno de ellos, El Transformer. Un avance que no solo ha revolucionado el mundo de la inteligencia artificial por completo sino que también ha revolucionado a todos los ámbitos con los que este tipo de arquitectura de red neuronal ha sido vinculado.

1 Introducción

Como brevemente se ha comentado, El Transformer, es un tipo de arquitectura de red neuronal que principalmente fue creada con el objetivo de mejorar la traducción así como para tareas de procesamiento de lenguaje natural, no obstante, con el tiempo se ha ido demostrando que su límite no se encontraba ahí. Debido a que este cuenta con algoritmos de propósito general, posee una gran versatilidad para adaptarse a las diferentes tareas de análisis, pudiendo tener así un uso muy amplio siendo capaz de emplearse para, además de traducir; generación de textos, resumen de textos,interpretación de audio, identificación de entidades, responder a preguntas y sobre todo, siendo este el tema principal de este artículo, la clasificación de imágenes.

Durante todo el artículo veremos cómo, con inteligencia artificial, ha sido posible clasificar más de un millar de imágenes de aves en sus correspondientes especies (llegando en torno a las cuatrocientas) de forma correcta, contando con una gran cantidad de imágenes (casi sesenta mil) con las que poder interactuar, llegando a dicha solución a través de diferentes caminos con resultados muy similares.

La clasificación de imágenes de forma automatizada y precisa hasta hace relativamente poco tiempo fue siempre una gran incógnita. Sin embargo, la situación ahora es muy diferente ya que nunca antes ha sido posible de forma tan sencilla como lo es hoy en día. En la actualidad, para realizar con éxito esta tarea podemos crear redes neuronales de diversos tipos; convolucionales, densas, perceptrones, etc...

Con esto puede parecer que la arquitectura El Transformer no es ninguna novedad ya que no hemos descubierto como hacer dicha tarea con ella, pero si podemos afirmar que el método seguido con esta arquitectura ofrece porcentajes de acierto muy superiores a los demás teniendo en cuenta el tiempo de entrenamiento, además de contar con muchos menos parámetros que entrenar, traduciéndose esto en un menor coste para la GPU. Aún así, más adelante se expondrán a fondo diversas comparativas de rendimiento y optimización entre:

- El Transformer
- Red neuronal Secuencial

Siendo esta última una red neuronal soporte que nos ayudará a entender de forma acertada como de revolucionario es El Transformer en el mundo de la inteligencia artificial.

2 El Transformer

Hemos estado hablando sobre la novedad de dicha estructura y hacia que ámbitos está enfocada, no obstante no hemos entrado a definir qué es y como funciona por dentro. En primer lugar empezaremos destacando a muy grandes rasgos cual es la idea principal de la estructura siendo esta gestionar completamente las dependencias entre la entrada y la salida con atención y recurrencia. Una vez que sabemos esa idea vamos a clarificarla definiendo su estructura y la función de cada una de las partes por la que está compuesto.

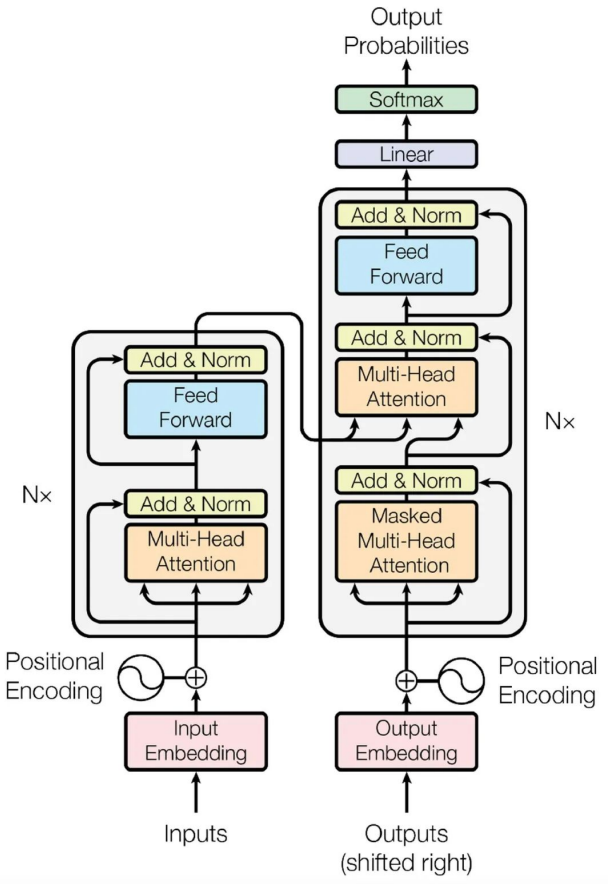

Figure 1: *El Transformer*

2.1 Bloques Encoder y Decoder

La estructura básica y el esqueleto del Transformer se basa en un bloque Encoder y un bloque Decoder. La idea principal aquí es la retroalimentación de ambas partes ya que se emplea el bloque Encoder para analizar el contexto de la secuencia de entrada (a partir de ahora será referenciada como "input" en el documento) y el bloque Decoder es el bloque encargado de generar la secuencia del salida (será referenciada como "output") a partir de este contexto. Para entender esta idea de forma más clara podemos apreciar en la parte inferior como funciona con texto en francés y su correspondiente traducción al inglés.

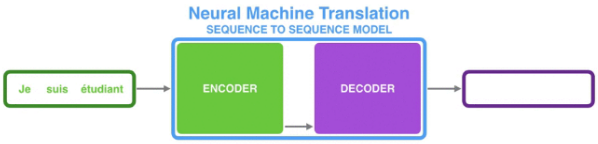

Figure 2: *Funcionamiento simplificado Encoder Decoder*

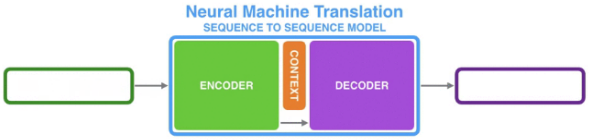

Figure 3: *Funcionamiento simplificado Encoder Decoder*

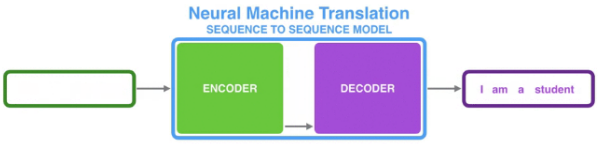

Figure 4: *Funcionamiento simplificado Encoder Decoder*

Para clarificar aún más si cabe, los dos bloques Encoder y Decoder son los resaltados en la siguiente imagen.

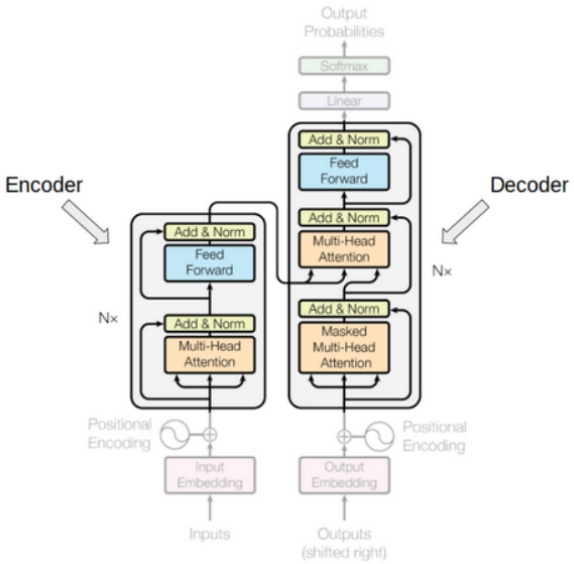

Figure 5: *Encoder y Decoder en la arquitectura*

2.2 Embedding

El objetivo principal de la arquitectura es que el bloque Encoder y el Decoder puedan trabajar bien, para ello es necesario realizar un embedding del texto, imagen o audio (diversidad de tipos de inputs) según sea el uso que tenga El Transformer. La palabra "Embedding", en este ámbito, significa una representación vectorial de cualquiera que sea su input. El objetivo de esta

transformación a vectores es que conseguir una representación mediante número de dicho input y que, además, el input que tenga una relación similar estará cerca en el espacio vectorial.

2.3 Positional Encoding

Pese a la pequeña posición que tiene esta parte en el diagrama de la arquitectura es una de las partes más importantes de esta, ya que juega un papel principal para que el modelo pueda entender la secuencia que le hemos pasado.

El Transformer no cuenta con un mecanismo de recurrencia, como otras redes neuronales, ya que utiliza el mecanismo "Multi-Head Attention" siendo esta característica uno de los principales pilares del Transformer. Como en la introducción se ha resaltado, gracias a dicha característica esta arquitectura permite capturar dependencias mucho más largas y obtener mejores resultado en un menor período de tiempo. Esta característica se verá en mayor profundidad más adelante.

Como se puede ver en el gráfico superior (Figuras 2,3 y 4) el input fluye de manera simultánea a entre el encoder y el decoder. Así pues, necesariamente debemos añadir información sobre la posición de cada input en el vector de secuencia. Para todo ello es necesario aplicar una función sinusoidal, siendo esta:

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

Figure 6: *Función sinusoidal*

2.4 Atención propia (Self-Attention)

Como se ha comentado anteriormente, El Transformer cuenta con un mecanismo de auto atención que le permite saber con qué otro input de toda la secuencia está relacionado el input que se procesa en ese instante de tiempo. A diferencia del resto de redes, El Tansformer gracias a esta cualidad no sufre de memoria a corto plazo a la hora de recordar un dato anteriormente procesado necesario a la hora de interpretar una situación en el presente. Con un ejemplo simple aplicado a texto debería verse de forma más clara esta idea.

- Pepa y Manuel han trabajado en el proyecto de IA pero ella no espera aprobar

Para otros tipos de red neuronal, la asociación de "ella" con "Pepa" era un gran problema debido a su memoria a corto plazo y la gran distacia entre estas dos palabras en el contexto de la oración. No obstante, este mecanismo de auto-atención resuelve por completo este problema, obteniéndose así una memoria prácticamente infinita (limitada por computación).

2.4.1 Atención mediante producto escalar

En primer lugar, antes de intentar entender como funciona el verdadero mecanismo de atención que emplea El Transformer, es necesario entender como funciona la versión en la que se basa dicho mecanismo. La atención mediante producto escalar se basa en tres valores de entrada:

- Valor Q: Representando a la Query que contiene el vector del input
- Valor K: Siendo este valor Key, el resto de inputs introducidos
- Valor V: Value obtiene el Valor vectorial de dicho input en el momento

Este mecanismo se basa, en el producto de los valores Q y K para después tratar al resultado con un escalado, una máscara (opcional) y la función softmax para que devuelva los valores en rango [0,1]. Una vez finalizado ese proceso volvemos a realizar el producto entre el resultado obtenido y el valor V.

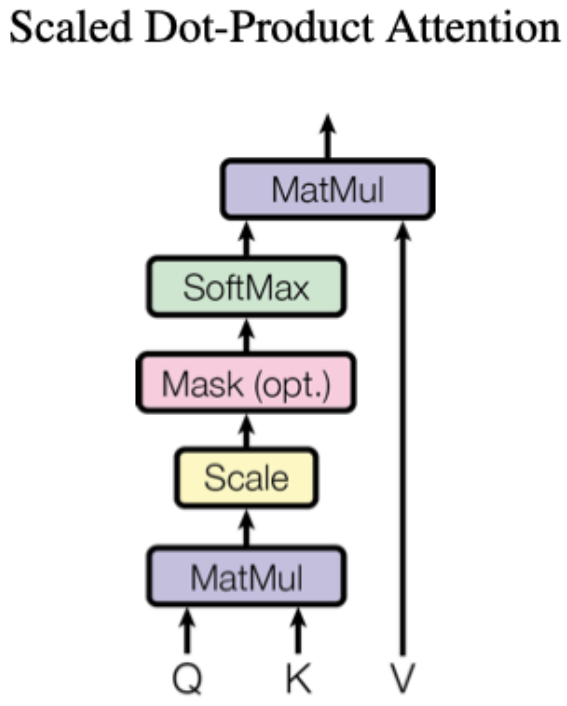

Figure 7: *Atención mediante producto escalar*

A través de la siguiente fórmula propuesta se puede calcular este índice de atención descrito e ilustrado anterioremente, destacando el producto entre los valores Q y K que intenta conseguir una aproximación de la alineación de los propios vectores para devolver

un peso para cada input, la división entre la raíz del tamaño de K para evitar valores de gran tamaño perjudiciales para softmax y el producto con V para quedarnos solo con los inputs relevantes y deshechar los inputs que no necesitemos, siendo esta una gran forma de optimizar recursos y no procesar datos de cierta irrelevancia.

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

Figure 8: *Fórmula de la atención mediante producto escalar*

2.4.2 Atención mediante diferentes cabezas

Una vez que hemos aprendido como funciona la atención mediante producto escalar, la atención mediante diferentes cabezas emplea un sistema muy parecido. No obstante, en este caso El Transformer emplea una proyección de Q, K y V en H espacios lineales. En este caso aparece una nueva variable H, que representa el número de cabezas con las que cuenta el sistema. Así pues, con un sistema formado por diversas cabezas podemos hacer que cada cabeza se centre en aspectos diferentes, para así después concatenar los diferentes resultados. Pudiendo tener múltiples representaciones de importancia de cada input, permitiendo que dicho input no sea el dominante en el contexto en el que se emplee dicho Transformer.

Con todo ello podemos afirmar el gran potencial que tiene esta arquitectura ya que nos permite aprender dependencias mucho más complejas en el mismo tiempo de entrenamiento gracias a que la proyección lineal reduce el tamaño de cada vector.

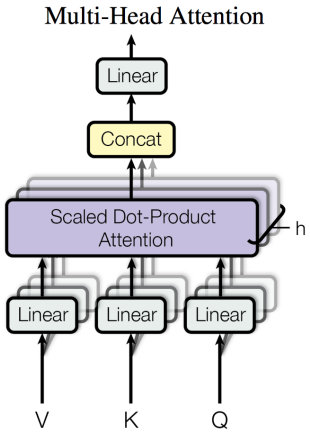

Figure 9: *Atención mediante diferentes cabezas*

Para hacernos una pequeña idea y entender de forma completa el potencial final de este mecanismo de atención mediante diferentes cabezas, en la siguiente imagen podrá apreciarse una comparativa en la traducción para que se vea de forma intuitiva y gráfica los niveles de atención entre una oración en inglés y en francés (siendo el léxico de ambos idiomas completamente diferente y en orden distinto).

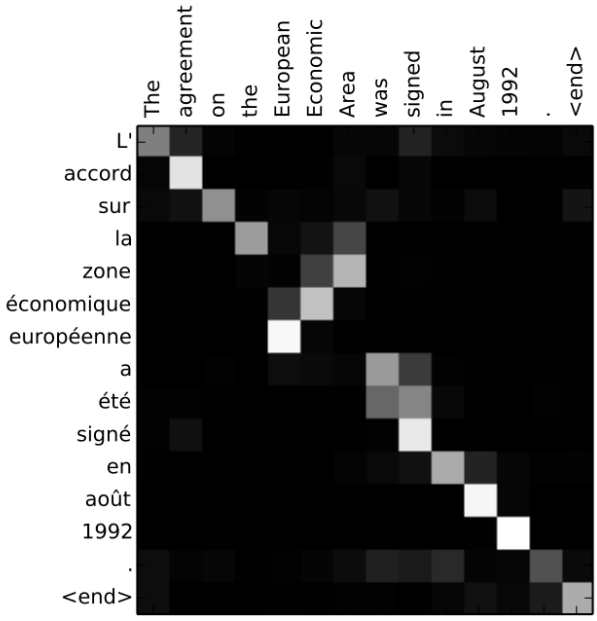

Figure 10: *Ejemplo de atención*

2.5 Funcionamiento final

En primer lugar se realiza la codificación en el bloque Encoder, estando este proceso formado por los siguientes pasos.

- Embedding del conjunto de inputs con el objetivo de transformar cada input para que el sistema pueda trabajar de forma numérica, en este caso en vectores. Gracias a esta propiedad es posible que acepte tanto texto, imágenes, audio, etc... Ya que cualquier input va a ser traducido a su forma vectorial.
- Pasamos por el componente Positional Encoding para agregarle la componente de la posición a cada vector.
- Aplicamos el mecanismo de atención de diferentes cabezas explicado anteriormente.
- Introducimos los datos que teníamos hasta ahora en la capa neuronal Feed Forward.

En segundo lugar se realiza la decodificación en el bloque Decoder, estando este proceso formado por los siguientes pasos.

- Al igual que antes hacemos un embedding pero en este caso debemos hacerlo del resultado del punto justo anterior.
- Pasamos por el componente Positional Encoding para agregarle la componente de la posición a cada vector.

- Aplicamos el mecanismo de atención de diferentes cabezas en este caso al output del resultado del punto justo anterior.
- Volvemos a emplear el mecanismo de atención, no obstante, en este caso como tenemos los resultados del bloque Encoder, empleamos dichos resultados como valor Q y K y el resultado obtenido en el mecanismo de atención anterior lo empleamos como V.
- Introducimos los datos que teníamos hasta ahora en la capa neuronal Feed Forward.
- Terminamos el proceso con una capa Lineal y Softmax para conseguir el objetivo final, obtener una probabilidad y devolver el input que tenga esta más alta como el siguiente.

3 Modelos empleados para resolver el problema propuesto

Una vez que hemos hablado sobre los beneficios del descubrimiento de la arquitectura El Transformer y sobre cómo son sus componentes y su funcionamiento a fondo, es hora de hablar de nuestras implementaciones.

3.1 Vision Transformer (ViT)

En primer lugar, para la propuesta en la que no se han utilizado librerias externas hemos utilizado un modelo de Vision Transformer (a partir de ahora será referenciado como "ViT" en el documento). No obstante, ¿cuál es la diferencia entre este ViT y la arquitectura El Transformer explicada anteriormente?

Para entender esta arquitectura de forma correcta debemos dividirla en tres componentes.

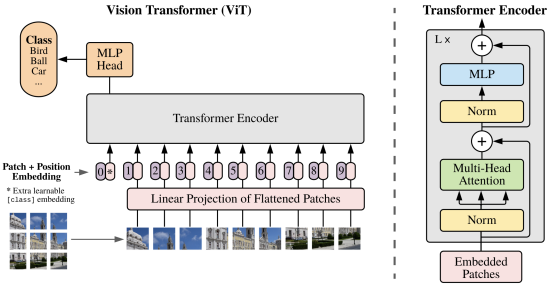

Figure 11: *Ejemplo de atención*

3.1.1 Embedding

Este modelo ViT esta formado por un bloque Encoder que está minuciosamente pensado para que reciba ciertos inputs, en este caso la imagen del ave separada en diferentes pedacitos (parches) siendo estos proyectados. Tras este paso, deberemos calcular para cada uno de estos Patch y Position Embedding.

Para la clasificación, concatenamos una clase embedding con las otras proyecciones de parches, cuyo estado en la salida sirve como información de clase. Este token de clase adicional se añade al conjunto de tokens de la imagen que se encarga de agregar la información global de la imagen y la clasificación final. Es capaz de aprender esta agregación global mientras pasa y aprende a través de las capas de atención. También añadimos una posicional embedding a los parches, para establecer un cierto orden en los parches que hemos creado como entrada. No obstante no debemos subestimar esto último último ya que es muy necesario para que el modelo aprenda el orden de la imagen ya que los Transformers no son capaces de recordar el orden de las entradas. Si los parches de la imagen se reordenan, se pierde el significado de la imagen original. Por ello, añadimos ese posicional embedding a nuestros parches para seguir la secuencia.

Figure 12: *Patches desordenados*

3.1.2 Encoder

Este bloque Encoder es muy similar al explicado anteriormente procedente de la arquitectura de El Transformer. Los únicos cambios significativos son el cambio de orden entre la normalización y la capa de atención, siendo esta de tipo diferentes cabezas (explicado en el apartado 2.4.2), y la normalización y la capa Feed Forward o en este caso MLP (Perceptrón Multicapa) haciéndose siempre después la conexión residual.

3.1.3 MLP Head

La arquitectura de este Transformer está diseñada para que el output del Bloque Encoder solo emplee la primera salida para conectarse a un MLP (Perceptron Multicapa) que hará la clasificación. Esta primera salida viene dada por la primera entrada la cual es una entrada vacía (creada en la primera posición de inputs antes que todos los patches) que solo sirve para realizar la conexión al clasificador MLP.

3.2 Red Neuronal Secuencial

Tras la construcción de nuestro propio ViT, pasamos ahora a construir una red neuronal sencilla que nos va a servir de gran ayuda para tener una comparativa entre ambas redes y ver sus respectivos resultados al final de este documento.

En primer lugar debemos aclarar que el objetivo de esta segunda red neuronal es exactamente el mismo que el del ViT, por lo que para su creación decidimos que debía contener lo siguiente.

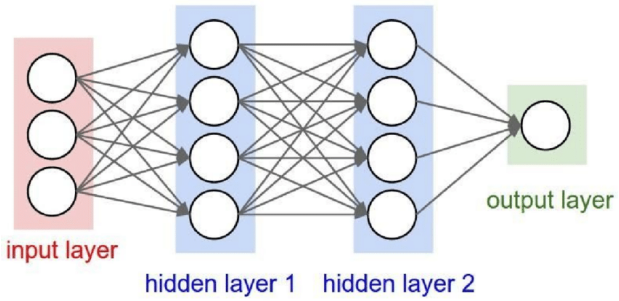

Figure 13: *Ejemplo de Red Neuronal Secuencial*

3.2.1 Algoritmo de descenso de gradiente en mini lotes

La red neuronal debía seguir el algoritmo de descenso de gradiente en mini lotes, para entender que es este algoritmo y como funciona podemos decir que este algoritmo se encarga de dividir el conjunto de datos de entrenamiento en pequeños lotes que se utilizan para calcular el error del modelo y actualizar los coeficientes del mismo. Este algoritmo no es único en su ámbito, existen más como el gradiente estocástico o el gradiente por lotes, pero este trata de encontrar un equilibrio entre la robustez del descenso de gradiente estocástico y la eficiencia del descenso de gradiente por lotes. Suele ser la implementación más común del descenso de gradiente utilizada en el campo del deep learning.

3.2.2 ResNet18

Para una eficiencia tanto computacional y como del tiempo decidimos que sería una gran idea contar con un modelo ya preentrenado como es ResNet18, por lo que los resultados con menos tiempo de entrenamiento eran mucho mejores que sin el preentreno de ResNet18.

3.2.3 Funcionamiento Secuencial

Puesto que es una red neuronal secuencial, su funcionamiento es realmente simple ya que, debido a su secuencialidad es necesario que su entrenamiento también lo sea, por lo que el método de entrenamiento de dicha red en un bucle que finaliza cuando el número de épocas llega al número de épocas solicitado.

4 Decisiones tomadas y problemas durante el desarrollo

Una vez que hemos entendido como funcionan nuestras redes, es necesario también entender cuales han sido nuestras decisiones a la hora de la creación del mismo.

- Una de las primeras decisiones a las que nos hemos enfrentado fue el entorno de desarrollo que íbamos a utilizar, siendo en primera instancia Google Colab ya que fue uno de los aconsejados por el profesorado. No obstante, diversos problemas con la memoria RAM y con Cuda hicieron replantearnos cambiar de entorno, por lo que tras un fugaz paso por la plataforma Kaggle decidimos que lo mejor era instalar el entorno local y usar nuestro propio hardware para evitar problemas (ya que este es de gama alta).
- El principal e inicial problema ha sido la falta de información que teníamos sobre el tema el cual solucionamos haciendo una larga investigación en diversas páginas webs y revisando videos que eran de gran utilidad durante varios días incluso antes de ponernos con algo de código. Una apreciación a recalcar es que la inmensa mayoría del contenido necesario era en inglés.
- Una de las siguientes dificultades fue la división de imágenes procedentes del dataset ofrecido. Tuvimos ciertas dudas con este tema que fueron resueltas con un correo electrónico al profesor. Con todo ello realizamos la división de la parte de "Entrenamiento" y de "Testeo" teniéndola que hacer a mano cogiendo las primeras 24 imágenes de cada especie de pájaro para el testeo y las demás para el entrenamiento, solucionándose así el problema. Sabemos que es una decisión muy extrema pero no encontramos una forma automatizada de hacerlo.
- Tras esto, otra dificultad fueron los diversos fallos al importar las librerías de pytorch en local. Para solucionarlo debimos instalar correctamente algunas librerías necesarias para Pytorch.
- Una vez conseguimos tener una red que podía entrenarse (Red Neuronal Secuencial), el principal problema fue que no conseguíamos entender de forma correcta esa red, teniendo unos resultados mediocres de precisión (0.05 por cierto) aún entrenándola una gran cantidad de tiempo. Después de muchas pruebas y muchos intentos decidimos desechar la red actual creada y nos propusimos crear otra para evitar fallos de base.
- Finalmente tras tener creados ambos clasificadores funcionando de forma correcta, nos faltaba pasar el resultado obtenido a un archivo .csv. Nuestras dudas fueron también resueltas por correo electrónico con el profesor. Con las recomendaciones del profesor y una búsqueda descubrimos que existía un método que transformaba matrices en csv por lo que creamos esas matrices a partir de arrays hechos con dos listas con la id y las prediciones de la imagen. Con todo lo anterior finalmente conseguir el objetivo y subirlos a la plataforma Kaggle.

5 Resultados obtenidos y comparativa

Finalmente, tras estudiar en profundidad el funcionamiento de Pytorch, exponer y realizar correctamente los clasificadores de imágenes, es hora de ponerlos a prueba con el conjunto de imágenes que nos daban para ese objetivo, submission test. Para analizar los resultados, contaremos con una tabla y una gráfica por cada resultado obtenido.

5.1 Resultados de la red neuronal secuencial

En primer lugar vamos a ver el desempeño de la red neuronal secuencial durante el proceso de entrenamiento.

Red Neuronal Secuencial		
Dataset	Porcentaje acierto	Época
$Train$	60.41	38
$Test$	62.28	38

Table 1: *Resultados simplificados Red Neuronal Secuencial sin ResNet18*

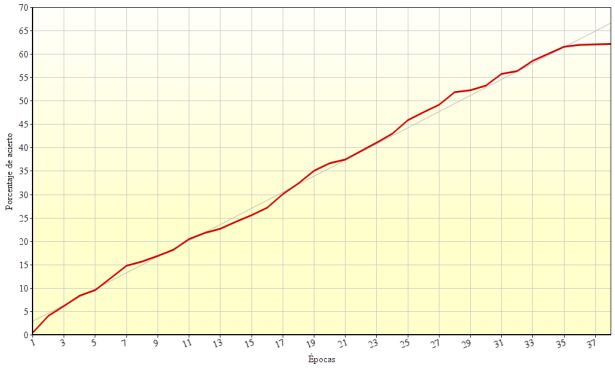

Figure 14: *Gráfico entrenamiento Red Neuronal sin preentrenamiento*

En este caso, esta tabla y el correspondiente gráfico muestra como la red neuronal secuencial ha actuado en primer lugar sin un modelo preentrenado como soporte para la clasificación. Esto nos ofrece unos resultados aceptables, pero a cambio de un gran coste computacional, energético y temporal.

Red Neuronal Secuencial		
Dataset	Porcentaje acierto	Época
$Train$	83.52	5
$Test$	85.08	5

Table 2: *Resultados simplificados Red Neuronal Secuencial con ResNet18*

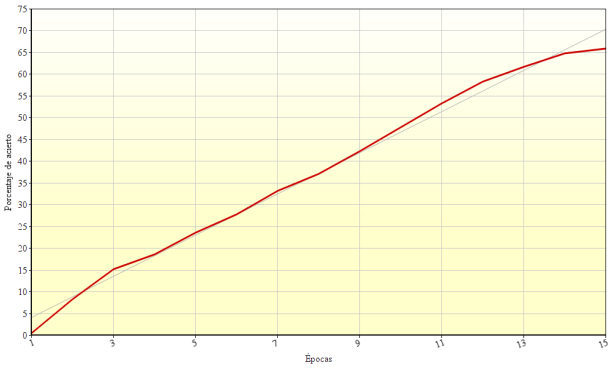

Figure 15: *Gráfico entrenamiento ViT*

Como podemos visualizar en los datos superiores, vemos como los resultados han sido muy decentes y mejores que los anteriores, en gran parte gracias a aplicar el modelo preentrenado ResNet18, ya que este ha sido de gran ayuda para conseguir unos buenos resultados a base de muchas menos épocas de entrenamiento. Lo que se traduce en una mejor optimización en todos los ámbitos.

5.2 Resultados del ViT

En segundo lugar vamos a ver el desempeño del ViT durante el proceso de entrenamiento.

ViT		
Dataset	Porcentaje acierto	Época
$Train$	63.78	15
$Test$	65.92	15

Table 3: *Resultados simplificados ViT*

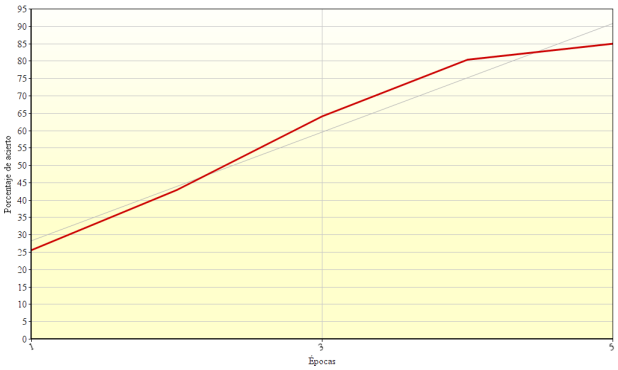

Figure 16: *Gráfico entrenamiento Red Neuronal con preentrenamiento*

Observando los datos vemos como nuestro ViT ha sido capaz de conseguir un buen resultado tras una cantidad moderada de épocas. Lo que significa que su papel desempeñado ha sido correcto y moderado durante las pruebas realizadas con él.

5.3 Resultados con una imagen propia de mascota doméstica

Entrenar los clasificadores para cumplir el objetivo de clasificar correctamente unas imágenes previamente obtenidas está bien pero, ¿y si los probamos con una imagen de una mascota propia?

En este caso probamos esta imagen sacada en el momento.

Figure 17: *Imagen de mascota*

Una vez importada, solo quedaba mostrárselo a los clasificadores para que hiciesen su trabajo, estos los resultados siendo RNSA la Red Neuronal Secuencial sin ayuda de ResNet18 y RNSB la Red Neuronal Secuencial que si cuenta con la ayuda de ResNet18.

Resultados		
Clasificador	Porcentaje acc	Predicción
RNSA	62.22	R F LOVEBIRD
RNSB	85.08	R F LOVEBIRD
ViT	65.92	R F LOVEBIRD

Table 4: *Resultados simplificados ViT*

Viendo los resultados podemos afirmar que todas fueron correctas ya que en español, esta especie es conocida como "Inseparable" siendo su traducción al inglés "Love bird" por lo que todas hicieron un buen trabajo como clasificadoras de imágenes.

5.4 Resultados en la plataforma Kaggle

Por lo general cuando los entrenamientos de ambas redes finalizaban, los resultados se pasaban a formato csv y los mejores se probaban en Kaggle. Normalmente, los resultados solían ser ligeramente superiores cuando Kaggle hacía la predicción y no nosotros, por lo que tomamos eso como una buena señal de trabajo realizado correctamente.

5.5 Comparativa entre ambos resultados

Tras analizar los resultados plasmados en las tablas anteriormente mostradas, se pueden sacar diversas conclusiones muy interesantes acerca del rendimiento de los diferentes modelos y su versatilidad. Por una parte, contamos con la Red Neuronal Secuencial, una red verdaderamente simple, sin demasiada tecnología,al contrario que la arquitectura de ViT, que sin un modelo preentrenado que la reespalde no puede competir con ViT ya que se queda muy lejos de los resultados obtenidos por este tanto en porcentaje de acierto, pero sobre todo en optimización. No obstante, por otro lado tenemos un ViT que ejerce de forma correcta su función con buenos resultados, pero que es eclipsado por una Red Neuronal Secuencial que tiene un modelo preentrenado detrás.

RED NEURONAL SECUENCIAL SIN RESNET18 < *ViT* < *RED NEURONAL SECUENCIAL CON RESNET18*

Figure 18: *Comparativa*

6 Conclusión

Para finalizar, analizando los resultados y la comparativa anterior del rendimiento de los clasificadores, podemos llegar a la conclusión de que un ViT está muy por encima en cuanto a rendimiento y optimización, gracias a toda su arquitectura sofisticada, de una red Neuronal Secuencial, siempre y cuando esta no cuente con el apoyo de un modelo entrenado como VGG, Xception o, en este caso, ResNet18. No obstante la conclusión no se queda ahí, a lo largo de este documento y sobre todo a lo largo de las semanas de desarrollo detrás del propio, nos hemos dado cuenta de los grandes pasos que se han dado sobre todo estos últimos años en el mundo de la inteligencia artificial. Grandes cambios

que han propiciado que áreas nunca antes vistas relacionadas con la inteligencia artificial ahora no puedan permanecer si ella, como la medicina o el deporte. Para terminar, nos hemos dado cuenta de que la inteligencia artificial es un mundo en contínua y rápida evolución al que no se le puede poner un límite porque aún queda mucho por venir, teniendo certeza plena en que, en un futuro cercano, todo lo aprendido para realizar este documento será reemplazado por algo mejor y más eficiente.

7 Bibliografía

- Comprensión y funcionamiento de redes neuronales, DotCSV
- Comprensión del algoritmo de descenso de gradiente en mini lotes, MachineLearningMastery
- Compresión de ViT, TheIaSummer
- Compresión de Transformers, ViT y Pytorch, Juansensio
- Idealización de estructura de Transformers, TheMachineLearners
- Uso de Latex, Manual de latex
- Funcionamiento de redes neuronales, If else stament
- Funcionamiento de un clasificador en Pytorch, Pytorch
- Funcionamiento de redes neuronales en Google Colab, Sanchit Tanwar
- Modelos de secuencia, TheMachineLearners
- Pasar las predicciones a formato CSV, Pandas
- Atention Is All You Need, Arxiv
- Crear tablas a partir de arrays, Pandas
- Pasar las predicciones a formato CSV, Pandas